Università degli Studi di Napoli
Federico II

Ph.D. Thesis

Learning Paradigms for Neural Networks: from Backpropagation to Locally Backpropagated Forward-Forward

by

Fabio Giampaolo

Tutor: **A. Mercaldo**
Supervisor: **F. Piccialli**
Co-Supervisor: **S. Cuomo**

December, 2023

Abstract

Nowadays, the realm of neural networks, an indispensable component of modern artificial intelligence, is marked by transformative developments and profound insights, but not without its challenges. These systems, once inspired by rudimentary biological models, have rapidly evolved into fundamental tools for a vast array of research and applications. Within this framework, a detailed understanding of the foundational components and learning processes of these networks is essential: such knowledge enables the development of new architectures, improved learning methods, and refined neural components to enhance results in existing tasks and starting to address new ones. So, every part of a neural network offers a chance for innovation, emphasizing the importance of comprehensively understanding them for both their current applications and future roles in computational research.

Contents

Chapter 1

Preface

In recent decades, technological advancements have been driven by the progression of machine learning [1], with a special emphasis on the capabilities of neural networks. Central to this evolution is the principle of computational methods adapting from data, wherein neural networks have become the predominant actors.

Modern artificial neural networks, characterized by layers of interconnected nodes, have shown impressive competence across various fields [2, 3]: for instance, they underpin the accuracy of facial recognition systems, thereby innovating authentication procedures; they serve as the computational brains of autonomous vehicles, assisting in real-time decision-making; in healthcare, neural networks aid doctors in diagnostic processes, analyzing medical images, and offering predictive insights. The domain of linguistics, once uniquely human, is now also accessed by neural networks, catalyzing developments in natural language processing, real-time language translation, and interactive voice response systems [4, 5]. These networks are, moreover, integral to recommendation algorithms, adapting to user preferences and behaviors, and in online search, where these models comb through vast data repositories providing relevant outcomes.

Beyond high-tech applications, neural networks influence commonplace daily tasks - be it predicting meteorological patterns or streamlining industrial logistics. From agriculture to finance, the utilization of neural networks is pervasive and transformative.

1.1 Historical Notes on Artificial Neural Networks

The concept of artificial neural networks (ANNs) found its inception in the 1950s, drawing inspiration from the intricate workings of the human brain. Although the initial ANNs were simplistic and possessed limited capabilities, they laid the cornerstone for the evolution of more sophisticated networks. One of the pioneering contributions to artificial neurons came in 1943 when Warren McCulloch and Walter Pitts proposed a foundational model - a simple mathematical function emulating the behavior of biological neurons: termed the McCulloch-Pitts neuron, it featured multiple inputs and one output. Inputs underwent weighted multiplication, where the weights were fixed and the same for all the inputs, and the summation of these products passed through a nonlinear function, tipically a Heaviside one, to yield the output [6].

The pivotal moment arrived in 1957 when Frank Rosenblatt introduced the perceptron, marking the first ANN capable of learning from data. The perceptron was a first example of network composed by a single-layer of artificial neurons (also called perceptrons). This single layer took input values, applied weights to those inputs, different for each neuron, computed a weighted sum, and then applied a threshold function to produce an output [7]. Training the perceptron involved employing a supervised learning algorithm known as the perceptron learning rule: this rule orchestrated adjustments to the perceptron's weights, ensuring accurate outputs for given inputs. Despite its initial success, the perceptron exhibited limitations, in particular its inability to handle non-linearly separable patterns like the XOR problem. In 1969, Marvin Minsky and Seymour Papert authored "Perceptrons", a book critical of the perceptron's potential and casting doubts on the future of ANNs. Minsky and Papert argued that perceptrons were too constrained to address real-world challenges [8]. Nevertheless, a dedicated group of researchers continued to explore ANNs, leading to a resurgence of interest in the 1980s. This revival stemmed from the development of new algorithms and the availability of more powerful computing resources. However, the early algorithms for training neural networks were often rudimentary: the perceptron learning rule, ad discussed before, or the delta rule, which exhibited more potential than the perceptron learning rule, were anyway restricted in their capacity to comprehend complex patterns [9].

A pivotal breakthrough emerged when David Rumelhart, Geoffrey Hinton, and Ronald Williams introduced backpropagation in 1986: rooted in the chain rule of calculus, backpropagation revolutionized the training of multilayer perceptrons (MLPs). This iterative algorithm adjusted the ANN's weights to minimize errors, expanding the ANN's capacity to solve a broader range of problems, including those whose data exhibited not linearly separable patterns [10]. The subsequent decade witnessed ANNs being integrated into various applications, such as image recognition and natural language processing. The 2000s brought forth deep learning, a subfield harnessing ANNs with multiple layers of neurons. Deep learning architectures like convolutional neural networks (CNNs) and recurrent neural networks (RNNs), for example, spearheaded significant advancements in various applicative tasks.

It could be said that the development journey of ANNs has been characterized by perseverance and innovation, with each era building upon the foundations of the preceding one. From their humble beginnings, ANNs have evolved into a potent tool capable of addressing a diverse array of complex problems [5, 11, 12]. The future of ANNs is promising, as they are poised to assume an increasingly pivotal role in the realms of machine learning and artificial intelligence, while continuing research focuses on addressing current challenges and broadening their applicability.

1.2 Importance of the Learning Process

In the realm of neural networks and AI, grasping how these models learn from data is pivotal: their varied applications, from medical diagnostics to autonomous machinery, highlight their significant influence on society; yet, with their achievements come inherent challenges that accentuate the necessity of understanding their learning processes. Delving into the learning mechanisms of neural networks requires insight into complex data patterns and relationships [10, 13, 14]. A deep comprehension of these networks is vital for improving model performance, developing efficient architectures [15], refining training methods, and optimizing hyperparameters [3, 16]: this understanding is key to advancing AI innovations and ensuring these systems align with evolving challenges. Further insight into the learning mechanisms of neural networks can, for example, enhance their robustness, promoting improved generalization and defense against adversarial situations, or given that these networks often train on datasets with inherent biases, comprehending and mitigating these biases is crucial for the development of ethically responsible AI systems [17, 18]. Ultimately, analysing neural network learning processes aligns with the broader, and most futuristic, objective of the AI: the pursuit of artificial general intelligence [19, 20].

Such understanding paves thus the way for models with the adaptability to handle diverse tasks, reflecting the flexibility of human cognition. By shedding light on the mechanisms of machine data interpretation and learning, this research contributes to the harmonious integration of artificial and human intelligence, amplifying their combined potential for societal advancement.

1.3 Research Objectives and Contributions

Neural networks have cemented their position as a cornerstone in both scientific and industrial applications due to their unparalleled ability to learn from data. This work delves into the intricate mechanisms that govern a neural network's learning capabilities: by dissecting a neural network and understanding its components and their interactions during the training process, we aim to enhance its utility in solving new and challenging tasks and foster further advancements in the field.

In the exploration of the fundamental components and standard learning dynamics of neural networks, this work sheds light on the essential building blocks of these versatile models: a deep comprehension of their current operational mechanisms is crucial to harness their full potential. Utilizing this knowledge allows for the judicious adaptation of various components and strategies, paving the way for the development of methodologies that are adept at addressing real-world challenges. Within this context, a comprehensive study encompassing various applicative fields, focusing on the creation of a forecasting framework has been presented. This framework leverages the merits of neural learning, demonstrating both robustness and feasibility, especially in the intricate task of predicting time series.

The last part of this work is dedicated to exploring alternatives to the conventional learning paradigms for neural networks. By delving deeper into learning dynamics and examining diverse learning strategies, we open the door to novel opportunities in modifying the foundational mechanisms of the training process. The flexibility and adaptability of neural networks become apparent as we navigate through recent paradigms and strategies, highlighting their intrinsic ability to evolve. With this perspective, we present methodological research on an alternative learning strategy to backpropagation: his investigation underscores not only the immense promise and relevance of this research domain but also its applicability in tackling unresolved issues, such as the challenges associated with heterogeneity in federated learning.

In conclusion, the vast landscape of neural networks, with its intricacies and nuances, continuously presents challenges and opportunities. Delving deep into the mechanisms of how these networks learn is not merely an academic exercise but a necessity. As technology advances and demands more sophisticated solutions, the importance of understanding and innovating within the realm of neural networks becomes paramount. Every breakthrough, every nuanced understanding, pushes the frontier of what's possible. The amalgamation of thorough understanding and innovative approaches will continually redefine what is achievable. Such rigorous and continuous exploration holds the potential to significantly impact numerous fields, solidifying the pivotal role of neural network research in the advancement of science and technology.

Part I

Fundamentals of Neural Networks

Chapter 2

Components of a Neural Architecture

2.1 Introduction

Artificial Neural Networks (ANNs) are situated at the confluence of computational science and biologically-inspired modeling. They reflect the continuous aspiration to harness computational constructs that are reminiscent of the cognitive capabilities of the human brain, thereby equipping machines with the ability to learn from data and perform intricate tasks. Previously, many tasks that ANNs now excel at were regarded as exclusive domains of human cognition. The meteoric rise of ANNs can be traced back to a synthesis of factors: progressive algorithmic innovations, significant enhancements in computational infrastructure, and the unparalleled proliferation of extensive datasets [3]. Together, these elements have augmented ANNs, transforming them into versatile tools exhibiting unparalleled efficacy across an array of domains.

Central to the architecture of ANNs is the concept of the artificial neuron. While this computational unit is inspired by its biological counterpart, it's crucial to underscore the pronounced distinctions in both its structural and operational paradigms. Within ANNs, data processing and signal transmission are facilitated via interconnected networks of these artificial neurons. In subsequent sections, we shall explore the nuances of artificial neurons, their activation functions, and the hierarchical organization typical of these networks. Artificial Neural Networks (ANNs) represent an intricate amalgamation of mathematical exactitude, computational strength, and engineering skill. A profound understanding of these underlying principles is crucial for comprehending their capabilities, intrinsic complexities, and the dynamic avenues of research within the domain of neural networks.

2.2 The Neuron: Foundation of Neural Architecture

As said, at the core of neural architectures lies a pivotal component: the neuron. Serving as the fundamental building block, the neuron encapsulates the basic principles and mechanisms that give rise to the intricate networks we recognize in artificial neural systems. It draws inspiration from its biological counterpart in that it receives multiple inputs, processes them, and yields a singular output. However, unlike its biological counterpart, which operates through electrochemical signals, the artificial neuron deals with numerical values and mathematical operations. At a high level, the operation of an artificial neuron can be conceived as a two-step process:

1. A weighted linear combination of its inputs.
2. The application of a non-linear transformation, typically represented by the activation function λ, to this combination.

From a mathematical point of view, given a classical artificial neuron receiving n inputs, the output o is expressed as:

$$o = \lambda \left(\sum_{i=1}^{n} w_i x_i + b \right) \tag{2.1}$$

where:

- x_i are the input values to the neuron.

- w_i represents the weights, which are numerical parameters that determine the influence or importance of the corresponding input x_i on the neuron's output.
- b is the bias term. It can be conceived as an offset or threshold that adjusts the neuron's output independently of its inputs.
- λ denotes the activation function, which introduces non-linearity to the neuron's operation. The discussion on various types of activation functions will be covered in detail in a subsequent section.

Weights and biases are vital for the neuron's learning capability: initially, they might be set randomly or based on specific strategies; however, as the neural network learns, these values undergo refinement. Weights, specifically, calibrate to determine the influence of each input on the output. Training algorithms, like gradient descent, iteratively fine-tune these weights to bridge the gap between the network's predictions and the actual targets. In essence, they encapsulate the network's acquired knowledge. Bias, on the other hand, isn't merely an additive afterthought. It's fundamental in granting the neuron an added flexibility: while weights modulate the inputs, the bias adjusts the overall output, crucial when input scaling isn't enough. For instance, when all inputs are zero, in the absence of a bias, the pre-activation output would be unequivocally zero. Bias ensures the neuron can still activate, shifting the activation function to better align with the data.

Having understood the foundational workings of the classical artificial neuron and its mathematical representation, it is crucial to recognize that the landscape of neural architectures is vast and varied. Beyond this basic neuron structure, advancements in neural networks have introduced specialized neuron types tailored to handle specific tasks and data modalities. Each of these distinct neuron types builds upon the foundational principles discussed here, yet introduces nuances and modifications that make them uniquely suited to their intended applications.

2.2.1 Convolutional Neurons

Convolutional Neurons form the core components of Convolutional Neural Networks (CNNs), designed specifically for processing grid-like structured data, most notably images. The underlying architecture draws its inspiration from the animal visual system, where specific neurons respond selectively to particular regions in the visual field, identifying localized patterns [21]. Neurophysiologists David Hubel and Torsten Wiesel, during the 1960s, conducted comprehensive studies on the mammalian visual system, leading to the discovery of 'simple' and 'complex' cells in the cat's visual cortex. Their work indicated a hierarchical structure in visual processing, where basic features combine to form more complex patterns. Leveraging these biological observations, Kunihiko Fukushima introduced the Neocognitron in 1980, an early artificial neural network model. This model, with its layers designed for local feature extraction and pooling, demonstrated robustness against input variations, a principle that remains central to current CNN architectures [22].

The wide-scale recognition and refinement of CNNs can be attributed to the efforts of Yann LeCun and his team in the late 1990s. Their introduction of the LeNet-5 architecture, aimed at recognizing handwritten and machine-printed characters, integrated convolutional operations, subsampling, and backpropagation training. This architecture highlighted the practical efficacy of CNNs [23]. The prominence and ubiquity of CNNs in today's landscape can be traced back to a synthesis of biologically-informed insights, algorithmic advancements, and empirical successes. As computational resources expanded and large datasets became available, CNNs began to surpass conventional image analysis techniques, solidifying their central role in the deep learning landscape.

Mathematically, the operation of a convolutional neuron is captured as a convolution between the input data (often an input feature map) and a filter or kernel. Let X represent the input feature map and K denote the filter. The convolution operation at coordinates (j, k) is articulated as:

$$o_{ij} = \lambda \left(\sum_m \sum_n X_{i+m,j+n} K_{m,n} + b \right) \tag{2.2}$$

Where:

- o_{ij} signifies the convolution's output at location (i, j).
- λ stands for the activation function, infusing non-linearity.
- $K_{m,n}$ represents the filter weights, with the filter methodically sliding over the input data to discern local patterns.
- b is a bias term, augmenting the neuron's flexibility.
- The twofold summation traverses the spatial dimensions of the filter.

A notable characteristic of the convolutional operation is its translational invariance. This ensures that once a CNN learns a specific feature in a certain region of an image, it can detect the same feature in different locations without the need for additional learning. As a result, CNNs possess the ability to recognize patterns irrespective of their spatial location within the input data. This makes them particularly effective for pattern recognition tasks, even when subjected to diverse transformations [24].

2.2.2 Recurrent Neurons

Recurrent Neurons are the cornerstone of Recurrent Neural Networks (RNNs), primarily designed to handle sequential or temporal data. Unlike feedforward neural architectures, the essence of recurrent neurons lies in their ability to retain and utilize historical information. This "memory" capability is achieved by incorporating feedback loops, where the neuron's output from a previous time step is recycled as an input in the subsequent time step.

The genesis of Recurrent Neural Networks (RNNs) dates to the 1980s, marked by pivotal work from Elman who introduced the Simple Recurrent Networks (SRNs), also known as Elman networks. These networks featured feedback loops within hidden layers, enabling them to retain previous states and thus laying the groundwork for later recurrent architectures [25]. In a similar timeframe, the emergence of Jordan networks took place, characterized by their unique recurrent connection from the output back to the hidden layer [26].

Notwithstanding these initial breakthroughs, the broader utilization of RNNs was stymied by the challenging vanishing or exploding gradient problem, complicating their training over long sequences. This significant barrier was highlighted and addressed by Hochreiter, Schmidhuber, and colleagues: their research not only provided clarity on the gradient problem but also introduced the Long Short-Term Memory (LSTM) network as a solution [27]. With its innovative gate-based architecture, encompassing input, forget, and output gates, LSTMs successfully circumvented the constraints of conventional RNNs, signifying a transformative moment in the domain of sequence modeling and prediction.

Subsequent to the introduction of LSTMs, another variant of RNN emerged to address similar challenges – the Gated Recurrent Unit (GRU) [28]. This unit streamlined the LSTM architecture by integrating the forget and input gates into a single "update gate". This simplification not only reduced the model's parameters, making it computationally more efficient, but also maintained competitive performance in sequence modeling tasks. The evolution of both LSTMs and GRUs underscores the continuous effort within the research community to optimize recurrent architectures for improved handling of sequential data while managing computational complexities.

Mathematically, the operation performed by a basic recurrent neuron can be articulated as follows. At a given time step t:

$$h_t = \lambda_h \left(w_{hh} h_{t-1} + w_{xh} x_t + b_h \right) \tag{2.3}$$

$$o_t = \lambda_o \left(w_{ho} h_t + b_o \right) \tag{2.4}$$

Where:

- x_t denotes the input at time t.
- h_{t-1} represents the hidden state (or "memory") from the previous time step.
- o_t is the output at time t.
- w_{xh} and w_{hh} are the input and recurrent weight matrices, respectively.
- w_{ho} is the weight matrix for generating the output from the hidden state.
- b_h and b_o are bias terms.
- λ_h and λ_o are activation functions.

The above formulation elucidates how an RNN captures sequential information: the hidden state h_t is a function of both the current input x_t and the previous hidden state h_{t-1}. This recurrent nature allows the network to "remember" patterns or sequences across time steps. This basic formulation, moreover, can be extended to more complex architectures such as LSTMs and GRUs, as will be shown in following sections. These structures have intricate gating mechanisms that control information flow, enabling the network to effectively learn and remember over extended sequences. To encapsulate, recurrent neurons, characterized by their unique architecture and feedback loops, are adeptly engineered to discern patterns inherent to sequential or temporal data. Serving as the linchpin of RNNs, they have instigated significant advancements in domains such as time series forecasting and natural language processing, reaffirming their pivotal stature in the domain of artificial neural networks.

2.2.3 Beyond the Standard Neurons

While neurons in feedforward, convolutional, and recurrent architectures have played pivotal roles in the rise of deep learning, the tapestry of artificial neural networks is richer and more diverse than these foundational units alone. The realm of deep learning has evolved rapidly, leading to the genesis of other computational units, each inspired by different challenges, data structures, and applications. In the following, some of them will be discussed so to provide a broader perspective on the dynamic landscape of neural network architectures, emphasizing the ongoing innovation and adaptability inherent to this field.

Radial Basis Function (RBF) Neurons

Originating from function approximation, RBF neurons differ from traditional ones by evaluating the distance between their input and a predetermined center [29]. They excel in detecting specific patterns in the input space and are prevalent in regression, function approximation, and classification tasks. Central to RBF neurons is their Gaussian activation function. The neuron's output hinges on the Euclidean distance between the input vector x and the center c:

$$y(\mathbf{x}) = e^{-\beta \|x-c\|^2} \tag{2.5}$$

Here, x is the input vector, c is the Gaussian function's center adjusted during training, and β influences the Gaussian width, thus the neuron's response intensity. The notation $\|\cdot\|$ denotes the Euclidean norm. With their Gaussian nature, RBF neurons yield a localized response, activating robustly for inputs nearer their center. Their efficacy in discerning localized patterns largely stems from β and the center $\mathbf{c}$ positioning [30].

Spiking Neurons

Spiking Neurons offer a closer approximation to biological neurons by emphasizing discrete dynamics over continuous outputs. Their operation can be mathematically characterized by a discrete-time output signal $s(t)$, where $s(t) = 1$ signifies a spike and $s(t) = 0$ denotes its absence, reflecting biological spike-time processes [31].

Grounded in Maass' foundational work, the Leaky Integrate-and-Fire (LIF) model describes spiking neurons as:

$$\tau_m \frac{du}{dt} = -u(t) + RI(t) \tag{2.6}$$

Here, $u(t)$ is the membrane potential, R signifies membrane resistance, $I(t)$ is the input current, and τ_m denotes the membrane time constant. When $u(t)$ hits a threshold, a spike occurs, followed by a reset. This model captures the balance of excitatory and inhibitory inputs found in biological neurons.

Stochastic Neurons

Stochastic neurons, in contrast to their deterministic counterparts, incorporate randomness directly into their operations, inspired by the naturally stochastic behavior of biological neurons [32]. Given a traditional neuron that computes a weighted sum of its inputs as $s = W \cdot x + b$, where W is the weight vector, x denotes the input vector, and b is a bias term, a stochastic neuron's output differs: instead of a deterministic activation function, the output is derived by sampling from a distribution influenced by s. Commonly, the Bernoulli distribution is employed:

$$y = \begin{cases} 1 & \text{with probability } f(s) \\ 0 & \text{with probability } 1 - f(s) \end{cases} \tag{2.7}$$

This stochasticity can act as a regularizer, potentially aiding optimization by mitigating overfitting and facilitating escape from local minima. However, it introduces challenges in training stability and may necessitate modifications to conventional training algorithms [33].

Quantum Neurons

Quantum neurons integrate quantum mechanics principles into neural computation, leveraging quantum properties – particularly superposition and entanglement – to potentially overcome classical computational constraints [34].

Unlike classical neurons, quantum counterparts can harness superposition, allowing them to process multiple inputs simultaneously, and entanglement, which interlinks particle states regardless of distance. This potential for concurrent processing hints at enhanced computational efficiency. While the quantum approach is promising, its practical implementation remains a challenge, with much of the current research being theoretical.

2.3 Activation Functions

Activation functions are pivotal to neural networks, primarily introducing non-linearity into the model. This non-linearity is imperative because it allows the network to capture intricate relationships and patterns in data, transcending the capabilities of a mere linear model. Furthermore, from a mathematical standpoint, the introduction of non-linear activation functions is what enables neural networks to be universal function approximators [35].

In the progression of neural network research, particularly as it relates to their mathematical underpinnings and optimization, it became evident that the choice of activation function could profoundly impact the learning dynamics. The foundational activation functions, like the sigmoid and hyperbolic tangent, have well-understood mathematical properties. For instance, they are smooth, differentiable, and bounded. Yet, as neural network architectures grew deeper and the objective landscapes of loss functions became more intricate, the limitations of these functions – especially regarding gradient – based optimization—came to the fore. These limitations manifest mathematically in various ways, with the vanishing and exploding gradient problems being paramount among them. These phenomena can severely hamper the convergence properties of the stochastic gradient descent (SGD) and its variants, which are the standard de facto algorithms for training deep networks.

Responding to these challenges, more advanced activation functions have been proposed. These aren't just heuristics but are often backed by rigorous mathematical analysis and are shaped to specifically address optimization challenges. Some of these functions, for instance, are designed to ensure bounded gradient norms or to introduce controlled non-linearities that facilitate better conditioning of the optimization landscape. Moreover, as neural networks are being applied to a wider array of mathematical problems, from approximating functions in high-dimensional spaces to modeling intricate dynamical systems, there's a growing need for activation functions tailored to these specific contexts.

2.3.1 Sigmoid

The sigmoid function, typically denoted by $\sigma(x)$, is defined as

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{2.8}$$

This function scales its output to lie strictly within the open interval (0, 1). Historically, its use in neural networks can be traced back to early models of artificial neurons, making it one of the archetypal activation functions [6].

Mathematically, the sigmoid function possesses several properties that render it appealing, especially in the context of early neural network designs [10]. It is smooth, monotonic, and continuously differentiable, which facilitates the gradient-based optimization techniques that underpin the standard training of deep neural networks. Despite its mathematical allure and historical significance, the sigmoid function has some notable drawbacks when used in deep neural networks [36]. One of the most prominent challenges it poses is the so-called vanishing gradient problem: for large positive or negative values of x, the function saturates, leading to its derivative tending towards zero. Consequently, if techniques as the backpropagation are applied, during this phase of neural network training, the gradients can become extremely small, causing weight updates to be minuscule. This results in slower convergence during training or, in extreme cases, causes the network to stagnate, effectively halting further learning [13]. The squashed output range of the sigmoid, although ideal for binary classification, also means that the outputs are never exactly zero or one, which can occasionally lead to numerical instability or misinterpretations [37].

2.3.2 Hyperbolic Tangent

The hyperbolic tangent function, often abbreviated as $\tanh(x)$, serves as another choice of activation function in neural network architectures. Mathematically, the function can be expressed in terms of exponentials:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.9}$$

The primary distinction between the tanh function and the sigmoid is its output range. While the sigmoid function maps input values to the (0, 1) range, the hyperbolic tangent maps to (-1, 1). This zero-centered characteristic often facilitates faster convergence in neural networks, since it allows weights to be adjusted in a more balanced manner – both in positive and negative directions. This is especially beneficial during the initial phases of training when weights are typically initialized with small random values. Some empirical studies have shown that networks employing tanh activations, especially in the hidden layers, can occasionally outperform those using sigmoid, due to its zero-centered property, which can alleviate the effects of certain optimization challenges [37]. Despite its advantages, the hyperbolic tangent function shares a common limitation with the sigmoid: the vanishing gradient problem. However, like all activation functions, the suitability of tanh is largely context-dependent and should be chosen based on both theoretical considerations and experimental validation.

2.3.3 Rectified Linear Unit (ReLU) and its variants

The Rectified Linear Unit, colloquially known as ReLU, has in recent years emerged as a popular choice of activation function for deep neural networks, especially convolutional ones. This can be attributed to its conceptual simplicity combined with practical efficacy. Mathematically, the ReLU function is represented as:

$$f(x) = \max(0, x) \tag{2.10}$$

One of the defining characteristics of ReLU is its non-saturating nature. Unlike sigmoid or tanh functions, which squash their input into a bounded range, ReLU allows positive values to pass without alteration, thereby alleviating the vanishing gradient problem for positive input values [38]. This ensures that the gradient remains large for positive inputs, leading to more robust learning. However, the ReLU function is not without its own set of challenges as the so-called "dying ReLU" problem: this phenomenon arises when neurons get stuck during training and consistently output zero; any neuron that enters this state ceases to update, effectively becoming inactive [39]. This situation can be exacerbated if the neuron's weights are initialized poorly or if a large gradient flows through a ReLU neuron, updating weights in such a manner that the neuron will consistently output negative values. Solutions and variants like Leaky ReLU and Parametric ReLU have been proposed to address this very concern, by introducing non-zero gradients for negative values [40].

Leaky ReLU

The Leaky ReLU activation function emerges as a direct response to the shortcomings of its predecessor, the vanilla ReLU. It is mathematically represented as:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{2.11}$$

Here, α is a small positive constant, conventionally set to a value less than 1. The introduction of this coefficient for the negative region ensures a minor slope, thereby permitting gradient flow even for negative input values. This property is essential in mitigating the issue of the "dying ReLU". Such an improvement fosters a scenario where neurons maintain their active state throughout the learning process, thereby enhancing the network's learning capability [41].

Parametric ReLU (PReLU)

PReLU can be perceived as an evolutionary step from Leaky ReLU, broadening its horizons by introducing adaptability. The function is mathematically delineated as the Leaky ReLU. What differentiates PReLU is the dynamic nature of the coefficient α. Instead of being a predetermined constant, this coefficient becomes a learnable parameter during the training phase. Such an attribute offers PReLU a notable flexibility, enabling the function to self-adjust based on the idiosyncrasies of the training data. Empirical studies have suggested that this adaptability can frequently lead to enhanced performance metrics, particularly in architectures with deeper layers where the risk of gradient vanishing or exploding becomes more pronounced [40].

Exponential Linear Unit (ELU)

The ELU activation function is an attempt to bring together the advantages of ReLU and its variants while addressing some of their limitations. Its formal representation is:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{for } x \leq 0 \end{cases} \tag{2.12}$$

Distinctively, the ELU activation encompasses an exponential component for negative input values. This design pushes the mean activations closer to zero – a desirable trait referred to as zero mean activations. Such behavior promotes faster learning due to reduced bias shift effects. Moreover, the non-zero gradient for the negative region helps in mitigating the dying neuron issue, a challenge frequently observed in standard ReLU-based functions. The consequence of these features is often accelerated training and enhanced network stability [42].

2.3.4 Gaussian Error Linear Unit (GeLU)

GeLU has recently gained traction, notably with the advent and widespread adoption of transformer-based architectures. The function is given by:

$$\lambda(x) = \frac{1}{2}x\left[1 + \text{erf}(x/\sqrt{2})\right] \approx 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right) \tag{2.13}$$

What distinguishes GeLU is its smooth transition from non-activation to activation, unlike the more abrupt transitions in functions like ReLU. This smoothness is postulated to help in various ways: for instance, during the training of deep architectures, it might offer better gradient propagation, especially in regions near the non-activation threshold. The transformative impact of GeLU is especially evident in NLP scenarios: when transformer-based models such as BERT (Bidirectional Encoder Representations from Transformers) adopted GeLU as their primary activation function, they achieved state-of-the-art results on a multitude of tasks, ranging from sentence classification to question-answering [43]. It's speculated that the smoother nature of GeLU allows for better capturing of linguistic nuances in data, which are often non-linear and intricate. Moreover, empirical investigations suggest that networks employing GeLU often demonstrate improved convergence properties, particularly in the presence of large model sizes and extensive data [44]. This is of paramount significance in the training of colossal models like GPT-3 (Generative Pre-trained Transformer 3) and their subsequent versions, where convergence dynamics become increasingly intricate with the depth and width of the architecture [45].

2.3.5 Swish

Swish is a self-gated activation function introduced by researchers at Google [46]. It is mathematically expressed as:

$$f(x) = x \cdot \sigma(\beta x) \tag{2.14}$$

where σ denotes the sigmoid function and β is a parameter. Uniquely, this parameter can either be set as a constant or treated as a learnable parameter during the training process. The motivation behind Swish arises from the desire to have a smoother, non-monotonic function that can adaptively adjust its output based on its input. This adaptability is achieved through the self-gating property, where the function can regulate the information flow. Empirical studies, as presented in the originating paper, have shown that Swish often outperforms traditional activation functions, such as ReLU, especially in deeper networks. This enhanced performance is attributed to its ability to capture a more diverse range of functions and mitigate issues associated with other activations, offering a more balanced trade-off between representational power and computational efficiency.

The realm of activation functions in neural networks represents a dynamic interplay of theoretical foundations and empirical discoveries. Each function, carved out of both mathematical insights and real-world challenges, holds a unique spectrum of properties – be it in addressing vanishing gradients, facilitating faster convergence, or ensuring neuron activity. While certain activation functions have risen to prominence due to their consistent performance across a plethora of tasks, no single function can be universally hailed as superior in all contexts. The intricate balance of selecting an apt activation function is dictated not merely by its mathematical appeal but also by its harmony with the specific problem domain, the peculiarities of the dataset in consideration, and the nuances of the selected network architecture. As neural networks continue to grow in complexity and application, the criticality of an informed choice becomes even more pronounced. Empirical evaluations, when conducted in tandem with rigorous mathematical analyses, remain at the heart of this decision-making process. As the field marches forward, the exploration and understanding of activation functions will undeniably remain a cornerstone of neural network research, bearing testament to their foundational role in the domain of deep learning [3].

2.4 Layers in Neural Networks

Neural networks, at their core, are hierarchical structures composed of computational units organized in an intricate manner. These units, commonly referred to as neurons, are arrayed into distinct strata, each termed as a ‘layer’. Each layer in this hierarchy carries out specific transformations on its input, further refining and molding it as the data courses through the network. The assemblage of neurons into layers isn’t arbitrary. Specific configurations or topologies have emerged over time, each tailored for particular data structures or computational tasks. The design and configuration of these layers play a pivotal role in determining the network’s capability and the nature of tasks it can efficiently address.

2.4.1 Fully Connected Layers

Often referred to as *Dense layers*, these are foundational components in many classical neural network architectures. Mathematically, a dense layer performs a linear transformation on its inputs which can be expressed as:

$$o = \lambda(Wx + b) \tag{2.15}$$

Note that, to introduce non-linearity, the activation function λ is applied element-wise on $(Wx + b)$. In this formulation:

- x is the input vector of size $n \times 1$, where n is the number of inputs.
- W represents the weight matrix of dimensions $m \times n$, where m is the number of neurons in the current layer and n is the number of neurons (or input dimensions) in the preceding layer.
- b is the bias vector of size $m \times 1$, adding an offset to each neuron in the current layer.
- o is the final output vector of size $m \times 1$ after applying the activation function.

In this context, each row of W corresponds to the weights associated with a neuron in the current layer. Given that each neuron in a fully connected layer is linked to every neuron from its preceding layer, there are $m \times n$ weights if m is the number of neurons in the current layer and n is the number in the previous layer. This expansive mesh of inter-neuronal connections enables intricate function approximations. From a theoretical standpoint, these layers have an essential property described by the *universal approximation theorem* [35], which states that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate any continuous function on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function. However, the dense interconnections inherently imply a considerable number of trainable parameters. Specifically, for an input of dimension n connected to a layer of size m, the total number of parameters is $m(n+1)$, accounting for both weights and biases. This count can become substantial, particularly for high-dimensional input spaces, potentially leading to challenges like overfitting and necessitating more data for effective training [3].

2.4.2 Convolutional Layers and their Variations

Convolutional layers are central to Convolutional Neural Networks (CNNs). The primary motivation behind these layers, as aforementioned, is the principle of spatial hierarchy and local connectivity, which is inspired by the human visual system's architecture and functionality. Mathematically, a convolution operation in a convolutional layer involves the use of a set of kernels or filters, which are smaller-sized matrices. These filters slide or convolve around the input data (like an image) to produce multiple feature maps. For a given filter K, the convolution operation can be formally expressed as:

$$Y^f(i,j) = \lambda \left(\sum_m \sum_n X_{i+m,j+n} K^f_{m,n} + b^f \right) \tag{2.16}$$

where:

- X is the input matrix.
- K^f is the f^{th} kernel or filter of the convolutional layer.
- Y^f is the f^{th} output feature map.
- b^f is the bias term associated with the f^{th} filter.
- λ represents the non-linear activation function applied element-wise to the feature maps.
- The summations iterate over the spatial dimensions of the kernel.

In a convolutional layer, this process is repeated for each filter to generate a set of feature maps. The collective set of these feature maps can be thought of as the output of the layer. The convolution operation aims to detect local patterns, like edges, corners, and textures. Different filters can detect different types of patterns, and thus, multiple filters are often used in each convolutional layer. As the network deepens, these patterns become more abstract, allowing higher layers to recognize more complex structures by combining features from earlier layers.

A salient feature of convolutional layers is weight sharing: instead of learning separate weights for every neuron (as in fully connected layers), the same weights (filter values) are shared among all locations in the input. This significantly reduces the number of parameters, making CNNs more parameter-efficient compared to their fully connected counterparts. The reduced number of parameters not only mitigates overfitting but also promotes translational invariance, ensuring that once a feature is learned in one part of the image, it can be recognized everywhere in the input. Strides and pooling operations further allow these layers to reduce the spatial dimensions of feature maps, making them more computationally manageable and robust to variations in the input. The concatenation of multiple such operations and layers enables CNNs to capture hierarchical patterns, making them particularly adept at tasks like image classification, object detection, and even areas beyond image data [2].

Transposed Convolutional Layers

Transposed convolutions, often mistakenly referred to as deconvolutions, are used to upsample the spatial dimensions of feature maps. While standard convolutional layers might reduce the spatial dimensions of an input (e.g., from 28x28 to 14x14), transposed convolutional layers aim to increase it (e.g., from 14x14 back to 28x28). Mathematically, the operation is akin to a convolution, but the spatial relationship between input and kernel is inverted:

$$Y^{f}_{i+m,j+n} = \lambda\left(\sum_{m}\sum_{n} X_{i,j}\cdot K^{f}_{m,n} + b^{f}\right) \tag{2.17}$$

This method is pivotal in architectures like Generative Adversarial Networks (GANs) for generating high-resolution outputs and in certain types of autoencoders to recover original input dimensions after encoding [47].

Locally-connected Layers

Locally-connected layers are a variant of the fully connected layers where each neuron is connected only to a local region of the input, similar to the receptive field in convolutional layers, but without weight sharing. Mathematically, if we let $W^f_{i,j,m,n}$ denote the weight from the input at location $(i+m, j+n)$ to the f^{th} feature map's neuron at position (i,j), then the output $Y^f_{i,j}$ for a given input X is given by:

$$Y^f_{i,j} = \lambda \left(\sum_m \sum_n X_{i+m,j+n} W^f_{i,j,m,n} + b^f_{i,j} \right) \tag{2.18}$$

Here, unlike convolutional layers, each weight $W_{i,j,m,n}$ is unique and not shared across spatial locations. Such layers are useful in scenarios where spatial invariance isn't desirable, and unique patterns across different regions of the input need to be captured distinctly.

Separable Convolutional Layers

Separable convolutions, as the name suggests, involve breaking down the conventional convolution process into separate operations to reduce computational cost. Referring back to Eq. (2.16), in the case of separable convolutions, this is factorized into a depthwise convolution, which applies separate filters to each of the c channels of the input, followed by a pointwise convolution (a 1×1 convolution). Specifically, the filter for the depthwise convolution $K^c_{m,n}$ operates on the spatial dimensions, producing intermediate feature maps:

$$D^c(i,j) = \sum_m \sum_n X^c_{i+m,j+n} K^c_{m,n} \tag{2.19}$$

Then, the pointwise convolution is applied, with P_c^f denoting the weight connecting the c^{th} channel from the depthwise convolution to the f^{th} feature map in the output:

$$Y^f(i,j) = \lambda\left(\sum_c D^c(i,j)P_c^f + b^f\right) \tag{2.20}$$

This factorization significantly reduces the number of parameters and computational complexity. Such layers are particularly prominent in architectures aiming for efficiency, like MobileNets [48], and in models like Xception, which posits that channel-wise spatial correlations and cross-channel correlations can be mapped separately [49].

2.4.3 Pooling Layers

Pooling layers, often interspersed between successive convolutional layers in Convolutional Neural Networks (CNNs), play a crucial role in reducing the spatial dimensions of the data, thereby simplifying the computational demands of the network and augmenting its robustness to input variations. The operation can be mathematically visualized as partitioning the input data (typically a feature map) into a set of non-overlapping rectangles and then outputting a summary statistic for each such rectangle. The exact nature of this statistic differentiates the common types of pooling:

- **Max Pooling:** Here, the maximum value from each of the segmented regions of the input data is taken. Formally, if X is the input matrix and $P_{\max}$ is the pooled output, then for each region R in X:

$$P_{\max}(R) = \max(X(R)) \tag{2.21}$$

- **Average Pooling:** In this variant, the mean value from each of the segmented regions is taken. Using a similar notation as before:

$$P_{\text{avg}}(R) = \frac{1}{|R|}\sum_{i,j\in R} X(i,j) \tag{2.22}$$

where $|R|$ denotes the size of region R.

The choice between max pooling and average pooling usually depends on the specific application. Max pooling tends to capture the most salient features in a region, while average pooling provides a broader summary, potentially preserving more contextual information. More specifically, pooling layers confer several benefits to the network:

1. **Dimensionality Reduction:** They significantly cut down the spatial size of the feature maps, leading to reduced computational costs and memory usage in subsequent layers.

2. **Translational Invariance:** By summarizing regions, minor translations or distortions in the input are smoothened out, allowing the network to be more robust to such variations.

3. **Overfitting Mitigation:** The reduction in spatial dimensions and the consequent decrease in parameters can help in preventing overfitting, especially when training data is limited.

The above-mentioned attributes highlight the importance of pooling layers in the design of effective CNN architectures, particularly for tasks that benefit from a condensed representation of the input space while retaining essential informational aspects [3].

2.4.4 Recurrent Layers and their Variations

Recurrent Neural Networks (RNNs) have been specifically architected to handle sequential data, where the temporal dynamics and dependencies are of primary importance. Mathematically, the recurrent layer at time step t can be expressed as:

$$h_t = \lambda_h(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \tag{2.23}$$

$$o_t = \lambda_o(W_{ho}h_t + b_o) \tag{2.24}$$

Where:

- x_t is the input vector at time step t.
- h_t represents the hidden state vector at time step t.
- h_{t-1} denotes the hidden state vector from the previous time step.
- W_{xh} and W_{hh} are weight matrices for the current input and previous hidden state, respectively.
- W_{ho} is the weight matrix for generating the output from the hidden state.
- b_h and b_o are the bias vectors.
- λ_h and λ_o are typically non-linear activation functions.

Despite this formulation enrich the RNNs with memory capabilities, it is worth recalling that conventional form of recurrent networks suffer from certain limitations:

- **Vanishing and Exploding Gradient:** Due to the recurrent nature, gradients can either vanish or explode during backpropagation, especially over long sequences, causing training difficulties [50].
- **Short-term Memory:** Standard RNNs struggle to remember long-term dependencies because of the aforementioned gradient issues.

To address these challenges, advanced RNN architectures like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) have been proposed, as introduced in Section 2.2.2. These architectures introduce gating mechanisms to control the flow of information and are better suited for tasks requiring modeling of long-term dependencies [27].

Long Short-Term Memory (LSTM) Layers

LSTMs were designed as a response to the shortcomings of conventional RNNs, especially the challenge of learning long-term dependencies due to the notorious vanishing gradient problem [27]. This issue, where gradients become too small for practical computation, renders traditional RNNs incapable of retaining memory over long sequences. LSTMs tackle this by introducing a sophisticated internal memory cell structure governed by distinct gates, each having a specific function:

- Input gate (i_t): Determines how much of the new information should be stored in the cell state.

- Forget gate (f_t): Decides what proportion of the current cell state should be discarded or retained.

- Output gate (o_t): Regulates the amount of information from the cell state to transfer to the hidden state.

Mathematically, these gates function as:

$$i_t = \sigma(W_{ii}h_{t-1} + W_{xi}x_t + b_i) \tag{2.25}$$

$$f_t = \sigma(W_{if}h_{t-1} + W_{xf}x_t + b_f) \tag{2.26}$$

$$o_t = \sigma(W_{io}h_{t-1} + W_{xo}x_t + b_o) \tag{2.27}$$

The cell state, representing the "memory" of the network, undergoes controlled modifications at each time step, defined by:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{ic}h_{t-1} + W_{xc}x_t + b_c) \tag{2.28}$$

where $\odot$ represents the element-wise product. The hidden state, which conveys information to subsequent layers or time steps, is given by:

$$h_t = o_t \odot \tanh(c_t) \tag{2.29}$$

This structure brought to LSTM the capacity to capture and retain long-range dependencies in sequential data.

Gated Recurrent Units (GRU) Layers

The Gated Recurrent Units (GRU) were introduced as an alternative to the more complex LSTM architecture, aiming to achieve similar performance but with a reduced parameter set [51]. One of the most prominent distinctions of GRUs over LSTMs is the consolidation of the cell state and hidden state into a singular representation, simplifying the recurrent mechanism. GRUs employ two critical gates to modulate the flow of information:

- Update gate (z_t): This gate determines the blend of past information (from the previous state) and new information from the current input.
- Reset gate (r_t): It controls how much of the past information (previous state) should be discarded or considered while calculating the candidate state.

These gates are mathematically articulated as:

$$z_t = \sigma(W_{uz}h_{t-1} + W_{xz}x_t + b_z) \tag{2.30}$$

$$r_t = \sigma(W_{ur}h_{t-1} + W_{xr}x_t + b_r) \tag{2.31}$$

Given the gates, the next state (or the updated hidden state) for GRU is expressed by:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tanh(W_{uh}(r_t \odot h_{t-1}) + W_{xh}x_t + b_h) \tag{2.32}$$

The simplicity and reduced number of gates in GRUs compared to LSTMs make them more computationally efficient, especially for certain tasks where model compactness and speed are prioritized. Nonetheless, the debate between the supremacy of LSTMs and GRUs is ongoing, with empirical evidence suggesting that their performance is largely task-dependent [52]. While LSTMs generally offer superior performance on more complex tasks, GRUs have showcased competitive results on various sequence modeling challenges, especially when the available data for training is limited.

Bidirectional RNN (BiRNN) Layers

Bidirectional RNNs (BiRNNs) process sequences from both the beginning and end, offering a comprehensive view of data. For an input x_t, the forward pass at time t is:

$$\overrightarrow{h}_t = \lambda_h(\overrightarrow{W}_{hh}\overrightarrow{h}_{t-1} + \overrightarrow{W}_{xh}x_t + \overrightarrow{b}_h) \tag{2.33}$$

Simultaneously, the backward pass reads:

$$\overleftarrow{h}_t = \lambda_h(\overleftarrow{W}_{hh}\overleftarrow{h}_{t+1} + \overleftarrow{W}_{xh}x_t + \overleftarrow{b}_h) \tag{2.34}$$

In applications where context from both past and future inputs influences the present understanding, BiRNNs excel. This symmetry proves crucial in tasks like language modeling where meaning often relies on surrounding words.

Peephole LSTM Layers

Peephole LSTMs augment the traditional LSTM architecture by allowing gates to draw context from the cell state, not just the previous hidden state and current input. The enhanced gate definitions are:

- Input gate: $i_t = \sigma(W_{ii}h_{t-1} + W_{xi}x_t + W_{ci}c_{t-1} + b_i)$
- Forget gate: $f_t = \sigma(W_{if}h_{t-1} + W_{xf}x_t + W_{cf}c_{t-1} + b_f)$

- Output gate: $o_t = \sigma(W_{io}h_{t-1} + W_{xo}x_t + W_{co}c_t + b_o)$

By affording gates this deeper insight, Peephole LSTMs potentially refine decision-making in scenarios where the cell state holds intricate temporal clues overlooked by conventional LSTMs.

2.4.5 Convolutional-Recurrent (ConvRNN) Layers

Convolutional Recurrent Neural Networks (ConvRNNs) merge the spatial processing of Convolutional Neural Networks (CNNs) with the sequential memory capabilities of Recurrent Neural Networks (RNNs). This combination is particularly useful for data with both spatial and temporal dimensions, such as videos or sequences of images [53]. Mathematically, a ConvRNN layer's update can be expressed as:

$$h_t = \lambda_h(K_{hh} * h_{t-1} + K_{xh} * X_t + b_h) \tag{2.35}$$

Where:

- X_t is the input tensor (e.g., an image frame) at time step t.
- h_t represents the hidden state at time step t.
- $*$ denotes the convolution operation.
- K_{xh} and K_{hh} are convolutional kernels for the current input tensor and previous hidden state tensor, respectively.
- b_h is the bias term.
- λ_h is a non-linear activation function.

Considering the sequential outputs, they can be expressed as:

$$o_t = W_{ho} * h_t + b_o \tag{2.36}$$

With W_{ho} being a convolutional kernel that transforms the hidden state tensor to the output tensor, and b_o is the associated bias term. ConvRNNs, by leveraging the spatial feature extraction capabilities of CNNs and maintaining memory through RNNs, can recognize spatial patterns over time. Of course, LSTM or GRU components could be used within the same structure, addressing the challenges of vanishing or exploding gradients that ConvRNN inherits from its foundational architectures.

2.5 A Bridge Towards Neural Architectures

In the study of Artificial Neural Networks (ANNs), the effectiveness and complexity of an architecture are deeply intertwined with the precise configuration and integration of its constituent layers. Each layer, which is an ensemble of computational units governed by specific activation functions, plays a pivotal role in the multi-stage transformation process, iteratively processing and refining the input data to produce a designated output. While the individual layers exhibit a broad range of typologies and functionalities, reflecting the diverse challenges and needs they address, a common structural paradigm is discernible in the design of neural architectures, a paradigm that transcends the particularities of any given task. At a foundational level, a neural network can be distilled into three primary segments: the input layer, which serves as the gateway for data ingestion; the hidden layers, which work in tandem to extract intricate patterns from the data and craft an (often non-linear) functional representation of these patterns; and the output layer, which is meticulously tailored to articulate the final output in a manner that aligns with the specific objectives of the task at hand.

Input Layer

The input layer, the neural network's antechamber, dictates the system's overall functioning. It not only intakes the data but also sets the dimensionality and structure for the forthcoming layers.

A neural network's efficacy stems largely from its ability to adapt to various types of data - be it images, sound waves, textual information, or numerical sequences. Depending on the nature of the data, the input layer transforms the data into a structured format, commonly a vector or a tensor, that is amenable for further processing by the subsequent layers. Mathematically, consider a dataset that presents information as features: each instance of this data can be captured as a vector $\in \mathbb{R}^n$, where n represents the number of distinct features. For instance, in an image that's grayscale and sized 28x28 pixels, n would be 784, thereby translating the 2D image into a 784-dimensional vector. The input layer, in this scenario, ingests this vector. For multichannel inputs, like colored images, higher-order tensors are employed. A common inquiry pertains to the rationale behind refraining from data transformation or manipulation at the input layer. This approach is rooted in the foundational design principles of neural networks. The primary function of the input layer is to provide an unaltered representation, ensuring that the neural architecture receives raw data in its original form. Introducing substantial transformations or manipulations at this initial stage might inadvertently introduce biases or obscure subtle details inherent in the data. However, it's essential to note that this doesn't preclude preliminary normalization or scaling processes that are typically performed prior to feeding data into neural networks. Such processes are designed to expedite the learning process without distorting the inherent structure of the data.

In essence, the input layer sets the stage. By dictating the initial data structure and dimensionality, it shapes the configuration of subsequent layers, ensuring they are aptly designed to process the input data.

Hidden Layers

Situated between the input and output layers, the hidden layers function as the neural network's processing core. The name "hidden" is suggestive of these layers' function: they operate beneath the surface, invisible to external observers, and yet, their configurations and processes shape the network's learning capability. Each hidden layer can be seen as a level of abstraction: with the progression from one layer to the next, the network transitions from identifying rudimentary features in the data, such as edges in images or specific phonemes in audio sequences, to more complex structures and patterns, like shapes or entire words.

With reference to the neurons' and layers' structures described in the previous sections, it is easy to imagine that weights and biases manage how the input is adjusted, while the non-linear activation function ensure the network doesn't just perform linear transformations. Without these non-linearities, irrespective of how many layers the network boasts, it would still be limited to linear operations. The cascade of these layers, each introducing its level of transformation and non-linearity, equips the ANN with the profound capacity to model intricate relationships in data. This is not merely a computational marvel but is theoretically supported by the *universal approximation theorem* [35], which underscores the capability of neural networks with hidden layers to approximate any continuous function, given appropriate parameters.

Yet, the depth – i.e., the number of hidden layers – and the breadth – i.e., the number of neurons in each layer – aren't trivial choices: while deeper networks can represent more complex functions, they are also more susceptible to overfitting and might require more data and computational power to train effectively. Balancing these considerations is a critical aspect of neural network design.

Output Layer

The output layer is the concluding part of the neural network's architecture. It collects the processed data from the previous layers and produces the network's final predictions or classifications.

The unique aspect of the output layer compared to other layers is its specific design according to the task the network is set to perform. While internal layers manage data transformation and refinement, the output layer is directly structured towards the objective, be it predicting a value, classifying an input, or generating a sequence of values. Consider the following scenarios:

- For *regression tasks*, where the objective is to predict a continuous value, the output layer typically comprises a single neuron. The activation function for this neuron can be linear, allowing the network to predict a range of values.

- In *classification tasks*, especially multi-class scenarios, the output layer contains as many neurons as there are classes. A softmax activation function is often employed, ensuring the outputs can be interpreted as a probability distribution across potential classes. This not only identifies the most probable class but also quantifies the confidence of the network in its prediction.

However, the responsibilities of the output layer extend beyond merely producing the final output. During training, the discrepancies between the network's predictions and the actual data, measured as "loss" or "error", are used to adjust the weights and biases throughout the network, thereby refining its predictive capabilities. This intricate process, which will be explored into in the subsequent chapter, is fundamental to the ANN's ability to learn and significantly influences what a neural network derives from the data. In this scenario, it's imperative to note the pivotal influence of loss functions in defining a neural network, almost on par with its architectural design. The choice of loss function not only dictates the optimization landscape but also provides insights into the specific tasks the architecture is most suited for. It is, in essence, a compass that guides the training process, ensuring convergence towards desired outcomes and steering the design towards optimal performance for particular objectives. Through, and thanks to, this process, while regression and classification represent two of the most prevalent tasks in the realm of neural networks they are by no means exhaustive: data generation, embedding extraction, anomaly detection, dimensionality reduction and recommendation systemsare just a few examples of tasks to which a neural network can be tailored.

2.5.1 Enhancements and Peculiarities in Neural Network Design

While the fundamental structure of an Artificial Neural Network (ANN) rests on the division into input, hidden, and output layers, contemporary architectures often introduce additional features to mitigate certain challenges, improve training dynamics, and achieve better performance. In this section, some of these design enhancements are explored. These, serving as a testament for the flexibility that this kind of models introduce, can be briefly summarized in the following elucidating the motivations behind their incorporation.

Attention Mechanisms

The attention mechanism emerged as a transformative innovation in deep learning, particularly in the realm of sequence-to-sequence tasks. At its core, attention enables models to dynamically weight and focus on pertinent parts of the input space during computation, thus emulating the human capacity to attend selectively to specific stimuli. The mathematical intuition behind the attention mechanism can be seen as a weighting scheme. Given a set of values V and their corresponding keys K, the attention mechanism computes weights for each value based on a given query Q. These weights determine the relevance or importance of each value with respect to the query. Formally, the attention weights α can be computed as:

$$\alpha = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \tag{2.37}$$

and the output or attended value O is:

$$O = \alpha V \tag{2.38}$$

where:

- d_k represents the dimensionality of the keys, and
- The division by $\sqrt{d_k}$ is a scaling factor introduced to stabilize the softmax output, especially when d_k is large [5].

The attention mechanism's beauty lies in its ability to compute a context-dependent weighted sum of values, allowing models like Transformers to capture long-range dependencies and relationships in the data without being constrained by fixed architectural windows or recurrences. Furthermore, attention mechanisms have shown remarkable versatility: while initially championed in natural language processing tasks, their applicability has expanded to other domains, proving beneficial in computer vision, audio processing, and even graph-structured data, thereby cementing their foundational status in modern neural network architectures [54].

Skip and Residual Connections

In standard feed-forward networks, the output from each layer is passed directly to the next layer. However, specific architectures have introduced the concept of *skip connections*, which allow for the bypassing of one or more layers. If o_h denotes the output from the h^{th} layer, a skip connection could permit o_h to be used directly in determining o_{h+k}, with $k > 1$.

A specialized variant of skip connections, known as *residual connections*, are structured so that the skipped output is added element-wise to the subsequent layer's output. This can be represented as

$$o_{h+k} = F(ho_{h+k-1}, w) + o_h \tag{2.39}$$

where F stands for a residual function and w represents the weights associated with that function [15].

These connections serve several purposes, for example they address the notorious vanishing and exploding gradient issues by offering alternative pathways for gradients during the backpropagation process. Moreover, these connections enable the network to effectively learn identity functions. Such functions can be indispensable for tasks that benefit from the depth of the network, allowing for increased layer counts without necessarily introducing added complexity. Lastly, by fostering such connections, the architecture encourages the reuse of features: this promotes a more profound and richer hierarchical representation of features, ensuring that the network can efficiently capture and represent the intricacies of the data it processes.

Batch Normalization

Batch normalization is a technique that has significantly transformed the dynamics of neural network training, by normalizing the activations of each layer to have a mean of zero and a variance of one. For a specified mini-batch $\mathcal{B} = \{x_1, x_2, \ldots, x_m\}$, with associated mean $\mu_\mathcal{B}$ and variance $\sigma^2_\mathcal{B}$, the input x_i is normalized as:

$$\hat{x}_i = \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma^2_\mathcal{B} + \epsilon}} \tag{2.40}$$

The term ϵ is a small constant ensuring numerical stability. Following this normalization, the outputs undergo a scaling and shifting transformation dictated by learnable parameters γ and β [55].

The introduction of batch normalization offers a series of profound benefits in the training of neural networks. Primarily, by ensuring consistent activation distributions across different mini-batches, it mitigates the challenges associated with internal covariate shifts, leading to a noticeable acceleration in training convergence. Additionally, the process introduces a regularizing effect, sometimes alleviating the need for other regularization techniques like dropout. And importantly, it smoothens the optimization landscape, thereby stabilizing and often enhancing the training process.

Layer Normalization

While Batch Normalization operates by standardizing activations across batches, *Layer Normalization* adopts a distinct approach: it centralizes its mechanism on the feature dimension. Considering a feature vector x, the normalization can be mathematically represented as:

$$\hat{x}_i = \frac{x_i - \mu_x}{\sqrt{\sigma_x^2 + \epsilon}} \tag{2.41}$$

In this equation, μ_x and σ_x^2 are the computed mean and variance across the features, respectively. Also here, ϵ is a small constant added for stability [56].

Two primary motivations underlie the adoption of Layer Normalization: firstly, it promises consistent activations within layers, a trait that remains unaffected by any variations in batch sizes. This consistency is especially advantageous for models that operate with dynamic batch sizes. Secondly, recurrent architectures, known for their sensitivity to the scale and temporal distribution of activations, find in Layer Normalization a strategy particularly well-suited to their nuances.

Weight Normalization

The *Weight Normalization* technique reparameterizes weight vectors, denoted as w, in neural networks as follows:

$$w = g\frac{v}{\|v\|} \tag{2.42}$$

In this equation, v is a weight vector and g is a scalar. The term $\|v\|$ is the L_2 norm of the vector [57]. This technique's advantages include improving gradient flow, which promotes faster convergence. By separating the magnitude and direction of the weight vectors, it makes the optimization process more straightforward for gradient-based algorithms.

Group Normalization

Group Normalization divides the channel dimension into groups and normalizes within these groups. For a feature map of shape $C \times H \times W$, it creates G groups. Within each group, the normalization process is similar to layer normalization [58].

The primary motivations for using Group Normalization are its ability to provide consistent normalization regardless of batch size, making it suitable for memory-intensive models, and its role in ensuring training stability across different architectures and batch sizes.

Further Techniques

Besides the aforementioned techniques, in the ever-evolving landscape of artificial neural networks, continuous innovations are developed to refine their operations, address inherent challenges, and improve performance metrics. Two such noteworthy examples include the adoption of Self-Normalizing Neural Networks (SNNs) and the development of Dynamic Network Architectures. The former utilize specific activation functions, namely the *scaled exponential linear unit (SELU)*, defined as:

$$\mathrm{SELU}(x) = \beta \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases} \tag{2.43}$$

where the parameters β and α are preset scaling values. Through SELUs, deep networks inherently achieve self-normalization [59], offering stable activation distributions across layers without extra normalization. Additionally, it leverages fixed points in activation behavior to combat challenges like vanishing or exploding gradients in deeper architectures.

As concerns Dynamic Network Architectures, the focus shifts to adaptability, with architectures altering their computational graph based on input data. Formally, for an input x and potential operations $\mathcal{O}$, a gating mechanism G determines which operations to employ:

$$y = \sum_{o \in \mathcal{O}} G(o, x) o(x) \tag{2.44}$$

With G emitting binary values to decide if an operation o is applied on $\mathbf{x}$ [60], such architectures adeptly balance computational complexity. This ensures optimal resource utilization based on input complexity, and can even reduce computational needs for simpler data.

Chapter 3

Foundations of the Learning Paradigm in Neural Networks

3.1 Introduction

The power of neural networks lies not only in their architectural intricacies or the huge number of parameters they can accommodate. Instead, it is their unprecedented ability to learn and adapt that sets them apart. This learning process, where networks adjust to represent and infer complex relationships in data, remains one of the most profound achievements in the realm of artificial intelligence. Yet, it is underpinned by a series of mathematical frameworks and principles that are as elegant as they are intricate.

To truly appreciate the capabilities and potential of neural networks, it's imperative to understand the very foundation on which they operate: the mechanism of learning. While the architecture and design of a neural network determine its capacity, it's the process of learning that actualizes its potential, enabling it to model complex non-linear relationships, discover latent patterns, and, more broadly, infer knowledge from data.

In this introductory section, we aim to elucidate this learning paradigm, shedding light on what it means for a neural network to learn and why this process is of paramount significance. We embark on this exploration by providing a brief overview of the learning process and then delving into its importance within the context of neural networks.

3.1.1 Brief Overview of Learning in Neural Networks

Neural networks, inspired by biological neural systems, stand as a class of mathematical models adept at recognizing patterns and making decisions. At the heart of these networks lies the principle of learning – a systematic approach to adjusting the internal parameters. Unlike many traditional mathematical models often built on predefined assumptions and where parameters are explicitly defined, neural networks utilize vast numbers of parameters which are learned from data. This data-driven approach allows neural networks to adapt and evolve, giving them the capability to solve, without any a priori assumptions, complex, non-linear problems which may be intractable for conventional techniques. At its core, a neural network can be defined as a composite function structured by a series of parameterized transformations. Considering a feed-forward network with L layers, as previously discussed, for each layer h, the transformation can be described as:

$$x^h = \lambda \left(W^h x^{h-1} + b^h \right) \tag{3.1}$$

where W^h and b^h are the weight matrix and bias vector, respectively, and λ is the activation function. The iterative application of these transformations across layers facilitates the extraction and refinement of features from input data.

The learning in neural networks, fundamentally, is an optimization problem: given a set of inputs and corresponding outputs, the objective is to find a set of parameters (the weights W^h and biases b^h) such that the difference between the predicted outputs of the network and the true outputs is minimized. So, the act of learning in neural networks pivots on optimizing a pre-defined objective function, termed the “loss” or “cost” function. The task then is to navigate the high-dimensional parameter space, often aided by gradient-based optimization techniques, to locate a parameter configuration that minimizes the empirical loss. However, the landscape presents challenges, including non-convexity, local minima, and saddle points, as will be also discussed in Section 6.6: navigating this landscape becomes a meticulous balance between efficient exploration and ensuring convergence to good minima, preferably ones that generalize well to unseen data.

3.2 The Concept of a Loss Function

As outlined before, in machine learning, particularly within neural networks, a model's ability to approximate functions based on data is vital. This necessitates a mechanism to gauge the accuracy of this approximation: the loss function. These functions quantify the difference between a model's predictions and the actual data, thereby serving as crucial indicators of performance. Moreover, they not only embody the objective of the task but also the model's deviation from this objective. On the other hand, optimization algorithms traverse the landscape of the loss function to find areas where errors are minimized. It means that the selection of the right loss function is pivotal since it directly influences the learning strategy. While different problems require different functions, a judiciously chosen one can enhance training efficiency and improve model generalization. In contrast, an inappropriate function could hinder optimization, leading to subpar performance.

At its essence, a loss function can be mathematically represented, in the case of a supervised learning task with N samples, as:

$$\mathcal{L}(f(x;\theta), y) = \frac{1}{N}\sum_{i=1}^{N} \ell(\mathcal{N}(x_i;\theta), y_i) \tag{3.2}$$

where ℓ measures the loss for an individual data point, $\mathcal{N}(x;\theta)$ denotes the model's prediction influenced by parameters θ, and x_i, y_i represent the input-output data pair.

Recent advancements in neural network research have introduced a variety of loss functions designed for specific tasks and objectives. Understanding these functions, their mathematical properties, and their effects is crucial for researchers working with neural networks.

3.2.1 Practical Dynamics of Loss Functions

While we have conceptualized loss functions in terms of their definitions and foundational roles in the previous section, it is paramount to grasp the intricacies surrounding their practical implications and dynamics during the training process.

Different tasks necessitate distinct loss functions. Consider regression and classification, where predictions provided by the neural model are indicated as $\hat{y}$: while the former might predominantly utilize Mean Squared Error (MSE), the latter often gravitates towards the Cross-Entropy loss, especially for binary scenarios. It's not merely a choice: each function's mathematical underpinnings profoundly affect the model's learning dynamics. In particular, the role of convexity is central in these dynamics: the convex nature of a loss function generally promises a smoother gradient descent towards a global minimum. A convex function f over interval I satisfies the condition

$$f(\gamma x_1 + (1-\gamma)x_2) \leq \gamma f(x_1) + (1-\gamma) f(x_2)$$

for all $x_1, x_2 \in I$ and any $\gamma \in [0,1]$. However, the landscapes of deep learning models are often replete with local minima, complicating optimization processes.

Moreover, the susceptibility of some loss functions to data anomalies cannot be overlooked: for instance, the MSE's quadratic nature intensifies the impact of outliers. Such issues birth the need for more robust alternatives, like the Huber loss, which bridges the gap between linear and quadratic losses, offering resilience against anomalous data points. Furthermore, loss functions extend beyond mere error quantification: they can be imbued with regularization terms like the L1 or L2 norms, steering models towards specific mathematical behaviors.

To encapsulate, while loss functions serve as error metrics at a surface level, they are, in reality, intricately designed mathematical tools that shape and guide the training trajectory of neural networks, influencing their efficiency, robustness, and generalizability.

3.3 Common Types of Loss Functions

In the realm of neural networks and machine learning, loss functions serve as a compass guiding the optimization algorithms. By minimizing the value of these loss functions, a model hones its ability to make accurate predictions. Therefore, the elucidation of their characteristics, applications, and underlying mathematics is essential to select the most appropriate loss function tailored to specific tasks and challenges. Different loss functions possess unique properties that make them better suited for certain types of problems, from regression to classification and beyond. Moreover, understanding the intricacies of these functions aids in deciphering the convergence behavior of optimization algorithms and ensures a more informed and effective model training.

3.3.1 Regression Oriented Loss Functions

Mean Squared Error (MSE)

The Mean Squared Error (MSE) is a widely-used loss function, particularly relevant for regression tasks. The objective of the MSE is to compute the average of squared differences between predicted and true values. It can be formally defined as:

$$\text{MSE}(\hat{y}, y) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \tag{3.3}$$

The primary attribute of the MSE is its quadratic nature. By squaring the prediction errors, deviations between the predictions and actual values become more pronounced, particularly for larger discrepancies; this property reinforces a model's objective to minimize substantial prediction errors [61].

Several factors contribute to the popularity of the MSE in regression tasks: its mathematical simplicity and continuous differentiability make it a convenient choice for gradient-based optimization techniques. Moreover, the MSE can be interpreted geometrically, representing the Euclidean distance between predicted and actual values in the output space. Despite its merits, the MSE has limitations: due to its squared terms, it is highly sensitive to outliers. A substantial error in prediction for even a single data point can disproportionately affect the computed loss, potentially guiding the optimization towards suboptimal model parameters. Furthermore, its underlying assumptions, based on Gaussian error distributions, might not be ideal for data that do not adhere to such distributions.

Mean Absolute Error (MAE)

The Mean Absolute Error (MAE) is a quintessential loss function, chiefly utilized for regression analyses. It serves to compute the mean of the absolute discrepancies between the predicted and actual values, thus avoiding any predilection for the direction of the errors. The MAE can be concisely expressed as:

$$\mathrm{MAE}(\hat{y}, y) = \frac{1}{N} \sum_{i=1}^{N} |\hat{y}_i - y_i| \tag{3.4}$$

One of its salient characteristics is its linearity with respect to penalties. In contrast to the quadratic nature of the Mean Squared Error (MSE), the MAE metes out linear penalties for deviations, endowing it with notable robustness against outliers. This property is a consequence of the absolute value operation, ensuring a linear contribution of each error to the aggregate loss, precluding the substantial amplification observed in the case of squared errors [62].

From an analytical perspective, the MAE offers certain merits which enhance its suitability for specific regression scenarios.: its computational simplicity, coupled with a resistance to the exaggerated influence of outliers, renders it an apt error metric, especially in situations where the data is characterized by aberrations or a non-Gaussian distribution. Geometrically, it is noteworthy that the MAE corresponds to the L_1 norm between predicted and actual outputs, in contradistinction to the L_2 norm representation by the MSE. However, it is imperative to acknowledge that the MAE's resilience to outliers, while frequently advantageous, can occasionally be its Achilles' heel. In instances where Gaussian assumptions are valid, the MAE may exhibit diminished statistical efficiency in comparison to the MSE [63].

Huber Loss

The Huber Loss function offers a compromise between the Mean Squared Error (MSE) and the Mean Absolute Error (MAE). The mathematical design of the Huber Loss is such that it combines characteristics from both MSE and MAE, making it less sensitive to outliers compared to MSE alone. The Huber Loss is defined as:

$$L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| \leq \delta \\ \delta|y - \hat{y}| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \tag{3.5}$$

Within the threshold parameter δ, the Huber Loss uses a squared error similar to MSE. For errors exceeding this threshold, it employs a linear penalty, resembling the MAE. This combination allows the Huber Loss to effectively penalize errors without being overly influenced by outliers.

Compared to MSE, it demonstrates a reduced sensitivity to outliers, making it a preferable choice for datasets with potential anomalous values. Moreover, because it is continuously differentiable, the Huber Loss can be efficiently used with gradient-based optimization algorithms. However, the Huber Loss introduces the parameter δ, which requires careful tuning to match the specific distribution and nature of the data.

Log-Cosh Loss

The Log-Cosh Loss is a specialized loss function designed for regression tasks. It is defined using the logarithm of the hyperbolic cosine of the prediction error. Mathematically, it is given by:

$$\text{Log-Cosh}(\hat{y}, y) = \log(\cosh(\hat{y} - y)) \tag{3.6}$$

The use of the hyperbolic cosine function is motivated by its smoothness, offering a distinct alternative to the quadratic behavior of the Mean Squared Error (MSE). This smooth characteristic makes the Log-Cosh Loss more resilient to large prediction errors, which can be advantageous in datasets with occasional significant deviations. Moreover, its smooth error surface can aid optimization algorithms, particularly those that are gradient-based, by providing more consistent convergence paths. Additionally, its formulation naturally diminishes the impact of large, infrequent prediction errors, helping to prevent disproportionate penalties on the model for such deviations. On the downside, the hyperbolic cosine operation of the Log-Cosh Loss, while contributing to the function's favorable traits, adds computational complexity. This might be a point of concern in situations where computational speed is critical.

3.3.2 Classification Oriented Loss Functions

Cross-Entropy Loss and its Variants

The Cross-Entropy Loss, grounded in information theory, is a foundamental loss function tailored for classification tasks. It quantifies the divergence between the true label distribution y and the predicted probability distribution $\hat{y}$. This loss function manifests in several variations, each catered to different classification scenarios.

Binary Cross-Entropy: This is the simplest form of the cross-entropy loss, designed specifically for binary classification tasks. For the binary case, the formula is:

$$\mathrm{BCE}(\hat{y}, y) = -\sum_{i} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i) \tag{3.7}$$

Binary Cross-Entropy is especially fitting when the classes are nearly balanced or when each instance needs individual probabilistic scoring.

Categorical Cross-Entropy: Employed for multi-class classification, it quantifies the divergence when multiple classes are involved. Assuming C is the number of classes, it is delineated as:

$$\mathrm{CCE}(\hat{y}, y) = -\sum_{c=1}^{C} y_c \log(\hat{y}_c) \tag{3.8}$$

This is typically used when the labels are one-hot encoded.

Sparse Categorical Cross-Entropy: This is a slight variation of the Categorical Cross-Entropy, tailored for scenarios where the ground truth labels are integers rather than one-hot vectors. The mathematics remains akin to the Categorical Cross-Entropy, but the indexing and encoding mechanisms differ, ensuring computational efficiency in scenarios with a large number of classes [64].

The Cross-Entropy Loss and its variants are particularly advantageous for a myriad of reasons. Primarily, they are intrinsically aligned with classification objectives, prioritizing prediction accuracy. Their continuous and differentiable nature makes them inherently suitable for gradient descent-based optimization methods. However, they are not without challenges: specifically, numerical instabilities can manifest, especially when the predicted probabilities approach the extremes of 0 or 1. To address this, it becomes crucial to incorporate measures that guarantee computational stability throughout the optimization process. Within the scope of deep learning, variations of the Cross-Entropy Loss are widely adopted, and this is particularly evident in neural network architectures that utilize softmax or sigmoid activations in their output layers, as they are well-suited to the characteristics of this loss function.

Kullback-Leibler Divergence Loss

The Kullback-Leibler (KL) Divergence [65], measures the dissimilarity between two probability distributions. Specifically, it quantifies the difference between a proposed distribution P and a reference distribution Q. The divergence is mathematically expressed as:

$$\mathrm{KL}(P||Q) = \sum_i P(i) \log\left(\frac{P(i)}{Q(i)}\right) \tag{3.9}$$

for discrete distributions, and in integral form for continuous ones. Intuitively, the KL divergence evaluates the inefficiency of approximating Q using P in terms of expected logarithmic loss [66]. In the realm of variational autoencoders (VAEs), KL divergence plays an instrumental role in measuring the difference between the learned latent variable distribution and a specified prior [67].

While the KL divergence offers a robust and theoretically grounded means of comparing distributions, its asymmetry, $\mathrm{KL}(P||Q) \neq \mathrm{KL}(Q||P)$, necessitates careful consideration in practical scenarios. In the domain of classification, the KL divergence often finds utility in scenarios where both predicted and ground truth labels are expressed as probability distributions. Such representations are especially prevalent in soft-label classification tasks where the true labels are not strictly categorical but carry probabilistic interpretations. By measuring the divergence between predicted and true distributions, the KL divergence serves as an effective loss function, penalizing models based on their deviation from the true label probabilities.

Hinge Loss

The Hinge Loss function is a widely-used loss function tailored for binary classification tasks, and it plays a central role in the formulation of Support Vector Machines (SVMs). Its primary objective is to penalize not only the incorrect predictions but also those that are correct yet lack a sufficient margin of confidence. The Hinge Loss is mathematically defined as:

$$\text{Hinge}(\hat{y}, y) = \max(0, 1 - y \cdot \hat{y}) \tag{3.10}$$

The above formula indicates that even predictions that are correct will be penalized if they are within a certain margin (i.e., when $y \cdot \hat{y}$ is less than 1). This feature pushes the model to make predictions that are further from the decision boundary, increasing the model's confidence in its decisions.

There are several benefits to using the Hinge Loss: it promotes the max-margin principle, wherein it actively pushes models to make predictions that are not just accurate but also confidently placed away from the decision boundary. Furthermore, it often leads to sparse models, which can improve interpretability and reduce the risk of overfitting. However, it should be taken into account that the Hinge Loss is not differentiable everywhere, specifically at $y \cdot \hat{y} = 1$, which complicates the use of certain optimization methods that rely on gradient computations. Also, the Hinge Loss provides decision values instead of probability scores, which might restrict its use in cases where probabilistic outputs are desired.

Focal Loss

Introduced by Lin et al. in the context of RetinaNet [68], the Focal Loss offers an adaptation of the conventional cross-entropy loss, with a pronounced emphasis on remedying the prevalent issue of class imbalance inherent in object detection endeavors. Through the integration of a modulating factor into the standard cross-entropy formula, the Focal Loss reallocates weights, diminishing emphasis on confidently classified samples and amplifying focus on misclassified instances. The formal representation of the Focal Loss is:

$$\text{FL}(\hat{y}, y) = -\sum_i \alpha_i (1 - \hat{y}_i)^\gamma y_i \log(\hat{y}_i) \tag{3.11}$$

Herein, α_i functions as a class-balancing coefficient, γ denotes a modifiable focusing parameter, $\hat{y}_i$ signifies the computed probability corresponding to class i, and y_i delineates the veritable label.

The primary reason for introducing the Focal Loss is to address the significant challenge presented by the numerous easy-to-classify negatives within dense object detectors. The Focal Loss modifies the standard loss function to place more emphasis on difficult-to-classify negatives; this ensures that the overwhelming number of negative samples doesn't disproportionately influence the loss, preserving the detector's performance integrity. On the other hand, the complexities arising from the focusing and class-balancing parameters must also be considered. These add to the challenge of hyperparameter optimization. Despite these complexities, the Focal Loss has proven particularly valuable in one-stage object detection frameworks like RetinaNet, which often face significant class imbalances.

3.3.3 Embedding and Similarity Losses

Contrastive Loss

The Contrastive Loss is a loss function based on distance metrics, designed specifically for the development of embeddings in tasks that necessitate distinguishing between similar and dissimilar pairings. Its foundational principles are rooted in methodologies like one-shot learning and face verification. However, its predominant utilization is within the context of Siamese networks and related twin architectures [69].

Given a pair of samples, denoted as x_1 and x_2, their associated embeddings, $f(x_1)$ and $f(x_2)$, as defined by a neural structure, the Contrastive Loss can be formally expressed as:

$$L(y, d) = \frac{1}{2} y d^2 + \frac{1}{2}(1 - y) \max(\epsilon - d, 0)^2 \tag{3.12}$$

Within the given formula, d denotes the Euclidean distance between the embeddings, defined as $d = \|f(x_1) - f(x_2)\|_2$. Meanwhile, y serves as a binary label, identifying the pair as either similar (1) or dissimilar (0). The parameter ϵ acts as a set margin, ensuring that dissimilar pairs possess a minimum spatial distance of ϵ in the embedding space.

The Contrastive Loss operates with two main objectives: to reduce the distance between embeddings of similar pairings and to maintain a distance for dissimilar pairings that surpasses the specified margin [70]. This approach ensures that the produced embeddings effectively cluster similar data points while clearly distinguishing them from dissimilar ones. One of the notable advantages of the Contrastive Loss is its effectiveness in few-shot learning scenarios, where traditional classification-based losses might falter due to the paucity of labeled samples. However, a potential challenge involves the choice of pairings. Striking an optimal balance between positive and negative pairings is essential, as it can significantly impact model convergence and the quality of the derived embeddings [71].

Triplet Margin Loss

The Triplet Margin Loss, commonly referred to as triplet loss, is a specialized loss function designed for metric learning and embedding tasks. Its development was motivated by the desire to establish meaningful distance metrics within feature spaces, which have found extensive applications in areas like face recognition, one-shot learning, and advanced recommendation systems [72].

In contrast to standard loss functions that often rely on individual data points or pairwise combinations, the Triplet Margin Loss employs a three-part structure. This triadic setup comprises an anchor sample A, a matching positive sample P, and a non-matching negative sample N. The primary goal is to ensure that in the resulting embedding space, the distance between the anchor and the positive sample is smaller than the distance between the anchor and the negative sample by at least a predetermined margin. The mathematical representation is as follows:

$$L_{triplet} = \max\left(||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + \text{margin}, 0\right) \tag{3.13}$$

In the formulation, the function $f(\cdot)$ denotes the embedding transformation that maps data points into a specified feature space. The primary objective of the Triplet Margin Loss is to enlarge the distance between A and N relative to the distance between A and P, while maintaining a fixed margin.

A notable characteristic of the Triplet Margin Loss is its capacity to shape an embedding space where similar samples cluster together; this feature is particularly advantageous for applications focused on data similarity, including retrieval and verification tasks. However, the performance of the Triplet Margin Loss heavily depends on the selection of triplets: utilizing overly simple triplets, where the desired inequality is easily satisfied without necessitating model adjustments, can hinder the model's learning progression. Consequently, careful selection of triplets becomes paramount for the effective utilization of this loss function [73].

Cosine Similarity Loss

The cosine similarity metric quantifies the cosine of the angle between two non-zero vectors, providing a measure of their directional congruence. Predominantly in the realms of natural language processing and computer vision, the cosine similarity has been utilized, particularly when the angular relationship between vectors, as opposed to their magnitudes, elucidates meaningful connections, such as in retrieval tasks of text or images [74]. Mathematically, the cosine similarity for vectors A and B is expressed as:

$$\text{sim}(A, B) = \frac{A \cdot B}{\|A\|\|B\|} \tag{3.14}$$

Here, $\cdot$ signifies the dot product and $\| \cdot \|$ corresponds to the L2 norm.

The cosine similarity can be adapted into a loss function for machine learning models that could leverage this geometric orientation. This loss function, frequently referred to as the cosine distance loss, is defined by:

$$L_{\text{cosine}} = 1 - \text{sim}(A, B) \tag{3.15}$$

Minimizing this loss function emphasizes enhancing the similarity between the two vectors. The notable characteristic of the cosine similarity loss is its insensitivity to fluctuations in vector magnitudes. By focusing on directional alignment, this loss promotes representations that emphasize structural and semantic relationships over pure magnitudes [75]. Such a property is advantageous, especially when dealing with high-dimensional data or scenarios where vector magnitudes might be skewed by external influences, rendering them unreliable. Nevertheless, there are aspects of the cosine similarity loss that require consideration. It inherently assumes that the vectors' directional orientation captures all relevant information, potentially neglecting patterns rooted in their magnitudes. Moreover, ensuring that input vectors are non-zero is crucial to avoid potential degeneracies.

In various fields, from word embeddings to deep metric learning, the cosine similarity loss has proven its efficacy, underlining models that prioritize angular relationships over mere magnitude-centric proximity [76].

3.3.4 Object Detection and Segmentation Losses

Localization Loss in Object Detection

Object detection, a sub-field of computer vision, aims to classify and demarcate objects' spatial boundaries within images. This requires predicting bounding boxes that encompass the objects. In this context, the localization loss serves as a crucial metric, measuring the differences between the predicted bounding boxes and their actual ground truths.

The localization loss, commonly based on the Mean Squared Error (MSE) between the actual and predicted bounding box coordinates, is formulated as:

$$L_{loc} = \sum_{i} (x_i - \hat{x}_i)^2 \tag{3.16}$$

Here, x_i and $\hat{x}_i$ denote the ground truth and predicted coordinates, respectively.

This loss function's significance is evident in its role in ensuring accurate object localization, thus refining the detailed accuracy of the detection task. However, the effectiveness of the localization loss is greatly dependent on the accuracy and precision of the ground truth annotations. Inaccurate or imprecise annotations can negatively influence the learning trajectory, highlighting the critical need for carefully annotated datasets in object detection.

Confidence Loss in Object Detection

Object detection frameworks, in addition to localizing objects within an image, also provide an estimation of the confidence associated with these localizations. This estimation quantifies the likelihood that a predicted bounding box correctly encompasses the intended target object. To evaluate the disparity between the predicted confidence values and their associated ground truths, the confidence loss is employed.

The confidence loss can be formally expressed as:

$$L_{conf} = \sum_{i} (c_i - \hat{c}_i)^2 \tag{3.17}$$

where c_i denotes the ground truth confidence and $\hat{c}_i$ signifies the predicted confidence for the ith bounding box.

The primary objective of the confidence loss is twofold: Firstly, it aims to reduce the number of false positives, which refer to bounding boxes that incorrectly predict the presence of objects. Secondly, it seeks to enhance the confidence scores associated with valid detections, ensuring that the model's predictions are both confident and accurate. Such accurate calibration is of utmost importance, especially when object detection systems are deployed in applications that require high levels of precision and reliability. However, it is essential to recognize the susceptibility of the confidence loss to the challenges posed by class imbalances: in situations where certain object classes are less prevalent, the model might exhibit diminished confidence in predictions corresponding to these classes or may inadvertently favor predictions aligned with more common classes. Such tendencies highlight the imperative need for either datasets with balanced representations or the adoption of techniques such as oversampling or employing class-weighted losses to mitigate these imbalances.

Dice Loss

In image segmentation tasks, the goal is to delineate the boundaries of objects with precision at the pixel level. While conventional loss functions such as cross-entropy treat each pixel as an independent entity, this approach can be suboptimal for segmentation. The Dice Loss, derived from the Dice Coefficient (also known as the Sørensen–Dice index), provides a more suitable evaluation metric, focusing on the overlap between the predicted segmentation and the ground truth. The Dice Loss is mathematically defined as:

$$L_{dice} = 1 - \frac{2 \times \text{intersection}(y, \hat{y})}{\text{volume}(y) + \text{volume}(\hat{y})} \tag{3.18}$$

Here, y signifies the ground truth segmentation, and $\hat{y}$ represents the predicted segmentation. The function intersection computes the number of pixels common to both segmentations, while the function volume determines the total pixel count of a given segmentation.

The primary objective of the Dice Loss is to optimize the overlap between the predicted segmentation and the true segmentation: by formulating the loss based on both the intersection and the individual volumes of the segmentations, it encourages models to closely align the predicted segmentation with the ground truth. However, a potential limitation of the Dice Loss is its behavior in scenarios with class imbalances, particularly when certain classes have significantly fewer instances. Given its formulation, the Dice Loss might inadvertently prioritize dominant classes, potentially overlooking less frequent ones. Such characteristics underscore the significance of utilizing balanced datasets or implementing strategies to mitigate these imbalances.

Intersection over Union Loss

Intersection over Union (IoU), also known as the Jaccard Index, is a commonly used metric in image segmentation tasks. It quantifies the overlap between predicted regions and the corresponding ground truth annotations. While traditionally employed as an evaluation metric, recent advancements in deep learning have repurposed IoU as a loss function, especially for tasks related to object detection and semantic segmentation [77]. For two sets A and B, the IoU is mathematically expressed as:

$$\mathrm{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.19}$$

Here, $|\cdot|$ denotes the cardinality of a set. Within the framework of image segmentation, sets A and B typically represent binary masks corresponding to the predicted and ground truth regions, respectively. Building on the IoU metric, the IoU loss is defined as:

$$L_{\mathrm{IoU}} = 1 - \mathrm{IoU}(A, B) \tag{3.20}$$

The objective of minimizing this loss function is to enhance the overlap between the model's predictions and the true segments within an image.

The appeal of the IoU loss is its direct alignment with a standard evaluation metric in segmentation, offering a clear optimization target [78]. Distinct from pixel-based loss functions, the IoU loss focuses on the geometric consistency between segments and offers resilience against class imbalance. However, employing the IoU loss in models is not without challenges. Its set-based formulation can introduce non-differentiability in the loss landscape, which may hinder gradient-based optimization techniques. Several methods have been proposed to circumvent this issue, including the use of smoothed approximations or surrogate losses that capture the essence of the IoU, while being more conducive to optimization [79].

The IoU loss has been integrated into contemporary segmentation models, directing them towards outcomes that are more consistent with established evaluation criteria, thereby enhancing the fidelity of segmentation outputs [80].

3.3.5 Loss Functions for Generative Models

Reconstruction Loss for Autoencoders

Autoencoders are deep learning models designed to encode data into a compressed latent space and then decode it back to its original form. Their effectiveness is typically assessed using the reconstruction loss, which quantifies how well the decoded output matches the original input. Mathematically, the reconstruction loss is often defined as the Euclidean distance (or L2 norm) between the input data x and its reconstructed version:

$$L_{recon} = ||x - \text{Dec}(\text{Enc}(x))||^2 \tag{3.21}$$

This loss function reinforces the primary goal of an autoencoder: to reconstruct the input as accurately as possible from its encoded representation. While the reconstruction loss ensures the model's fidelity to the input, a potential limitation is its emphasis on pixel-level accuracy. This might lead to reconstructions that capture minute details at the expense of higher-level semantic features. Hence, it's crucial to consider other evaluation metrics that might capture broader data patterns.

Generator Loss for GANs

Generative Adversarial Networks (GANs) are prominent tools in unsupervised machine learning. They consist of two components: a generator and a discriminator. In GANs, the generator produces samples aiming to replicate real data. The effectiveness of the generator is measured by the generator loss, which quantifies its success in deceiving the discriminator. Given a latent variable z from a specified distribution, the generator, G, generates a sample. The discriminator, D, then provides a probability score indicating its perceived authenticity of this sample. The generator loss is expressed as:

$$L_G = -\log(D(G(z))) \tag{3.22}$$

This equation describes the generator's aim: to increase the likelihood that the discriminator accepts the generated sample as real. As the generator improves, it becomes more challenging for the discriminator, driving the system toward a Nash equilibrium where the generated samples closely resemble real data.

The primary benefit of the generator loss in GANs is its ability to guide the generation of high-quality samples that closely match the true data distribution. However, a notable drawback is the risk of mode collapse. This occurs when the generator predominantly produces samples from specific regions of the data distribution that easily deceive the discriminator, resulting in a lack of diversity in the generated samples.

Discriminator Loss for GANs

Within the Generative Adversarial Networks (GANs) framework, the discriminator's role is to differentiate between authentic data samples and those generated by the generator. The efficacy of the discriminator is quantified using the discriminator loss.

The mathematical formulation of the discriminator loss consists of two components. The term $-\log(D(x))$ quantifies the discriminator's ability to correctly classify authentic samples. In contrast, the term $-\log(1 - D(G(z)))$ measures its skill in identifying generated samples as non-authentic. The combined loss is thus:

$$L_D = -\log(D(x)) - \log(1 - D(G(z))) \tag{3.23}$$

This formulation underscores the discriminator's dual responsibilities: to recognize genuine data and to correctly label the generator's output as synthetic.
The discriminator loss effectively focuses on the essential task of binary classification: categorizing samples as real or generated. By linking the loss to both kinds of classification errors, it ensures that the discriminator rigorously evaluates each sample. Despite its utility, an overly competent discriminator might outperform the generator, leading to an imbalance in the adversarial relationship. This imbalance can disrupt training equilibrium and potentially hinder the generator's development, causing instability during training.

3.3.6 The Multifaceted Landscape of Loss Functions

The field of loss functions in machine learning is extensive and perpetually evolving, mirroring the advancements and shifts in the broader domain. The selection of loss functions discussed in this document represents a noteworthy fraction of the broader range available. Each loss function, characterized by its mathematical design, embodies specific objectives and assumptions regarding data distributions, model targets, and the characteristics of the solution space.
The mathematical structure and details of each loss function articulate the objectives it aims to achieve: for instance, loss functions designed for regression tasks commonly aim to minimize prediction discrepancies, while those for classification focus on accurate instance labeling, and generative tasks emphasize the authenticity of produced data. As machine learning delves into increasingly complex problem areas, there arises a growing necessity to tailor or adapt loss functions to particular requirements. This often involves integrating mathematical insights with domain-specific knowledge, whether it means developing entirely novel loss functions to address unique challenges or amalgamating existing ones to capitalize on combined benefits.
In conclusion, the array of loss functions not only showcases the mathematical foundations of machine learning but also indicates the objectives set for a given model.

Their multiplicity and intricacy underline the significance of the optimization process in determining the final quality and efficacy of learned solutions.

3.4 Gradient Descent and Backpropagation

3.4.1 A Geometric Perspective: The Topological Landscape of Loss Surfaces

Training neural networks can be conceptualized as a challenging endeavor within high-dimensional parameter spaces punctuated by intricate loss topographies. To elucidate the gradient descent algorithm's mechanics, one might consider it akin to navigating a manifold embedded in a higher-dimensional space. Let's instantiate a manifold $\mathcal{M}$ in a Euclidean space $\mathbb{R}^n$, where each point on $\mathcal{M}$ corresponds to a distinct set of model parameters. The function value at each of these points, which we might term the "altitude", represents the loss function's magnitude, capturing the divergence between the model's predictions and the ground truth. The aim is to identify a point $p^* \in \mathcal{M}$ such that the loss function $L(p^*)$ is minimized, signifying an optimal configuration of model parameters. In the absence of global knowledge of $\mathcal{M}$, the optimization process necessitates reliance on local geometric properties. Particularly, the gradient at a given point provides essential cues about the steepest ascent direction. Analogously, in gradient descent optimization, this gradient information guides the iterative refinement of model parameters.

This geometric representation underscores the intricacies associated with gradient descent, particularly when contending with the high-dimensional, non-convex landscapes frequently encountered in deep learning scenarios:

- **Local Minima and Saddle Points:** The manifold $\mathcal{M}$ need not be convex and may exhibit multiple basins (local minima) and plateaus (saddle points). One might erroneously converge to a suboptimal local minimum or stagnate near a saddle point. These topological features analogously manifest as optimization challenges in neural network training.

- **Step Magnitude and Learning Rate:** The magnitude of the adjustment at each iteration, governed by the local gradient, is analogous to the learning rate in gradient descent. An aggressive update might expedite convergence but risks overshooting. Conversely, conservative updates, while ensuring stability, might render the convergence protracted.

- **Incorporation of Previous Gradients:** Just as momentum derived from preceding steps can aid in traversing plateaus or surmounting slight ascents, variants of gradient descent incorporate historical gradients to influence the current update direction. Such strategies facilitate faster convergence and ameliorate challenges induced by irregular topologies.

- **Incorporating Stochastic Elements:** Introducing stochasticity in the optimization trajectory, mirroring techniques like stochastic gradient descent, can prove advantageous in circumventing local minima.

- **Complexity of the Loss Surface:** The dimensionality and intricacy of the loss landscape escalate with increasing model sophistication. While deep neural networks correspond to high-dimensional manifolds, challenging to intuitively visualize, the foundational principle of exploiting gradient information for optimization persists.

In essence, this geometric perspective, while theoretical, helps clarify the intricacies and approaches relevant to optimizing neural networks. As further discussions delve into the finer details of gradient descent, this geometric viewpoint provides an intuitive foundation for the mathematical details.

3.4.2 Mathematical Framework of Gradient Descent

An incisive exploration of the gradient descent algorithm necessitates a rigorous mathematical treatment. The gradient, a seminal construct in the domain of multivariable calculus, elucidates the essence of the optimization dynamics. Let $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$ be a scalar-valued function defined over an n-dimensional Euclidean space. Within our discourse, $\mathcal{L}$ represents the loss function, and n corresponds to the cardinality of parameters in the model, encapsulated by the vector θ. The gradient of $\mathcal{L}$ evaluated at a point θ, represented as $\nabla\mathcal{L}(\theta)$, manifests as an n-dimensional vector. Each of its components is the partial derivative of $\mathcal{L}$ relative to the respective parameter:

$$\nabla\mathcal{L}(\theta) = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \theta_1} \\ \frac{\partial \mathcal{L}}{\partial \theta_2} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \theta_n} \end{bmatrix} \tag{3.24}$$

Inherently, this vector signifies the direction of maximum ascent of the function $\mathcal{L}$. In contradistinction, the negation of the gradient delineates the trajectory of steepest descent. During the iterative refinement, the gradient imparts crucial information regarding the requisite modifications to the parameters θ to achieve minimization of $\mathcal{L}$. The quintessential update schema for gradient descent can be articulated as:

$$\theta_{t+1} = \theta_t - \alpha\nabla\mathcal{L}(\theta_t) \tag{3.25}$$

In the preceding expression, α, referred to as the learning rate, is a non-negative scalar that demarcates the amplitude of the update, oriented in the antithetical direction of the gradient.

3.4.3 The Mechanics of Backpropagation

Backpropagation, an abbreviation of "backward propagation of errors", stands as a pivotal algorithmic approach for training feedforward neural networks. Fundamentally, it determines the distribution of error emanating from the network's terminal layer across preceding layers, thereby furnishing insights on parameter adjustments to attenuate this error. This error distribution, aligned with the network parameters, materializes as the gradient of the loss concerning individual weights.

For a neural network characterized by L layers, let S^h symbolize the weighted input, and x^h denote the activations at the h-th layer. The associated weights and biases for the h-th layer are represented by w^h and b^h respectively. The error at the h-th layer, represented as δ^h, is expressible in terms of the error in its succeeding layer:

$$\delta^h = (w^{h+1})^\top \delta^{h+1} \odot \lambda'(S^h) \tag{3.26}$$

Herein, λ' represents the derivative of the activation function, while $\odot$ denotes element-wise product. The gradient of the loss function $\mathcal{L}$ relative to the weights and biases materializes as:

$$\frac{\partial \mathcal{L}}{\partial w^h} = \delta^h (x^{h-1})^T \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial b^h} = \delta^h \tag{3.27}$$

Backpropagation's inherent elegance emerges from its recurrent application of the aforementioned relations. Starting from the output layer and progressing inversely, it efficiently derives the gradient for each constituent parameter.

3.4.4 Chain Rule: The Mathematical Foundation of Backpropagation

At its core, backpropagation employs the chain rule from calculus to determine the manner in which errors propagate through a multi-layered architecture. This rule delineates the differentiation procedure for composite functions.

Let's consider a sequence of functions x, g, f, and $\mathcal{L}$ such that $y = g(x)$, $z = f(y)$, and $\mathcal{L} = \mathcal{L}(z)$. The derivative of $\mathcal{L}$ concerning x, articulated through its intermediate functions, is given by:

$$\frac{d\mathcal{L}}{dx} = \frac{d\mathcal{L}}{dz} \cdot \frac{dz}{dy} \cdot \frac{dy}{dx} = \frac{d\mathcal{L}}{df} \cdot \frac{df}{dg} \cdot \frac{dg}{dx} \tag{3.28}$$

Within the domain of neural networks, characterized by their layered configurations, the chain rule is of paramount significance. The influence of a weight in a given layer on the final loss is determined by its combined effects as they propagate through all the subsequent layers. This interconnected influence is effectively described by the chain rule of differentiation. As instance, for a rudimentary network comprising two layers, and given an error δ^{h+1} in the subsequent layer, the impact attributable to weights w^h, adopting the same notation as before, in the preceding layer can be represented as:

$$\frac{\partial \mathcal{L}}{\partial w^h} = \frac{\partial \mathcal{L}}{\partial x^{h+1}} \cdot \frac{\partial x^{h+1}}{\partial S^{h+1}} \cdot \frac{\partial S^{h+1}}{\partial x^h} \cdot \frac{\partial x^h}{\partial S^h} \cdot \frac{\partial S^h}{\partial w^h} \tag{3.29}$$

Each term in this product denotes a direct derivative. Yet, their compounded effect, traversing layers, encapsulates the essence of the backpropagation algorithm.

3.4.5 Challenges: Vanishing and Exploding Gradients

Training deep neural networks, especially those with many layers, introduces a set of challenges that can impede or even disrupt the learning process. Central to these challenges are the phenomena of vanishing and exploding gradients. Both issues can significantly affect the convergence properties and stability of the optimization process.

Vanishing Gradient Problem

Consider a deep feedforward neural network, where the error at layer h, δ^h, can be expressed in terms of the error in the subsequent layer $h + 1$ as shown by Eq. (3.26) where λ' represents the derivative of an activation function, like the sigmoid function $\sigma(s)$, which is given by:

$$\sigma'(s) = \sigma(s)(1 - \sigma(s)) \tag{3.30}$$

For values of s that are either very large (positive or negative), the derivative becomes close to zero. As backpropagation proceeds through the layers, repeated multiplications involving small derivatives can cause the gradient to diminish to minuscule values. In mathematical terms, if the product of the Jacobians and the derivatives of the activations have magnitudes less than 1, they can lead to this decay, formally expressed as:

$$\prod_{l=1}^{L} ||J_{\lambda^h}|| < 1 \tag{3.31}$$

where J_{λ^h} represents the Jacobian matrix for layer h. This decay means that the weight updates for earlier layers become negligible, leading to stalled learning, especially for layers close to the input.

Exploding Gradient Problem

Conversely, when the gradients are propagated back, especially in architectures like RNNs where the same weights are applied at each time step, gradients can accumulate and grow exponentially with the sequence length. If we again consider the product of Jacobians, the exploding gradient problem occurs when:

$$\prod_{l=1}^{L} ||J_{\lambda^h}|| > 1 \tag{3.32}$$

Under such circumstances, the gradient updates can be excessively large, leading to unstable training dynamics and weight values that could diverge to infinity.

Mitigating the Challenges

Training deep neural networks is fraught with challenges, particularly those related to the behavior of gradients. As networks grow deeper, issues like the vanishing and exploding gradient problems become more pronounced, which can lead to unstable training dynamics or even render networks intractable. Such challenges stem from the cumulative multiplicative effects of layer weights and activations, which can squash or amplify gradients exponentially as they propagate back through the network. Recognizing the critical impact these problems have on the viability of deep learning, the research community has invested significant effort in devising solutions. Various strategies have emerged to counteract these issues:

- **Weight Initialization:** Proper initialization schemes, like He or Xavier initialization, aim to set the initial weights in a manner that balances the forward and backward flows of signals [40] (as will be discussed in Section 6.2.2).
- **Batch Normalization:** By normalizing activations within a layer, it ensures that they have a stable distribution, mitigating the extreme values that can exacerbate gradient issues [55] (as discussed in Section 2.5.1).
- **Gradient Clipping:** A simple yet effective method for RNNs where gradients larger than a threshold are scaled down, preventing them from exploding [50].
- **Architectural Choices:** Using activation functions like ReLU and its variants (Leaky ReLU, Parametric ReLU) can help reduce the vanishing gradient problem since their derivatives do not squash values as the sigmoid or tanh functions do (as discussed in Section 3.2).
- **Skip Connections:** Architectures like Residual Networks (ResNets) incorporate direct paths from earlier layers to later ones, ensuring gradient flow across layers [15] (as discussed in Section 2.5.1).

In essence, while the intricacies of gradient flow in deep networks introduce challenges, the research community's understanding and solutions have evolved, enabling the training of even very deep architectures with stability and efficiency.

3.4.6 The Importance of Data Preprocessing and Normalization

The process of data preparation, particularly preprocessing and normalization, holds paramount importance in the training dynamics of neural networks. At its core, the intent behind these operations is to transform the raw data into a more digestible form for the neural network, subsequently promoting efficient and stable learning. One of the primary goals of normalization is to transform the raw data, usually denoted as X, into a form that eases the optimization landscape. This can result in a loss surface that is more symmetrical, less skewed, and generally easier for optimization algorithms to navigate. Mathematically, this improvement can often be quantified by a reduced condition number of the Hessian matrix associated with the loss function [81].
Two popular methods of data normalization are z-score normalization and Min-Max scaling. Z-score normalization is given by:

$$X' = \frac{X - \mu}{\sigma} \tag{3.33}$$

where X' is the normalized data, μ is the mean, and σ is the standard deviation of the feature vector X. Normalizing data in this manner, for example, often leads to a more symmetrical loss landscape, which in turn aids optimization algorithms.
Min-Max scaling, which normalizes data into a predetermined range, usually [0, 1], serves as an effective method for preserving the original distribution of the data while scaling it to a tractable range. The formula for Min-Max scaling is given by:

$$X' = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{3.34}$$

One of the key advantages of this approach is that it is well-suited for data with a bounded range. Unlike z-score normalization, which could potentially transform the data to unbounded values, Min-Max scaling ensures that the data fits within a tight, predictable range. This is particularly useful when working with activation functions that are sensitive to the input scale [82], such as the sigmoid or hyperbolic tangent functions. Moreover, Min-Max scaling often eases the learning process by allowing optimization algorithms to find a suitable learning rate more quickly.

Both methods, as well as other normalization techniques, have their own merits and use-cases. Beyond aiding the optimization algorithm in navigating the loss landscape, normalization also promotes numerical stability [83]. Without such preprocessing steps, features with varying scales can introduce notably large activations and gradients, leading to numerical issues or dominating the optimization process. For example, in the context of backpropagation, a skewed scale of features can result in a gradient $\nabla\mathcal{L}$ that may cause numerical instability or overshooting during the optimization steps [14]. Normalizing inputs, therefore, acts as a safeguard, ensuring that activations remain within a reasonable range and preventing potential saturation of neurons, especially in architectures that employ activation functions like sigmoid or tanh.

Another pivotal aspect is the harmonization of learning dynamics across the network. By ensuring that features have similar scales through normalization, weights update in a more synchronized manner. This leads to smoother and more predictable learning dynamics, making it easier to diagnose issues and fine-tune hyperparameters [84].

3.5 Optimization Algorithms

Optimization plays a pivotal role in the learning process of neural networks [3]. It steers the iterative refinement of model parameters to improve performance on a given task. While the foundational idea is rooted in gradient descent, the high-dimensional, non-convex nature of the loss landscapes in deep learning presents unique challenges [85], necessitating the evolution of advanced optimization techniques.

3.5.1 Beyond Basic Gradient Descent: Advanced Optimizers

The elementary gradient descent algorithm serves as a prototype optimization technique, where the primary idea is to adjust the model parameters iteratively in the direction of the steepest decrease of the loss function. However, the topological intricacies of the loss landscape associated with deep networks can impede the algorithm's efficiency and effectiveness. Several concerns emanate from the practical application of basic gradient descent [86]:

- **Convergence Rate:** Basic gradient descent might converge exceedingly slowly, especially in regions where the loss surface is flat or when faced with ill-conditioned optimization problems.

- **Local Minima and Saddle Points:** In high-dimensional spaces, while local minima are less of a concern, saddle points (where some dimensions have a positive curvature and others have a negative curvature) can stall optimization. The gradient is close to zero at saddle points, much like at local minima.

- **Oscillations and Overshooting:** A constant learning rate might lead to oscillations, especially in steep ravines of the loss landscape. An overly large learning rate can cause the optimization to overshoot minima, leading to divergence, while an excessively small one might result in slow convergence.

- **Hyperparameter Sensitivity:** The algorithm's performance is often sensitive to the choice of hyperparameters, particularly the learning rate. Choosing an optimal learning rate is non-trivial and might require extensive experimentation.

Recognizing these challenges, researchers have proposed a gamut of advanced optimization algorithms, ingeniously designed to address the aforementioned issues [87].

These algorithms often incorporate adaptive learning rates, momentum, or second-order optimization techniques to expedite convergence and navigate the intricate loss landscapes effectively. In the subsequent sections, we will discuss some of the most prominent of these algorithms, such as AdaGrad, Adam, and RMSProp, elucidating their mathematical foundations and practical implications.

3.5.2 Gradient Descent Augmented with Momentum

The momentum method, drawing inspiration from classical physics, enhances gradient descent by incorporating a memory of prior gradients. This momentum, viewed as an accumulated velocity for gradient updates, ensures consistent update directionality, thus facilitating more efficient traversal of the optimization landscape.

In optimization, the concept of momentum is pivotal, especially when addressing challenges in the loss landscapes of deep models. Deep models possess high-dimensional parameter spaces, often peppered with numerous local minima. Despite many of these minima having comparable loss values, their influence on model generalization can differ: momentum, by accumulating gradient directions over iterations, helps in circumventing potential ensnarement in these suboptimal zones. Furthermore, deep models frequently present extensive plateau regions with near-zero gradients, which can stymie basic gradient descent. Momentum, with its history-driven approach, aids in traversing these flat expanses more adeptly, guiding the optimization towards the optimal solution. Formally, the momentum-based update rule is given by:

$$\begin{aligned} v_{t+1} &= \beta v_t + (1-\beta)\nabla\mathcal{L}(\theta_t) \\ \theta_{t+1} &= \theta_t - \alpha v_{t+1} \end{aligned} \tag{3.35}$$

Where:

- v_t denotes the accumulated gradient (or velocity) at iteration t, combining past gradients in a manner that places exponential emphasis on their recency.

- β is the momentum coefficient, dictating the extent of historical gradient influence in the update. Empirically, values in the range of 0.8 to 0.9 have proven effective, although domain-specific tuning may occasionally be necessary [3].

While the addition of momentum introduces another dimension of hyperparameter tuning, the empirical benefits – quicker convergence and heightened resilience against suboptimal parameter initialization – frequently justify its inclusion in deep learning optimization frameworks [84].

3.5.3 AdaGrad

The AdaGrad (Adaptive Gradient Algorithm) optimization method seeks to address one of the fundamental challenges in the gradient-based optimization landscape: the selection and adjustment of the learning rate. While a universal learning rate (as used in standard gradient descent) may be suboptimal due to the disparate gradient behaviors across parameters, AdaGrad proposes a dynamic adjustment mechanism that scales the learning rate for each parameter based on its historical gradient magnitude.
Key insights of AdaGrad emphasize the importance of adaptive learning rates based on the observed gradients of parameters. Parameters that consistently experience high gradients should undergo updates with a reduced learning rate to avoid overshooting the optimal values. On the other hand, parameters that exhibit smaller gradients – especially those associated with sparse features – deserve more substantial updates to ensure that they receive adequate attention and aren't overlooked during optimization.
The update rules for AdaGrad can be expressed as:

$$\begin{aligned} g_{t+1} &= g_t + \nabla L(\theta_t)^2 \\ \theta_{t+1} &= \theta_t - \frac{\alpha}{\sqrt{g_{t+1} + \epsilon}} \nabla L(\theta_t) \end{aligned} \tag{3.36}$$

Here:

- g_t represents an accumulator for squared gradients up to time t.

- ϵ is a small constant added for numerical stability, ensuring there's no division by zero.

While AdaGrad provides a robust mechanism for adaptive learning rates, a potential drawback is its accumulation method. Since it continues to add squared gradients, the accumulated value can grow large, causing the effective learning rate to diminish, potentially stalling optimization in later phases [88].

3.5.4 RMSProp

Recognizing the potential pitfalls of AdaGrad's monotonically decreasing learning rates, RMSProp (Root Mean Square Propagation) introduces a modification to the gradient accumulation process. Instead of summing all historical squared gradients, RMSProp calculates an exponentially decaying moving average. This modification guarantees that the method retains sensitivity to recent gradient behaviors and is not unduly influenced by earlier gradient contributions. The RMSProp update equations are:

$$\begin{aligned} g_{t+1} &= \beta g_t + (1-\beta)\nabla L(\theta_t)^2 \\ \theta_{t+1} &= \theta_t - \frac{\alpha}{\sqrt{g_{t+1}+\epsilon}}\nabla L(\theta_t) \end{aligned} \tag{3.37}$$

Here:

- g_t is the accumulated squared gradient, updated as a moving average.

- β represents the decay rate, determining the weight of the previous squared gradient in the current accumulation. Typically, β is set close to 1, such as 0.9.

Through this mechanism, RMSProp maintains a more balanced and adaptive learning rate, especially for deep neural network training where AdaGrad might face challenges due to its ever-decreasing rates [89].

3.5.5 Adam

The Adam (Adaptive Moment Estimation) optimization algorithm [16] represents a fusion of momentum and RMSProp's principles. It maintains exponential moving averages for both gradients (akin to momentum) and their squared values (akin to RMSProp's adaptive learning rate). This hybrid approach endows Adam with the benefits of both adaptive gradient methods and momentum-based methods.
The update rules for Adam are as follows:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla L(\theta_t) \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)\nabla L(\theta_t)^2 \\ \hat{m_t} &= \frac{m_t}{1-\beta_1^t} \\ \hat{v_t} &= \frac{v_t}{1-\beta_2^t} \\ \theta_{t+1} &= \theta_t - \frac{\alpha \hat{m_t}}{\sqrt{\hat{v_t}}+\epsilon} \end{aligned} \tag{3.38}$$

Where:

- m_t and v_t represent the moving averages of the gradient and squared gradient, respectively.
- $\hat{m_t}$ and $\hat{v_t}$ are bias-corrected versions of the moving averages to counteract their initializations at the origin.
- β_1 and β_2 are exponential decay rates for the moving averages, typically set close to 1, such as 0.9 and 0.999, respectively.
- ϵ is added for numerical stability.

In summary, the Adam optimizer combines the strengths of momentum and RMSProp, offering an algorithm that is both adaptive and momentum-aware. This dual capability contributes to its robustness in navigating intricate loss landscapes common in deep neural networks. Since its introduction, Adam has gained substantial traction and is widely recognized as one of the most utilized optimization algorithms in deep learning. Its ability to consistently deliver competitive results across various tasks and models, without the need for intricate hyperparameter tuning, underscores its utility and effectiveness in the realm of neural network training.

3.5.6 The Criticality of Learning Rate and Hyperparameter Interplay in Gradient-Based Optimization

In the intricate domain of gradient-based optimization for neural networks, hyperparameters play an essential role in determining the trajectory and efficiency of the optimization process. Among these, the learning rate, denoted α, holds a particularly central position, serving as the linchpin that can make or break the training dynamics. The learning rate governs the magnitude of parameter updates based on computed gradients. Consequently, its value has direct ramifications on the convergence properties of the optimization algorithm:

- A learning rate that is set too high risks erratic parameter oscillations, as the updates become too aggressive, potentially overshooting minima and even causing the optimization to diverge [3].

- On the opposite end of the spectrum, an overly conservative learning rate, while ensuring stability, can lead to protracted convergence. The diminutive steps might not only make the training process inordinately lengthy but also risk the optimization getting ensnared in shallow, sub-optimal local minima or saddle points [85].

While the primacy of α is undeniable, it is crucial to appreciate its synergistic interplay with other hyperparameters. Parameters like β in momentum-based methods, or ϵ in algorithms such as AdaGrad, RMSProp, and Adam, don't merely act in a supplementary capacity. Their values, in concert with the learning rate, define the optimization's overall behavior and resilience against various challenges presented by the loss landscape. This ensemble of hyperparameters necessitates a coordinated tuning strategy to holistically shape the optimization trajectory.

The process of hyperparameter selection, however, is far from trivial. More often than not, it involves a blend of empirical evaluations, cross-validation techniques, and occasionally, insights derived from domain-specific expertise [90]. While grid or random search might provide initial insights, more nuanced methods like Bayesian optimization are gaining traction for hyperparameter tuning due to their efficiency [91].

Recognizing the dynamic nature of deep learning training, recent methodologies advocate for adaptive hyperparameter strategies. One such strategy is learning rate annealing, which involves the systematic reduction of the learning rate as the optimization advances, catering to the intuition that as we approach an optimum, more granular steps can help in finer convergence. Conversely, learning rate warm-up, wherein the learning rate is progressively increased during the initial phases of training, can stave off premature convergence and navigate regions of high curvature without destabilizing the optimization process [84].

In conclusion, while the individual roles of hyperparameters like the learning rate are evident, a holistic understanding of their intricate interdependencies and the development of adaptive strategies remain active research frontiers, emphasizing the profundity and dynamism inherent in the neural network optimization landscape.

3.6 Regularization Techniques

Overfitting is a prevalent challenge in machine learning and is particularly pronounced in deep learning due to the vast number of parameters in neural networks. An overfitted model has learned the training data too well, capturing not just the underlying patterns but also the noise or random fluctuations. Consequently, while the model excels on the training set, it struggles with new, unseen data, leading to poor test set performance.

The mathematical manifestation of overfitting is an increasing disparity between the training loss and the validation/test loss as training progresses. The repercussions of overfitting are multifaceted: a decline in model robustness, heightened sensitivity to input changes, and diminished predictive efficacy on novel samples.

3.6.1 Regularization as Loss Term

To counter overfitting, regularization introduces a penalty to the loss function, deterring the emergence of overly intricate models.

L1 Regularization (Lasso)

L1 regularization, or Lasso, adds the sum of the absolute values of the parameters to the loss function. This approach promotes sparse weight vectors, often nullifying many weights. The mathematical representation of the L1 penalty is:

$$L_{L1} = \alpha \sum_i |\theta_i| \tag{3.39}$$

Here, α signifies the regularization strength. A more considerable α amplifies the regularization effect, simplifying the model. In contrast, a smaller α allows the model to fit the training data more closely, potentially causing overfitting.

L2 Regularization (Ridge)

L2 regularization, known as Ridge, integrates the sum of the squared values of the parameters into the loss function. It tends to reduce weight magnitudes towards zero without necessarily inducing sparsity. The L2 penalty is mathematically expressed as:

$$L_{L2} = \alpha \sum_i \theta_i^2 \tag{3.40}$$

Elastic Net Regularization

Elastic Net strikes a balance by combining both L1 and L2 regularization techniques. It's especially useful when dealing with correlated features. The penalty term for Elastic Net is:

$$L_{ElasticNet} = \lambda\alpha \sum_i |\theta_i| + (1-\lambda)\alpha \sum_i \theta_i^2 \tag{3.41}$$

Where λ governs the mix of L1 and L2 regularization.

Given the huge parameter count in deep neural networks, regularization becomes indispensable for preventing overfitting. The choice between L1, L2, or Elastic Net depends on the problem at hand and the nature of the data. Furthermore, it's crucial to remember that regularization plays a pivotal role in the bias-variance trade-off, ensuring models strike the right balance and generalize effectively to unseen data.

3.6.2 Dropout, Early Stopping, and Data Augmentation

Dropout

Dropout is a regularization technique introduced to mitigate overfitting in large neural networks [92]. During training, a random subset of neurons is temporarily deactivated, ensuring that no single neuron is dependent on the presence of others, pushing them to develop more robust features. Mathematically, for an output vector o_h from layer h, the dropout operation is represented by:

$$o'_h = o_h \odot d \tag{3.42}$$

where $\odot$ denotes element-wise multiplication, and d is a binary mask vector derived from a Bernoulli random variable with probability p, indicating the dropout rate.

This technique can be viewed as training a pseudo-ensemble of neural networks sharing parameters. It reduces the co-adaptation of neurons, ensuring each neuron contributes significantly to the task without over-relying on others. This approach enhances the model's generalization capabilities for unseen data. During inference, while all neurons are active, their outputs are adjusted to maintain consistency with the dropout applied during training.

Early Stopping

Early stopping is a simple yet effective technique to prevent overfitting in iterative training algorithms like gradient descent [93]. As the training progresses and the model starts fitting to the noise in the data, its performance on a held-out validation set often starts to deteriorate. By monitoring the validation performance, training can be halted once the validation error ceases to improve, or even starts increasing. This way, the model parameters are "saved" at a point where the validation performance was optimal, thereby providing a balance between underfitting and overfitting.

Data Augmentation

Data augmentation is a strategy primarily used to increase the effective size of the training dataset without collecting new data [94]. By applying various transformations such as rotations, translations, zooming, and horizontal flipping, new training samples are created. These augmented samples introduce variability and force the model to learn more invariant features. For instance, in image recognition tasks, regardless of the position, orientation, or scale of an object in an image, the model should recognize it. Data augmentation mimics such real-world variability, enhancing the model's ability to generalize across diverse, unseen data.

3.6.3 Other Regularization Techniques

Several other techniques also deserve mention in the realm of neural network regularization and stabilization. Among these is Weight Noise, where the introduction of noise to weights can thwart overfitting. This strategy both deters weights from settling into narrow minima and simulates a kind of ensemble learning. Another noteworthy method is Batch Normalization. While its initial purpose was to address the internal covariate shift dilemma, it inadvertently offers a regularizing effect on models [55]. Lastly, Spectral Normalization has garnered attention, especially in the context of Generative Adversarial Networks. By constraining the spectral norm of weight matrices, it ensures a more stabilized training regime [95].

Regularization remains a cornerstone in the training of deep neural networks. Such methodologies, when aptly applied, offer a buffer against overfitting, ensuring that these mathematical models maintain their extrapolative prowess when confronted with previously unobserved data. The techniques elaborated upon in this section, underscore the manifold and nuanced regularization strategies at the disposal of researchers. Their utility transcends mere diversity; they impose constraints on the model's complexity, which in turn solidifies the model's ability to generalize effectively. This generalization is paramount, especially when one considers deployment scenarios that might deviate markedly from the environments wherein the models were trained. Furthermore, it has been documented that regularization augments a model's resilience against adversarial perturbations, lending models a heightened reliability in empirical contexts [3, 92].

3.6.4 Regularization and the Bias-Variance Trade-off

In statistical learning and machine learning theory, one frequently encounters the interplay between two types of errors: bias and variance. Understanding this trade-off is pivotal in grasping the essence and implications of regularization. Bias refers to the error introduced by approximating the real-world complexity with a potentially too simplistic model. In essence, it measures how much our model's predictions differ from the true values if we were to reuse the same training data. A high bias can lead to an underfitting scenario, where our model fails to capture the underlying structure of the data [96]. On the other hand, variance measures our model's sensitivity to small fluctuations in the training set. Essentially, it quantifies how much our predictions would vary if we were to train our model on a different dataset. A model with high variance might be so flexible that it captures not just the genuine data patterns but also the random noise, a phenomenon termed overfitting [61].

The role of regularization, then, emerges as a tool to manage this trade-off: by introducing penalties for certain model complexities, regularization effectively restricts the potential variance of the model. As the strength of regularization increases, the variance of the model decreases, preventing it from overfitting to the training data. However, there's a caveat: as the model is simplified, its bias tends to increase. This introduces the potential risk of the model becoming too generalized, such that it might miss important underlying patterns in the data. The challenge, therefore, is to find a harmonious balance where both bias and variance are minimized. This is often achieved by carefully tuning the regularization parameters [96]. In practice, one might visualize this trade-off by plotting training and validation errors as functions of model complexity. As the model becomes more complex, training error typically decreases, while validation error follows a U-shaped curve. The point of minimum validation error represents our desired optimal balance between bias and variance.

3.7 Batch Learning versus Online Learning

Neural networks, like many machine learning algorithms, can be trained using various approaches that depend on how data is fed into them. Among these, batch and online learning stand out [3]. Both strategies have their unique strengths, and the choice between them often hinges on the specific requirements of a given problem and computational constraints.

3.7.1 Definitions, Benefits and Challenges of Batch and Online Learning

Batch and online learning represent two fundamental paradigms in machine learning, each with its distinct characteristics, advantages, and challenges. The former focuses on the utilization of the entire dataset for each model update, ensuring stability, while the latter emphasizes adaptability by updating the model incrementally using individual data points. Batch Learning relies on the complete dataset for every update cycle: it ensures a *stable convergence* due to the consistent influence of the complete dataset, resulting in reduced variance in gradient estimates and, consequently, fewer oscillations in the learning trajectory [3]. Contemporary computational libraries are optimized for batch operations, making this approach particularly efficient with matrix operations. However, its memory-intensive nature and potential inertia in dynamic environments – stemming from its aggregated approach – can sometimes be limitations.

Alternatively termed as incremental learning, Online learning updates the model parameters for each individual data point. Thanks to this, it offers *high adaptability*, allowing the model to quickly respond to changing data distributions, which is invaluable in dynamic settings. Its granular approach also ensures *memory efficiency.* Nevertheless, the method can lead to *noisy updates* due to the inherent variance among data points. This can cause the model to exhibit an unstable learning path, susceptible to frequent fluctuations [3]. Moreover, while it is memory-efficient, online learning can sometimes be more computationally taxing.

3.7.2 The Concept and Advantage of Mini-batch Learning

In the realm of neural network training paradigms, *Mini-batch Learning* elegantly bridges the gap between batch and online learning. The approach is characterized by its use of small, randomly selected subsets of the dataset, known as "mini-batches", for each update. Represented mathematically, the weight update for a given mini-batch can be articulated as:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_k(\theta; X_k, y_k) \tag{3.43}$$

In this equation, $\mathcal{L}_k$ stands for the loss computed over the k^{th} mini-batch, denoted by data X_k and corresponding labels y_k.

So, mini-batch learning emerges as an optimal compromise between the rigor of batch learning and the adaptability of online learning. One of the primary advantages of this paradigm is its computational efficiency. Modern hardware architectures, notably Graphics Processing Units (GPUs), are inherently designed for parallel processing. By breaking the dataset into manageable mini-batches, this method is able to effectively harness the parallelism inherent in such architectures, resulting in an expedited training process. Empirically, mini-batch learning often exhibits superior computational efficiency compared to its online counterpart, especially when tailored to leverage GPU architectures. Furthermore, mini-batch learning offers a harmonious amalgamation of stability and flexibility. While batch learning is characterized by its stable convergence, it can be less responsive due to its reliance on the entire dataset for each update. Online learning, in contrast, updates the model with each individual data point, rendering it highly adaptable but susceptible to noisy trajectories. Mini-batch learning ingeniously mitigates these extremes. By processing subsets of the dataset, it achieves a form of data aggregation that lends stability to the gradient updates. This ensures a more consistent learning path, devoid of the pronounced oscillations that might be observed with purely online methods. Simultaneously, by not being constrained by the entire dataset for every update, it retains a commendable degree of adaptability, positioning it as more responsive than traditional batch learning [87]. Lastly, in terms of memory efficiency, mini-batch learning stands out. Analogous to online learning, it obviates the need to accommodate the complete dataset in memory during each gradient update. This inherent design choice results in a methodology that is not only nimble in its adaptability but also economical in its memory requirements. Such efficiency becomes crucial, especially when dealing with large datasets or when computational resources are at a premium.

Chapter 4

Neural Network Architectures

4.1 Introduction

The expressiveness of artificial neural networks fundamentally emanates from the composition of their layers and units, akin to how the richness of polynomials arises from the combination of their terms and degrees. At its core, each layer in a neural network transforms and refines the representation of its input data, building progressively more abstract and discerning features. Layers can be combined and sequenced in myriad configurations, yielding a vast space of potential architectures. Mathematically, consider a single layer defined by a transformation T_i. In a sequential architecture of L layers, the output is the composite function:

$$T_L \circ T_{L-1} \circ \ldots \circ T_2 \circ T_1(x) \tag{4.1}$$

However, this representation only captures the essence of vanilla feed-forward networks. Introducing skip, residual, and other such connections, we venture into a more intricate space of functions. Further versatility is introduced with network enhancements like dropout, normalization techniques, and dynamic architectures. These not only influence the representational power but also shape the optimization landscape, facilitating or impeding the learning process.

While the combinatorial assembly of layers grants expressivity to neural networks, it's paramount to recognize that this expressiveness isn't devoid of challenges or guarantees. The *universal approximation theorem* affirms that a neural network can approximate any continuous function given sufficient hidden units. However, the theorem doesn't prescribe how to find the appropriate weights or even hint at the practical feasibility of the network's depth and width. It's also imperative to note that the architecture alone isn't the sole determinant of a network's success. The loss function, which quantifies the discrepancy between predictions and ground truths, plays an equally pivotal role. Architectures, while defining the representational power, must be paired with apt loss functions tailored to the task at hand, be it regression, classification, or more intricate endeavors like generative modeling. Furthermore, the inherent intricacies of the learning process, gradient flow, and convergence properties are inextricably tied to the chosen architecture. Aspects like vanishing or exploding gradients, overfitting, or even adversarial vulnerabilities are shaped by the interplay between architecture, loss, and the optimization algorithm.

In the following sections of this chapter, we will delve into the detailed study of neural network architectures. The objective is to examine the intricate relationship between these architectures, specific loss functions, and their targeted applications. We will analyze how this combination of elements impacts the learning process in neural networks. This analysis aims to build a thorough understanding of the multifaceted aspects of neural network learning and its various implementations.

4.2 Feedforward Neural Networks

Feedforward neural networks, also known as Multilayer Perceptrons (MLPs), serve as the foundational bedrock in the domain of artificial neural networks and have paved the way for the development of more specialized architectures [3, 61].

Composed as a stack of Dense Layers, described in Section 2.4.1, in mathematical terms a feedforward network is defined by a sequence of transformations $T_1, T_2, \ldots, T_L$ from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$, represented as $\mathcal{N}_\theta : \mathcal{X} \to \mathcal{Y}$. Here, θ contains all trainable parameters. For a network with L layers, the mapping $\mathcal{N}_\theta$ is essentially a composition of these transformations [3, 61]. Each of these transformations T typically involves a linear operation followed by a non-linear activation.

One of the defining features of this architecture is its unidirectional data flow. The network processes information linearly from the input to the output layers without any feedback loops, thus simplifying the forward computational pass [2, 61]. In summary, the architecture of feedforward neural networks combines linear transformations and non-linear activations in a streamlined, unidirectional flow. This elegant construction is central to its broad applicability and serves as a prototype for more specialized neural network architectures.

4.2.1 Use-cases of FFNs

Feedforward neural networks are renowned for their versatility, finding applications across a range of domains that require function approximation, classification, and regression. In the area of classification, FNNs are frequently employed to categorize data into predefined classes. Applications span diverse fields including natural language processing for text categorization, healthcare for medical diagnosis, and finance for credit risk assessment [61]. Moving to regression problems, FNNs have proven their mettle in tasks that require predicting continuous output variables. Examples range from real estate, where housing prices are forecasted, to finance, where stock market trends are projected [3]. The Universal Approximation Theorem substantiates the usage of FNNs for scientific computing tasks such as solving differential equations and modeling complex systems [97]. In addition to these tasks, FNNs are also used as feature extractors. Their hidden layers can transform the original high-dimensional feature space into a more manageable, and often more interpretable, form [2]. This transformational capability is particularly useful for anomaly detection, where FNNs are trained to learn the distribution of normal data and hence can effectively identify outliers.

While they lack the inherent design to handle sequential data, FNNs are occasionally used for specific problems in natural language processing where fixed-size input features like bag-of-words representations are sufficient [4]. However, it is essential to note that their unidirectional and memory-less architecture makes them less suitable for tasks requiring the understanding of temporal or spatial dependencies, such as time-series prediction or image recognition [37].

4.2.2 Properties of FFNs

Feedforward Neural Networks exhibit a unique set of properties that confer both advantages and limitations. One of their most significant properties, as outlined before, is their adherence to the Universal Approximation Theorem. This theorem highlights the profound computational potential concealed within their seemingly simple structure.

FNNs are noted for their computational efficiency, both in memory and runtime [3]: this efficiency makes them well-suited for tasks involving static input-output mappings, where the absence of memory or recurrent connections is not detrimental. Their straightforward architecture also renders them relatively interpretable, making them an appealing option for a wide array of machine learning problems.

However, the absence of more specialized structures like recurrent or convolutional layers can result in a large number of parameters when handling complex or high-dimensional data such as images [37]. They are also prone to overfitting if the network's size is disproportionately large relative to the complexity of the problem [61].

In summary, while FNNs offer a robust foundation for various machine learning tasks, their specific properties must be carefully weighed when dealing with more complex or specialized data structures.

4.3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) have become a cornerstone in the realm of artificial neural networks, specifically tailored for handling grid-structured data such as images. Typically relying on alternating stacks of Convolutional layers (refer to Sec. 2.4.2) and Pooling layers (with reference to Sec. 2.4.3), CNNs often incorporate Dense layers towards the latter stages of the architecture. These layers are adept at preserving spatial hierarchies and extracting local features from input data. Batch normalization and dropout techniques may also be interwoven to enhance model generalization and combat overfitting. Thanks to this, CNNs have transcended their initial applications in image recognition to become versatile tools for a plethora of tasks involving spatial hierarchies and local receptive fields [2, 98].

4.3.1 Use-cases of CNNs

Convolutional Neural Networks have proven indispensable in various fields, primarily owing to their exceptional performance in tasks requiring the understanding of spatial hierarchies. Originally designed for image recognition tasks, they have become the gold standard in computer vision applications [98]. In image recognition, CNNs can discern intricate details at varying levels of abstraction. Early layers often detect basic features like edges and corners, while deeper layers capture complex patterns and object parts. This property has been harnessed in facial recognition technologies, allowing for accurate identification and verification across a multitude of scenarios [99]. Beyond basic image classification, CNNs are integral to object detection systems, where the task is not just to classify an image but to identify multiple objects within the image along with their positions. These capabilities have had a transformative impact on fields such as surveillance and robotics. Another area where CNNs have demonstrated their prowess is medical imaging. The learned features and hierarchical abstractions are valuable in tasks such as tissue classification, anomaly detection, and even complex diagnostics. For instance, CNNs are increasingly used to analyze X-rays, MRIs, and CT scans, thereby aiding in the early detection of diseases and abnormalities [100]. In natural language processing, they have been employed to capture local semantic features within a sequence of text, thus contributing to tasks like sentiment analysis and text summarization [101]. This demonstrates the flexibility of CNN architectures in capturing local structures, even when the input data is not inherently grid-like. Interestingly, CNNs have also penetrated into the realm of video analysis and autonomous vehicles. In such dynamic environments, CNNs offer real-time object detection and tracking. For instance, in autonomous driving, they can identify pedestrians, track surrounding vehicles, and recognize traffic signals, thus enabling safe and reliable navigation [102]. In summary, the spatial feature learning capability of CNNs has made them a versatile tool, applicable to a wide range of problems well beyond their initial remit. Whether it is static images or sequential data, CNNs have set new performance standards across various domains.

4.3.2 Properties of CNNs

CNNs exhibit several distinct properties that furnish them with both advantages and limitations, particularly when contrasted with other neural network architectures. A defining feature is the use of local receptive fields, which ensures that neurons in convolutional layers are attuned to specific, localized regions of the input. This design decision optimizes the network for the detection of local spatial hierarchies.

Another salient aspect of CNNs is their incorporation of pooling layers, commonly situated after convolutional layers. These pooling layers serve a dual purpose: they reduce data dimensionality and introduce translational invariance into the network, features that contribute to the model's computational efficiency and robustness. The architecture further enhances efficiency through the principle of weight sharing across convolutional layers. This reduces the number of trainable parameters, curtailing the risk of overfitting and rendering CNNs particularly effective for high-dimensional inputs [2].

However, this specialization also imparts certain limitations: CNNs are generally less suitable for tasks that demand an understanding of sequences or temporal dynamics. For such tasks, it is often beneficial to integrate CNNs with other architectures like Recurrent Neural Networks (RNNs) to capture both spatial and temporal dependencies [103].

4.4 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) stand as a pivotal advancement in the domain of artificial neural networks, specifically designed for the analysis of sequential and temporal data. Stemming from the early work by Elman [25], RNNs distinguish themselves from feedforward and convolutional architectures through their unique topological structure. At the crux of this structure lie cyclical connections, allowing RNNs to perpetuate information across time steps, effectively endowing them with a dynamic form of "memory". This inherent capacity to retain and process information from preceding states has rendered RNNs indispensable in tasks demanding an understanding of temporal dependencies and sequential patterns. This is realized through a structure composed by stacks on RNN layers (see Section 2.4.4), that receive input not only from the previous layer in the stack but also from the output of the current layer's previous time step. Consequently, these layers effectively act as a looping mechanism over time steps, maintaining a latent state that evolves as the sequence progresses.

4.4.1 Use-cases of RNNs

Recurrent Neural Networks (RNNs) have carved out a significant niche for themselves in the vast landscape of machine learning, thanks to the ability to remember and leverage information from previous states, which naturally suits it for tasks that require understanding patterns over sequences and time. Arguably one of the most transformative applications of RNNs lies within the realm of NLP. Their sequential nature and memory-driven architecture render them ideal for comprehending the nuances and temporal dependencies intrinsic to languages. Machine translation, for instance, is a complex task that involves converting sentences from one language to another while preserving the semantic essence. Traditional methods often fell short in capturing long-range dependencies between words and phrases. RNNs, with their sequential memory, brought a paradigm shift to this domain. Works such as that by Sutskever et al. [4] showcased the potential of RNNs in bridging linguistic gaps across languages. Similarly, sentiment analysis, which delves into discerning the underlying sentiment of a given text, also reaped the benefits of RNNs. The inherent ability of RNNs to process text in chunks and remember past words provides a contextual understanding, vital for tasks like these [104]. Furthermore, the world of speech recognition, a domain rife with temporal sequences, has been revolutionized by RNNs, capturing the temporal dynamics of spoken words with remarkable accuracy [105].

Beyond the confines of text and speech, RNNs have demonstrated significant prowess in predicting future values based on historical data. Be it the volatile terrains of stock markets, the unpredictable nature of weather patterns, or the nuanced fluctuations in electrical loads, RNNs have emerged as reliable prognosticators [106]. Their strength lies in understanding past patterns and extrapolating them into the future, making them particularly well-suited for these tasks.

While the aforementioned applications underscore the analytical strength of RNNs, their foray into creative domains showcases their versatility. The music industry, for instance, has been enthralled by the capability of RNNs to generate novel compositions. By training on vast datasets of existing music, RNNs can produce melodies that resonate with human-created compositions, highlighting their potential to understand and replicate the intricate structures and temporal patterns in music [107]. Similarly, in the realm of entertainment, RNNs have been employed to generate scripts, suggesting a promising avenue where machines can potentially complement human creativity.

4.4.2 Properties of RNNs

The inherent design of RNNs allows them to effectively manage sequences of variable lengths, which is a critical property for many practical applications. Unlike feedforward neural networks, which require fixed-size input, RNNs can process input sequences of varying lengths, making them ideal for tasks where the sequence length cannot be predetermined. However, this sequential processing nature of RNNs also introduces certain application limitations. One significant limitation is the challenge in parallelizing computations: since RNNs process data sequentially, each step depends on the computations of the previous step, making it difficult to parallelize operations across multiple processing units. This dependency constrains the ability to leverage modern parallel processing hardware to its full extent, which is a notable drawback compared to architectures like CNNs that are more amenable to parallelization. This limitation becomes particularly pronounced in tasks involving very long sequences or when processing large datasets, where the sequential nature of RNNs can lead to longer training times.

Moreover, while RNNs are adept at capturing short-term dependencies, they often struggle with long-term dependencies due to the vanishing gradient problem [50]. This challenge is somewhat mitigated by advanced architectures like LSTMs and GRUs (as discussed Section 2.4.4). These versions, with their refined architectures, are designed to overcome the limitations commonly associated with traditional RNNs; this enhancement broadens their applicability and improves their effectiveness in tasks that involve processing sequential data. However, the aforementioned drawback still remains an area of active research to improve RNNs' ability to capture and process long-range dependencies in sequential data more effectively.

4.5 Encoder-Decoder Architectures

The encoder-decoder architecture has established itself as an essential construct in the domain of sequence-to-sequence learning. Introduced in the context of neural machine translation, the encoder-decoder paradigm has found utility in a range of applications from speech synthesis to time series forecasting [4, 28]. Formally, let us consider a source sequence $x = (x_1, x_2, \ldots, x_T)$ of length T. The encoder maps this sequence to a fixed-size context vector **c** in the latent space, given by:

$$c = \mathcal{N}_{\text{encoder}}(x; \theta_{\text{encoder}}) \tag{4.2}$$

where $\mathcal{N}_{\text{encoder}}$ represents the encoder function parameterized by θ_{encoder}. The decoder then generates an output sequence $y = (y_1, y_2, \ldots, y_{T'})$ of length T', using the context c as:

$$y = \mathcal{N}_{\text{decoder}}(c; \theta_{\text{decoder}}) \tag{4.3}$$

where $\mathcal{N}_{\text{decoder}}$ denotes the decoder function with parameters θ_{decoder}. In this framework, the role of the context vector is pivotal: it serves as a bridge between the encoder and decoder, summarizing the input sequence's information into a dense representation that the decoder then utilizes to generate the output sequence.

Thus, the encoder-decoder architecture exemplifies a versatile and effective model for sequence-to-sequence tasks. Its strength lies in its capacity to condense complex input sequences into a compact context vector, and then to extrapolate from this representation to produce varied output sequences. This unique ability to transform and translate between different forms and lengths of data makes it an invaluable tool in a large variety of applicative scenarios.

4.5.1 Use-Cases for Encoder-Decoder Architectures

The encoder-decoder paradigm has found extensive applications across diverse domains, underlining its versatility and efficacy in sequence-to-sequence challenges. Perhaps the most canonical application of encoder-decoder frameworks lies in the domain of machine translation [4, 28]. The task is to transform a sentence or paragraph from a source language into a coherent and contextually appropriate translation in a target language. The encoder captures the semantic essence of the source text, condensing its intricacies into a fixed-size context vector. The decoder then, equipped often with attention mechanisms to discern and prioritize relevant parts of the input sequence [108], renders this context into the target language, thereby achieving translation.

Another domain of applications for this kind of neural structure is speech synthesis: this task demands the conversion of textual information into its spoken counterpart [109]. This involves not just lexical and syntactic comprehension, but also a nuanced understanding of intonation, stress, and rhythm. Encoder-decoder architectures facilitate this by ingesting textual sequences, encapsulating them into meaningful context vectors, and then decoding these into appropriate audio sequences, thus enabling machines to generate human-like speech.

Beyond mere sequence-to-sequence transformations, encoder-decoder paradigms have also demonstrated prowess in bridging modalities. Image captioning is a quintessential example of this cross-modal translation [110]. Here, the encoder, often a convolutional neural network, processes an image and distills its myriad features into a context vector. The decoder, typically a recurrent neural network, then translates this contextual understanding into a textual description, providing a semantically apt caption for the image.

4.5.2 Properties of Encoder-Decoder Architectures

The encoder-decoder architecture specializes in sequence-to-sequence mapping, particularly useful when input and output sequences differ in length or structure, thus offering flexibility for diverse sequence-related problems [4]. Central to this is the fixed-size context vector, which the encoder populates after processing the input. This vector holds the input's semantic essence, bridging the encoder and decoder. One of the most transformative advancements in this realm is the integration of attention mechanisms with these architectures: they enable the decoder to dynamically emphasize various input segments, mitigating challenges in long sequence translations. This is particularly essential when a segment of the input critically influences a distant segment of the output [108].

The encoder-decoder's strengths lie in its adaptability across tasks like machine translation and image captioning, demonstrating its versatility with different data modalities [28]. Furthermore, it effectively handles variable sequence lengths and supports end-to-end training using backpropagation, eliminating the need for task-specific tweaks or handcrafted features. However, foundational encoder-decoder models, without attention mechanisms, face challenges with long-term dependencies. The fixed-size context vector's constraint may lead to data loss for extensive sequences. Additionally, their scalability can be a concern; especially attention-equipped models can be computationally intensive, necessitating significant computational resources and extended training [5].

4.6 Autoencoders

Rooted in the Encoder-Decoder architectures, Autoencoders, which fall under the umbrella of unsupervised learning within neural network architectures, serve as a practical application of data compression in conjunction with deep learning principles. Their primary function is to identify and extract latent patterns from large datasets. The data processing in an autoencoder consists of a dimensionality reduction phase, leading to a condensed latent representation, and a subsequent reconstruction phase aiming to replicate the original data. The overarching goals are to maintain a concise yet informative latent representation and to ensure that the difference between the reconstructed and original data is minimized. Mathematically, let (x be the input data vector. An autoencoder aims to find an encoder function φ and a decoder function ψ such that:

$$\psi(\phi(x)) \approx x \tag{4.4}$$

Where $\varphi : \mathbb{R}^d \to \mathbb{R}^k$ and $\psi : \mathbb{R}^k \to \mathbb{R}^d$ with typically $k < d$. The middle layer, with the reduced dimensionality k, is the compressed or encoded representation of the data, also named *latent space*.

4.6.1 Use-Cases for Autoencoders

Autoencoders have found applicability in numerous domains due to their inherent capabilities. One of their primary roles is in dimensionality reduction. Analogous to traditional techniques like Principal Component Analysis (PCA), autoencoders can transform data into a lower-dimensional space, which is advantageous not only for visualization purposes but also for mitigating computational burdens in subsequent analyses [111]. Another significant application of autoencoders is in anomaly detection. By calibrating an autoencoder with standard data, the reconstruction error for new data points can be evaluated. If the model registers a conspicuously high reconstruction error, it can hint at the presence of anomalies or outliers, as these are instances the model is not well-versed in reconstructing [112]. Furthermore, there exists a specialized variant known as denoising autoencoders that are tailored for denoising. These models are trained using corrupted data as input and the corresponding clean data as the desired output. Through this training paradigm, they acquire the ability to rectify and counteract the introduced disturbances, effectively restoring the original data [113]. Lastly, autoencoders are valuable tools for feature extraction. They can discern and extract pivotal features from data that can subsequently be utilized for a variety of machine learning endeavors. Given that the latent space's compressed representation encapsulates essential information, it emerges as a potent reservoir for feature extraction [114]

4.6.2 Properties of Autoencoders

Autoencoders are characterized by a symmetric architecture, where the encoder and decoder often have mirrored configurations. This ensures a balanced procedure for both compression into and expansion from the latent space, which is the pivotal juncture of the architecture. The latent space effectively compresses the most significant features of the data, functioning as a critical information bottleneck.

One of the notable strengths of autoencoders is their flexibility. In contrast to many conventional dimensionality reduction methods that operate linearly, autoencoders have the capability to encapsulate intricate, non-linear correlations within the data [111]. Additionally, they exhibit an inherent quality of self-supervision. Autoencoders ascertain representations without necessitating explicitly labeled data, positioning them as particularly beneficial resources when labeled datasets are sparse or entirely unavailable. On the other side, a predominant concern about this kind of architectures is overfitting: due to their intricate architectures, autoencoders might at times merely reproduce the input without capturing substantial representations, a scenario that becomes particularly pronounced when the neural network's scale outweighs the dataset's size [113]. Furthermore, the training phase of deep autoencoders is not devoid of optimization complexities. Often, the convergence during training can settle at local minima, presenting hurdles in achieving an optimal model configuration [115].

4.7 Variational Autoencoders (VAEs)

Variational Autoencoders (VAEs) represent a confluence of autoencoders and probabilistic graphical models. These neural network architectures seamlessly blend deterministic and stochastic behaviors, resulting in the capacity to generate diverse yet coherent samples. Fundamentally, VAEs differ from traditional autoencoders by their introduction of a probabilistic layer: rather than mapping input data to a fixed latent vector, they map inputs to distributions within the latent space [67]. The encoder in a VAE outputs parameters of a probability distribution (typically Gaussian) defined in the latent space. When decoding, a sample is drawn from this distribution, ensuring that the decoded output incorporates inherent variations present in the data. This stochastic approach is coupled with a regularization term in the loss function, which encourages the learned latent space distributions to approximate a prior (typically a standard normal distribution). The equilibrium achieved between data fidelity, as quantified by the reconstruction loss, and compliance with the established prior, as measured by the Kullback-Leibler divergence, facilitates the extraction of both efficient and semantically relevant representations in the latent space [116].

$$\mathcal{L}(x, \hat{x}, \mu, \sigma^2) = \text{ReconstructionLoss}(x, \hat{x}) + \beta \cdot D_{KL}(\mathcal{N}(\mu, \sigma^2)||\mathcal{N}(0, 1)) \tag{4.5}$$

where ReconstructionLoss is typically the mean squared error between the original data and its reconstruction, D_{KL} represents the Kullback-Leibler divergence, and β is a hyperparameter that controls the trade-off between the two terms.

4.7.1 Use-Cases for VAEs

VAEs have found utility in an array of applications due to their generative capabilities. Data Generation is a direct application where VAEs are used to produce novel samples that are coherent with a given dataset, useful for data augmentation or synthetic data creation [67]. In the arena of Semi-supervised Learning, VAEs exploit both labeled and unlabeled data to enhance model performance, tapping into the vast swaths of unlabeled data available in many domains [117]. Their probabilistic nature makes VAEs apt for Anomaly Detection, as data instances that deviate significantly from the learned distributions result in high reconstruction errors. Lastly, the compact and structured latent space that VAEs produce can be explored and manipulated to discern and even modify inherent characteristics of data, facilitating tasks such as feature disentanglement and interpolation [118].

4.7.2 Properties of VAEs

Variational Autoencoders (VAEs) are distinguished by their unique blend of probabilistic encoding and deep learning architecture. While they share some architectural symmetry with traditional autoencoders, their defining characteristic lies in their ability to encode input data into probabilistic distributions rather than fixed latent representations. This probabilistic approach introduces an element of randomness in the decoding process, enabling the generation of diverse outputs from a given input [67].
The regularization of the latent space is another salient feature of VAEs. By imposing a regularization term in the loss function [116], VAEs ensure that the distributions in the latent space adhere closely to a predetermined prior, typically a standard normal distribution. This regularization not only contributes to a more structured and interpretable latent space but also helps in mitigating issues like overfitting, which are common in complex neural network models.
VAEs exhibit a remarkable generative capability, owing to their stochastic nature and deep learning foundations. They can create new data instances that are coherent with the characteristics of the input data, making them particularly useful in fields like synthetic data generation and semi-supervised learning. The structured latent space of VAEs also allows for intuitive operations such as interpolation and feature disentanglement, which are crucial for in-depth explorations and manipulations of data

representations. However, the intricacies of VAEs, stemming from their probabilistic elements and regularization, introduce certain complexities in their implementation and optimization. Achieving the right balance between accurate data reconstruction and effective regularization can be challenging, often leading to training difficulties [119]. This complexity is further amplified by phenomena like posterior collapse, where the model may prioritize the regularization term at the expense of data reconstruction [120], thus hindering the effectiveness of the latent space in capturing meaningful data representations.

4.8 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) [11] represent a paradigm shift in unsupervised machine learning. At the heart of this structure are two adversarially training neural models: the generator and the discriminator. The generator seeks to produce synthetic data samples, attempting to imitate the genuine data distribution. In contrast, the discriminator endeavors to differentiate between actual data and the counterfeits produced by the generator. The mathematical underpinning of this adversarial contest is framed as a minimax problem:

$$\min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))] \tag{4.6}$$

between the generator loss and the discriminator loss (further details are discussed in Sections 3.2).

4.8.1 Use-cases of GANs

Generative Adversarial Networks (GANs) have significantly impacted a wide spectrum of applications. Among their prominent roles, GANs shine in data generation. By harnessing the potential of the generator, these networks synthesize novel data samples closely resembling their training datasets. This capability has been transformative, particularly in areas such as art generation and augmenting datasets with synthetic examples.

GANs are not limited to data generation but also find utility in applications such as *Image-to-Image Translation.* Architectures like CycleGAN [121] have been developed to convert images from one domain to another, facilitating operations like style transfer and modality conversion. Another application of GANs is *Super-Resolution*, where the objective is to improve the resolution and clarity of images. Super-resolution GANs (SRGANs) [122] are employed in this context to upgrade images that might be blurry or of a lower resolution to a higher resolution with enhanced details. Additionally, these architectures have been adapted for specific data generation tasks through conditional generation. In this regard, architectures such as Conditional GANs (cGANs) [123] are prominent. These networks, when provided with specific conditions or labels, generate outputs in line with those conditions, allowing for more controlled content generation.

4.8.2 Properties of GANs

Generative Adversarial Networks (GANs) possess specific properties that distinguish them from other machine learning models. Central to their design, GANs are proficient at approximating complex data distributions due to their adversarial training mechanism, eliminating the need for explicit probabilistic modeling. Additionally, GANs operate on a *non-equilibrium game* principle, wherein during the training process, the generator and discriminator continuously adjust to each other, leading to a dynamic optimization process.

Among the notable advantages of GANs is their capacity for generating high-quality data. Properly trained GANs can produce synthetic samples that closely resemble real data. Furthermore, GANs are recognized for their modularity, as they can be conditioned for specific data generation tasks, making them valuable in applications requiring targeted outputs. However, GANs are frequently noted for their training instability [124]. The balance between the generator and discriminator, while central to their operation, can also lead to complications in training. Effective training often requires careful hyperparameter selection. Another limitation is the occurrence of mode collapse [125], where the generator produces a limited variety of samples, failing to capture the diversity of the real data distribution, which can undermine the purpose of the generative model.

4.9 Attention and Transformer Models

The Transformer architecture [5], presented a new paradigm in sequence-to-sequence learning. While conventional models such as RNNs and LSTMs demonstrated efficacy, their sequential nature posed challenges in handling long-range dependencies and leveraging parallel processing. The Transformer, utilizing attention mechanisms, mitigates these issues by allowing concurrent processing of different segments of an input sequence to capture its intricacies

As can be imagined, at the heart of the Transformer is the attention mechanism. Conceptually, it computes a weighted sum of values, where the weights are determined by a query's interaction with corresponding keys. The Transformer's "self-attention" mechanism allows the model to assess the significance of each position in the input sequence relative to a specific position in the output sequence. Notably, these assessments can be computed in parallel for different positions, enhancing computational efficiency. Additionally, the architecture incorporates "multi-head attention" to augment its representational capabilities. Instead of a single attention computation, the model performs multiple calculations, each potentially concentrating on various facets of the input data. Specifically, with reference to the Section 2.5.1 for the mathematical formalization, for each attention "head", distinct linear projections of the initial Q, K, and V are employed. The resultant attention outputs are concatenated and subjected to a linear transformation to yield the final output:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \dots)W_O \tag{4.7}$$

where each head_i represents a distinct attention output and W_O is an optimizable weight matrix [5].

4.9.1 Use-cases of Transformer Models

Transformer models have become instrumental in a myriad of applications, demonstrating their efficacy in reshaping sequence-to-sequence tasks.

In the realm of language translation, the capacity of Transformers to manage long-range dependencies and formulate coherent sequences has resulted in substantial advancements in machine translation. The absence of a need for explicit alignment models, coupled with their proficiency in parallel processing and self-attention, has enabled state-of-the-art performances on renowned benchmarks like WMT (Workshop on Machine Translation) [126]. This offers an improved approach compared to preceding models, such as phrase-based or RNN-based systems, that occasionally overlooked certain linguistic subtleties.

For text summarization, Transformers have proven indispensable. The objective of condensing extensive texts while preserving their essence aligns well with the attention mechanisms of the architecture. Notably, models like BERT and T5 have leveraged this capability to achieve exceptional outcomes in both extractive and abstractive summarization tasks [127, 128].

In the context of content generation, models derived from the Transformer architecture, such as GPT (Generative Pre-trained Transformer), have been noteworthy in their generation of text that is both contextually rich and coherent. Their applicability spans a variety of tasks, from generating narratives to answering open-ended queries [129]. Moreover, the adaptability of Transformers is not confined solely to text-based tasks.

Examples like the Vision Transformer (ViT) elucidate the architecture's competence in image classification, where images are segmented into fixed-size patches and treated analogously to textual tokens [130]. This adaptability across domains highlights the extensive potential of the Transformer architecture.

4.9.2 Properties of Transformers Models

A key aspect of these models is the self-attention mechanism, which dynamically assigns weights to different parts of the input, enabling the model to effectively handle long-range dependencies. This feature is particularly advantageous for complex tasks such as machine translation, where contextual understanding is crucial [5]. Unlike RNNs, which process information sequentially, and CNNs, which may be limited by their receptive field, Transformers excel in capturing interactions between distant elements in the input sequence. In terms of computational efficiency, Transformers are designed for parallel processing, a stark contrast to the sequential computation of RNNs. This architectural decision aligns well with the capabilities of modern hardware like GPUs and TPUs, enhancing both the training and inference processes.

However, Transformers also present certain challenges. One of the primary concerns is their significant memory requirements, particularly in models with multiple attention heads. This characteristic can lead to considerable computational resource demands [131]. Furthermore, the complexity and large parameter count of models such as GPT-3 can increase the risk of overfitting, necessitating careful regularization and model selection strategies [45]. Training deep Transformer networks also introduces stability challenges, with issues such as vanishing or exploding gradients. Addressing these requires techniques like gradient clipping, adaptive learning rate schedules, and layer normalization [132].

Despite these challenges, the transformative impact of Transformer models on deep learning research is profound. Their strengths have led to significant advancements across various domains, while their limitations continue to inspire ongoing research and development efforts aimed at optimizing and refining the architecture.

4.10 Emerging and Specialized Architectures

In the ever-evolving landscape of deep learning, specialized architectures have emerged, each tailored to address specific challenges or to offer a novel approach to neural computation. Among these are Capsule Networks, Siamese Networks, and Neural Turing Machines. Each of these architectures provides unique capabilities, from handling spatial hierarchies in vision tasks to learning similarity in input pairs and even enhancing neural networks with external memory capabilities.

4.10.1 Reservoir Computing and Echo State Networks

Reservoir Computing (RC) offers an alternative framework for training Recurrent Neural Networks (RNNs). Within this paradigm, the Echo State Network (ESN) is a standout model [133]. An ESN consists of a fixed, random recurrent layer termed the "reservoir" and trainable readout weights. The main principle behind ESNs is the "echo state property", which posits that the network's state becomes a fading echo of past inputs and can be represented uniquely by the input history.

The reservoir is characterized by its sparsity and random initialization, creating a diverse range of dynamic responses to incoming signals. Let's denote by $u(t)$ the input at time t, by W_{in} the input weights of dimensions $m \times n$ (where m is the number of reservoir neurons and n is the input dimension), by W the $m \times m$ reservoir weights, and by $x(t)$ the state vector at time t of dimension m. Mathematically, the state update can be represented as:

$$x(t) = \tanh(W_{\text{in}}u(t) + Wx(t-1)) \tag{4.8}$$

One of the main advantages of ESNs is the efficiency in training. Since only the readout weights are adjusted, the model can be trained rapidly using linear regression. The echo state property, formalized as the spectral radius of W being less than 1, ensures that past states fade and are not retained indefinitely. Despite the simplicity of their training process, ESNs have demonstrated proficiency in tasks like time series forecasting and pattern recognition, capturing long-term dependencies in data.

4.10.2 Capsule Networks

Capsule Networks (CapsNet) [134] challenge the status quo of traditional convolutional neural networks (CNNs) by introducing a more sophisticated approach to manage spatial hierarchies within visual data.

A CapsNet is structured around capsules, which are groups of neurons whose outputs, termed as activity vectors, represent different properties of a particular feature and its presence in an image. The length of the activity vector signifies the probability of the feature's presence, while its orientation encodes the properties of the feature, such as pose, texture, and hue. A key mathematical innovation in CapsNets is the "squashing" non-linearity function, which ensures that short vectors get shrunk to almost zero length, and long vectors get shortened to a length slightly below 1. It can be formalized as:

$$v_j = \frac{||s_j||^2}{1 + ||s_j||^2} \frac{s_j}{||s_j||} \tag{4.9}$$

where s_j is the input to the capsule and v_j is the output vector of the capsule [134].

The dynamic routing mechanism is another salient feature of CapsNets. Unlike traditional pooling methods in CNNs, dynamic routing allows a capsule in one layer to send its output to a higher-level capsule based on the level of agreement between their activity vectors. This routing by agreement mechanism is pivotal for recognizing objects in different poses and spatial configurations [135]. Given the robustness of their internal representation, CapsNets have demonstrated resilience to adversarial attacks and can potentially reduce the need for extensive data augmentation [134]. Their capacity to understand spatial hierarchies in visual data, coupled with their generalization abilities, make CapsNets a promising avenue for future research in computer vision.

4.10.3 Siamese Networks

Siamese Networks [69] were architected with the primary purpose of tasks necessitating the determination of similarity or relationships between two analogous inputs. These neural structures have shown potential in one-shot learning scenarios, wherein the network is conditioned to recognize new classes with minimal examples.

Structurally, a Siamese Network is characterized by two identical subnetworks, each taking an individual input but sharing identical parameters. If $f(x)$ denotes the feature extraction function of one of these subnetworks, then the similarity between two inputs x_1 and x_2 can be computed as:

$$S(x_1, x_2) = ||f(x_1) - f(x_2)|| \tag{4.10}$$

Here, $||\ .\ ||$ denotes a norm (often L_2-norm) that measures the distance in the embedded space. The twin networks converge to a point in the training where semantically similar inputs are mapped to nearby points in the embedded space. This feature makes Siamese Networks especially suitable for tasks like face verification and signature verification.

The training of Siamese Networks typically involves a contrastive loss or a triplet margin loss that ensures similar samples are brought closer in the embedded space while dissimilar samples are pushed apart. In the contemporary era, their applications have expanded beyond signature and face verification to tasks such as anomaly detection, image captioning, and even in medical imaging to identify similarities among patterns [70].

4.10.4 Neural Turing Machines

Neural Turing Machines (NTMs) [136] combine neural networks with external memory capabilities, similar to Turing machines. They comprise a controller network (either feedforward or recurrent) and memory matrix, with read and write heads managing memory interactions via attention mechanisms. In NTMs, the memory matrix M of size $N \times M$ (memory locations by vector size) is manipulated through read and write operations. The read operation, for instance, generates a read vector r_t as a weighted sum of memory content, with weights $w_t(i)$ given by the controller. Writing involves modifying memory content at specific locations, guided by erase and add vectors e_t and a_t, also produced by the controller.

These architectures excel in tasks requiring long-term memory, such as complex sequence predictions, due to their ability to store and retrieve information over extended periods. NTMs represent an advancement in neural networks, offering algorithmic problem-solving capabilities alongside adaptive learning.

4.10.5 Neural Ordinary Differential Equations (Neural ODEs)

Neural ODEs [137] mark a significant shift in neural network architecture from discrete layers to continuous-time representations. Central to Neural ODEs is the idea of defining hidden state evolution as a continuous process governed by a differential equation:

$$\frac{dh(t)}{dt} = f(h(t), t; \theta) \tag{4.11}$$

where f is a smooth function, $h(t)$ the hidden state, and θ the function's parameters. Neural ODEs are characterized by reduced memory usage, as they only store the initial state and the governing function, in contrast to traditional networks that store intermediate states. They are particularly adept at handling irregularly sampled data, adjusting their complexity dynamically based on data specifics. This adaptability makes them suitable for tasks with inconsistent temporal resolutions, such as time series forecasting and physical simulations. The continuous framework of Neural ODEs also allows for adaptive computation, tailoring the number of computation steps to the data's intricacies, thereby optimizing computational resources. These properties position Neural ODEs as a promising tool in various applications where a continuous-time approach is beneficial.

4.10.6 Graph Neural Networks (GNNs)

Introduced in the late 2000s, Graph Neural Networks (GNNs) emerged as a response to the need for models that can effectively process graph-structured data, especially as data in non-Euclidean domains, such as social networks and molecular structures, became prevalent [138]. The essential operation underpinning GNNs is the *message-passing* mechanism, reminiscent of message-passing in graphical models. At each layer, nodes in the graph exchange and aggregate information with their neighbors, allowing them to accumulate information from their local surroundings and capture both the structure and feature dependencies present in the graph. Formally, the message-passing procedure can be expressed as:

$$m_v^h = \sum_{u \in \mathcal{N}(v)} M^h(k_v^{h-1}, k_u^{h-1}, e_{vu}) \tag{4.12}$$

$$k_v^{(l)} = U^h(k_v^{h-1}, \mathbf{m}_v^h) \tag{4.13}$$

where k_v^h is the feature vector of node v at layer h, $\mathcal{N}(v)$ denotes the neighboring nodes of v, $\mathbf{e}_{vu}$ represents edge features, and M^h and U^h are the message and update functions, respectively.

Graph Neural Networks (GNNs) draw parallels with convolutional neural networks (CNNs) in their operational mechanics: specifically, CNNs utilize kernels to aggregate information from spatially adjacent regions in images; conversely, GNNs generalize this methodology, gathering data from regions that are topologically proximate within graphs. This methodology is anchored in spectral graph theory, leading to the inception of graph convolutions, a foundational concept for numerous GNN variants [139]. GNNs demonstrate a commendable aptitude for managing data sampled irregularly and can modulate their complexity contingent upon the task's demands. Their proficiency in recognizing long-range dependencies in graph-structured data has been instrumental in various applications, including but not limited to bioinformatics, specifically in protein-interaction network analysis, and in natural language processing for dependency parsing.

Over time, numerous GNN architectures have been introduced, with each refining the inherent message-passing mechanism to address different challenges posed by graph-structured data. Noteworthy among these are several distinct variants: Graph Convolutional Networks (GCNs) [140] occupy a significant position within the GNN research domain. As an introductory variant of GNNs, GCNs employ convolutional techniques to gather and process information, emphasizing the local neighborhood structure of nodes. This straightforward yet robust methodology paved the way for the development of more advanced architectures. Graph Attention Networks (GATs) [141] incorporate the concept of attention mechanisms within the GNN framework. These networks permit nodes to dynamically allocate importance weights to neighboring nodes during the information aggregation phase. Such a mechanism refines the conventional message-passing process, enhancing its adaptability and granularity.

Beyond tasks focusing on individual nodes, Graph Pooling Networks [142] concentrate on assimilating information from the entire graph to produce a uniform graph-level descriptor. This level of abstraction becomes critical, particularly in tasks such as graph classification. The objective is to distill the myriad properties of a graph into a singular representation, facilitating effective graph comparisons. Spectral-based GNNs [143] emphasize capturing global properties of graphs. They operate within the graph's spectral domain, employing tools like Chebyshev Polynomial-based layers to unveil the underlying global structural properties of graphs. Relational GNNs, on the other hand, cater to the intricacies of multi-relational graphs. Acknowledging the possibility of various relationship types within a single graph, these networks employ distinct aggregation and transformation procedures for each relationship type, leading to more accurate and nuanced graph representations.

Owing to their diverse architectural designs and inherent flexibility, GNNs have found applicability in numerous domains. They are employed in fundamental tasks such as node classification, link prediction, and graph regression, as well as in more complex operations like graph generation and spatial-temporal graph forecasting. Their emergence has extended the reach of deep learning into fields where understanding relational and topological intricacies is paramount.

4.11 Comparison of Architectures

Neural network architectures, mirroring the dynamic and evolving landscape of artificial intelligence, exhibit a broad spectrum of designs, each tailored to address distinct challenges and enhance specific functionalities. This variety spans several domains, including image processing, sequential data analysis, natural language processing, and complex graph-structured data handling. An in-depth comparative analysis of these architectures is vital, focusing on their computational efficiency, memory requirements, training dynamics, adaptability to various tasks, and real-world applicability.

4.11.1 Comparative Analysis of Different Architectures

This section delves into the intricate differences between various neural network architectures, examining their unique attributes and practical implications in diverse application contexts.

Computational Complexity CNNs have become a staple in image and video processing due to their ability to effectively capture spatial hierarchies. However, their reliance on convolution operations contributes to significant computational overhead, particularly as network depth and complexity increase, posing challenges in resource-constrained environments [98]. Conversely, RNNs, designed for sequential data processing, exhibit a recursive structure that, while efficient per time step, often requires extensive training periods. This prolonged training can be a hindrance in rapidly evolving scenarios where model agility is paramount [50].

Parameter Efficiency Emerging architectures like Capsule Networks represent a leap towards optimizing parameter usage. They offer a nuanced understanding of spatial hierarchies, potentially achieving higher representational efficiency than conventional deep CNNs. This efficiency is particularly evident in their ability to encode complex spatial relationships with fewer parameters, suggesting a pathway to more compact and computationally efficient models in the future [135].

Flexibility and Generality The advent of Transformer models marks a significant advancement in handling diverse data types. Originally developed for natural language processing tasks, Transformers have demonstrated remarkable adaptability across various domains, including computer vision. The key to their flexibility lies in the self-attention mechanism, which allows the model to dynamically focus on different parts of the input data, adapting its processing strategy based on the task at hand [5].

Memory Usage One of the persistent challenges in deep learning is managing the substantial memory requirements of traditional architectures. Neural Ordinary Differential Equations (Neural ODEs) offer an elegant solution to this issue. By conceptualizing the network's transformations in a continuous-time framework, they circumvent the need for storing large numbers of intermediate states during training, thus reducing memory overhead. This approach is not only mathematically elegant but also highly practical in scenarios with limited memory resources [137].

Scalability Scalability remains a critical consideration, especially in architectures designed for complex tasks. CNNs, for instance, often require deeper and more intricate structures to handle sophisticated visual tasks, leading to an exponential increase in parameters. Graph Neural Networks, while exceptionally proficient at capturing relational data, face their own set of scalability challenges. These challenges become pronounced when dealing with large-scale graphs, where the computational and memory demands can quickly escalate [144].

Interpretability The quest for model interpretability is a key driver in the evolution of neural architectures. Deep learning models, particularly those based on complex architectures, often operate as 'black boxes', obscuring the rationale behind their decisions. The introduction of attention mechanisms, particularly in Transformer models, has shed light on this aspect, offering a window into the model's focus and decision-making process [108].

Training Stability and Convergence Diverse training dynamics across architectures present both opportunities and challenges. For instance, while GNNs excel in capturing complex graph structures, they can exhibit instability during training, particularly in deep networks where the convergence can be erratic. This instability is in stark contrast to more traditional architectures, like CNNs, which typically exhibit more predictable and stable training trajectories [145].

In conclusion, the selection of a neural network architecture for a given task is a multifaceted decision, influenced by the specific characteristics and demands of the task, as well as the inherent strengths and limitations of each architecture. This comprehensive analysis underscores the importance of a balanced approach that considers both theoretical foundations and practical implications when deploying neural network models in real-world scenarios.

4.11.2 Suitability for Specific Applications

Neural network architectures are developed not merely as theoretical constructs but as pragmatic solutions addressing real-world challenges. This section explores how certain architectures are optimally suited for specific application domains.

Image Processing Convolutional Neural Networks (CNNs) [23] have established themselves as the primary choice for image processing tasks. Their convolutional layers effectively capture spatial hierarchies, making them highly suitable for analyzing visual data. However, in scenarios involving adversarial vulnerabilities where minimal perturbations can mislead models, Capsule Networks [135] have emerged as a robust alternative. Their unique design, which encodes spatial relationships within multi-dimensional capsules, offers potential advantages over traditional CNNs in terms of resilience to adversarial attacks.

Sequential and Time-Series Data For sequential and time-series data, which require memory and recurrence capabilities, Recurrent Neural Networks (RNNs) [25], Long Short-Term Memory (LSTM) networks [27], and Gated Recurrent Units (GRUs) [28] have been traditionally favored. However, the Transformer architecture [5] has recently emerged as a significant disruptor in this area, thanks to its parallel processing capabilities and efficient self-attention mechanism.

Graph-Structured Data Graph Neural Networks (GNNs) [138] are ideally suited for handling graph-structured data, common in areas such as social network analysis and molecular modeling. They excel in capturing the relational dynamics of such non-Euclidean data. Additionally, Relational GNNs [144] are specifically designed to deal with the complexities of multi-relational data in knowledge graphs, offering refined representations through differentiated aggregation and transformation processes for various relationship types.

Irregularly Sampled Data Neural Ordinary Differential Equations (Neural ODEs) [137] are particularly effective for datasets with irregular sampling intervals. By modeling data generation in a continuous-time framework, they allow for flexible handling of non-uniform data sequences, a significant advancement over traditional methods that require uniform sampling.

Natural Language Processing In the domain of natural language processing, the Transformer architecture and its variants like BERT [43] and GPT [146] have set new performance benchmarks. Their ability to understand the context and subtleties of language through self-attention mechanisms has led to significant advancements in tasks ranging from text classification to generative language models.

Complex Decision-Making Tasks The fusion of deep learning with reinforcement learning, exemplified by models like AlphaGo [147], has proven effective in complex decision-making tasks. These hybrid models illustrate the power of combining deep learning's pattern recognition capabilities with strategic analysis derived from reinforcement learning.

sectionOpen Problems and Future Directions The exploration of neural network architectures has led to significant progress across various fields. However, the evolution of this technology continues to reveal new challenges and opportunities for advancement.

A critical issue in current neural network research is the balance between model complexity and interpretability. As architectures become more complex, their transparency and understandability diminish, indicating a need for future research to focus on enhancing interpretability without sacrificing performance. Another pressing concern is the computational cost of state-of-the-art models. Future efforts should aim to develop architectures that are resource-efficient yet maintain high levels of performance. Achieving this goal may require innovative architectural designs, improved training methods, and the implementation of sparse representations.

Moreover, current neural networks, while inspired by the human brain, are still rudimentary compared to the intricate mechanisms of human cognition. Integrating advanced cognitive processes such as few-shot learning, cognitive reasoning, and knowledge consolidation into artificial models presents a formidable challenge with potentially rewarding outcomes. Additionally, enhancing the resilience of these networks to adversarial attacks, incomplete data sets, and changing data distributions is critical. Future models should be robust to these challenges while retaining the capacity to generalize effectively across diverse applications.

As neural networks become more integrated into sensitive areas, ethical considerations become increasingly important. Future research should focus on ensuring that outputs from these models are unbiased and equitable, promoting transparency and accountability. Exploring the integration of symbolic reasoning with neural approaches offers another promising research direction. This convergence could combine the pattern recognition capabilities of neural models with the logical reasoning strengths of symbolic AI, potentially bridging a significant gap in current AI technology.

Finally, the rapidly evolving field of neural networks is likely to encounter unanticipated challenges and explore new applications. These developments will require ongoing adaptation and innovative solutions.

In conclusion, while neural network architectures have already transformed many areas of technology and research, there remains a vast landscape of challenges and opportunities to explore. Addressing these issues is key to advancing our understanding of neural networks and unlocking new, groundbreaking applications.

Chapter 5

Applications to Time-Series Forecasting

5.1 Introduction

In the vast landscape of data science, the capacity to interpret and forecast future patterns stands at the forefront of many academic and industrial endeavors. Central to this challenge is the analysis of time-series data. Mathematically, a time-series is defined as a sequence $\{y_t\}$, where y_t denotes an observation recorded at time t, and these observations are indexed in increasing chronological order [148]. There are fundamentally two types of time-series: univariate and multivariate. A univariate time-series consists of singular observations recorded sequentially over time. In this format, one tracks a single variable's progression, such as the daily temperature of a city or the monthly sales of a product. Formally, for a given time t, a univariate time-series records a single observation y_t; contrarily, multivariate time-series extends this concept by recording multiple observations at each time point, resulting in a vector $\mathbf{y}_t = (y_{1,t}, y_{2,t}, \ldots, y_{n,t})$ for each t, where n represents the number of observed variables [149]. These observations might comprise a combination of different but related metrics.

Predicting time-series, whether univariate or multivariate, possesses immense benefits since accurate forecasts can provide pivotal insights, fostering optimized strategies and facilitating improved decision-making processes across multiple domains. Due to its relevance, the domain of time-series forecasting, rich in its historical evolution, has witnessed an array of methodologies. Initial efforts were grounded in statistical models – often linear – that were adept at capturing rudimentary temporal patterns in the data [150]. The advent of Machine Learning (ML) and Deep Learning (DL) revolutionized the landscape, introducing methods that could discern complex non-linear relationships, especially prominent in multivariate time-series [3]. However, as the richness and complexity of data have escalated, challenges have concurrently intensified. Multivariate time-series, especially, confront high dimensionality, intricate inter-series dependencies, noise, and real-world unpredictabilities and conventional models, though invaluable, occasionally fall short against these intricacies. To counter these challenges, recent years have seen a pronounced shift towards ensemble and hybrid methods [151]: by amalgamating the strengths of multiple forecasting techniques, these models target comprehensive, robust, and precise predictions. The integration of neural networks into this mix, particularly, brings depth and adaptability, demonstrating promising results for the nuanced challenges of multivariate series prediction.

5.2 Historical Evolution of Forecasting Methods

Forecasting has historically been an essential aspect of analytical efforts, with its main objective being the anticipation of future outcomes based on past data. The shift from empirically-driven forecasting, which largely depended on observation, to systematic methods marked a significant progression in the field. As these methodologies matured, probabilistic techniques were developed, thereby enhancing predictions by considering inherent uncertainties.

By the mid-20th century, the discipline of time-series forecasting saw marked advancements, primarily through the introduction of models rooted in rigorous mathematical foundations. A seminal contribution during this era was the Exponential Smoothing method proposed by Holt [152]. Originating from the basic premise that recent observations are often the most indicative predictors for future events, this method considered historical data but applied differential weights to them. Such an approach facilitated more adaptable predictions, emphasizing recent data trends. Holt's model is mathematically expressed as:

$$\hat{y}_{t+1} = \alpha y_t + (1 - \alpha)\hat{y}_t. \tag{5.1}$$

Building on the foundation established by Holt, subsequent advancements in time-series forecasting were directed towards accommodating the intricacies of diverse datasets. For instance, the Holt-Winters Exponential Smoothing method incorporated mechanisms specifically to address seasonality, rendering it more effective for datasets exhibiting cyclical patterns [153]. A notable evolution during this period was the introduction of the ARIMA model. By integrating autoregression (AR), differencing (I), and moving average (MA) components, ARIMA showcased a broader applicability, efficiently modeling various time-series characteristics such as underlying trends, seasonal patterns, and autocorrelation structures [154]. The ARIMA(p, d, q) model is described by:

$$(1 - \sum_{i=1}^{p} \phi_i L^i)(1 - L)^d y_t = (1 + \sum_{i=1}^{q} \theta_i L^i)\varepsilon_t, \tag{5.2}$$

where y_t denotes the value at time t, L is the lag operator, d specifies the order of differencing, ϕ_i and θ_i are parameters associated with the AR and MA segments, respectively, and ε_t is white noise. To address seasonal patterns, the Seasonal ARIMA (SARIMA) model was devised. It brings in seasonal differencing as well as seasonal autoregressive and moving average terms. This model is represented as SARIMA$(p, d, q)(P, D, Q)_s$, where P, D, Q are the seasonal orders, and s indicates the number of periods within a season. Building upon SARIMA, the SARIMAX model incorporates exogenous variables (X) – supplementary time-series data that can influence predictions for the main series. This model's ability to account for external factors, which can be expressed as time-series data, emphasizes its relevance, especially when external variables impact the main time-series [155].

As time-series forecasting methodologies evolved, the challenges presented by non-linearities, missing data, and abrupt structural changes became evident. In response, State Space Models, complemented by Kalman Filtering, were introduced to enhance forecasting under such complexities [156]. Transfer Function Models further allowed analysts to gauge the impact of exogenous variables on primary time-series. Additionally, the Seasonal Decomposition of time-series (STL) method segmented time-series into its core elements: trend, seasonality, and residuals, enabling more refined forecasting approaches [157].

In the latter half of the 20th century, the field of forecasting saw marked advancements, particularly due to the confluence of enhanced statistical methodologies and the rise of computational capabilities. Fourier and spectral analyses became key techniques, focusing on the frequency aspects of time-series data [158]. By the onset of the 21st century, several advanced models augmented traditional statistical approaches. For instance, ARIMA and Exponential Smoothing were complemented by methods like TBATS and the multilinear orthogonal autoregressive (MOAR) model, effectively addressing varied forecasting challenges from oceanic temperatures to financial metrics [159]. Additionally, the Block Hankel Tensor ARIMA (BHT-ARIMA) and Prophet, this latter developed by Facebook's research team, emerged as noteworthy contributions [160, 161]. Parallel to these advancements, machine learning (ML) introduced a new dimension to forecasting: ML's strength in discerning non-linear patterns, absent the constraints of rigid model assumptions, became evident. Support Vector Machines (SVM), originally for classification, were adapted to forecasting through Support Vector Regression (SVR), offering insights especially in the realm of financial forecasting due to its ability to handle high-dimensional data and non-linearities [162]. Ensemble techniques like Random Forests provided another layer of sophistication: by combining insights from multiple decision trees, they not only increased prediction accuracy but also addressed challenges like overfitting, making them particularly valuable in domains such as energy forecasting where multiple factors influence predictions [163].

Within this framework, the recent rise of Neural Networks has had a profound impact on the forecasting landscape. Their ability to approximate a wide range of continuous functions highlights their adaptability in various forecasting challenges [35]. Particularly for time-series data, which often embodies complex temporal patterns, architectures like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) have become indispensable. Noteworthy developments in this arena include Amazon's RNN-based DeepAR [164] and the N-Beats neural network model, which surpassed many contemporaries in the M4 competition [165]. Hybrid models, which combine traditional forecasting techniques with neural architectures, have also gained traction: for instance, hybrid ARIMA-Neural Network models are discussed in [166] and [167], while the integration of state space models with deep learning is explored in [168]. Matrix factorization, employed for dimensionality reduction before prediction, is discussed in works such as [169], [170], and [171]. Further, hybrid models like the one presented in [172] employ a sequential convolutional-recurrent approach for high-dimensional series, while [173] presents an award-winning model from the M4 Competition that merges Exponential Smoothing with RNNs. More recently, the potential of Graph Neural Networks in predicting time-series has been explored, as they excel in recognizing inter-variable connections [174]. Moreover, the emergence of Transformer-based architectures has marked another significant milestone; innovations such as the Temporal Fusion Transformer (TFT), which focus on multi-step prediction interpretability [175], or the Autoformer uses auto-correlation techniques to address long-term forecasting challenges inherent to standard Transformers [176] have further demonstrated the flexibility of this architecture.

5.3 Ensemble Methods

As the historical panorama of forecasting evolved, researchers recognized both the strengths and limitations intrinsic to each method. Traditional statistical methods, though adept at capturing linear relationships and patterns, often struggled with non-linear, complex datasets, especially those characterized by high dimensionality and intricate interdependencies. The rise of Machine Learning and Deep Learning techniques sought to address these challenges, but even they had their own limitations, such as susceptibility to overfitting or sensitivity to noisy data.

Ensemble methodologies, having gained traction in the realm of machine learning, are grounded in the principle that aggregating outputs from multiple models can yield superior results than relying on any single one: recognizing that no individual method is universally optimal, the idea of combining predictions emerged as a strategy to harness collective strengths while mitigating individual weaknesses. As a result, ensemble techniques have profoundly influenced predictive modeling paradigms, being the subject of extensive research for their ability to enhance generalization, reduce overfitting, and improve predictive accuracy [177].

5.3.1 Fundamental Pillars of Ensemble Approaches

Ensemble methods rest on a triad of guiding principles, and each of these principle addresses a distinct challenge, offering a comprehensive solution that a singular model may not achieve. The first principle, *Diversity in Decision Making*, emphasizes the advantages of integrating multiple models. When decisions from varied models are aggregated, ensemble methods can more thoroughly analyze intricate datasets, potentially offering more robust conclusions than a singular model [178]. A quantitative basis for this concept emerges when contemplating model errors: when two models produce errors that aren't perfectly correlated, their aggregated output will, in expectation, have diminished error. The second principle, *Overcome Model Limitations*, articulates the inherent constraints of individual models. Supported by the idea behind the “No Free Lunch theorem” [179], it’s acknowledged that a single model is unlikely to be the best fit for every scenario. Ensemble approaches, by incorporating diverse models, aim to capitalize on their combined strengths and minimize the limitations intrinsic to each model. Lastly, the principle of *Uncertainty Quantification* highlights the ability of ensemble methods, especially within Bayesian settings, to provide a comprehensive view of outcomes; instead of merely offering point estimates, they present a distribution over possible results, granting a detailed perspective on the reliability of predictions [180]. Such granularity becomes paramount in scenarios where gauging the uncertainty associated with a forecast is as critical as the forecast itself.

5.3.2 Applications of Ensemble Methods in Time-Series Forecasting

As said, recent advancements in time-series forecasting are characterized by the increasing utilization of ensemble methodologies. Typically, these methodologies employ linear combinations, optimizing weights to achieve enhanced predictive precision. The MSEF technique [181] exemplifies this, merging multiple models following prior time-series decomposition tools like STL. The FYTS approach [182] amalgamates forecasts based on model performance metrics, determining weights for a composite prediction. A specialized application for the tourism sector [183] decomposes the time-series into primary components, models each via ensembles, and subsequently aggregates predictions. Distinguishingly, some methods prioritize dynamic weight adjustment; a case in point is an ensemble for wind energy forecasts [184], which iteratively recalibrates model weights based on recent performance, ensuring consistent efficiency by favoring recently effective models.

Beyond the realm of linear combinations, in the contemporary literature on ensemble methodologies, there is a marked exploration of approaches that determine the specific contributions of individual models. EASTPAS, for example, presents an integration of multiple forecasts using statistical techniques like bagging, boosting, and stacking [185]. Also, the role of genetic algorithms in ensemble forecasting has been a focal point in [186], where the approach consists in combining multiple classifiers optimizing their weights using this type of optimizers. This utility of genetic algorithms in ensemble methods has been further explored in [187, 188], where they are successfully applied to domain-specific forecasting tasks such as electricity demand and financial predictions, highlighting the algorithms' versatility in addressing a spectrum of forecasting complexities.

Concurrently, methods rooted in computational intelligence are emerging as significant contributors: one such approach involves training decomposed segments of wind power time-series using dedicated neural networks [189]; similarly, an ensemble technique has been proposed that amalgamates forecasts from diverse machine learning models to produce a holistic prediction [190]. The proliferation of deep learning, especially the refinement of Recurrent Neural Networks (RNNs), has opened avenues for bolstering ensemble frameworks. A notable methodology [191] fuses forecasts from varied RNN configurations using a stacking algorithm, leveraging the strengths of distinct RNN architectures. Hybrid ensemble techniques, such as those discussed in [192, 193], underscore the benefits of integrating disparate model categories, proving invaluable in tasks like wind speed prediction and ultra-short-term power demand forecasting. Beyond traditional ensemble approaches, a meta-learning strategy presented in [194] employs neural networks to discern connections between diverse data features and the corresponding optimal forecasting models. This not only streamlines the integration of predictions but also elucidates the relationship between data characteristics and model suitability, providing a refined model selection mechanism.

Despite the success of complex methodologies in the evolving landscape of ensemble forecasting, the value of fundamental techniques should not be understated. The comprehensive review in [195] underscores the efficacy of simple algebraic combinations, such as uniform aggregations or median-based selections. Impressively, these elementary methods frequently surpassed their more complex counterparts in performance. This perspective is further cemented by results from the M4 forecasting competition, which attested to the significance of these fundamental ensemble approaches [196]. Such outcomes serve as a reminder: as interest in advanced ensemble strategies increases, it is essential to maintain focus on the consistent effectiveness of classical techniques. Researchers should aim for a balance between the complexity of new methods and the straightforward effectiveness of traditional ensemble strategies in time-series forecasting, ensuring reliable outcomes

5.4 The Idea of a Neural Combiner

Following the established trajectories of ensemble strategies, it's evident that while combining predictions from simpler models has consistently showcased robust forecasting methodologies, challenges persist. Specifically, ensembles obtained through weighted combinations can inadvertently underestimate the prowess of an individual model to capture certain intrinsic dynamics of the target time-series. This underrepresentation becomes even more pronounced when we delve into multi-input multi-channel networks. Though these networks have traditionally been geared towards ensembling neural sub-models, they often miss out on the inherent diversity and adaptability brought forth by models of varied natures. In response to the intricate challenges present in time-series forecasting, the ensemble method introduced in the following offers a more advanced approach: instead of just combining forecasts, this method tries to understand the importance and details of initial predictions provided by other models. The method uses a hybrid neural network that combines convolutional layers, which detect spatial patterns, with recurrent layers that recognize temporal dependences. In particular, the recurrent branch of the proposed network assimilates the original data, features representative of seasonal patterns and predictions: this facilitates the delin-

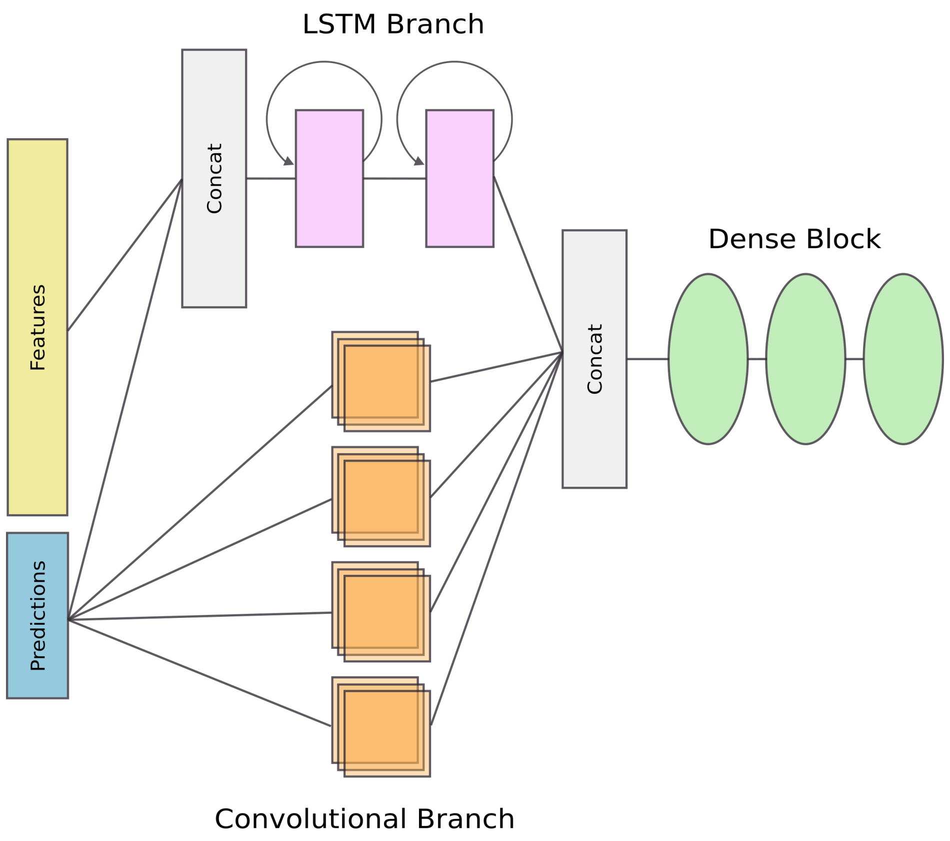


Figure 5.1: Sketched representation of the Hybrid Neural Network designed to enseble forecasts for univariate or multivariate time-series.

eation of temporal relationships between data points, offering a detailed understanding of their chronological interdependence. Such a representation of temporal dependencies is critical to ensure the integrity and reliability of the forecasting process. In contrast, the convolutional branch focuses on the spatial aspects of data relationships: it conducts a detailed analysis of horizontal patterns observed within the preliminary prediction outputs. This branch provides an in-depth spatial interpretation of data relationships, emphasizing the importance of understanding patterns beyond just temporal sequences. This spatial analysis is vital as it can reveal underlying correlations and intricate patterns that are essential for refining forecasting accuracy.

In the architecture, integration of insights from both convolutional and recurrent branches is realized within the dense layers. Serving as a junction, these layers amalgamate both the temporal information sourced from the recurrent branch and spatial patterns discerned by the convolutional branch, resulting in a consolidated prediction output. By incorporating initial forecasts as auxiliary regressors, the model's strength is improved, making it better suited to handle complex time-series data. The key idea behind this ensemble method is that different models interpret time-series data based on its unique characteristics and how it changes over time. The ensemble doesn't overly rely on one model's predictions; instead, it values the different insights that various models offer at different times. The hybrid neural network combines the knowledge from multiple models, allowing it to adjust easily to the changing nature of time-series data.

5.5 Case Study: Predictive Analytics for Smart Parking

Smart Cities exemplify the synergy between technology, governance, infrastructure, and citizen engagement powered by the Internet of Things (IoT) technologies. Leveraging Artificial Intelligence (AI) and Machine Learning (ML), these cities optimize urban processes, resulting in enhanced emergency response, real-time traffic management, safety protocols, and other innovations such as AI-driven public lighting [197].
A pivotal aspect of Smart Cities is Smart Mobility, emphasizing the efficient movement within urban environments. Within this ambit, smart parking emerges as a crucial component [198]. Smart parking systems utilize the prowess of IoT and AI, integrating sensors, cameras, and interconnected devices to produce real-time parking data. Such systems proffer advantages like:

- Delivering real-time parking information to drivers via mobile platforms.
- Swiftly identifying parking violations for law enforcement.
- Recommending alternative transportation during peak parking periods.

These systems aim to mitigate parking-related challenges, leading to reduced traffic congestion, pollution, and noise.

In this context, the synergy between AI and IoT emerges as a cornerstone in reshaping urban infrastructures, underscoring the growing demand for robust predictive analytics tailored to Smart City applications. Specifically, within the realm of smart parking, ML and DL have found applications across diverse settings [199–202]. Forecasting, indispensable for informed decision-making, has been diligently pursued in parking occupancy challenges, employing a spectrum of both statistical and machine learning approaches [203, 204].

With these premises, an initial study involving the Hybrid Neural *Combiner* detailed in Section 5.4 is developed: the primary aim is to establish a robust predictive framework capable of accurately forecasting daily parking occupancy with 1-hour and 3-hour horizons, using three months of historical IoT sensor data. To anchor this study in practicality, time-series data from the K-City project in the Campania Region, Italy, is utilized. This project is distinguished by its comprehensive network of over a thousand IoT sensors tailored for real-time parking space monitoring. These sensors, consistently transmit occupancy data to a centralized platform, setting the stage for subsequent AI-driven analysis.

5.5.1 The Project

The K-City project integrates Internet of Things (IoT) sensors into urban parking systems, addressing the pressing need for efficient parking space identification in populated urban areas. At its core, the project involves deploying sensors, known as “buoys”, in parking zones. These buoys, connected to a centralized IT platform via the Long Range Wide Area Network (LoRaWAN) protocol, provide real-time data on parking availability and payment statuses to users’ mobile devices. Positioned strategically over fee-required spaces, they not only monitor occupancy but also verify payment compliance. The overarching aim of the K-City project extends beyond parking management; by providing real-time data, it seeks to reduce the time spent by residents searching for parking, fostering both convenience and a more sustainable urban transit experience.

5.5.2 Data Preprocessing

The considered datasets contains time-series data representing the occupancy rates of specific parking zones within the city of Naples, Italy. The data encompasses six distinct parking zones located at: *piazza Vanvitelli*, *via Filangieri*, *via Gasparri*, *via Patturelli*, *via Turati*, and *via Unità Italiana*. Each data point in the time-series represents an occupancy rate, defined as a real number within the interval [0, 1]: an occupancy rate of 0 indicates that the parking zone is completely vacant, while a rate of 1 indicates full occupancy. Upon an initial analysis of the data, clear hourly patterns in the occupancy

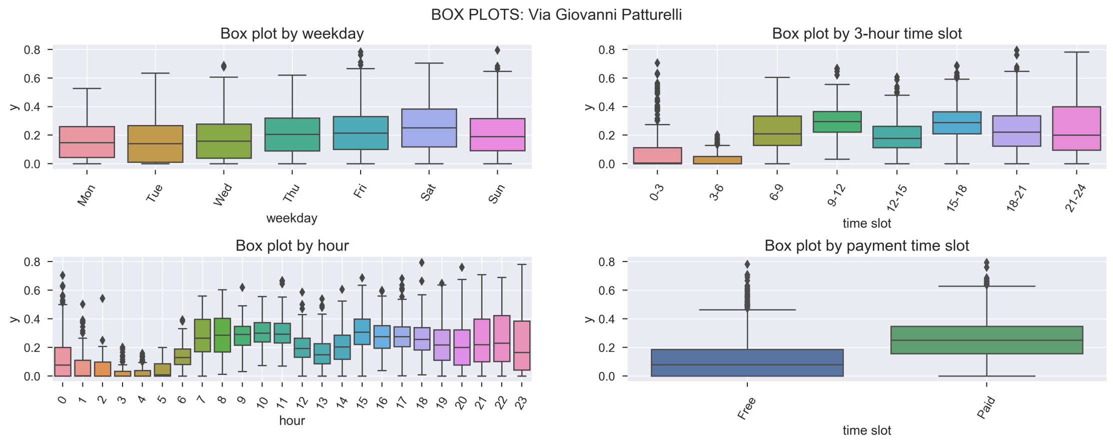


Figure 5.2: Boxplots displaying aggregated data categorized by weekday (upper-left), 3-hour time slot (upper right), specific hour (lower-left), and payment time slot (lower-right) for the occupancy rate time-series of *via Patturelli*. The data reveals a consistent hourly pattern, notably the heightened occupancy rates during daytime hours, present across the time-series under consideration.

rates become evident: this cyclical behavior is particularly observable in the upper right and lower left boxplots of Figure 5.2, which highlight the occupancy trends for *via Patturelli*. Notably, the higher variability in occupancy rates during nighttime hours might be attributed to longer parking durations, predominantly seen on weekends. A further observation reveals that during standard business hours, during which payment is required, tend to maintain a higher average occupancy. The designation *Payment time slot* pertains to predetermined hours wherein parking payment is obligatory. For example, from Monday to Thursday, the mandatory payment period spans from 8 a.m. to 9 p.m. During other days, this interval extends from 8 a.m. to midnight. Figure 5.3, again referencing *via Patturelli* as a prototypical instance, demonstrates

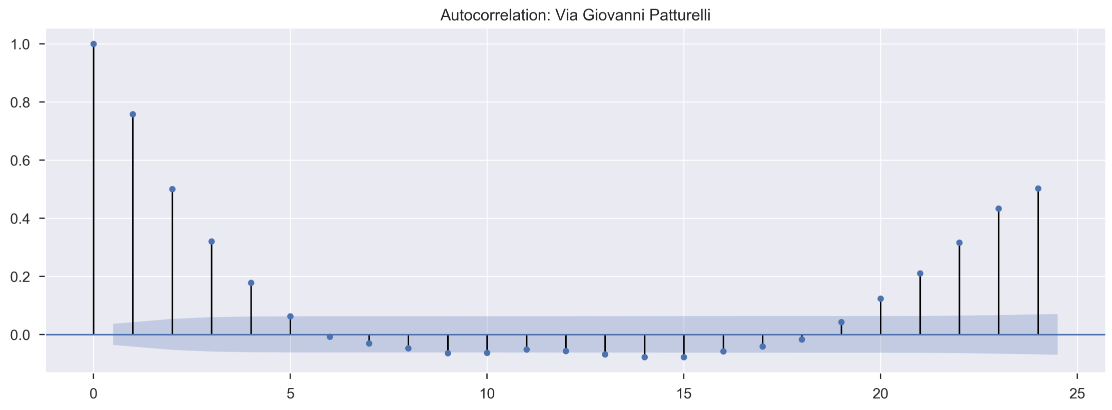


Figure 5.3: Autocorrelation graph of the occupancy rate time-series for *via Patturelli*. A marked correlation is evident in both short-term intervals (1 to 3) and long-term (23 to 24) hourly periodicity. Analogous patterns are identified across all examined time-series.

through autocorrelation analysis a pronounced correlation of each data entry with the preceding 24-hour information. Given this observation, lagged variables, spanning from *lag1* to *lag24*, have been integrated as regressive features into the model. An alternative configuration utilizing lags from *lag3* to *lag24* is also assessed. Additional variables in the dataset have been also considered to capture its natural patterns: these variables represent different time periods such as 3-hour intervals, payment times, weekdays, weekends, and holidays. Given these additions, each time sequence value is represented by:

$$y_t \to \mathcal{H}(t, \Theta), \tag{5.3}$$

where y_t is the target value. The function $\mathcal{H}(t, \cdot)$ can also be written as $\mathcal{H}([y_{t-s}, \ldots, y_{t-24}]^T, \cdot)$, where the delay s is either 1 or 3, and denotes the dependence on previous values in the series. The matrix Θ includes the aforementioned periodicity variables.

5.5.3 Methodology

For the preliminary analysis and preprocessing stages intended to generate individual forecasts, which are subsequently integrated through the Hybrid Neural Network, we have selected algorithms that offer robust predictive capabilities and can capture highly non-linear relationships. Specifically, tree-based regressors have been chosen. To maintain variability among predictors, these methods are complemented by two linear regressors, since even with their inherent assumption of simple relationships among features, they have shown commendable performance in forecasting contexts. Thus, initial predictions required for the combination phase are generated by a Random Forest [205] regressor, a Gradient Boosted Decision Trees [206] regressor, a Lasso [207] and a Ridge [208] Linear regressors. According to the pipeline sketched in Figure 5.4, data

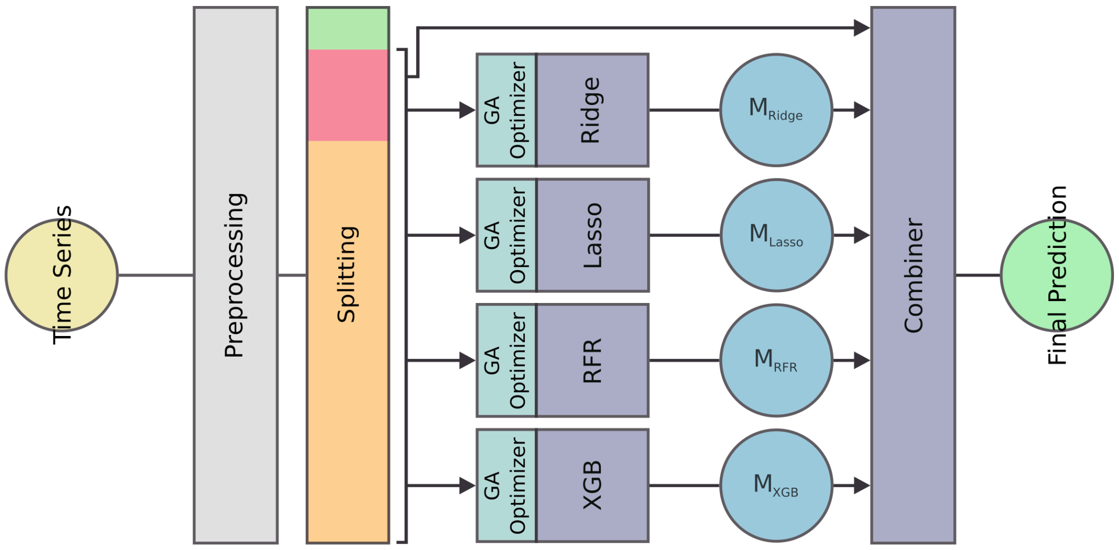


Figure 5.4: Schematic representation of the proposed framework: time-series is preprocessed, then divided in *train*/*validation*/*test* sets. Predictors of the first layer are trained over the *train set* (in yellow), their hyperparameters are optimized through *genetic algorithms* and forecasts produced on the validation set (in red) are fed into the Hybrid Neural Network together with lagged and periodicity features. The *Neural Combiner*, is then trained in recognising relations between time-series characteristics and forecasts produced by predictive models. The final forecast on the test set (in green) is then obtained combining predictions provided by *Lasso*, *Ridge*, *RFR* and *XGB* models with lagged and periodicity features relative to the same set.

undergo preprocessing to yield a dataset comprising lagged variables and periodicities.

Subsequently, this dataset is temporally partitioned into three sets: *Train*, *Validation* and *Test* sets.

$$\begin{aligned}\mathcal{D}_{train} &\rightarrow [y_0, ..., y_{N-(N_{val}+N_{test})}]^T \\ \mathcal{D}_{val} &\rightarrow [y_{N-(N_{val}+N_{test})+1}, ..., y_{N-N_{test}}]^T \\ \mathcal{D}_{test} &\rightarrow [y_{N-N_{test}+1}, ..., y_N]^T\end{aligned} \tag{5.4}$$

Here, N_{val} and N_{test} respectively represent the sizes of the *Validation* and *Test* sets. As suggested by their names, these partitions facilitate the training and validation of the framework: each predictive algorithm in the primary prediction layer is trained using the *Train* set and subsequently generates forecasts for the *Validation* set. Since better the predictions of these algorithms, more robust the final prediction, a Genetic Algorithm has been employed to optimize hyperparameter configurations for each predictor in the context of the specific time-series. These forecasts are then input into the Hybrid Neural Network alongside the time-series' regressive and periodic features: in this way the network discerns the relationships between forecasts and time-series characteristics during its training. Upon training the Hybrid Network, the final k-hours-ahead forecast (where $k = 1, 3$) is determined by feeding the *Combiner* with the single predictive model's forecasts on the *Test* set, in conjunction with lagged and seasonality variables pertinent to the period under consideration.

Genetic Hyperparameter Optimization

This class of algorithms was introduced in the context of a general theory of automation [209], and later expanded for machine learning applications [210]. The underlying concept is inspired by Charles Darwin's theory of evolution: here, species evolution proceeds through mechanisms such as cross-breeding, mutation, and selection, ensuring the propagation of the most advantageous genes for species survival, as specifically discussed in Section 6.3.2.

It is crucial to underline that, in the context being considered, the number of hyperparameter combinations can be substantial, particularly for tree-based methods, where it may amount to millions of potential configurations. Employing a traditional grid search algorithm in this scenario would incur prohibitive computational costs. In contrast, the genetic algorithm provides a pathway to identifying a near-optimal solution with significantly fewer model evaluations, thereby justifying its adoption for this purpose.

5.5.4 Results

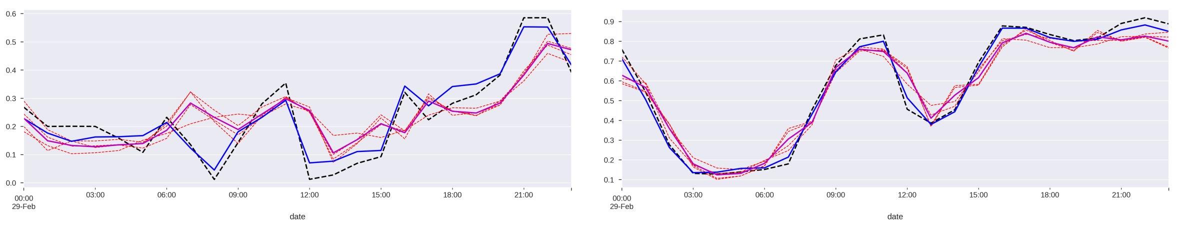
The dataset considered for evaluating the approach is constituted by univariate time-series concerning six parking areas, and covers a period of 4 months – from November 2019 to February 2020, inclusive. Since the standard k-fold Cross-Validation technique is unsuitable for time-series data as it fails to maintain temporal patterns, a modified version of Nested Cross Validation, as discussed in [211] and [212] has been used. To achieve a more accurate model prediction error estimation, we established five consecutive *Train*/*Validation*/*Test* splits: this approach ensures that the training set at step $k+1$ encompasses the test set from step k.

The six time-series under consideration comprise approximately 2,900 hourly samples pertaining to street occupancy rates. Adhering to the aforementioned testing procedure, a validation size of $N_{\text{val}} = 96$ samples and a test size of $N_{\text{test}} = 24$ have been set. Data from 01/Nov/2019 serve as the training foundation, while forecasting capabilities are assessed over a span of five days, targeting the closing days of February 2020. Results from Table 5.1 demonstrate the effectiveness of the framework in utilizing optimal first-layer predictions to improve the final forecast, which consistently yields better results compared to individual predictors and the average strategy. Notably, the R^2 coefficient values show that the *Combiner* can accurately reproduce the predicted time-series pattern, even when first-layer models underperform, highlighting the robustness of our approach. Figure 5.5a and Figure 5.5b present the final stage of the Nested Cross Validation procedure for the *via Patturelli* and *via Vanvitelli*, respectively. Especially

	Vanvitelli plaza			**Filangieri road**			**Gasparri road**		
Predictor	R^2	MAE	RMSE	R^2	MAE	RMSE	R^2	MAE	RMSE
Lasso	0.88980	0.06109	0.07823	0.90861	0.03782	0.05284	0.33051	0.12303	0.15401
Ridge	0.88879	0.06224	0.07953	0.90441	0.03857	0.05393	0.26685	0.12306	0.15177
XGB	0.89274	0.05833	0.07536	0.88741	0.04086	0.05605	0.25630	0.12335	0.15343
RF	0.83389	0.06716	0.08499	0.848488	0.04991	0.06357	0.05041	0.12775	0.15730
MEAN	0.89529	0.05767	0.07404	0.90119	0.03938	0.05328	0.25307	0.12211	0.15043
COMBINER	**0.97183**	**0.02891**	**0.03848**	**0.97355**	**0.02216**	**0.02792**	**0.89732**	**0.04713**	**0.06254**
	Patturelli road			**Turati road**			**Unità Italiana road**		
Predictor	R^2	MAE	RMSE	R^2	MAE	RMSE	R^2	MAE	RMSE
Lasso	0.17602	0.07649	0.1028	0.62019	0.05896	0.07986	0.02363	0.03895	0.05379
Ridge	0.08455	0.08111	0.10618	0.65397	0.06033	0.07846	0.23473	0.04072	0.05472
XGB	0.19167	0.07977	0.10424	0.70532	0.05494	0.07365	0.11499	0.03784	0.05265
RF	-0.17446	0.07816	0.10175	0.57331	0.06386	0.08049	0.22326	0.03467	0.04799
MEAN	0.10726	0.07669	0.10047	0.66442	0.05595	0.07470	0.17268	0.03651	0.05080
COMBINER	**0.83980**	**0.03562**	**0.04583**	**0.84134**	**0.03819**	**0.05068**	0.**72528**	**0.02189**	**0.02900**

Table 5.1: Comparison of Fivefold Nested Cross Validation Results for 1-Hour-Ahead Forecasts between Single Predictors (Lasso and Ridge, Boosted Trees, and Random Forest regressors), Equally Weighed Linear Combination and the Proposed Framework

in the first case the hybrid neural network's prediction effectively captures the original time-series' temporal pattern. It does so by leveraging both the first-layer forecast data and the inherent information from the time-series, augmented with periodicity features, resulting in a more accurate representation of occupancy rate fluctuations. These initial results indicate the Hybrid Neural Network's effectiveness in producing



(a) (b)

Figure 5.5: Comparision between Lasso, Ridge, Boosted Trees, and Random Forest predictions (red dotted lines), equally weighted combination (in magenta) and proposed *Combiner* (in blue). Results show for 1-hour-ahead prediction over a period of 24 hours for the TS of occupancy rate regarding *via Patturelli* (a) and *via Vanvitelli* (b). Best results are highlighted in bold.

accurate 1-hour-ahead predictions, surpassing both individual forecasts and their linear combinations. The results for 3-hour-ahead forecasts are presented in Table 5.2. As expected, the longer forecasting horizon decreases the predictive accuracy: this is

	Vanvitelli plaza			**Filangieri road**			**Gasparri road**		
Predictor	R^2	MAE	RMSE	R^2	MAE	RMSE	R^2	MAE	RMSE
Lasso	0.67569	0.08932	0.11161	**0.75485**	0.06751	0.08549	-0.38027	0.14824	0.17881
Ridge	0.69919	0.08692	0.10938	0.74527	0.06699	0.08656	-0.44685	0.14937	0.18071
XGB	0.75290	0.07905	0.09704	0.59845	0.07725	0.09686	-0.30723	0.14705	0.17960
RF	0.70399	0.07977	0.10012	0.69131	0.07088	0.08617	-0.40677	0.14926	0.17869
MEAN	0.73793	0.08124	0.09968	0.72612	0.06817	0.08617	-0.36131	0.14656	0.17632
COMBINER	**0.83001**	**0.06740**	**0.08473**	0.75319	**0.06503**	**0.079030**	**0.29847**	**0.12435**	**0.15098**
	Patturelli road			**Turati road**			**Unità Italiana road**		
Predictor	R^2	MAE	RMSE	R^2	MAE	RMSE	R^2	MAE	RMSE
Lasso	-0.68779	0.09266	0.11833	-0.01404	0.08342	0.10162	-0.10789	0.04432	0.05638
Ridge	-0.70927	0.09729	0.12274	0.11699	0.07933	0.09874	0.03287	0.04759	0.05843
XGB	-1.24476	0.10495	0.12893	0.30363	0.07730	0.09435	-0.12409	0.04009	0.05196
RF	-1.15450	0.09813	0.12205	0.34883	0.07477	0.09410	0.14002	**0.03874**	0.04975
MEAN	-0.90428	0.09643	0.12043	0.23236	0.07703	0.09433	0.03354	0.04047	0.05245
COMBINER	**0.00211**	**0.06936**	**0.08890**	**0.53018**	**0.06648**	**0.08526**	**0.17732**	0.03923	**0.04949**

Table 5.2: Comparison of five-fold Nested Cross Validation Results for 3-Hour-Ahead Forecasts between Single Predictors (Lasso and Ridge, Boosted Trees, and Random Forest regressors), Equally Weighed Linear Combination and the Proposed Framework

observed in the heightened errors of single-model predictions and the reduced effectiveness of the equally weighted linear combinations; while the hybrid neural network still demonstrates superiority in 3-hour-ahead forecasts, its advantage is less pronounced compared to the 1-hour forecasts, possibly due to its primary design geared towards shorter-term predictions. Nevertheless, the proposed approach frequently outperforms both the individual predictors and the average method in various performance metrics: except for *via Filangieri*, the *Combiner* consistently enhances the R^2 score, illustrating its capacity to more accurately reflect the original time-series' patterns. These preliminary observations suggested potential areas for refinement based on the forecasting timeframe.

In summary, the designed prediction model for parking space occupancy effectively constitute an enhanced ensemble techniques based on Deep Learning methods: tested on a real-world IoT dataset about parking occupancy and compared to common methods, it significantly improves prediction accuracy, making it a promising forecasting method.

5.6 Case Study: Forecasting of Medical Bookings through Multi-Source Time-Series Fusion

The field of healthcare industry which, being of vital importance, can significantly benefit from predictive analytics: these tools can serve to improve healthcare delivery by predicting potential events and enabling patient-specific treatments. Moreover, they play an essential role in ongoing care, tracking diseases and preventing chronic illnesses. The effective extraction and interpretation of patient data leads to improved service quality in healthcare facilities, ensuring accurate services for patients and efficient resource allocation. Artificial Intelligence-based approaches are instrumental in developing optimization tools specifically designed for resource planning; these tools assist stakeholders in setting up reservation systems or improving contact center queue management [213]. Predictive analytics also plays a role in predicting patient no-shows [214], forecasting scheduled hospital admissions [215], and anticipating hospitalizations due to chronic diseases – actions which optimize medical facility schedules and reduce operational expenses [216, 217].
Given this context, there is a pressing need for methods that offer superior predictive accuracy, especially in the domain discussed here, where in particular time-series forecasting becomes vital for enhancing scheduling systems and managing staff efficiently.

5.6.1 Data Fusion in Healthcare

The notion of amalgamating data from disparate sources to construct a comprehensive mathematical model has roots in literature as far back as the 1960s. Over the years, this concept has been employed across various research domains, from pattern recognition [218] to metrology [219]. Presently, with the vast interconnectivity of sensors and the capability to store extensive data sets, data fusion is garnering renewed interest, particularly in tandem with Deep Learning methodologies. Within the realm of time-series forecasting, data fusion often leads to enhanced models: incorporating information from diverse yet related sources facilitates a multi-faceted understanding of a given problem. Such holistic approaches often yield more dependable and robust forecasts [220–222] compared to conventional univariate and multivariate models.

Concerning healthcare contexts, strategies for integrating both clinical and non-clinical data to foresee health outcomes are prevalent in academic discussions, often intertwined with Deep Learning techniques at the predictive stage. Works such as [223, 224] introduce methodologies to enhance the recognition of patients' activities, employing a fusion of Kernel PCA and RNN-based sequence modelling, as well as swarm search feature selection for machine learning frameworks on merged data. The remote health monitoring strategies, utilizing wearable and environmental sensors, discussed in [225] apply data fusion via a weighting algorithm. In [226], structured and unstructured EHR data are processed through CNN-based and LSTM-based neural networks, respectively, with subsequent patient vector representations employed for classification. [227] merges sensor and EHR data via Information Gain and Conditional Probability techniques, followed by training a Deep Learning Classifier for heart disease prediction. [228] integrates bio-signals and medical imagery, adopting a group Lasso regularization technique for feature selection, aiming to discern the temporal progression of sparse data and forecast chronic disease advancement. Lastly, works such as [229–231] merge EHR data and medical imagery (MRI and blood cell images) using CNN-based models to extract pivotal features, subsequently predicting disease progression.

Within the scope under consideration, forecasting models predominantly address the prediction of service demand. For instance, [232] leverages school zones and census tracts data to shape the forecast of vaccine delinquency, framing the prediction of mobile clinic demand as non-convex optimization challenges. These models also forecast the spread of diseases: in [233, 234], predictions concerning influenza or influenza-like illnesses are enriched using Electronic Health Records (EHR), social media metrics, internet search trends, and medical surveillance data. In [235], an algorithmic framework is presented that harnesses multiple textual data sources to deliver forecasts for rare disease outbreaks, with predictions derived from a weighted combination of individual data source predictions.

Various strategies for data source amalgamation and robust forecasting have been outlined thus far and a recurring theme is the utilization of data fusion and feature selection via weighting procedures, statistical methodologies, or evolutionary algorithms. Nevertheless, when intricate relationships manifest both among the features and between the features and targets, the aforementioned strategies might falter in modeling these patterns. On the other hand, when Neural Networks, or broadly supervised learning approaches, are employed for the fusion process, the vectorized representation of the initial information is often fashioned such that the loss function directly aligns with the ultimate classification or regression objective. Consequently, this might yield a representation of the feature space that lacks self-consistency, rendering it ill-suited for modified prediction strategies or versatile extensions.

In light of the mentioned context and to address the mentioned challenges, the forecasting framework introduced in Section 5.5.3 has been expanded into a two-stage structure that focuses on multi-sequence fusion. Concerning the conducted experiments, in the data fusion phase, exogenous time-series – specifically *air-quality* and *weather* sequences – are compacted into a coherent representation through an AutoEncoder structure. This serves threefold purposes: (i) attenuating the noise inherent in the high-dimensional feature space stemming from multiple time-series, (ii) preserving the intricate temporal patterns interwoven among the features, and (iii) achieving a concise vector representation amenable to various forecasting approaches. With respect to the forecasting technique, the ensemble approach still utilizes a hybrid neural network, leveraging features distilled from the raw data and predictions sourced from diverse Machine Learning algorithms employed as regressors. Moreover, the proposed design allows for easy additions to the Machine Learning prediction layer without changing the compression or combination stages. This flexibility makes the whole process more versatile and suitable for different situations.

5.6.2 Correlation and Causality Analysis

The initial Electronic Health Record (EHR) dataset, for each entry, displays a scheduled appointment for a specific patient. The patient is designated by a unique numerical identifier corresponding to an encrypted version of their National Identification Number. Each record specifies the appointment's date in the regional health department of interest, alongside its status, which can be categorized as either *booked*, *confirmed*, or *cancelled*. A time-series that shows the number of medical services for respiratory diseases, based on the daily count of confirmed bookings from 2014 to 2019 has been extracted, and any date with missing data has been populated with a zero value, interpreting the absence as a lack of provisions on that specific day.
Moreover, an hypothesis arise regarding whether integrating external data sources, specifically those related to *air-quality* and *weather conditions*, might enhance the forecast's accuracy. Before integrating these potential correlations into the forecasting, it is essential to confirm the interdependencies between the *respiratory diseases* series and the mentioned external data. The investigation focused on several key quantities:

- PM2.5 (fine particles less than 2.5 μm in diameter, in μg/m^3);
- PM10 (particles with a diameter of 10 μm or less, in μg/m^3);
- CO (carbon monoxide, in mg/m^3);
- NO2 (nitrogen dioxide, in μg/m^3)
- Average temperature (in °C);
- Cumulative precipitation (in millimeters);
- Wind velocity, derived from the 100m u-component and v-component of the wind, presented in meters per second.

In particular, data pertaining to *air-quality* has been sourced from ARPAC-CEMEC (Agenzia Regionale Protezione Ambientale della Campania - CEntro MEteorologico e

Table 5.3: Summary of all the time-series considered for the prediction of *respiratory diseases bookings* (RDB) temporal sequence.

Date	RDB TS	Air-Quality TS				Weather TS		
		PM2.5 ($\mu g/m^3$)	PM10 ($\mu g/m^3$)	CO (mg/m^3)	NO2 ($\mu g/m^3$)	$\bar{T}(C)$	$TP(mm)$	$WS(m/s)$
2017/05/01	0	7.362	16.441	0.448	27.161	14.699	0.0337	1.532
2017/05/02	251	7.982	22.0477	0.505	27.041	15.133	1.565	1.063
2017/05/03	326	8.664	23.812	0.531	23.668	15.621	0.957	1.118
2017/05/04	331	9.641	26.659	0.523	28.497	15.565	0.0327	1.194
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2019/04/30	153	6.583	12.733	0.446	28.732	12.211	0.0758	1.272

Climatologico), encompassing averaged measurements collated from multiple Naples stations. Conversely, the *weather* data has been extracted from the ERA5 hourly archives provided by the Copernicus Climate Change Service Climate Data Store. For alignment with the *respiratory diseases* dataset, each of these series has undergone daily sampling. The daily readings were obtained by averaging the respective 24-hour measurements, with two exceptions: the *total precipitation*, which is recorded as the cumulative sum over a day, thus utilizing the day's concluding value, and the *wind speed*, which employs a daily average computed from the hourly magnitude determined from its *u-component* and *v-component.*

The considered period, starting on May 1st, 2017 and ending on April 30th, 2019, is obtained as the intersection of the available periods in the provided time-series. In Tab. 5.3 a summary of the entire data set of considered sequences for the forecasting problem is reported. An initial challenge emerges from the pronounced presence of high-frequency seasonal components: as illustrated in Fig. 5.7 on the left, a marked weekly seasonality is present. This can be attributed to the lack of bookings during weekends. When assessing other time-series, the *temperature* dataset exhibits an annual seasonality. However, this pattern is absent in both the *total precipitation* and the *wind speed* datasets. In relation to the *air-quality* datasets, all series manifest a subtle weekly seasonality, likely linked to the traffic congestion commonly observed on workdays.

To effectively engage in relation discovery, it's essential to mitigate any pronounced short-term seasonality across all sequences; consequently, a *7-samples* moving average was implemented to address this requirement. A foundational step in the analytical

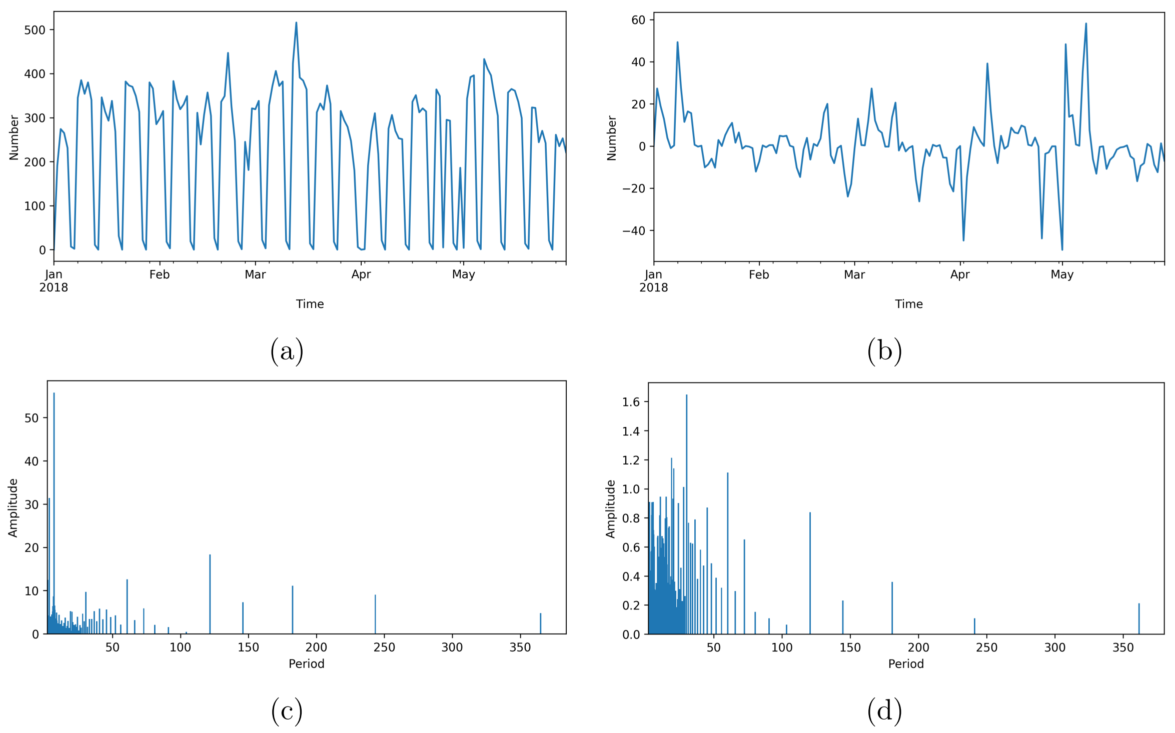


Figure 5.6: Time series of *respiratory diseases medical service*. The left side represents the original data and related spectrogram, while the right side shows data obtained after the preprocessing phase.

process involves ensuring the stationarity of the considered time-series, given that the applied statistical methods leverage causality tests, which are intrinsically based on this concept. We employ the Augmented Dickey-Fuller test [236] as the benchmark for determining the stationarity of the time-series. The essence of this test is to ascertain the presence of a unit root within the series. When the null hypothesis, indicating the presence of a unit root, fails to be rejected, the series undergo a first-order differencing, followed by a reiteration of the test. In our dataset, post-deseasonalization, a single differentiation proved adequate. The outcomes of this preprocessing stage are illustrated on the right side of Fig. 5.7.

After ensuring stationarity across all the series, we utilize tools from two predominant statistical domains: *correlation analysis* and *causality analysis*. While the former gauges the potency of relationships between variables – specifically employing the linear function in the case of the Pearson correlation – the latter evaluates the influence of one sequence as regressive data concerning the prediction of another. It's crucial to consider potential delayed effects in the given time-series, thus necessitating an analysis of time-lagged relationships. For the correlation analysis, in particular, the *time-lagged cross-correlation* is exploited, which affords insights into synchronicities between series, taking time lag into account. In order to discern causality, a suite of techniques, in-

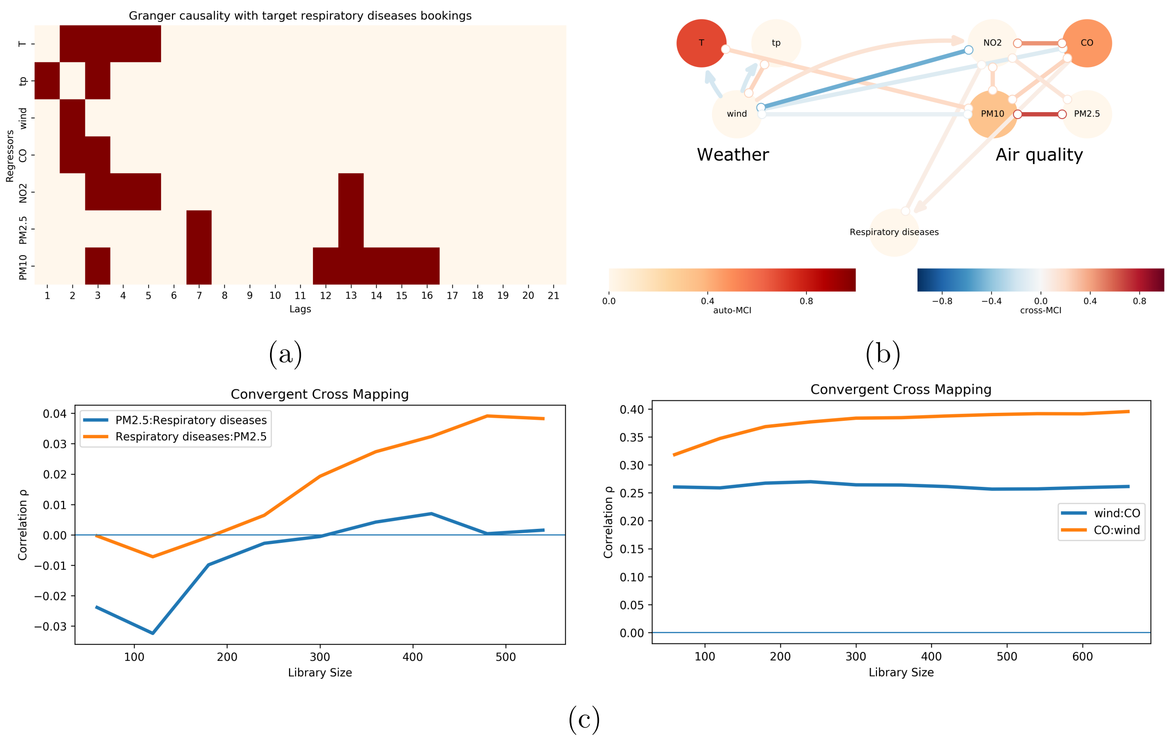


Figure 5.7: (a) Heatmap of the Granger causality between the regressors and the respiratory diseases bookings time-series at different lags . Dark colour indicates that a regressor time-series Granger-causes the bookings, light pink otherwise. The threshold for the positive response is obtained imposing the p-value less than or equal to 0.05. (a) Graph of causality relations obtained with *PCMCI*$^+$. The links with an arrow indicate direct causality towards the pointed variable, while the non-arrowed links indicate unclear direction of causality. The more the link is blue, the more adverse is the effect, while the more the link is red, the causality is proactive. (c) Results of CCM on relation between respiratory disease booking time-series and PM2.5 concentration, and on the relation between wind speed and CO.

cluding *Granger causality*, *Convergent Cross Mapping* (CCM), and *PCMCI*$^+$ has been employed, since the usage of diverse methodologies mitigate potential artifacts linked to the limitations of individual methods, as corroborated by existing literature. In particular, *Granger causality* operates on the principle of temporal precedence: roughly speaking, it questions whether the historical values of variable A can better forecast variable B, compared to B's own antecedents. While its simplicity and ubiquity across sciences make it appealing, it's constrained by its linearity and universal assumptions. Applied to the context under consideration, as evident in Fig. 5.7a, air-quality time-series manifested protracted causality effects, in contrast to the fleeting effects displayed by weather variables. On the other hand, *CCM* delves into non-linear causality: by gauging one time-series' ability to map onto another, it uncovers causal links where linear methodologies may falter. The application of this technique to the considered dataset, as presented in Fig. 5.7c, reveals a mild positive predictive capacity of the regressors on respiratory diseases. Amongst these, air-quality time-series and the interaction between wind speed and CO stood out, whereas total precipitation displayed a marginal one-sided causality on particulate concentration. *PCMCI*$^+$, an enhancement over its predecessor *PCMCI*, is the state of the art of the causal discovery: it evaluates relations as a multivariate time-series graph, where nodes signify time lags, and links indicate causal connections. As illustrated in Fig. 5.7b, this test reveals that the relationships among the regressors are subtle, with the notable exception of the direct causality from CO to respiratory diseases.

5.6.3 Prediction Workflow

Preliminary investigations discussed so far highlight interrelationships within the three time-series groups analyzed for predictive purposes: specifically, the *respiratory diseases bookings* series seems to be impacted by both the *weather* and the *air-quality* sequences, albeit not in a purely linear manner. Furthermore, there are evident links between the *weather* and *pollution* series, with the latter seemingly being influenced by the former. Given these insights, the objective is to leverage these *mild short-to-mid-term* dynamics to enhance the short-term forecasting of the *respiratory diseases bookings* series. Recognizing the intricate nature of the aforementioned relationships, a direct approach to extract predictive insights from these interactions seems infeasible. As a solution, we expanded the forecasting framework described in the previous Section: the design was improved in such a way multivariate time-series can be exploited to provide *7-days ahead* predictions. Additionally, due to the high-dimensional feature space arising from the time-lagged versions of the external time-series (which serve as input to the predictive segment, in tandem with autoregressive and periodicity elements of the *respiratory diseases bookings* series), we've incorporated an LSTM Autoencoder into the general architecture. Its chief function is to streamline the dimensionality of this feature set, ensuring the preservation of relationships between the *weather* and the *air-quality* sequences.

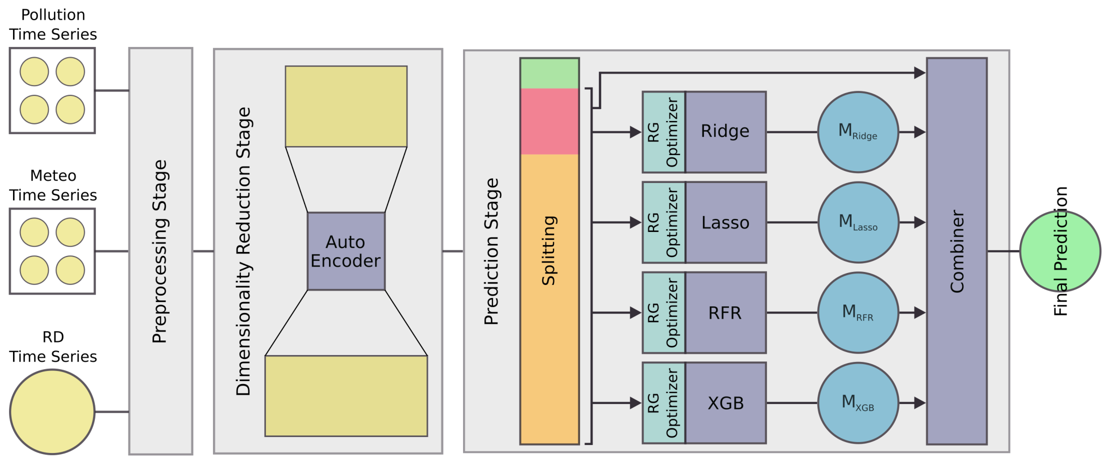


Figure 5.8: Schematic design of the proposed framework: temporal sequences of *respiratory diseases bookings*, *weather* and *air-quality* variables are preprocessed; then the dimensionality reduction strategy is applied on the subset composed by lagged variables extracted from exogenous time-series; Finally, the obtained dataset is split in *train/validation/test* sets, used to train, optimize and validate the whole procedure: the machine learning predictors are trained over the *train set* (in yellow), their hyperparameters are optimized through a *random grid search*, and their forecast on the validation set (in red) fed the *Combiner*, with lagged and periodicity features, which is trained in recognizing patterns between time-series attributes and forecasts produced by predictive models. The final forecast is then obtained on the *test set* (in green).

Preprocessing Stage

The *respiratory diseases bookings* series displays a distinct multimodal periodicity, as observed from spectrogram analysis, which correlates with yearly and weekly cycles; these intrinsic time-series properties are utilized to craft categorical features reflecting such periodic patterns. Variables accounting for weekends and holidays are also integrated into the dataset. The autocorrelation analysis indicates a significant relationship between each data point and the series' activity from the preceding seven days. Consequently, for the intended *7-step-ahead* prediction, lags from *lag7* to *lag14* are incorporated.

Given the subtle interconnections among the three sets of sequences, lagged variables from both the *weather* and *air-quality* series were incorporated into the feature space. Various tests for both linear and non-linear causality exposed inconsistent short-to-medium-term correlations within these two time-series sets; as a result, lags ranging from 0 to 28 for the *weather* series and 7 to 28 for the *air-quality* series were chosen as the predictive variables for the forecasting challenge. It's important to recognize that the framework in discussion intends to forecast a span of seven days into the future. Thus, data from *lag0* to *lag6* of the *weather* series, theoretically, would not be available since they represent future values. However, considering that *weather* predictions have been validated as reliable for up to a week, the experiments described subsequently simulate a scenario where the final prediction also leverages these accessible *weather* forecasts. Mathematically, the preprocessing described so far can be encapsulated as:

$$y_t \to \mathcal{H}(t, \Theta) \tag{5.5}$$

Here, y_t denotes the prediction target for the *respiratory diseases bookings* series. With y as the time-series of interest and $\tilde{y}_i$, ranging from i to k, as exogenous series, the map $\mathcal{H}$ associates each y_t with features ($[y_{t-7}, ..., y_{t-s}], [\tilde{y}_{1,t-s'}, ..., \tilde{y}_{1,t-28}], ..., [\tilde{y}_{k,t-s''}, ..., \tilde{y}_{k,t-28}], \Theta_t$), where $s = 14$, $s' = 0$ for *weather*, $s'' = 7$ for *air-quality*, and Θ_t delineates periodicities linked to sample t.

Dimensionality Reduction

During the preprocessing phase, a feature space with considerable dimensionality emerges. Within this context, interactions among the $\tilde{y}_i$ sequences are observed, as mentioned previously. However, due to their subtle nature, these interactions might go unnoticed by standard ML prediction algorithms, especially in the face of the well-known "curse of dimensionality". Adopting a *data-driven* perspective, and to mitigate challenges associated with *noise* and high dimensionality, an LSTM Autoencoder is integrated into the system, positioned directly before the forecasting component. The inclusion of LSTM cells in the Autoencoder's architecture stems from these recurrent units' ability to effectively process and recall sequences, thereby shaping the *latent space* efficiently. Since the feature subspace derived from lagged variables of external datasets (specifically from *weather* and *air-quality* sets) arises from sequences governed by nuanced non-linear relationships, the LSTM-Autoencoder facilitates a thorough dimensionality reduction process. This leads to a transformation into a denser space, ensuring both inherent temporal associations and relationships across different time-series are comprehensively accounted for.

5.6.4 Predictive Stage

Following data compression, the concluding step outlined in Fig. 5.8 aims to produce forecasts from the refined dataset. A data point at time t can be represented as:

$$y_t \rightarrow (\ [y_{t-7}, ..., y_{t-s}], \Gamma_t, \Theta_t\ ) \tag{5.6}$$

where Γ_t encompasses the latent representation of variables at time t, obtained from preceding compression processes. Clearly, this stage progresses through three key components: (i) data partitioning, (ii) an initial layer of ML techniques generating early predictions, and (iii) the Hybrid Neural Network creating the ultimate forecast by amalgamating the initial outputs.

Data is divided into three specific temporal subsets, in a manner analogous to that used in the smart parking case

Based on preliminary analysis findings, and drawing parallels with the smart parking case study, the selected algorithms encompass two tree-based regressors and two linear models. Specifically, we employed a Random Forest regressor and a Gradient Boosted Decision Trees Regressor. Additionally, Ridge and Lasso linear predictors are incorporated. In the learning segment of this workflow, these models are trained on $\mathcal{D}_{train}$ and generate forecasts for $\mathcal{D}_{val}$. For this study, each machine learning model undergoes Hyperparameter Optimization (HO) based on predictions for the validation set, using a Random Grid Search. The decision to replace GA with Random Grid Search for this task primarily stems from considerations of computational efficiency: given the complexity of the data input into the ML predictors, the Random Grid Search has demonstrated results comparable to GA while significantly reducing the time required for optimization. After optimizing these algorithms, predictions produced on $\mathcal{D}_{test}$ serves as the feature set for the *Combiner*, whose aim is to integrate the provided forecasts.

5.6.5 Results

In order to validate the discussed approach, forecasts of the hybrid *Combiner* network, obtained without exogenous variables and with them, both in the case they are compressed through the *Autoencoder* and considered without compression, have been compared.

Experimental setup

Due to the inherent temporal sequence of time-series data, also this scenario's validation follows a variant of the Nested k-fold Cross-Validation. Moreover, given the

Table 5.4: Hyperparameter configuration for the AutoEncoder and the *Combiner*. Regarding the AutoEncoder, n_{regr} represents the total number of features of the dataset generated from exogenous time-series.

Type	Layers	Units	Kernel sizes	Activation	Value
Autoencoder	Encoder LSTM	3	$(n_{regr}, n_{regr}/2, 36)$	$tanh$	-
	Decoder LSTM	3	$(36, n_{regr}/2, n_{regr})$	$tanh$	-
	Learning rate	-	-	-	$1e-4$
	Regularization	-	-	-	$l_2(5e-8)$
	Batch size	-	-	-	256
Combiner	LSTM	2	$(96, 48)$	$tanh$	-
	Conv1D	4	$(8, 8, 8, 8)$	$ReLU$	$(2, 3, 5, 7)$
	Dense	3	$(64, 32, 1)$	$ReLU$	-
	Learning rate	-	-	-	$1e-3$
	Regularization	-	-	-	$l_2(8e-3)$
	Batch size	-	-	-	32

consistent scale of our temporal sequence and the occurrence of zero-valued samples (attributable to weekends and holidays), the choice was to employ absolute error metrics over percentage-based ones, as suggested by [237]. The selected evaluation metrics encompass the Mean Absolute Error (MAE), the Root Mean Squared Error (RMSE), and the R^2 coefficient. While MAE and RMSE gauge the magnitude of the error, R^2 quantifies the proportion of variability captured by the prediction relative to the actual values.

For the Autoencoder and *Combiner* networks' configurations, Tab. 5.4 lists the hyperparameters. These selections were determined after thorough heuristic testing of various configurations. It is worth underlining that the hyperparameters for the *Combiner* remained consistent across all experiments, regardless of feature setup variations.

Encoding

Compression is executed before the prediction phase, specifically a single time prior to the k-fold operation. The encoding outcomes are illustrated in Fig. 5.9, detailing the reconstruction error associated with the decoded compressed features. With an aggregate mean error of 0.01218 ± 0.00136, and given the rescaling of regressors between 0 and 1, the reconstruction of lagged values is notably precise. The distinct peaks suggest that discrepancies in the reconstruction primarily arise from specific values across all lagged components of both *weather* and *air-quality* sequences.

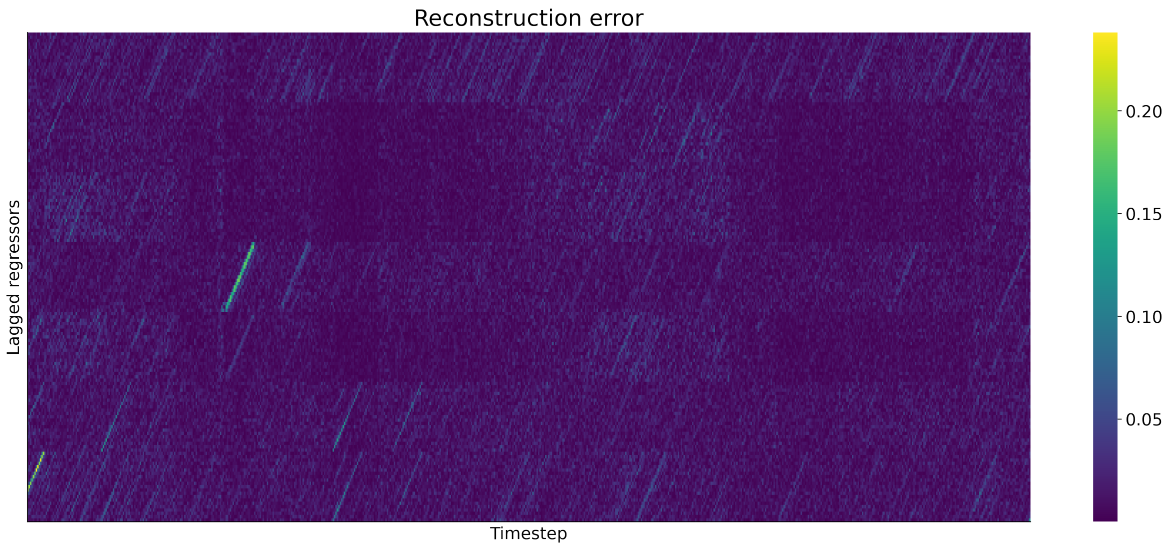


Figure 5.9: Heatmap of the reconstruction error, computed as the absolute difference between the original regressors and the reconstructed regressors. Darker color means lower reconstruction error.

Predictions

To assess the forecasting efficacy of the introduced approach, performance metrics, specifically those related to error, were calculated based on a 5-fold cross-validation. These results were juxtaposed against those of several prevalent methodologies. This comparison set includes univariate forecast models such as classical ARIMA, SARIMA, and ETS, along with their hybrid derivatives: ARIMA-ANN [238], SARIMA-ANN [239], and ETS-ANN [240]. The autoregressive components of these models underwent optimization via an automatic grid search on the hyperparameters, informed by the Akaike Information Criterion. Additionally, Machine Learning techniques, including Support Vector Regression (SVR), Multi Layer Perceptron (MLP), and vanilla LSTM, were evaluated. Each was refined according to a procedure analogous to that previously described. Individual predictors integrated within the proposed framework were also assessed. Exploration further extended to weighting ensemble strategies, applied to predictors similar to the *Combiner*. This set encompassed methods like average and median [178], Particle Swarm Optimization (PSO) [241], Genetic Algorithm (GA) [242], and the Kalman Filter (KF) [243], which operates as a time-variant weighing model.

Referring to Tab. 5.5, the ensemble method consistently demonstrates superior performance in terms of both average MAE and RMSE, surpassing individual regressors and other combination techniques by delivering predictions with considerably reduced errors. When evaluating the R^2 metric, a nuanced perspective is required: in scenarios without exogenous sequences or when the *weather* and *air-quality* data are integrated as compressed features, our Hybrid Neural Network captures a refined temporal pattern for the *respiratory diseases bookings* time-series, resulting in an enhanced modeling performance based on this metric. However, when the dimensionality reduction step is bypassed, while the R^2 exhibited by the *Combiner*'s forecasts might not always be the best result, it closely trails the top performers. This underlines the *Combiner*'s proficiency in leveraging machine learning model forecasts as enriching factors and discerning the most trustworthy among them.

The comparative analysis among the three configurations highlights the models' behavior relative to the incorporation of exogenous time-series into the regression problem. As indicated in the Tab. 5.5, when compression is omitted, the high-dimensional feature space leads linear regression models and the boosted tree regressor to outperform the baseline. However, while deep learning models (MLP, LSTM) and the SVR don't significantly reduce average error measures, they exhibit a better understanding of the time-series patterns, as evidenced by their improved R^2 values. This improvement affects the standard deviation of MAE and RMSE, even if they, on average, aren't superior to previous results. Conversely, the Random Forest struggles to deliver consistent predictions, especially noticeable in its mean R^2 score and significant variability. This can be attributed to the "curse of dimensionality" and noise in the expanded dataset. The commendable performance of Lasso, Ridge, and XGB can be traced back to two

Predictor	Baseline			All regressors			Encoded regressors		
	R^2	MAE	$RMSE$	R^2	MAE	$RMSE$	R^2	MAE	$RMSE$
ARIMA	−0.065 ± 0.825	41.454 ± 16.730	49.369 ± 22.037	-	-	-	-	-	-
SARIMA	0.618 ± 0.397	34.118 ± 16.424	41.014 ± 22.893	-	-	-	-	-	-
ETS	0.607 ± 0.424	34.530 ± 19.776	41.410 ± 24.448	-	-	-	-	-	-
ARIMA-ANN	0.177 ± 0.705	38.580 ± 21.934	46.545 ± 27.007	-	-	-	-	-	-
SARIMA-ANN	0.610 ± 0.415	33.907 ± 16.894	40.644 ± 23.666	-	-	-	-	-	-
ETS-ANN	0.587 ± 0.454	33.910 ± 19.993	40.919 ± 25.008	-	-	-	-	-	-
MLP	0.106 ± 0.568	41.217 ± 20.253	51.677 ± 17.313	0.399 ± 0.176	53.794 ± 8.541	62.286 ± 12.109	0.419 ± 0.177	49.981 ± 10.015	55.632 ± 10.944
LSTM	0.526 ± 0.293	42.309 ± 11.669	49.309 ± 10.131	0.643 ± 0.134	46.401 ± 7.047	55.192 ± 5.515	0.681 ± 0.078	47.264 ± 8.834	51.861 ± 8.933
SVR	0.491 ± 0.217	52.338 ± 14.216	58.707 ± 13.597	0.552 ± 0.098	47.622 ± 8.497	57.459 ± 8.176	0.654 ± 0.151	52.252 ± 10.891	57.595 ± 7.741
Lasso	0.627 ± 0.131	50.731 ± 10.365	59.021 ± 11.042	0.749 ± 0.067	46.360 ± 6.367	52.360 ± 8.360	0.733 ± 0.095	46.071 ± 7.279	54.527 ± 9.417
Ridge	0.648 ± 0.135	48.997 ± 10.302	57.276 ± 10.473	0.749 ± 0.070	49.473 ± 7.797	55.061 ± 9.295	0.747 ± 0.086	46.226 ± 7.688	54.644 ± 10.147
XGB	$\underline{0.776 \pm 0.112}$	29.034 ± 11.780	39.342 ± 15.659	**0.80 ± 0.087**	$\underline{25.782 \pm 10.206}$	35.849 ± 11.359	$\underline{0.786 \pm 0.070}$	26.237 ± 3.338	34.903 ± 4.854
RFR	0.679 ± 0.156	33.509 ± 5.266	44.027 ± 8.795	−0.210 ± 0.841	58.905 ± 9.791	68.195 ± 14.139	0.563 ± 0.276	36.712 ± 11.661	47.881 ± 14.494

Ensemble	Baseline			All regressors			Encoded regressors		
	R^2	MAE	$RMSE$	R^2	MAE	$RMSE$	R^2	MAE	$RMSE$
Average	0.717 ± 0.110	38.009 ± 10.300	46.888 ± 12.075	$\underline{0.764 \pm 0.077}$	36.651 ± 6.814	43.201 ± 8.643	0.759 ± 0.1017	37.608 ± 5.514	44.413 ± 7.498
Median	0.709 ± 0.112	38.501 ± 10.762	47.822 ± 11.330	0.760 ± 0.079	38.983 ± 8.360	46.094 ± 9.084	0.751 ± 0.0896	35.470 ± 6.771	44.358 ± 8.556
PSO	0.541 ± 0.365	24.365 ± 8.340	31.402 ± 10.569	0.248 ± 0.428	31.329 ± 6.713	38.476 ± 7.177	0.637 ± 0.1774	$\underline{24.225 \pm 4.181}$	$\underline{30.163 \pm 5.484}$
KF	0.672 ± 0.080	35.914 ± 9.428	43.705 ± 7.268	0.714 ± 0.064	34.340 ± 3.537	41.257 ± 4.441	0.693 ± 0.0505	34.750 ± 1.616	42.127 ± 2.935
GA	0.577 ± 0.306	$\underline{23.908 \pm 6.691}$	$\underline{30.714 \pm 9.138}$	0.436 ± 0.341	28.487 ± 5.146	$\underline{35.646 \pm 4.408}$	0.605 ± 0.2083	24.420 ± 4.139	30.741 ± 5.573
Combiner	**0.817 ± 0.117**	**20.707 ± 5.665**	**26.184 ± 7.175**	0.746 ± 0.209	**23.926 ± 10.418**	**30.177 ± 13.002**	**0.855 ± 0.139**	**18.088 ± 6.082**	**21.544 ± 7.699**

Table 5.5: Comparison of 5-Fold Cross Validation results for *7-days ahead* forecasts between single predictors, models and combination strategies discussed, and the proposed framework. The measures are expressed as mean ± standard deviation of the k-fold results. Best values are highlighted in bold, while second best ones are underlined.

factors: i) their inherent mechanisms to filter out or minimize the impact of irrelevant features, and ii) the adaptability in their hyperparameter optimization given variations in the feature space.

Ensemble strategies generally mirror the baseline results, showing reduced error variability due to the commendable performance of three out of the four combination methods. However, the *Combiner* lags behind the baseline in both average error metrics and their consistency, despite achieving, on average, the best RMSE and MAE values among the compared methods. This discrepancy arises from the combination methodology. While a weighted combination relies solely on machine learning algorithm predictions, the *Combiner* also factors in the predictions, aggregated dataset, and features extracted from the *respiratory diseases booking* series. Consequently, the high dimensionality and noise from the *weather* and *air-quality* lagged variables interfere with the accurate reconstruction of temporal patterns. The rightmost section

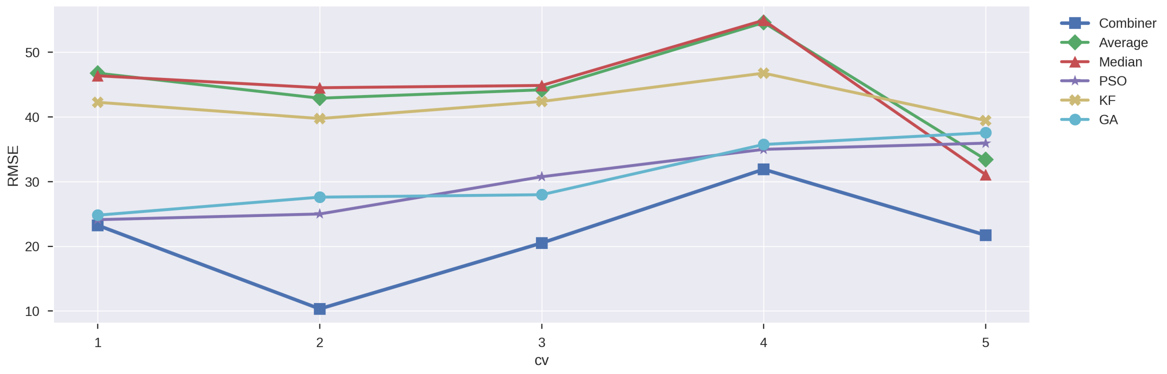


Figure 5.10: Comparision of the RMSE for the different combination strategies with respect to each fold of the 5-fold cross validation procedure. Predictions are obtained in the case the low-dimensional feature space is used.

of the table presents error metrics when encoding is applied to lagged variables from external time-series. Notably, dimensionality reduction has minimal impact on average MAE and RMSE for single regressors, yielding results comparable to those without compression. Furthermore, the Random Forest regains its predictive prowess, showing performance slightly below the baseline. Weighting strategies largely maintain their accuracy, with some (like PSO and GA) improving due to the enhanced Random Forest predictions. This attests to the LSTM-Autoencoder's capability to retain subtle correlations, as discussed before. In this context, the *Combiner* leverages these correlations to boost its prediction accuracy across all error metrics. While the improvement in the R^2 score is modest compared to the baseline, the reductions in MAE and $RMSE$ are more pronounced. This solidifies the proposed approach's superiority, delivering the most consistent and accurate predictions, even against other combination strategies, as further visualized in Fig. 5.10.

5.7 Case Study: ENCODE

Recognizing the ubiquity of time-series in both applied and scientific domains, the investigations into smart parking and healthcare have underscored the necessity for a systematic and resilient forecasting framework. Multivariate time-series forecasting is acknowledged as one of the most intricate challenges in data analytics since this task requires the discernment of intricate patterns both within and between series, often complicated by factors like high dimensionality and noisy observations. As highlighted by [244], for a predictive technique to serve real-world applications effectively, it must exhibit not only accuracy – with respect to a tailored loss function addressing the unique characteristics of the problem – but also robustness. Consequently, the recent years have witnessed rigorous endeavors to attain these benchmarks. The literature reveals a preponderance of methodologies crafted to harness the capabilities of several predictive techniques, as evinced by the outcomes of the M4 [245] and M5 [246] competitions. Predominantly, such strategies converge on the amalgamation of techniques, each attuned to discern particular patterns, with the goal of procuring a final prediction more resilient than its individual constituents. Broadly, these methodologies fall into two pivotal categories: *ensembles*, which meld numerous predictions via a unifying layer, and *hybrid methods*, wherein disparate techniques function cohesively.

Foundational insights into the efficacy of these methodologies are expounded in [247] and [248]. In light of these studies, and based on the investigations conducted in the two cases study discussed in previous Section, a pioneering ensemble framework tailored for multivariate time-series forecasting has been designed. This structure effectively discovers correlationsand hidden patterns in multivariate data utilizing deep learning paradigms. Thus, it is named Ensemble Neural Combination for Optimal Dimensionality Encoding in Time-series Forecasting, or ENCODE for brevity. In addressing

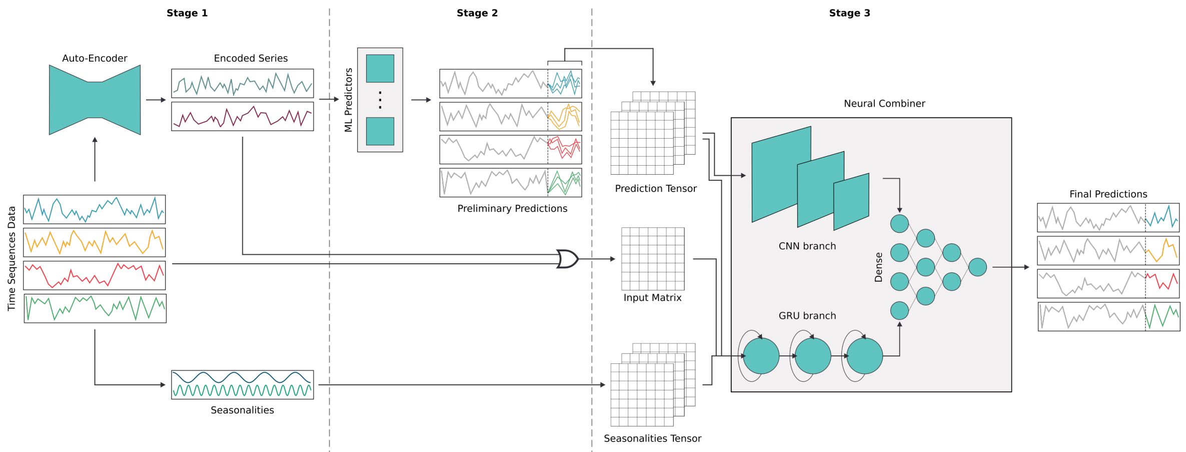


Figure 5.11: The framework is made up of three components: the AE encodes the lagged time-series to reduce the dimensionality and clean the noise; the single predictions attempt to predict the future values starting from the encoded variables; and the *Combiner* takes as input the single predictions, the seasonality, and the encoded variables to obtain the final prediction.

challenges such as high dimensionality, data noise, missing values, and varying scales, a modular approach is proposed. This begins with a custom pre-processing phase, followed by a three-stage structure to extract new insights from the data. In the first stage, an autoencoder (AE) is used to encode lagged variables into a more compact latent space, which is then utilized in subsequent stages. In the second stage, a variety of machine learning and statistical methods generate individual predictions. Concluding the process, an hybrid neural network inspired to the method discussed in the beginning of this chapter, acts as a *Combiner* of these predictions to produce the final forecast. With respect to the previous design, GRU layers have been adopted in place of LSTM ones, in order to reduce the number of parameters of the network and facilitate the convergence procedure. Inputs to the *Combiner* include individual predictions, seasonal patterns, and an input matrix containing the original or encoded time-series, depending on the dimensionality of the data space. The final goal is to capitalize on the temporal and spatial interdependencies among input variables, thereby enhancing the prediction quality.

As concerns the novelty of the proposed approach, while the proposed framework can fall in the ensemble-based methods, its uniqueness doesn't lie in the individual components but in their combination to boost performance. In particular two main features stand out: the AE phase's ability to handle outlier values effectively and the neural network's architecture, purposefully designed to capture temporal patterns and the relationships between individual predictors.

Seeking a position for ENCODE in the literature among competitor methods, it is necessary to underscored the robustness of ensemble and hybrid methods for time-series prediction. As some hybrid methods, ENCODE focus on dimensionality compression and noise reduction. However, through its AE, ENCODE guarantees non-linearity in the projection of data into the low-dimensional space. Transitioning to ensemble techniques, they predominantly follow a two-pronged approach: first, the generation of diverse predictions, and subsequently, a synthesis strategy to create the final forecast. Although similar steps are also present in ENCODE, the *Neural Combiner* not only serves as aggregation step for preliminary forecasts, but also directly contribute to the prediction thanks to the knowledge extracted from time-series data. Given our proposal's alignment with these typologies of methods, our focus sharpens on frameworks captured succinctly in Table 5.6. This table delineates the characteristics differentiating our proposal from analogous techniques.

5.7.1 ENCODE Methodology

As previously highlighted, the ensemble consists of three primary components, detailed in Fig. 5.11 and Algorithm 1. Summarizing, our approach unfolds across three stages:

Ref.	Methods	Steps	Application Field	Multivariate	Multi Steps-Ahead
[249]	SL, ML and DL with NN combiner	2	General Applications	No	No
[250]	NN with median and globally weighted averages	4	Wind Energy	Yes	Yes
[192]	Ensemble of LSTM with SVR combiner	2	Wind Speed	No	Yes
[186]	ML and Heterogeneous Ensembling	2	General Applications	No	No
[191]	Ensemble of LSTM with RFR and Ridge as combiners	2	General Applications	No	Yes
[251]	SL with NN selector adequate forecasting	2	Industry Sales	No	No
[252]	ML and Wavelet NN	3	Sunspots	No	No
[253]	LSTM based on Ensemble Empirical Mode Decomposition	2	Oil Production	No	No
[254]	Regime identification model and multiple regression trees	2	Traffic Flow	Yes	No
[187]	ML methods and weighted Iterative Ensembling	4	Electricity	Yes	No
[255]	Ensemble of cloud and fuzzy models	2	Weather Time Series	No	No
[193]	Hybrid Ensemble Strategy with LSTM	2	Power Demand	No	Yes
[188]	Neuro-Fuzzy Ensemble Model	2	Finance	No	No
[256]	Four kinds of NN and dynamic weight combination	2	General Applications	No	No
[183]	Decomposition-Ensemble approach with NN	4	Hong Kong tourism demand	No	Yes
[257]	Set of NN with bagging ensembling approach	2	Chinese Stock Market Index	No	No
[258]	Sparse Representation and NN	3	Crude Oil Price	No	Yes
[259]	SL and weighted mean	3	Regional Photovoltaic Power	No	No
[260]	SL and SVR combiner	2	Electric Load Data	No	No
[261]	AutoEncoder and bootstrap aggregation	2	Crude Oil Price	Yes	No
ENCODE	AutoEncoder and Hybrid Neural Combiner	3	General Applications	Yes	Yes

Table 5.6: A list of methodologies, with references in literature, which can be compared to ENCODE. For each, the methods applied, the number of steps, the application field, the uni/multivariate framework, and the 1/multi-steps ahead predictions are highlighted.

initially, the AE processes the lagged time-series to reduce data dimensionality and filter out noise; subsequent to this, individual predictors utilize the encoded variables to forecast upcoming time-series values; finally, the *Combiner* assimilates individual forecasts, along with the seasonality and encoded variables, to produce the final prediction.

Introductory Notes

Assuming a multivariate framework the following notations will be used:

- $T = \{1, ..., N\}$ denotes the time set;
- M is the number of time-series $X^1, ..., X^M$, with $X^j = \{x_t^j\}_{t\in T}$, $\forall j \in \{1, ..., M\}$;

Algorithm 1 ENCODE

```
1: {$\textbf{\textit{Pre-processing Stage}}$}
2: Merge time-series $X^j$ into a dataset $X$.
3: Divide time interval $\{1,...,N\}$ into $TrainingP = \{1,...,n_{TrainingP}\}$, $TC\_Val = \{n_{TrainingP}+1,...,n_{Validation}\}$, and $Test = \{n_{Validation}+1,...,N\}$.
4: Split $TC\_Val$ into $TrainingC = \{n_{TrainingP}+1,...,n_{TrainingC}\}$ and $Validation = \{n_{TrainingC}+1,...,n_{Validation}\}$
5: Fit the $[0,1]$ Scaler and transform $X$.
6: Decompose time-series $\{I_t\}_{t\in TrainingP}$ to obtain seasonalities $\{S_t\}_{t\in\{1,...,N\}}$.
7: Obtain the vectors $\{I_t\}_{t\in\{1,...,N\}}$ of lagged inputs.
8: {$\textbf{\textit{Encoding Stage}}$}
9: Train AutoEncoder using $\{I_t\}_{t\in TrainingP}$ as training set and $\{I_t\}_{t\in val}$ as EarlyStopping.
10: Encode all variables to obtain $\{\tilde{I}_t\}_{t\in\{1,...,N\}}$ vectors.
11: {$\textbf{\textit{Single Predictions Stage}}$}
12: for Predictor $f \in \{1,...,F\}$ do
13:     Define Hyperparameter Grid for predictor $f$: $\Theta_f$.
14:     Obtain the optimal hyperparameter $\theta_f$ via Randomized Grid Search.
15:     Fit $f$ on $\{\tilde{I}_t\}_{t\in TrainingP}$ using $\theta_f$ hyperparameter and obtain the predicted values $\{P_{f;t}\}_{t\in TC\_Val}$.
16:     Fit $f$ on $\{\tilde{I}_t\}_{t\in TrainingP\cup TC\_Val}$ using $\theta_f$ hyperparameter and obtain the predicted values $\{P_{f;t}\}_{t\in Test}$.
17: end for
18: Merge predictions $\{P_{f;}\}_{t\in TC\_Val\cup Test}$ to obtain $\{P\}_{t\in TrainingC\cup Test}$.
19: {$\textbf{\textit{Combiner Stage}}$}
20: Define Hyperparameter Grid for Combiner: $\Theta$.
21: Obtain the optimal hyperparameter $\theta$ via Randomized Grid Search.
22: Train Combiner on $\{I_t\}_{t\in TrainingC}$, $\{P_t\}_{t\in TrainingC}$, and $\{S_t\}_{t\in TrainingC}$ data using $\theta$ hyperparameter and $\{I_t\}_{t\in Validation}$, $\{P_t\}_{t\in Validation}$, and $\{S_t\}_{t\in Validation}$ for the early stopping.
23: Exploit $\{I_t\}_{t\in TrainingC}$, $\{P_t\}_{t\in TrainingC}$, and $\{S_t\}_{t\in TrainingC}$ data to obtain final predictions for the Test data $\{\hat{X}_t\}_{t\in Test}$.
```

- $\mathbf{x}_t \in \mathbb{R}^M$, $\forall t \in T$ represents the M-dimensional vector of time-series observations at time t, i.e., $\mathbf{x}_t = [x_t^1, ..., x_t^M]$;
- H is the forecast horizon, i.e., the expected output corresponds to the matrix $X_{t:t+H} = [\mathbf{x}_j]_{j\in\{t,...,t+H-1\}} \in \mathbb{R}^{H\times M}$;
- the prediction exploits K previous lags, i.e., the input corresponds to $X_{t-K:t} = [\mathbf{x}_j]_{j\in\{t-K,...,t-1\}}$;
- for algorithmic purposes, the matrix $X_{t-K:t}$ is flattened to obtain the vector

$$\mathbf{i}_t = [x_{t-K}^1, ..., x_{t-1}^1, x_{t-K}^2, ..., x_{t-K}^M, ..., x_{t-1}^M]_{j\in\{1,...,MK\}} \in \mathbb{R}^{KM}; \qquad (5.7)$$

- S seasonalities are detected and grouped into a matrix $S_t \in \{0,1\}^{H\times S}$ where each row represents the variables expressing such features for a step-ahead prediction.

Hence, the ensemble strategy can be written as the function:

$$\phi = \phi(\cdot;\theta) : (\mathbf{i}, S) \in \mathbb{R}^{MK} \times \{0,1\}^{H\times S} \to \hat{X} \in \mathbb{R}^{H\times M} \tag{5.8}$$

So, the encoder processes an input vector of lagged values along with a matrix signifying seasonalities, ultimately producing the predicted outputs. The function ϕ hinges significantly on a hyperparameters vector θ, determined via the Randomized Grid Search. For algorithmic clarity, the input vector $\mathbf{i} \in \mathbb{R}^{MK}$ is treated as a single-row matrix $I \in \mathbb{R}^{1\times MK}$.

In terms of data subset partitioning, a train-validation-test division has been adopted. Each subset is utilized as follows: the *Training-P* set is designated for training individual predictors, and in data-scarce situations, the *Combiner* too; the *Training-C* set is employed to determine the optimal hyperparameters vector for each predictor and for *Combiner* training; the best hyperparameters vector for the *Combiner* is identified using the *Validation* set; finally, the *Test* set serves to contrast our ensemble's forecasts against other leading methods and benchmarks. This data partitioning methodology is visualized in Figure 5.12. Moreover, to assess the performances of the proposed

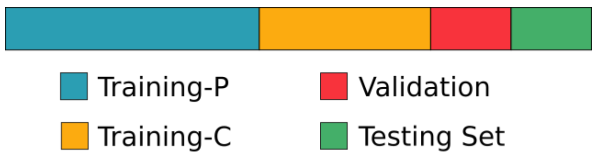


Figure 5.12: The available data are divided into four subsets: the *Training-P* set, the *Training-C* set, the *Validation* set, and the *Test* set. The first is used to train the encoder and the single predictors; the *Training-C* set is used to optimize these algorithms and to train the *Combiner*, whose best hyperparameters are chosen with respect to the Validation set; and the *Test* set is used for the comparison.

framework, it is compared with the following methodologies:

- The standalone predictions of the models integrated within our *Combiner*. This includes Lasso, Ridge, Elastic Net, RFR, XGB, ARIMA, and SVR.

- The random walk approach, specifically the ARIMA(0,1,0) model. This model is defined as $x^j_{t+1} = x^j_t + \mu^j + \sigma^j \varepsilon^j_{t+1}$, where μ^j and σ^j are constants, and ε^j_{t+1} represents white noise. Based on the training data, we forecast x^j_{t+h} as $x^j_t + h\mu^j$.
- The mean and median ensemble strategies derived from the predictions of the models mentioned in the first point. In essence, from the prediction vector of these models, we compute the mean and median values, respectively.
- Ensembles based on Particle Swarm Optimization (PSO, [262]) and Genetic Algorithm (GA, [263]), applied to the models in the first point. Here, PSO and GA determine the weightage of each prediction.
- The N-Beats model [165], in a Python version provided on GitHub[1].
- The BHT-ARIMA model [160], as per the official GitHub repository by its authors[2].
- The Prophet model [161].
- The MTGNN model [174], tailored to our objectives using its official GitHub repository[3].

First Stage: Encoding

In the initial phase, as highlighted by Lines `7-9` of Algorithm 1, we employ an Autoencoder (AE) to transform the lagged variables of the time-series. The primary objective is to convert the input vector $\mathbf{i}$ into a reduced representation $\tilde{\mathbf{i}}$, aiming to preserve the inherent characteristics of the original data. Mathematically, this transformation can be expressed as:

$$\phi_1 : \mathbf{i} \in \mathbb{R}^{MK} \to \tilde{\mathbf{i}} \in \mathbb{R}^{V} \quad \text{where } V \ll MK \tag{5.9}$$

[1]`https://github.com/philipperemy/n-beats`
[2]`https://github.com/huawei-noah/BHT-ARIMA`
[3]`https://github.com/nnzhan/MTGNN`

Such an encoding process is advantageous for augmenting the strategy's resilience: specifically, it minimizes the influence of anomalies, which could arise due to unforeseen events or measurement inaccuracies, on the forecasts. Moreover, the encoding can also play a pivotal role in data denoising. When dealing with a vast collection of time-series, the product of the series count and the selected lags may introduce computational challenges during the training of individual predictors and the final *Combiner*.

Regarding the AE's architecture, two GRU layers for both the encoder and decoder have demonstrated efficacy across various scenarios. The selection of the latent space dimension is contingent on the complexity of the problem and available computational resources. Specifically, a latent dimension ranging from $\frac{1}{4}$ to $\frac{1}{40}$ of the initial input matrix's dimension is selected, based on the problem context. For the learning rate (α), a dynamic approach has been applied: the training starts with a high α subsequently decremented each time the training is halted by early stopping. A dual criteria for halting is established: an α threshold and a discernible improvement metric. If the α is deemed minimal and no notable enhancements in the loss function are observed, the training is terminated. Finally, the mean of the original and the reduced dimensions proves optimal for determining the sizes of the central layers for both the encoder and decoder, negating the need for extensive hyperparameter tuning at this juncture.

The AE is trained on both the *Training-P* and *Training-C* datasets, incorporating early stopping based on the *Validation* set. The encoded variables obtained from the *Training-P*, *Training-C*, *Validation*, and *Test* sets are subsequently utilized in the succeeding stages.

Second Stage: Single Predictions

The subsequent phase, denoted by Lines `10-17` of Algorithm 1, pertains to individual forecasting. Within this phase, the encoded vector $\tilde{\mathbf{i}}$ serves as an input for diverse Machine Learning (ML) and Statistical Learning (SL) methodologies to derive predictions for every series and contemplated forecast horizon. The inherent modularity of our approach allows the incorporation of a wide array of predictive methods to bolster the diversity of employed techniques: linear regression algorithms (Lasso, Ridge, and ElasticNet), SVR, tree-based models (RFR and XGB), and the specialized time-series model ARIMA (uniquely fitted using the original time-series $\mathbf{i}$ rather than its encoded variant $\tilde{\mathbf{i}}$). After rigorous experimentation, the aforementioned methods were heuristically selected, showcasing commendable performance both as independent predictors and as inputs to the *Combiner*. Given F predictors, the function corresponding to the f-th predictor can be defined as:

$$\phi_{2,f} : \tilde{\mathbf{i}} \in \mathbb{R}^{V} \rightarrow P_f \in \mathbb{R}^{H \times M} \tag{5.10}$$

Here, the resultant matrix $P_f = \phi_{2,f}(\tilde{\mathbf{i}})$ is structured such that its primary dimension relates to the prediction intervals, while the secondary pertains to the count of analyzed time-series. Extending this, the collective stage can be mathematically described as:

$$\phi_{2} : \tilde{\mathbf{i}} \in \mathbb{R}^{V} \rightarrow P \in \mathbb{R}^{H \times MF} \tag{5.11}$$

In this equation, the resulting matrix emerges from the concatenation of individual prediction matrices.

As regards the hyperparameter optimization of the regressors, a Randomized Grid Search has been employed following this protocol:

1. Every hyperparameter vector undergoes training for each designated algorithm across every time-series in the *Training-P* dataset.

2. The outcomes are assessed on the *Training-C* dataset.

3. The hyperparameter configuration, which mitigates the loss function (the mean square error), is subsequently adopted.

Post this, the individual predictors are trained on the *Training-P* dataset, with predictions drawn from the *Training-C* dataset. It's imperative to note that predictions from *Training-C* facilitate the *Combiner*'s optimization, as it's vital to gauge the out-of-sample prediction errors and not just the in-sample deviations. In the terminal phase, these models are retrained (retaining the identified hyperparameters) on the unified *Training-P*, *Training-C*, and *Validation* datasets. The final predictions for the *Test* dataset are then obtained.

Third Stage: *Combiner*

In the final part of our methodology, as outlined in Lines `18-22` of Algorithm 1, the *Combiner* is introduced. This component combines the individual predictions to generate the final forecast.

The CNN component processes the predictions P, identifying potential relationships that might enhance the overall forecast performance. The GRU component uses the predictions P, the encoded variables $\tilde{\mathbf{i}}$, and the seasonal variables S. Its purpose is to identify temporal patterns within the data and assess the performance of each predictor under varying conditions. The outputs from the CNN and GRU are then integrated using fully connected layers to produce the final forecast. This process can be represented by:

$$\phi_3 : (\tilde{\mathbf{i}}, P, S) \in \mathbb{R}^V \times \mathbb{R}^{H \times MF} \times \{0,1\}^{H \times S} \to \hat{X}_t \in \mathbb{R}^{H \times M} \tag{5.12}$$

From our experiments, we have found that a *Combiner* architecture consisting of three convolutional layers, two GRU layers, and two fully connected layers prior to the output layer provides reliable results across various scenarios. Hyperparameter Optimization (HPO) is applied to this *Combiner*, targeting specific hyperparameters such as learning rate, weight decay, and layer dimensions. Also in this case, Randomized Grid Search is used for this purpose. Additionally, a learning rate decay strategy, as used with AE, is also applied.

Final Remarks

In the domain of time-series forecasting, leveraging the data from exogenous variables can be beneficial. Within the ENCODE framework, these exogenous variables are taken into account: this integration is achieved by concatenating the exogenous variables with the target series, and then inputting this concatenated data into the AE stage. Consequently, the latent variables $\tilde{\mathbf{i}}$ encapsulate patterns not only from the target series but also from the exogenous series. Given that this data is used in both the individual prediction stages and the *Combiner*, the insights from the exogenous variables are carried forward through all stages of ENCODE.

Addressing seasonality is another critical aspect: the proposed method adheres to the approach recommended by [155]; specifically, a seasonal decomposition is performed for each detected seasonality and subsequently, the proportion of the seasonal component relative to the combined magnitude of the seasonal and error components are computed. Seasonalities with a ratio exceeding 0.05 are retained; practically, these are input into the *Combiner* as tensors.

Another important consideration is the potential non-stationarity of time-series data: by design, the ENCODE methodology is equipped to handle non-stationary data and, in particular in the initial phase, the AE, which is built upon GRUs, filters the series. Given the inherent capability of GRUs to manage non-stationarity, the data is aptly processed. This attribute is maintained in the individual predictors of the second phase, and the *Combiner*, which also incorporates GRUs. Thus, our methodology is adept at processing non-stationary time-series without compromising performance.

5.7.2 Experimental Setup

In the experiments, five datasets are used: three public and two private. Moreover, AutoCorrelation Function (ACF) and Partial AutoCorrelation Function (PACF) are exploited to select lagged values to be taken into account. The public datasets include:

- **Electricity**[4]: this dataset consists of daily power consumption from 370 customers between 2012 and 2014. The dataset, after a preprocessing stage, contains 320 series with 1096 observations. We break it down into *Training-P* (700 observations), *Training-C* (196 observations), and *Validation* and *Test* sets (100 observations each). The prediction lags are $1-8$, $3-8$, and $7-14$ for 1, 3, and 7-step forecasts. Weekly and annual seasonalities are considered. Figure 5.13 illustrates a sample time-series.

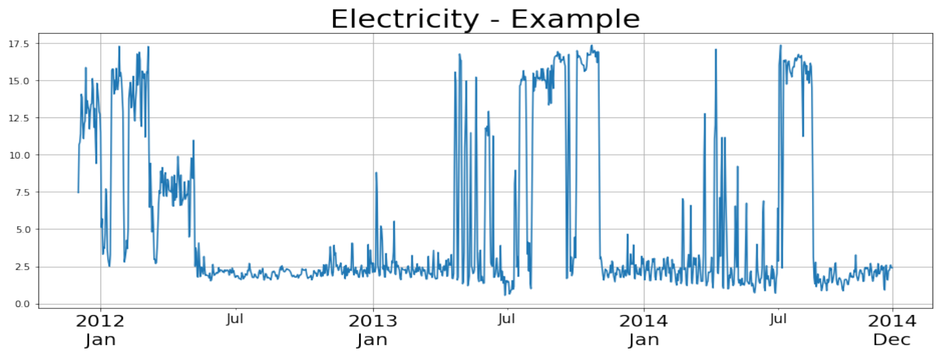


Figure 5.13: Sample from the Electricity dataset.

- **SST**[5]: This dataset features ocean surface temperatures from 67 buoys in the Pacific from 2000 to 2019. After filtering, it has 30 series with $2,556$ observations. The dataset uses the first $1,461$ samples as *Training-P*, the middle ones as *Training-C* and *Validation*, and the last 122 as *Test* sets. Forecasting considers 1, 3, and 7-steps ahead with prediction lags of $1-7$, $3-7$, and $7-13$. Figure 5.14 depicts a sample series.

- **PeMS**[6] ([264]): This other dataset contains data from California highways from March to May 2021. The dataset has 46 series with $1,463$ observations, divided into *Training-P* (840), *Test* (168), and the remainder for *Training-C* and *Validation*. Forecasts are 12, 24, and 36-steps ahead, with prediction lags of $12-38$,

[4]UCI Machine Learning Repository, `https://archive.ics.uci.edu/dataset/321/electricityloaddiagrams20112014`

[5]NOAA Tropical Atmosphere Ocean Project, `https://tao.ndbc.noaa.gov/`

[6]California Department of Transportation, `https://dot.ca.gov/programs/traffic-operations/mpr/pems-source`

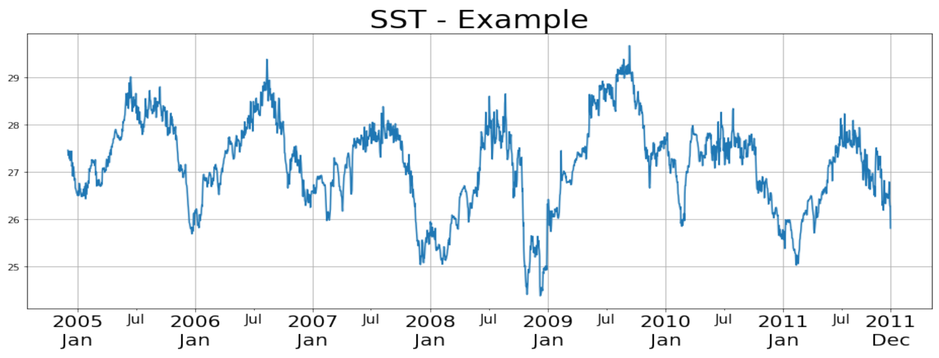


Figure 5.14: Sample from the SST dataset.

24 − 50, and 36 − 62, respecting daily and weekly seasonalities. A sample is shown in Figure 5.15.

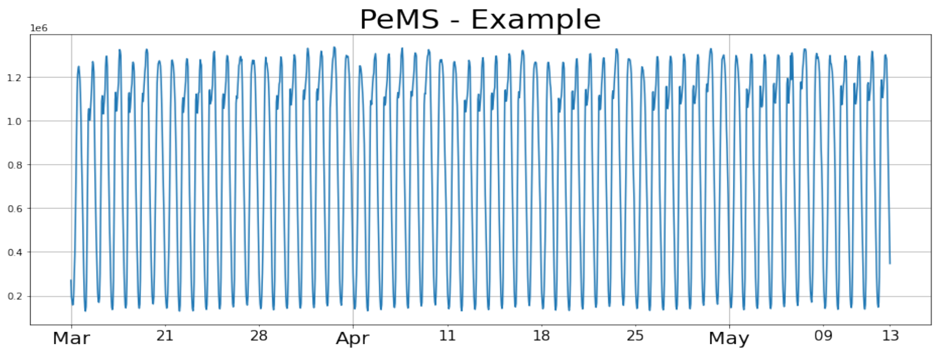


Figure 5.15: Sample from the PeMS dataset.

The two private datasets pertain to previously discussed applications:

- **Healthcare**: Holds daily hospital booking counts for allergy and pneumology in Campania, Italy, along with weather data. With 730 observations across 13 variables, the dataset is divided into *Training-P* (first 300 observations), *Test* (last 73 observations), and the remaining used for *Training-C* and *Validation* sets. Forecasts range from 1 − 7 and 1 − 14 steps ahead, with 7 endogenous and 21 exogenous lags. Yearly and strong weekly seasonality, due to weekends, are observed.

- **ToIT**: Features hourly parking occupancy rates. The 2,099 observations are allocated as *Training-P* (839), *Training-C* (924), *Validation* (168), and *Test* sets

(168). Forecasts span $1-24$ steps with 26 lags. The dataset shows daily and weekly seasonality and includes nine exogenous weather series for Naples and two Caserta districts.

The efficacy of our approach has been evaluated using the Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) as loss functions. A block Cross-Validation technique has been developed to consider the time-related nature of the dataset. For the RMSE, as an instance, the *Test* set is partitioned into Q segments while retaining the temporal sequence:

$$Test = Test_1 \uplus \cdots \uplus Test_Q \tag{5.13}$$

With $\uplus$ signifying disjoint union, and:

$$Test_q = \{t_{q,1}, \ldots, t_{q,n_q}\} \quad \forall q = 1, \ldots, Q \tag{5.14}$$

The error is subsequently assessed for every segment as:

$$RMSE_q = \sqrt{\frac{1}{n_q}\sum_{j=1}^{n_q} \|X_j - \hat{X}_j\|_2^2} \tag{5.15}$$

Where $X_j, \hat{X}_j \in \mathbb{R}^{H\times M}$ and $\|.\|_2$ denotes the element-wise l^2 norm. The resulting vector is:

$$\mathbf{RMSE} = [RMSE_1, \ldots, RMSE_Q] \in \mathbb{R}^Q \tag{5.16}$$

In this context, as evaluation metrics the mean (*Mean*), the standard deviation (*Std*), and their cumulative value (*Mean+Std*) which indicates the confidence interval's upper boundary have been adopted. The MAE follows an identical procedure. As concerns the cross-validation, 10 folds for Electricity, SST, and Healthcare have been considered, whereas for PeMS and ToIT, 7 folds due to the hourly data and one-week test set are used. This ensures each segment corresponds to a day. Prior to calculations, data is standardized to ensure uniform weights across all series during the performance assessment. The data is confined between 0 and 1, and scaling is applied only to the training and validation datasets, avoiding the use of future data points.

5.7.3 Results

A unified metric is introduced to facilitate a clearer comparison between different prediction techniques. Consider each dataset and each forecast horizon as a distinct test: for every test, the errors are standardized to ensure they range between 0 and 1. Let's denote the error of the j-th prediction method as ϵ_j. Its standardized version, $\hat{\epsilon}_j$, can be computed as:

$$\hat{\epsilon}_j = \frac{\epsilon_j - \min \epsilon}{\max \epsilon - \min \epsilon} \quad \text{for methods such as } LASSO, \textit{Ridge}, \text{ and so on.} \tag{5.17}$$

Subsequently, for each prediction technique, the average error over all the tests is computed. The outcomes of this computation are presented in Table 5.7, where the best result for each metric is highlighted in bold.

Discussions

From our earlier experimentation results, the resilience and effectiveness of our proposed model are evident: it consistently ranks at the top or nearly so based on our chosen evaluation criteria. Examining the Healthcare dataset, N-Beats stands out as the top performer, yet ENCODE provide results that are closely matched with it. In the PeMS dataset, ENCODE's performance metrics, especially for *Mean* and *Mean+Std*, are predominantly superior, with few exceptions: for instance, in the 12-step ahead prediction, its *Std* excels, and in the 24-step ahead forecast, the *Mean+Std* score is nearly the best. With the ToIT dataset, the outcomes are promising, as our model showcases the lowest error in both *Mean* and *Mean+Std* metrics.

However, a potential vulnerability surfaces when evaluating the Electricity dataset. ENCODE's performance is subpar across all measures for *Mean* and *Mean+Std.* Yet,

MODEL	**RMSE**			**MAE**		
	Mean	**Std**	**Mean+Std**	**Mean**	**Std**	**Mean+Std**
LASSO	0.3188	0.653	0.3712	0.1686	0.6940	0.2661
Ridge	0.2692	0.6304	0.3495	0.1496	0.6996	0.2715
Elastic Net	0.2834	0.5949	0.3576	0.1667	0.6588	0.2770
XGB	0.1937	0.3998	0.2347	0.0827	0.3763	0.1232
Random Forest	0.1738	0.3845	0.2109	**0.0640**	**0.3692**	**0.1002**
SVR	0.2670	0.4239	0.2748	0.4004	0.5069	0.4287
ARIMA	0.5019	0.3084	0.4895	0.3969	0.4544	0.4705
Mean	0.2058	0.4490	0.2574	0.1276	0.4888	0.2020
Median	0.2169	0.5184	0.2773	0.0785	0.5433	0.1712
PSO	0.2513	0.5487	0.2879	0.2118	0.6250	0.2950
Genetic	0.2507	0.4197	0.2889	0.1958	0.5099	0.2709
Random Walk	0.6924	0.4346	0.6857	0.5145	0.5530	0.5814
N-Beats	0.4908	0.6256	0.4982	0.4343	0.4972	0.4719
Prophet	0.3210	0.4380	0.3391	0.3581	0.4129	0.3876
BHT-ARIMA	0.7153	0.4806	0.7069	0.5416	0.5759	0.6188
MTGNN	0.1761	0.427	0.2177	0.0896	0.4095	0.1527
ENCODE	**0.1160**	**0.0959**	**0.0589**	0.2417	0.1515	0.1981

Table 5.7: Comparison between the different predictors. The values are obtained in this way: for each dataset and step ahead, we normalized the errors made by predictors and then averaged the normalized values. The normalization is done by scaling each error measure in each dataset and time horizon considered between 0 and 1. We can compare the losses obtained on different datasets in such a way. Finally, we exploit the same exogenous used by ENCODE also for all the baseline models that support exogenous variables.

it's worth noting that it consistently reports a very low *Std.* The dip in performance is likely attributable to the encoding phase: the intricacy of the input variable encoding prevents the predictors from recovering information from the original data. Table 5.8 contrasts the 1-step ahead RMSE metrics for regressors using encoded and non-encoded variables. It becomes evident that the encoding process in this scenario strongly impacts the predictive accuracy, leading to a performance downturn In the SST dataset,

MODEL	**No Enc**			**Enc**		
	Mean	**Std**	**Mean+Std**	**Mean**	**Std**	**Mean+Std**
LASSO	0.075	0.040	0.114	0.139	0.035	0.174
Ridge	0.074	0.039	0.113	0.128	0.039	0.167
Elastic Net	0.074	0.039	0.113	0.128	0.034	0.162
XGB	0.075	0.035	0.110	0.144	0.032	0.175
RFR	0.076	0.035	0.111	0.134	0.034	0.168
SVR	0.086	0.039	0.124	0.149	0.040	0.189

Table 5.8: Comparison between RMSE 1-step ahead results from basic regressors, with and without encoded inputs.

the results yielded by our approach are especially noteworthy. Errors associated with *RMSE* are significantly lower than our competitors, and the *MAE* results are comparably impressive. This can be primarily attributed to the data outliers, which significantly impact the results of auto-regressive techniques, causing their predictions to deviate considerably from the actual values during peaks and subsequent values. However, our model is less influenced by these outliers, due to the inclusion of the AE stage which mitigates peak effects. In essence, this dataset underscores the effectiveness of the AE (AutoEncoder) stage and emphasizes its pivotal role in enhancing resilience to outliers when decoupled from predictive stages. This phenomenon isn't just empirically evident but is also theoretically grounded: the AutoEncoder can be interpreted as a function $\phi_1 : \mathbf{i} \in \mathbb{R}^{MK} \to \tilde{\mathbf{i}} \in \mathbb{R}^V$. Simplifying the scenario with a single encoding layer, we have $\phi_1(\mathbf{i}) = tanh(W\mathbf{i})$ where $W \in \mathbb{R}^{V \times MK}$. With the encoder training relying on the convergence of back-propagation, it's reasonable to state that the matrix norm doesn't exceed 1, i.e., $|W| \leq 1$. The nature of the *tanh* function lends it a standardizing characteristic (considering its first derivative $tanh'(z) = \frac{4}{(e^z+e^{-z})^2} \leq 1$), which dampens the impact of outliers.

To summarize, these experimentation underscores a key insight: while some basic statistical models might occasionally outperform ENCODE, the proposed framework finds its best application when dealing with time-series characterized by pronounced patterns like strong seasonalities or sporadic anomalies. It's also pivotal to note that while ENCODE demands significant computational resources during the training phase, its testing phase is computationally efficient, especially when benchmarked against the granularity and predictive spans of the examined time-series.

Statistical Comparison

In order to delve deeper into the efficacy of the proposed methodology, a rigorous statistical analysis has been conducted with the aim of discerning whether the average error measures of the ENCODE framework is demonstrably lower than the counterparts. To this aim the *corrected paired Student's t-test* [265]. This test is designed to compare the error vectors across the distinct folds, specifically the **RMSE** as discussed before. The foundational hypothesis, or the null hypothesis, postulates that the mean vector of errors from ENCODE, denoted as $\mathbf{RMSE}^{ENCODE}$, does not exhibit a significant reduction when compared with the competitor's $\mathbf{RMSE}^{Comp}$. Contrarily, the alternative hypothesis posits that $\mathbf{RMSE}^{ENCODE} < \mathbf{RMSE}^{Comp}$. It is noteworthy that the standard deviation utilized in the test statistic is refined to account for the inherent correlation amongst the errors.

The derived p-value from the test pertinent to the RMSE, across all algorithms, datasets, and predictive horizons, is elucidated in Table 5.9. To augment clarity and facilitate enhanced visual discernment, p-values less than 0.05 have been underscored, while those below 0.01 have been emboldened. As depicted in the table, ENCODE demonstrates marked efficacy on the SST, PeMS, and ToIT datasets, consistent with observations in previous discussions. However, the SST dataset exhibits an unusual characteristic: even though ENCODE's mean value and standard deviation

Dataset	**Electricity**			**SST**			**PeMS**			**Healthcare**		**ToIT**
Steps	**1**	**3**	**7**	**1**	**3**	**7**	**12**	**24**	**36**	**7**	**14**	**24**
LASSO	0.9999	1.0	0.9948	**0.0013**	**0.0012**	**0.0012**	**0.0**	**0.0009**	**0.0001**	0.8447	0.9891	**0.0**
Ridge	0.9998	0.9998	0.9835	**0.0012**	**0.0011**	**0.0012**	**0.0**	**0.0014**	**0.0001**	0.2432	0.0629	0.0152
Elastic Net	0.9999	1.0	0.9925	**0.0012**	**0.0011**	**0.0012**	**0.0**	**0.0013**	**0.0001**	0.2555	0.065	**0.0002**
XGB	1.0	0.9988	0.9736	**0.0012**	**0.001**	**0.0011**	**0.0028**	0.0747	**0.0001**	0.0269	0.021	0.0164
Random Forest	1.0	0.9987	0.9432	**0.0012**	**0.001**	**0.0011**	**0.0004**	0.4471	**0.0007**	0.6624	0.9169	0.0741
SVR	0.9722	0.9753	0.6771	**0.0008**	**0.0007**	**0.0008**	**0.0018**	0.0135	**0.0002**	0.3607	0.0374	0.0406
ARIMA	0.22	0.9891	0.9662	**0.0016**	**0.0014**	**0.0015**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0004**
Mean	1.0	1.0	0.9968	0.0103	**0.0088**	**0.0098**	**0.0**	**0.0049**	**0.0**	0.6757	0.4361	**0.0031**
Median	1.0	1.0	0.9988	**0.0015**	**0.0014**	**0.0016**	**0.0001**	0.0243	**0.0001**	0.8066	0.8351	0.0147
PSO	0.0544	0.1618	0.0176	**0.0018**	**0.0008**	**0.0**	**0.0**	**0.0**	**0.0**	0.0137	0.2246	**0.0002**
Genetic	0.2972	0.0376	0.0569	**0.0024**	**0.0017**	**0.0**	**0.0**	**0.0014**	**0.0**	**0.0056**	**0.0003**	**0.0007**
Random Walk	0.1098	**0.0**	0.9992	0.0512	0.0432	0.0367	**0.0**	0.0938	**0.0**	**0.0003**	**0.0**	**0.0**
N-Beats	0.1404	0.0939	0.074	0.0267	0.0165	**0.0031**	**0.0011**	0.2495	**0.0004**	0.9924	0.9984	**0.0**
Prophet	0.9228	0.9958	0.9905	0.0237	0.0206	0.0209	**0.0027**	0.0313	**0.0021**	**0.002**	**0.0016**	0.2012
BHT-ARIMA	**0.0001**	**0.0001**	0.0481	0.0509	0.0424	0.0393	**0.0**	**0.0**	NaN	**0.0002**	**0.0001**	**0.0**
MTGNN	1.0	0.9999	0.9994	0.0363	0.0336	0.0308	0.2829	0.4755	**0.0088**	0.4962	**0.0068**	**0.0011**

Table 5.9: Statistical test p-values when considering RMSE as the loss function. Values lower than 0.05 and 0.01 are underlined or reported in bold, respectively.

are markedly better than other predictors, the statistical test does not robustly reject the null hypothesis, especially when compared to certain algorithms. This anomaly can be attributed to the significant presence of outliers, which magnify the variance of predictors and thereby skew the test results.

5.7.4 Computational Time Comparison

Upon presenting the outcomes in terms of precision and robustness, a concise examination of computational times is briefly provided. Experimental settings for this analysis encompassed the Electricity, SST, and PeMS datasets, on the following configurations:

- For the Electricity and SST datasets: Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz, equipped with 128 GiB RAM and a GeForce RTX 3070.
- For the PeMS dataset: Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, complemented with 16 GiB RAM and a GeForce GTX 970.

It is worth underlining that discrepancies in the hardware architectures are solely for the facilitation of parallelization and bear no implications on the impartiality of the experiments. Outcomes are delineated in seconds, as depicted in Fig. 5.16. For enhanced clarity and legibility, the durations against the maximal time have been normalized while also detailing the absolute maximum durations. It's evident that both the op-

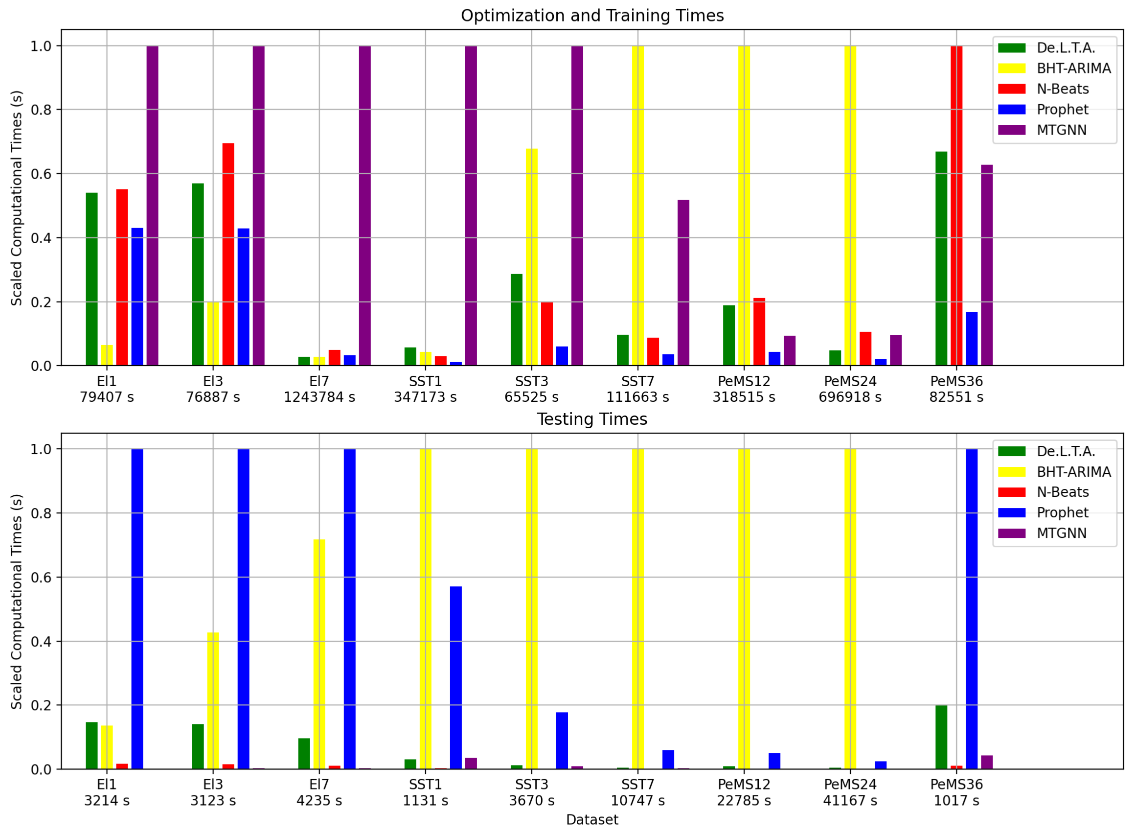


Figure 5.16: Chronological Analysis of Computational Durations. Timeframes are normalized against the peak duration for every dataset and forecast horizon. Below each dataset, the peak duration is furnished to provide an insight into the requisite computation times.

timization and training durations are often high due to the intricate structure of the *Combiner*, with the optimization phase being especially complex. It is noteworthy that ENCODE's computational times are more prolonged for datasets containing numerous time-series compared to those with a high number of observations. Nevertheless, while the optimization and training phases may be time-intensive, the prediction phase is remarkably fast. It's crucial to highlight that, from an industry perspective, prediction speed is paramount since only this phase integrates into the production cycle. Conversely, the model's training occurs offline and doesn't impede ongoing production processes.

5.8 Conclusions

The research works discussed in this chapter lay the foundation for the design and implementation of ENCODE, a pioneering ensemble approach for predicting multivariate time-series. Originating from the concept of intelligently amalgamating various predictions via a Neural Network, and navigating real-world challenges, the end product is a sophisticated framework. ENCODE integrates both Machine Learning (ML) and Deep Learning (DL) techniques, adeptly recognizing patterns within multivariate time-series to enhance the robustness and precision of its forecasts. This assertion is further corroborated by comprehensive experimental results.

A defining feature of ENCODE, which straddles the domains of both hybrid and ensemble forecasting methodologies, is its data regularization effect. This positions it as especially adept when handling time-series affected by outliers. Such resilience to anomalies can be credited to the encoding phase, which mitigates the impact of anomalous input values. Moreover, on datasets abundant with observations, ENCODE consistently outperforms many established methods in terms of forecasting accuracy. This suggests that the Hybrid Neural Network, when adequately trained, can effectively synchronize patterns drawn directly from data with insights generated by the regression models.

Additionally, the framework has evolved from its original conception in such a way both long-term and short-term predictions can be provided, rendering it apt for real-world scenarios where varied forecasting horizons are often a prerequisite.

Part II

Advanced Concepts on Learning Dynamics

Chapter 6

Advanced Perspectives and Challenges in Neural Learning Dynamics

6.1 Introduction

Advancements in neural network architectures necessitates a comprehensive exploration of learning mechanisms that extend beyond the foundational principles outlined in the preceding chapter. As the architectures evolve in complexity and depth, optimization and training methods become increasingly multifaceted, warranting a rigorous mathematical investigation. This chapter is devoted to an in-depth examination of such advanced topics that have profound implications for neural learning and optimization. Traditional optimization techniques like backpropagation, although widely adopted, represent only one category of methods within a broad landscape of optimization approaches. Many of these alternative methods originate from diverse branches of mathematics, computational science, and engineering. For instance, reinforcement learning offers optimization techniques that focus on decision-making in stochastic environments, while evolutionary algorithms provide heuristic-based optimization strategies inspired by natural selection processes.

Beyond the realm of optimization, other key factors also play a critical role in the efficiency and efficacy of neural network training. This includes an understanding of the dynamics of learning rates, the importance of weight initialization schemes, and the need for effective data preprocessing techniques. Recent innovations have added yet another layer of complexity to neural network training: transfer learning and meta-learning paradigms, for example, offer new avenues for improving network generalization by leveraging pre-trained models and by enabling networks to learn the learning process itself, respectively.

Through detailed analysis and conceptual discussion, this chapter aims to broaden the understanding of the state-of-the-art methodologies employed in modern neural network training and to elucidate the mathematical underpinnings that support these techniques.

6.2 The Role of Weight Initialization

In machine learning, particularly in the context of neural networks, initialization might initially seem like a rudimentary setup step, almost an afterthought compared to the sophistication of optimization algorithms or the architecture design [3]. However, as neural networks have evolved in depth and complexity, so has the understanding of the role that weight initialization plays in their performance [40]. It has become increasingly clear that the initialization of network weights is far from trivial; rather, it is a crucial step that can profoundly impact the learning dynamics and the ultimate efficacy of the network [13]. An improper initialization can lead to a host of problems, ranging from slow convergence to model instability, that can render the training process ineffective [84]. On the other hand, a well-thought-out initialization strategy can accelerate training, improve generalization, and make the network robust to various challenges.

6.2.1 Fundamental Objectives of Weight Initialization

The process of weight initialization serves multiple core objectives, each of which can have far-reaching implications on the performance and reliability of the neural network [3]. From breaking the symmetry among neurons to avoiding the pitfalls of vanishing and exploding gradients, weight initialization strategies aim to tackle several challenges inherent in the training of neural networks [13]. Understanding these objectives is not just academic; it can provide invaluable insights in training complex neural networks for real-world applications [84].

Symmetry Breaking

One of the fundamental objectives of weight initialization is to break the inherent symmetry that can arise in neural layers. If all neurons in a given layer were initialized with identical weights, during the forward and backward passes of training, they would undergo identical updates. Mathematically, this symmetry is exhibited when each neuron in a layer responds identically to incoming inputs, making them redundant. This issue exacerbates with depth, essentially leading to layers of replicated neurons that fail to capture diverse features from the data. The implications of such symmetry are profound: the capacity of the network remains largely underutilized as multiple neurons effectively learn the same features. Proper initialization introduces slight asymmetries that help neurons diversify during training, allowing the network to explore a broader functional space and better capture the underlying data distribution [13].

Avoiding Vanishing/Exploding Gradients

Deep neural architectures, especially when using certain activation functions, are susceptible to the notorious vanishing and exploding gradient problems. The essence of these issues lies in the chain rule of differentiation used in backpropagation.

Mathematically, when considering a deep feed-forward neural network with L layers, the gradient of the loss with respect to the weights in the initial layers can be expressed as a product of several terms. If these terms are consistently less than one in magnitude, the gradient diminishes exponentially with depth; conversely, if they're greater than one, it explodes. This phenomenon drastically hinders learning in the initial and final layers of the network. Modern initialization techniques, such as Xavier (Glorot) initialization, aim to maintain a balance. For instance, in the case of the Xavier initialization, weights are drawn from a distribution with variance $\frac{1}{n}$, where n is the number of input units to a neuron. This variance ensures that the activations and gradients do not exhibit high variance, promoting stable learning across layers [40].

Accelerated Convergence

The choice of initial weights can influence the trajectory that the optimization algorithm takes in the parameter space. Well-chosen initializations place the weights in regions of the loss landscape where the descent towards minima is smoother and more direct. In contrast, poor initializations might land the weights in regions fraught with saddle points or steep curvature, slowing convergence.

Furthermore, from a geometric perspective, if weights are initialized such that the loss is in a relatively flat region, gradients are small, causing slow progress. On the other hand, if the initialization is in a steep region, gradients can be large, leading to oscillations and potential divergence unless countered by a small learning rate. A balanced initialization ensures a faster, more stable descent towards optima, economizing on the computational budget and time [84].

6.2.2 Types of Weight Initialization

As said, the selection of a specific weight initialization technique is a crucial step that can have a tangible impact on the performance of a neural network [13, 40, 84]. Different initialization strategies are designed to work optimally under various conditions, such as the choice of activation functions or the architecture of the neural network.

Zero Initialization

Initializing all weights to zero seems like a straightforward strategy, but it is fraught with pitfalls. The primary issue is that of symmetry: initializing all weights to zero implies that every neuron in each layer will produce the same output during the forward pass and receive the same gradient during the backward pass. Mathematically, this can be seen by considering the weight update equation, neglecting the biases, $\Delta W = -\alpha \nabla \mathcal{L}(W)$, where $\mathcal{L}(W)$ is the cost function. Since the gradient $\nabla \mathcal{L}(W)$ is the same for all neurons with zero-initialized weights, ΔW will be the same for all neurons, effectively rendering them redundant.

Random Initialization

Random initialization involves drawing initial weight values from a probability distribution, such as a uniform or Gaussian distribution. The most commonly used distributions are:

$$W \sim \mathcal{N}(0, \sigma^2) \quad \text{or} \quad W \sim \mathcal{U}(-a, a) \tag{6.1}$$

Mathematically, the distributions can be parameterized to control the spread of initial weight values; this spread influences the gradient descent steps, especially during the early stages of training. The choice of standard deviation σ in Gaussian initialization or the range a in uniform initialization can thus affect convergence speed and the quality of the final model [84].

Xavier (Glorot) Initialization

Xavier initialization, also known as Glorot initialization, is particularly suitable for networks with sigmoid activation functions. The weights are initialized as:

$$W \sim \mathcal{N}\left(0, \frac{1}{n_{\text{in}}}\right) \tag{6.2}$$

Xavier initialization controls the variance of initial weights based on the layer size. The strategy minimizes the risk of vanishing/exploding gradients by maintaining the variance of activations across layers. This ensures that $Var(y) = Var(x)$, where $y = Wx + b$, resulting in a balanced training process [13].

He Initialization

He initialization is designed for ReLU activation functions. Similar to Xavier, it initializes the weights but adjusts the variance according to the layer size:

$$W \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}}\right) \tag{6.3}$$

He initialization extends Xavier initialization to ReLU activation functions. Mathematically, the factor of $\frac{2}{n_{\text{in}}}$ in the variance aims to account for the zero half of the ReLU function, thus helping to mitigate the dying ReLU problem. The result is a faster and more effective training process for networks with ReLU activations [40].

LeCun Initialization

LeCun initialization is optimized for Scaled Exponential Linear Units (SELUs). It is defined as:

$$W \sim \mathcal{N}\left(0, \frac{1}{n_{\text{in}}}\right) \tag{6.4}$$

LeCun initialization offers a solution for networks that employ Scaled Exponential Linear Units (SELUs), aiming to maintain a mean of zero and unit variance across layers. This choice of initialization becomes increasingly important when dealing with deep architectures, as it helps maintain stable gradients [2].

Orthogonal Initialization

Orthogonal initialization involves setting the weight matrix W as an orthogonal matrix. This initialization is beneficial when the network has more complex recurrent connections. Mathematically, an orthogonal matrix is a square matrix whose rows are mutually orthonormal unit vectors: this property helps in preserving the gradient's norm over time, especially when the network has deep or recurrent layers [266].

6.2.3 Practical Considerations and Guidelines

The task of selecting an appropriate weight initialization strategy goes beyond theoretical considerations; the choice is far from trivial and is dictated by multiple factors, including the architecture, depth, and activation functions employed in the network.
Different activation functions have distinct characteristics. For instance, networks utilizing ReLU-based activation functions are often better initialized with He initialization [40]. In contrast, sigmoid or hyperbolic tangent (tanh) activations usually benefit more from Xavier initialization [13]. The depth of the network further complicates the issue. Deeper architectures are particularly susceptible to the challenges of vanishing or exploding gradients, making He or Xavier initializations, which aim to balance the variances during forward and backward passes, more appropriate choices [13, 40].
When dealing with specialized architectures such as recurrent neural networks (RNNs) or convolutional neural networks (CNNs), the initialization strategy may require additional considerations: for RNNs, the challenge is often in preserving the gradient's magnitude across sequences during backpropagation through time, so orthogonal initialization is a strategy commonly employed in this context [266]. By initializing the recurrent weight matrix as an orthogonal matrix, the dot product between any two different rows or columns is zero, effectively preventing the gradients from diminishing or exploding too quickly. In the realm of CNNs, while He or Xavier initializations are commonly applied, some specialized initializations focus on preserving the rotational and translational invariances in image data. Such initializations may involve filter-based or spatial arrangements of the weights [40].
However, empirical validation remains a critical aspect of selecting an initialization strategy: sometimes, regardless of the theoretical merits of an initialization method,

the real measure of its effectiveness lies in its practical performance on a specific problem. In such cases, fine-tuning the initialization can be guided by hyperparameter optimization algorithms, such as grid search or random search, to arrive at the most effective initial setup [84]. Another aspect to consider is that, while adaptive learning rate methods like Adam ease the sensitivity of the training dynamics to the choice of initialization, they are not a panacea. Adam adjusts the learning rates during training, but this adaptivity doesn't obviate the need for a sound initialization scheme [16]. Poor initialization can still lead to slow convergence or get the optimization stuck in suboptimal solutions, underscoring the enduring importance of careful initialization [16]. Furthermore, the choice of initialization should be made in the context of other architectural and training considerations such as regularization techniques. For example, using larger initial weights may offer quicker convergence, but at the risk of overfitting. To counteract this, stronger regularization methods like dropout or L_2 weight decay may be required [92]. This creates a nuanced interplay between initialization and regularization, demanding careful tuning to optimize both the bias-variance tradeoff and the computational efficiency [92].

6.3 Alternative Optimization Techniques in Neural Networks

Optimization lies at the heart of training neural networks. Since the resurgence and mainstreaming of neural networks in the 1980s and 1990s, backpropagation has emerged as the de facto standard for updating network weights and reducing prediction errors. Its effectiveness in chain-rule-based gradient computation and suitability for gradient descent optimization has cemented its role in the annals of deep learning history. However, the increasingly complex and diverse landscape of applications for neural networks, coupled with the non-convex nature of many loss landscapes, has necessitated the exploration of alternative optimization strategies that seek to address some of the inherent challenges posed by backpropagation. Moreover, the sheer scale and dynamism of modern data have also prompted the search for techniques that can adapt in real-time, scale efficiently, or navigate the intricate topologies of high-dimensional loss surfaces. From the probabilistic explorations of reinforcement learning to the bio-inspired heuristics of swarm intelligence and the evolutionary pressures simulated by genetic algorithms, a spectrum of strategies that not only challenge the traditional paradigms of neural network optimization but also enrich its repertoire exists.

6.3.1 Reinforcement Learning-based Techniques

Reinforcement Learning (RL) fundamentally diverges from traditional supervised learning paradigms: at its core, RL is concerned with the problem of an agent learning by interacting with an environment, making sequential decisions to achieve a certain objective. While supervised learning is fed with explicit input-output pairs, RL thrives on exploration, discovery, and feedback loops. The agent in RL is not provided with the "correct" action at each step but rather discovers it through a process of trial and error. This trial and error are guided by rewards (or penalties) given by the environment. Over time, the agent learns a strategy or a policy to take actions that maximize its cumulative reward. This reward-driven framework offers a natural ground for optimization, as the agent is essentially navigating an optimal path in a vast space of possibilities [267].

Policy Gradient Methods

Among various techniques in RL, Policy Gradient (PG) methods [268] have gained considerable attention due to their efficacy in directly optimizing decision-making policies. While other methods might focus on estimating value functions or Q-functions to indirectly arrive at an optimal policy, PG methods cut through the chase, targeting the policy itself. In PG, the policy, denoted by $\pi_\theta(a|s)$, is parameterized by θ. It provides a mapping from states s to actions a, typically in the form of a probability distribution. The core concept of PG is that by modifying the parameters θ, we can steer the policy towards more rewarding actions. The objective function in PG, $\mathcal{L}(\theta)$, captures the expected return (cumulative reward) when following the policy π_θ. The gradient of this objective, $\nabla_\theta \mathcal{L}(\theta)$, points in the direction of the steepest increase in expected return. By adjusting the policy parameters in the direction of this gradient, the policy iteratively improves.

The gradient expression:

$$\nabla_\theta \mathcal{L}(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) R_t] \tag{6.5}$$

is particularly insightful. Here, $\nabla_\theta \log \pi_\theta(a|s)$ denotes the gradient of the logarithm of the policy (often referred to as the log-probability of the action). R_t represents the reward. The product of these terms essentially weighs the log-probability of an action by its associated reward, guiding the updates towards actions that accrue higher rewards [269]. This mechanism ensures that actions leading to better outcomes become more probable in subsequent interactions, refining the agent's strategy over time. The elegance of policy gradients lies in their adaptability: they are applicable to both discrete and continuous action spaces and have proven effective in complex scenarios, from robot locomotion to game playing.

Q-learning and Value-based Methods

Q-learning [270] is a quintessential off-policy algorithm within the Reinforcement Learning domain. Unlike policy-based methods that seek to learn a policy directing the agent's behavior, value-based methods, as the name suggests, emphasize learning the value of each action in each state. The core principle is that if the value of each action is known, then the optimal policy can be directly derived by choosing the action with the highest value at every state.

The fundamental object of interest in Q-learning is the Q-function, represented as

$Q(s, a)$. This function quantifies the expected return or cumulative discounted future reward of taking action a in state s. The primary objective of Q-learning is to learn this function. The term “Q” is derived from the word “quality”, indicating the quality of a certain action in a specific state. Q-learning is fundamentally a form of temporal difference learning, where the difference between the predicted Q-value and the updated estimate forms the basis for learning. The agent uses its current knowledge to estimate the expected reward of an action in a state and continuously updates this estimate as it gains more experience. Deep Q-learning, a recent and popular variation, uses deep neural networks as function approximators to generalize the learning across vast and continuous state spaces. This approach overcomes the limitations of traditional table-based Q-learning, which is only feasible for problems with limited and discrete state and action spaces.

For a neural network approximator $Q(s, a, \theta)$ with weights θ, the Q-learning update rule is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{6.6}$$

Here, r_{t+1} symbolizes the reward for the current action, and γ represents the discount factor, which determines the present value of future rewards. A higher γ indicates that the agent values future rewards more highly relative to immediate rewards. The term α is the learning rate, determining the step size or weight of the update.

Deep Q-learning, due to its ability to handle high-dimensional state spaces and generalize learning, has facilitated the successful deployment of RL in numerous complex tasks, most notably in game playing scenarios like the ones demonstrated by DeepMind with the game of Go and various Atari games [271].

6.3.2 Evolutionary Algorithms and Genetic Algorithms

The quest for efficient training and optimization methods in neural networks has led researchers to explore various strategies, diverging from the traditional gradient-based methods such as backpropagation. One of the remarkable avenues that has garnered interest in this domain is the adoption of heuristic optimization techniques inspired by natural evolutionary processes. Specifically, Evolutionary Algorithms (EAs) and their subset, Genetic Algorithms (GAs), offer a stochastic, population-based approach to optimization that contrasts sharply with the deterministic nature of backpropagation.

Foundational Concepts of Evolutionary Algorithms

Evolutionary Algorithms (EAs) encapsulate techniques inspired by biological evolution to find approximate solutions to optimization and search problems [210]. Central to EAs is the iterative progression of a population of candidate solutions towards improved regions in the search space. Three primary operators drive this evolution: natural selection, crossover (recombination), and mutation.

Natural Selection Natural selection serves as the steering mechanism, ensuring the persistence of fit individuals and the culling of the less fit. The likelihood of an individual being selected is proportional to its fitness, which is typically defined by a problem-specific fitness function, f [272]. Given a population P of size n and an individual $x \in P$, the probability $p_{\text{select}}(x)$ of selecting x can be expressed as:

$$p_{\text{select}}(x) = \frac{f(x)}{\sum_{i=1}^{n} f(x_i)} \tag{6.7}$$

Where x_i is the i^{th} individual in the population.

Crossover (Recombination) Crossover is a mechanism to generate new offspring by combining the genetic information of two parent individuals [273]. This promotes the propagation of good genetic material within the population. For binary-encoded individuals, a common crossover technique is the one-point crossover [274]. Given parents $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$, a random crossover point k is chosen. The offspring are then created as:

$$\begin{aligned}\text{offspring}_1 &= (x_1, x_2, \ldots, x_k, y_{k+1}, \ldots, y_n)\\ \text{offspring}_2 &= (y_1, y_2, \ldots, y_k, x_{k+1}, \ldots, x_n)\end{aligned} \tag{6.8}$$

Mutation Mutation injects randomness into the population by slightly altering the genetic material of individuals [275]. For binary representations, this might involve flipping a bit. Given an individual $x = (x_1, x_2, \ldots, x_n)$ and a mutation probability p_m, each gene x_i is mutated with probability p_m [276]. For a binary representation:

$$x'_i = \begin{cases} 1 - x_i & \text{with probability } p_m \\ x_i & \text{otherwise} \end{cases} \tag{6.9}$$

Where x' is the mutated individual.

The combination of these operators allows EAs to explore vast search spaces effectively [277]. While the stochastic nature of EAs doesn't guarantee convergence to a global optimum, it does offer resilience against becoming trapped in local optima, making it a potentially robust alternative for neural network optimization, especially in non-convex landscapes or when gradient information is sparse or unreliable [278].

Genetic Algorithms: A Specialized Evolutionary Approach

Genetic Algorithms (GAs) constitute a particular category of Evolutionary Algorithms, distinguished primarily by their structured solution encoding, referred to as chromosomes. Each chromosome is an encoded version of a potential solution to the optimization problem. While EAs could have a variety of representations for individuals, GAs often use more intricate encoding schemes.

1. **Encoding:** Chromosomes in GAs can adopt complex data structures, ranging from binary strings to real numbers or even trees, depending on the problem at

hand. This flexible encoding mechanism enables GAs to solve a broad array of optimization problems. For instance, a chromosome c can be represented as:

$$c = (g_1, g_2, \ldots, g_n) \tag{6.10}$$

where g_i is a gene in the chromosome.

2. **Selection Strategies:** Unlike the broader category of EAs, GAs often use sophisticated selection mechanisms, such as roulette wheel or tournament selection, which are designed to preserve genetic diversity while ensuring the survival of the fittest individuals.

3. **Specialized Crossover and Mutation:** GAs often implement specialized versions of crossover and mutation operations tailored to their unique chromosome structures. These might involve multiple-point crossovers or non-uniform mutations, among others.

The key strength of GAs lies in their balance between exploration and exploitation, which makes them especially effective for tackling optimization problems characterized by rugged search spaces. This capability enables GAs to navigate around local optima, offering a robust alternative for neural network optimization [210, 276].

Application to Neural Network Training

Training neural networks involves minimizing a loss function to tune the network's parameters, a process fraught with optimization challenges such as local minima and saddle points. Traditional gradient-based methods like backpropagation often struggle with these intricacies [3]. EAs and GAs present an alternative optimization paradigm. Their population-based methodology enables a broader exploration of the solution landscape, thus increasing the probability of finding diverse and robust solutions [210]. Unlike gradient-based methods, which can get trapped in local optima, the stochastic and evolutionary nature of EAs and GAs allows for more effective traversal of complex optimization landscapes [279]. Additionally, the inherent randomness in EAs and GAs provides a form of regularization, mitigating the risk of overfitting. This effect can be viewed as an implicit form of ensemble learning [275]. GAs offer additional flexibility through their versatile encoding schemes for neural network parameters, which can be tailored to specific architectural needs [280]. While EAs and GAs typically require more computational resources, their inherent parallelism is well-suited for modern computational architectures, like GPUs or distributed systems, making them increasingly practical for large-scale neural network training [281, 282].

6.3.3 Swarm Intelligence-based Techniques

Swarm intelligence is a subfield of artificial intelligence inspired by the collective behavior of decentralized natural systems, such as flocks of birds or schools of fish. These algorithms model simple agents interacting with their environment and other agents, leading to emergent, complex behaviors that can be harnessed for optimization tasks, including training neural networks [283].

Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is an iterative optimization algorithm whose fundamental concept is to have a swarm of particles where each particle represents a potential solution in a multidimensional space – in this context, a particular configuration of neural network weights [241].

Particle Swarm Optimization operates on the premise of collective behavior where each particle in the swarm represents a point in a multidimensional solution space. A particle's behavior is influenced by its own best-known position, denoted as *pbest*, and the best-known position in its neighborhood, termed as *gbest*. The neighborhood can be defined in various ways, such as a fixed topological structure or dynamically based on particle proximity.

The algorithm involves a two-step update rule for each particle: updating the velocity and then updating the position. The velocity $v_i(t)$ of the i^{th} particle at iteration t is updated according to:

$$v_i(t+1) = w \cdot v_i(t) + c_1 \cdot rand() \cdot (pbest_i - x_i(t)) + c_2 \cdot rand() \cdot (gbest - x_i(t)) \tag{6.11}$$

Here, w is the inertia weight that controls the impact of the particle's previous velocity. The parameters c_1 and c_2 are cognitive and social scaling coefficients, respectively, which moderate the influence of the personal and global best positions. The term $rand()$ is a random number drawn between 0 and 1, introducing stochasticity into the system. Following the velocity update, the position $x_i(t)$ is updated as:

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{6.12}$$

This two-step process encapsulates the core of PSO's iterative search mechanism.

When applied to neural networks, each particle in the PSO algorithm represents a unique set of network weights and biases. The fitness function, often corresponding to a loss metric like cross-entropy or mean squared error, is used to evaluate the quality of these weights. PSO's stochastic and population-based nature allows it to explore a wide range of configurations, effectively circumventing local minima in non-convex optimization landscapes commonly encountered in deep learning [284].

Despite its benefits, PSO is not without drawbacks. Like other heuristic methods, PSO can be computationally expensive, requiring the evaluation of the fitness function for each particle in the swarm at every iteration. This could become prohibitive for large-scale problems or complex neural network architectures. Additionally, the algorithm's performance is sensitive to the choice of hyperparameters, including the inertia weight w and the cognitive and social coefficients c_1 and c_2. However, its inherent parallelism [241] makes it amenable to acceleration via modern hardware like GPUs, thus partially mitigating the computational cost.

6.3.4 Other Alternative Methods

The quest for optimization methods to efficiently train neural networks is an ongoing area of rigorous exploration and development. While reinforcement learning, evolutionary algorithms, and swarm intelligence offer robust alternatives to traditional gradient-based methods like backpropagation, they represent only a fraction of the diverse techniques that have been investigated. This expansive landscape of alternative methods draws from various mathematical and computational paradigms, each contributing its unique set of advantages, drawbacks, and applicability conditions. Some of these techniques are designed to tackle specific challenges—such as the high computational costs, susceptibility to local minima, or lack of parallelization—that are associated with traditional methods. Others offer a radical departure from established practices, leveraging principles from fields as diverse as statistical physics, operations research, and dynamical systems theory to achieve efficient optimization.

Hessian-Free Optimization

Hessian-Free Optimization (HFO) is a second-order optimization technique that circumvents the need for explicitly computing the Hessian matrix H. Instead, it approximates the matrix-vector product Hv iteratively, often using techniques like conjugate gradients. This is particularly advantageous for deep networks, where computing the full Hessian could be computationally infeasible. The update rule for HFO can be given by:

$$w_{t+1} = w_t - \alpha H^{-1} \nabla \mathcal{L}(w_t) \tag{6.13}$$

Here, $\mathcal{L}$ is the cost function, and α is the learning rate [285].

Conjugate Gradient

The Conjugate Gradient (CG) method addresses some of the limitations of first-order methods like gradient descent by incorporating curvature information into the optimization process. In CG, the search directions are conjugate to each other with respect to the Hessian, which often leads to faster convergence. The update rule for CG is:

$$w_{t+1} = w_t + \alpha_t d_t \tag{6.14}$$

where α_t is the step size, and d_t is the conjugate direction [286].

Sparse Coding

Sparse coding focuses on learning a sparse representation of the input data and has been used as an unsupervised alternative to backpropagation for neural network training. In mathematical terms, given an input x, the objective is to find a sparse code z such that $x \approx Dz$, where D is the dictionary matrix [287].

Random Search

Random Search is a simple yet effective technique for hyperparameter optimization. Although it doesn't replace the actual training algorithm, it competes well in the hyperparameter space. Given a set of hyperparameters Θ, Random Search samples $\theta \sim \Theta$ uniformly at random and evaluates the performance [90].

Simulated Annealing

Inspired by metallurgy, Simulated Annealing (SA) employs probabilistic mechanisms to escape local minima. The algorithm replaces the current solution x with a random neighboring solution x' with probability $p(x, x', T)$, where T is the "temperature", often modeled as:

$$p(x, x', T) = \exp\left(\frac{f(x') - f(x)}{T}\right) \tag{6.15}$$

This ability to escape local minima makes SA versatile for global optimization tasks [288].

Coordinate Descent

Coordinate Descent (CD) involves optimizing one coordinate (or a subset of coordinates) at a time while keeping others fixed. Mathematically, the algorithm iteratively minimizes the objective function f along one coordinate i as:

$$w_i = \arg\min_{w_i} f(w) \tag{6.16}$$

CD can be particularly effective in scenarios where each individual coordinate-wise optimization is computationally more tractable than the full-dimensional optimization [289].

6.3.5 Future Prospects and Integrative Approaches in Neural Network Optimization

The increasing intricacy of neural network architectures and their growing range of applications have highlighted the limitations of relying solely on traditional gradient-based methods like backpropagation [3]. This evolving landscape has given rise to a rich tapestry of alternative optimization techniques, each with unique advantages and potential drawbacks.

Emerging trends point toward hybrid models that meld gradient-based optimization with alternative methods, such as reinforcement learning or evolutionary algorithms. These integrative approaches have opened new horizons in tasks like game playing through Deep Reinforcement Learning [271] and neural architecture search [282]. Moreover, they offer several advantages. First, they allow for a more versatile exploration of the solution space by leveraging both stochastic and deterministic methods. Second, their multi-faceted nature can lead to greater robustness, particularly when neural networks operate in non-stationary environments or face adversarial threats. Finally, these hybrid approaches are especially adept at handling the complex and high-dimensional optimization problems that are now commonplace in modern deep learning tasks. However, the benefits of such integrated optimization strategies are not without their challenges. The computational overheads often increase due to the incorporation of multiple paradigms. There is also a heightened risk of overfitting, requiring meticulous regularization and model evaluation. Moreover, successfully merging and tuning multiple optimization techniques often require a degree of expertise that can be a barrier to their broader adoption. Finally, it could be said that the path to advancing neural network capabilities lies not in the proliferation of isolated optimization techniques, but in their synergistic integration. As we tackle increasingly complex and high-stakes applications, from autonomous systems to healthcare, the optimization strategies we deploy will need to be as versatile, robust, and adaptable as the problems they are designed to solve [290].

6.4 Transfer Learning and Fine-tuning

Transfer learning has established itself as a pivotal paradigm within the machine learning domain. Its foundational premise centers around the utilization of pre-trained models, originally designed and optimized for one task, to serve as starting points for related tasks, potentially in disparate domains. By leveraging the knowledge acquired from the source task, transfer learning seeks to expedite the learning process of the target task, especially in scenarios characterized by limited labeled data. This capability to harness pre-existing knowledge and adapt it to novel contexts not only promotes computational efficiency but also provides a strategic avenue to circumvent challenges associated with data scarcity and overfitting. The inherent versatility and adaptability of transfer learning render it an indispensable tool in the machine learning arsenal, especially in domains where data acquisition is challenging or expensive.

6.4.1 Leveraging Pre-trained Models

Transfer learning, within the machine learning domain, addresses the computational challenges associated with the direct training of intricate neural models. To elucidate, consider training a neural network $\mathcal{N}$ on a dataset $\mathcal{D}$ to minimize a loss function $\mathcal{L}$:

$$\min_{\theta \in \Theta} \mathcal{L}(\theta; \mathcal{D}), \tag{6.17}$$

where θ represents the set of trainable parameters of the network, and Θ is the parameter space. Training on large datasets, such as ImageNet which consists of millions of labeled instances, requires considerable computational resources to converge to a suitable local minimum θ^*. After training, $\mathcal{N}$ approximates a function $f : \mathcal{X} \rightarrow \mathcal{Y}$, mapping from input space $\mathcal{X}$ to output space $\mathcal{Y}$. This function, parameterized by θ^*, embeds significant features extracted from $\mathcal{D}$. Instead of restarting the training for a related task $\mathcal{T}$, transfer learning suggests repurposing the pre-trained function f.

In practical applications, models pre-trained on substantial benchmark datasets are frequently used. These models are adept at recognizing a wide spectrum of features $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$. These features can be expressed as a mapping $g : \mathcal{X} \rightarrow \mathcal{F}$, converting input data into a feature space. Leveraging foundational layers from such models provides access to a vast feature space without extensive retraining [12, 291]. Mathematically, transfer learning can be defined as:

$$\min_{\theta \in \Theta} \mathcal{L}'(\theta; \mathcal{D}') \quad \text{s.t.} \quad d(\theta, \theta^*) < \epsilon, \tag{6.18}$$

where L' is the loss function for task $\mathcal{T}$, $\mathcal{D}'$ is the new dataset, and $d(\theta, \theta^*)$ measures the difference between the new and the pre-trained parameters, constrained by threshold ϵ. In situations where direct training is not feasible due to constraints, transfer learning using pre-trained models offers a robust alternative [292, 293].

6.4.2 Optimal Contexts for Transfer Learning

Transfer learning is particularly effective under specific conditions that would be challenging for conventional training methodologies. There are several such conditions where transfer learning exhibits superiority.

Data Scarcity

One primary scenario is when there's an insufficient amount of labeled data for the desired task $\mathcal{T}$. To elaborate, consider a target dataset $\mathcal{D}'$ with size $|\mathcal{D}'| = n$, where n is significantly less than the dataset $\mathcal{D}$ used for initial training. In such a situation, overfitting becomes a significant concern, causing the model to perform poorly on unseen data. Overfitting can be formally described as a large gap between the training loss $\mathcal{L}(\theta; \mathcal{D}')$ and the validation loss $\mathcal{L}(\theta; \mathcal{D}_{\text{val}})$:

$$|\mathcal{L}(\theta; \mathcal{D}') - \mathcal{L}(\theta; \mathcal{D}_{\text{val}})| > \delta, \tag{6.19}$$

where δ is an established threshold. Starting with a model pre-trained on a larger dataset $\mathcal{D}$ helps mitigate this risk [12, 292].

Cross-Domain Adaptation

Another situation where transfer learning is effective is in domain adaptation. This involves adapting a model from a source domain $\mathcal{S}$ to a different, yet related, target domain $\mathcal{T}$. The objective here is to develop a function $f : \mathcal{S} \to \mathcal{T}$ that reduces the difference between domains while retaining task-relevant features. This approach is particularly beneficial when the target domain lacks ample labeled data since prior knowledge from the source domain can assist the adaptation process [294].

Multi-Task Learning

There are instances where a primary task $\mathcal{T}_p$ can be enhanced by introducing related secondary tasks $\mathcal{T}_{a_1}, \mathcal{T}_{a_2}, \ldots, \mathcal{T}_{a_m}$. This setup, termed multi-task learning, utilizes shared features across the tasks. The combined learning goal is expressed as:

$$\min_{\theta \in \Theta} \left[\mathcal{L}_p(\theta; \mathcal{D}_p) + \beta \sum_{i=1}^{m} \mathcal{L}_{a_i}(\theta; \mathcal{D}_{a_i}) \right], \tag{6.20}$$

where $\mathcal{L}_p$ and $\mathcal{L}_{a_i}$ represent the loss functions for the primary and secondary tasks, and β controls the contributions from each task. Adopting this approach often enhances performance on the primary task [295].

6.4.3 Challenges and Considerations in Transfer Learning

Transfer learning, though replete with advantages, introduces its own set of complexities. A primary challenge in transfer learning is the risk of negative transfer. This phenomenon arises when the source task $\mathcal{S}$ and the target task $\mathcal{T}$ do not align adequately, thereby compromising the target task's performance. One can formally represent negative transfer if:

$$\mathcal{L}(\theta^*; \mathcal{T}) > \mathcal{L}(\theta_0; \mathcal{T}), \tag{6.21}$$

where θ^* denotes the transferred parameters and θ_0 signifies the parameters initialized randomly [296].

Another concern is overfitting, especially when dealing with limited target data. The use of pre-trained models does not immunize against overfitting on a small target dataset $\mathcal{D}'$. To mitigate this, regularization techniques, such as L_1 or L_2 regularization, can be applied:

$$\mathcal{L}_{\text{reg}}(\theta; \mathcal{D}') = \mathcal{L}(\theta; \mathcal{D}') + \beta \left(\alpha \|\theta\|_1 + (1-\alpha) \|\theta\|_2^2 \right), \tag{6.22}$$

with β controlling the strength of regularization and α balancing the regularization terms [297].

Moreover, the significant size and complexity of pre-trained models also present computational constraints. Models laden with an extensive number of parameters necessitate substantial computational capabilities for both training and inference. When resources are constrained, there exists a trade-off to consider regarding the computational overhead $\mathcal{O}(f(n, d))$ against the benefits derived from a robust pre-trained model [48]. In conclusion, while transfer learning is pivotal in machine learning, it demands astute decisions to exploit its advantages, all while managing its intrinsic challenges [298].

6.5 Meta-Learning: Learning to Learn

With the evolution of neural network architectures and the maturation of machine learning models, a need for refining the traditional learning mechanisms has been identified. Particularly in contexts where data availability is limited or when the objective is to devise models that can swiftly acclimate to new tasks using sparse data, conventional methodologies may prove inadequate [299]. This has led to the exploration of meta-learning, a sophisticated learning paradigm that seeks to equip models with the capability to optimize their own learning processes.

6.5.1 Limitations of Traditional Learning Approaches

Traditional learning paradigms, notably supervised learning, have garnered attention for their ability to fit complex models. However, they are fundamentally data-intensive, necessitating significant volumes of labeled data to accurately estimate the model parameters [3]. This data-dependency presents a limitation in real-world applications where acquiring a sufficient amount of labeled data is economically or logistically challenging. Furthermore, the temporality of machine learning applications – where speed in adapting to new tasks becomes increasingly critical – adds another layer of complexity [300]. The necessity to train a model from scratch for each distinct task poses a challenge, both computationally and in terms of resource allocation.

To provide a mathematical exposition, consider the supervised learning problem defined as finding an optimal mapping $f_\theta : \mathcal{X} \to \mathcal{Y}$, parameterized by θ. This optimization problem is generally framed as:

$$\theta^* = \arg\min_\theta \mathcal{L}(\theta; \mathcal{D}), \tag{6.23}$$

where $\mathcal{L}$ is a loss function and $\mathcal{D}$ represents the dataset. In contrast, meta-learning operates on a more abstract level. Let $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ be a set of tasks, each with its corresponding dataset $\mathcal{D}_i$. Meta-learning aims to find an optimal initialization θ_0 or a learning policy π such that:

$$\theta_i^* = \arg\min_\theta \mathcal{L}(\theta; \mathcal{D}_i), \tag{6.24}$$

can be efficiently computed for each T_i with minimal additional data or adjustments [299]. The ultimate goal is to construct a model that generalizes well across the distribution of tasks $\mathcal{T}$ with lesser computational and data requirements.

Meta-learning frameworks often involve a hierarchical or “nested” optimization structure that is mathematically intriguing. The outer optimization loop usually focuses on learning what can be referred to as meta-parameters, which in turn guide the inner loop’s optimization on individual tasks. For example, in Model-Agnostic Meta-Learning (MAML), the meta-objective aims to find a model parameterized by θ such that, after one or more gradient steps on a new task T with its loss function $\mathcal{L}_T$, the model performs well on that task. Formally, this can be expressed as:

$$\min_\theta \sum_T \mathcal{L}_T(f_{\theta'}), \tag{6.25}$$

where $\theta' = \theta - \alpha \nabla \mathcal{L}_T(f_\theta)$. The challenge here is to optimize θ in a manner that it serves as a good initializer for a broad range of tasks, which is a non-trivial optimization problem.

6.5.2 Examples and Applications

The efficacy of meta-learning is best demonstrated through its myriad applications, which span a multitude of scenarios that challenge the efficacy of traditional machine learning paradigms. These applications provide compelling use-cases that not only establish the versatility of meta-learning but also underscore its capability to tackle problems in a data-efficient manner.

Few-shot Learning

Few-shot learning emerges as a seminal application domain for meta-learning, aiming to construct models capable of performing well with an exceedingly small number of labeled examples [301]. From a mathematical standpoint, let T be a specific task where the available labeled dataset comprises k examples, with k being considerably small. We can formulate the few-shot learning problem as:

$$\min_{\theta} \mathcal{L}(\theta; \mathcal{D}_k), \tag{6.26}$$

where $\mathcal{D}_k$ represents the k-sized dataset and $\mathcal{L}$ is the loss function. Meta-learning algorithms train on multiple such tasks $T_1, T_2, \ldots, T_n$, optimizing the model to adapt swiftly to novel tasks using these minimal examples [302].

Neural Architecture Search (NAS)

Neural Architecture Search (NAS) offers another compelling application area for meta-learning. Instead of utilizing handcrafted architectures, NAS techniques strive to discover optimal architectures automatically [303]. Within the meta-learning framework, the objective function $\mathcal{L}(A)$ for NAS can be considered, where A denotes a neural architecture. The meta-learner, therefore, seeks to optimize:

$$\min_{A} \mathcal{L}(A), \tag{6.27}$$

by learning a strategy π for effective architecture selection, which can generalize well across multiple tasks [304].

6.5.3 Limitations and Challenges of Meta-Learning

Meta-learning, while promising, faces computational and theoretical challenges that necessitate further research [305]. A significant challenge lies in the often used nested optimization, where each meta-training iteration entails solving multiple tasks. This approach exacerbates computational costs [306, 307], making the development of efficient algorithms a pressing concern [308].

The success of meta-learning hinges on the assumption that training tasks align with real-world application tasks in terms of similarity or relatedness [309]. A deviation from this assumption, where training tasks do not accurately represent real-world tasks, can undermine the meta-learner's generalization [310]. Hyperparameter sensitivity, especially concerning the learning rates for inner and outer optimization loops, poses another challenge. Incorrect hyperparameter values can destabilize training dynamics and slow convergence, impacting the model's efficiency and reliability [311]. Additionally, there's a risk of overfitting in meta-learning models, particularly when training tasks lack diversity. This can compromise the model's ability to generalize to new tasks [312].

In summary, while meta-learning offers notable advantages, addressing its inherent challenges is crucial for its evolution and effective application in the machine learning domain [305].

6.6 Challenges and Open Problems in Neural Learning

Deep neural networks have revolutionized the landscape of machine learning and artificial intelligence, leading to state-of-the-art performance across a plethora of domains, ranging from computer vision and natural language processing to healthcare and autonomous systems. These architectures have indeed achieved remarkable success in capturing intricate data distributions, generalizing across tasks, and even surpassing human performance in specific benchmarks. However, the advent of such powerful learning systems has also reignited an array of complex challenges that put into question the robustness, stability, and transparency of neural learning paradigms.

6.6.1 Non-convexity of Loss Surfaces

Recalling the discussion also held in Chapter 3, the loss surfaces associated with deep neural networks are inherently non-convex, a phenomenon rooted in their complex, hierarchical architecture [313]. This non-convexity complicates the optimization landscape and presents theoretical challenges in rigorously understanding the training dynamics. To formalize this, let $\mathcal{N}_\theta : \mathcal{X} \to \mathcal{Y}$ denote the neural network function, parameterized by $\theta \in \Theta$, where Θ is the high-dimensional parameter space. The objective function or the loss $\mathcal{L} : \Theta \to \mathbb{R}$ is defined as

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(\mathcal{N}_\theta(x_i), y_i), \tag{6.28}$$

where (x_i, y_i) are samples from the dataset, ℓ is a task-specific loss function, and N is the number of samples. Because of the composite nature of $\mathcal{N}_\theta$, $\mathcal{L}(\theta)$ becomes a highly non-convex function over Θ.

One significant issue introduced by non-convexity is the phenomenon of multiple local minima. In a convex optimization landscape, the gradient points toward a unique global minimum, and gradient-based methods can efficiently find it. However, in a non-convex setting, the optimization process becomes fraught with challenges. Mathematically, a local minimum θ^* can be defined as a point where the gradient $\nabla\mathcal{L}(\theta^*) = 0$ and the Hessian $\nabla^2\mathcal{L}(\theta^*)$ is positive definite. Due to the non-convexity of $\mathcal{L}(\theta)$, multiple such points can exist, and gradient descent can get trapped in any of these points, impacting the generalization of the learned model [314]. The notion of saddle points further complicates this landscape. In high-dimensional spaces, these points are statistically more prevalent than local minima [85]. A saddle point θ_s can be formally characterized as a point where the gradient $\nabla\mathcal{L}(\theta_s) = 0$, but the Hessian $\nabla^2\mathcal{L}(\theta_s)$ has both positive and negative eigenvalues. The implications of saddle points are twofold: First, they can act as traps for optimization algorithms, substantially slowing down the convergence [84]. Second, their ubiquity in high dimensions suggests that many optimization techniques might get stuck near these points rather than at local minima, raising questions about the common belief that local minima are the primary hindrances to optimization. Finally, the rich and intricate structure of the loss landscape necessitates a nuanced balance between exploration and exploitation during optimization. While gradient-based algorithms inherently focus on local descent, thus exploiting the immediate landscape, they may overlook better solutions that lie outside their neighborhood. Stochastic gradient descent, for example, introduces an element of randomness, partially solving this issue. Specifically, the stochastic nature of the gradient estimations allows the algorithm to 'jump' out of poor local minima or saddle points, thereby achieving a more extensive exploration of the landscape [315, 316]. This randomness can be mathematically formalized by incorporating a noise term ϵ in the gradient estimation, leading to $\nabla\mathcal{L}(\theta) \leftarrow \nabla\mathcal{L}(\theta)+\epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Another example of strategies to address the intricate nature of the loss landscape is through evolutionary algorithms (EAs) [281]: unlike gradient-based optimization, instead of refining a single solution iteratively, EAs maintain a population of candidate solutions and evolve this population over generations to optimize the objective function, so exploring the search space more broadly.

Moreover, since they maintain a diverse population of solutions, they are less susceptible to being trapped in local minima and can potentially uncover regions of the loss landscape that are inaccessible to gradient-based methods.
Despite the efforts, the non-convexity of loss landscapes in deep learning remains an open question, since it is not merely a computational obstacle but also a complex mathematical construct that presents several open challenges and questions in optimization theory.

6.6.2 Theoretical Guarantees (or the Lack Thereof)

In classical optimization theory, particularly for convex problems, a multitude of theoretical guarantees exist, including conditions under which algorithms like gradient descent will converge to a unique global minimum [81, 317]. However, the non-convexity of the optimization landscapes of neural networks presents a formidable obstacle in extending these guarantees to the realm of neural learning. The phenomenon often referred to as the "Generalization Paradox" poses a challenge to classical statistical learning theory, which includes metrics such as the Vapnik–Chervonenkis (VC) dimension to predict model performance. According to these traditional principles, a model with more parameters than training examples should suffer from overfitting. However, deep neural networks regularly contradict this expectation, generalizing effectively despite their large number of parameters [318]. This unexpected behavior has led to a significant research effort aimed at understanding the underlying reasons for this robust generalization. One prevailing hypothesis points to the influence of architectural choices, such as the use of convolutional layers in image recognition tasks [319]. These architectural elements may introduce specific inductive biases that guide the learning process toward useful solutions, thereby facilitating effective generalization even in situations where traditional learning theory would predict failure. Another line of inquiry focuses on the role of the optimization algorithms, particularly stochastic gradient descent (SGD) and its variants, which are commonly used to train deep neural networks.

These algorithms have been found to introduce a form of implicit regularization during the training process. From a mathematical perspective, the behavior of SGD in the parameter space can be modeled as a dynamical system, providing insights into this implicit regularization effect [320]. Research indicates that the stochastic nature of the algorithm's updates actually aids in avoiding poor local minima, thereby facilitating solutions that generalize well. This observation is helping to narrow the gap between theoretical expectations and empirical performance.

One of the foremost concerns in the optimization of neural networks is the guarantee of convergence, especially given the high-dimensional and non-convex nature of their associated loss landscapes. For example, the empirical success in avoiding suboptimal local minima of Stochastic gradient descent (SGD) and its manifold variants is well-documented; however, a formal understanding of this behavior remains significantly less concrete, often entailing assumptions that, while mathematically convenient, are far removed from the complex realities of practical implementations [321, 322]: existing mathematical analyses often invoke assumptions such as smoothness and strong convexity around local minima to prove convergence rates [321]. While such results provide valuable insights, they are usually based on idealized models of the loss landscape and thus may not generalize to more complex, real-world scenarios. Therefore, there is an urgent need for theoretical advancements that can provide more realistic models and, consequently, more reliable convergence guarantees.

Simultaneously, another axis of investigation revolves around the relationship between the architecture of a neural network – specifically its depth – and its expressive capabilities. Empirical evidence suggests that deep neural networks possess a computational advantage over their shallow counterparts, often requiring exponentially fewer parameters to approximate complex functions [323, 324]. Yet, despite this empirical robustness, a mathematically rigorous account of why depth bestows such expressive prowess remains elusive: current mathematical frameworks for understanding the expressive power of deep networks often rely on the concept of approximation theory anb theoretical results in this domain typically demonstrate the ability of deep networks to approximate any continuous function to an arbitrary degree of accuracy, under certain conditions. However, these conditions often involve an exponential dependence on the depth of the network, and translating these theoretical findings into practical, trainable models remains a challenge [318].

6.6.3 The Interpretability and Transparency Challenge

Deep learning models, especially those with a large number of parameters and layers, are often described as "black boxes", owing to the intricate connections and non-linear transformations that make the interpretive analysis non-trivial [325, 326]. As neural networks find their way into crucial sectors like healthcare, autonomous vehicles, and financial systems, the imperative for model interpretability and transparency escalates significantly. Mathematically, the notion of interpretability can be framed as elucidating the functional mapping $\mathcal{N}_\theta : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{N}_\theta$ represents the neural network and θ denotes its parameters. Addressing this complexity involves multiple layers of analysis. A primary layer pertains to the elucidation of *feature importance*. This investigation seeks to systematically determine the weightage or influence of each input feature on the network's decision-making process. This level of understanding becomes essential when refining models, especially when their applications span domains of paramount importance such as medical diagnostics and financial risk assessment [327, 328]. From a formal perspective, the challenge is then to ascertain the degree of sensitivity of f_θ to variations in each element x_i of the input vector $x \in \mathcal{X}$. Such sensitivity can be gauged using methodologies like partial derivatives $\frac{\partial \mathcal{N}_\theta}{\partial x_i}$ or advanced tools such as Integrated Gradients [329]. Additionally, the domain of information theory provides avenues like mutual information as a metric to quantify the amount of information a specific feature conveys about the output [330]. Formally assessing the mutual information $I(X_i; Y)$ between a feature X_i and the output Y offers a comprehensive metric for feature significance. Statistical methods, including ANOVA (Analysis of Variance) or regression coefficients in linear approximations, also provide insights into feature importance [331]. It's essential to note that while these methods might highlight overarching trends, the intricate non-linear interplay between features, inherent in deep neural networks, may elude such techniques.

The second consideration in neural network analysis revolves around the *internal representations* present within the hidden layers [332]. These intermediary layers are instrumental in processing raw input data by isolating hierarchical features that lay the groundwork for subsequent layer computations. To interpret these high-dimensional spaces, dimensionality reduction tools such as t-SNE (t-Distributed Stochastic Neighbor Embedding) and PCA (Principal Component Analysis) are commonly employed. However, a notable limitation of these methodologies is their linear disposition, in contrast to the predominantly non-linear feature spaces of neural networks [333]. Activation maximization is another approach that seeks to identify inputs causing substantial neuron activation, thereby shedding light on the specific patterns a neuron is attuned to [334]. From a more quantitative perspective, Representational Similarity Analysis (RSA) quantifies similarities in representations for varied inputs across layers, facilitating comparisons between neural network functionalities and human cognitive activities [335]. Notwithstanding their value, these techniques possess inherent constraints, especially as they are designed predominantly for supervised learning scenarios. Current research efforts aim to adapt these methods to alternate learning frameworks, including unsupervised and reinforcement learning [336].

The third area of emphasis is on the *interpretability of decisions* by neural networks, a significant concern, particularly in domains like healthcare, law, and finance [337]. In such contexts, the rationale behind a decision holds as much weight as the decision itself. For instance, understanding the cause behind a misclassification in medical diagnostics, such as wrongly identifying a benign tumor as malignant, is vital for both model improvement and patient care. Multiple techniques aim to enhance interpretability. LIME (Local Interpretable Model-agnostic Explanations) is one method that offers an approximation to the decision boundary of a complex model in a local vicinity of an instance [338]. Mathematically, LIME aims to find an interpretable model that closely matches the original model's predictions in this local region. Another approach, SHAP (SHapley Additive exPlanations), applies cooperative game theory principles to assign each feature's contribution to a prediction fairly [339]. Nevertheless, the computational demands of these methods can be extensive, leading to challenges in scaling with the dimensionality of inputs or the model's complexity [340]. Counterfactual explanations present an alternative, identifying the proximate instance to an input that alters the decision [341]. However, these methods might only yield local insights or approximations and could miss the broader complexities intrinsic to advanced models. Thus, the search for a comprehensive approach that seamlessly integrates accuracy, computational viability, and lucid interpretability is still in progress. While several techniques, like attention mechanisms and saliency maps, aim to alleviate these issues, a comprehensive solution remains elusive [337]. Hence, it serves as a vital area for future research, both from an applied and a theoretical standpoint.

In summary, while progress has been made in the development of techniques for model interpretability, fully understanding the complexities inherent in the architectures and decision-making processes of deep neural networks remains an open problem. This is an issue of both theoretical and practical importance, requiring ongoing research to make neural networks not just powerful, but also transparent and accountable.

Chapter 7

Beyond the Standard Learning

7.1 Introduction

As discussed, numerous research endeavors are dedicated to refining the learning procedures of neural networks to accommodate specialized tasks and confront challenges intrinsic to conventional learning approaches. An additional dimension to this discourse emerges from a more philosophical viewpoint: the biological credibility of the backpropagation algorithm. With the increasing recognition of backpropagation due to its success in fields such as image recognition [342, 343], natural language processing [344, 345], and speech recognition [346, 347], concerns about its veracity as a representation of cortical learning mechanisms have intensified. Furthermore, while various studies suggest the potential biological role of backpropagation [348–351], definitive evidence is still lacking: it remains to be established that the cortex disseminates error derivatives or retains neural activities for subsequent backward phases. Also, for example, the descending connections in the visual pathway do not correspond symmetrically to the ascending ones, contrary to expectations if backpropagation had a significant role in the visual system; these connections instead exhibit a cyclical pattern, directing neural activity through several cortical layers in both regions before returning to the initial point.

Moreover, in the context of biological neural networks, several challenges emerge when examining backpropagation. One primary issue is related to the continuous, real-valued gradient information that may switch its sign, which could be in contradiction with Dale's Law. Further, backpropagation necessitates a guiding signal to deliver target values: for the algorithm to operate efficiently, it is imperative to carry out all relevant linear transformations and consistently multiply the retrograde signal by the derivatives of forward activations. This demands an exact sequence between forward and backward computational stages. A distinct challenge of backpropagation is its requirement to relay error signals in the reverse direction, using synaptic weights that are expected to correspond with the forward weights. Ensuring this symmetry in weights throughout the learning process, especially in a biological framework, is challenging. Additionally, backpropagation assumes an in-depth understanding of the forward pass to compute derivatives accurately. Integrating an uninterpretable "black box" component within the model poses complications. In the absence of a clear model of the forward pass, the viability of the algorithm is called into question unless a differentiable representation for this component is introduced. Given this uncertainty in modeling the forward pass, considering reinforcement learning techniques becomes a plausible alternative since this method involves altering weights or neural activities randomly, linking these changes to variations in a reward function. Yet, reinforcement learning brings its own set of challenges: the pronounced variance, particularly when modifying several components at once, makes it difficult to determine the effects of individual component modifications. Addressing this demands adjusting the learning rate proportionally to the number of modified components. This presents scalability concerns in larger networks, suggesting that reinforcement learning might be less efficient compared to backpropagation.

Rethinking the error information's backward propagation is exemplified by works like [15], which presented the Residual Network – a design that incorporates skip connections to enhance gradient flow during training. Similarly, [352] introduced target propagation, a technique that backpropagates target outputs rather than error gradients. Many of these methods aim to streamline training by using auxiliary variables or gradient approximations [353]. Such a philosophy is further illustrated in [354], which presents efficient model-parallel training by segmenting a neural network into consistent "blocks". [355] offers a comprehensive approach to ensure the convergence of such techniques, specifically for loss functions with Lipschitz continuous gradients. A significant effort has been directed towards developing intra or inter-layer cost functions to sidestep whole model backpropagation. Feedback Alignment, for instance, leverages a fixed random matrix instead of transposed weight matrices for error signal propagation [348, 356]. Other strategies based on randomness are consolidated in [357].

Distinct alternatives to Backpropagation have also gained traction: for instance, kernel-based methods are showcased in [358] and [359], with one adopting a genetic algorithm and the other focusing on hypercube-based encoding. The Random Vector Functional Link Network (RVFL) [360] takes a different approach – it randomizes and locks the hidden weights while analytically computing only the output layer's weights. Yet, as of now, no alternative has genuinely managed to replace the Backpropagation process in neural network training.

7.2 General Framework

In the context of feedforward neural networks for supervised learning tasks, the general network structure can be described as:

$$\mathcal{N}[N_1, N_2, ..., N_h, ...N_L]. \tag{7.1}$$

Here, N_1 signifies the count of neural units in the input layer, while N_h and N_L depict the number of neurons in the h^{th} hidden layer and the output layer, respectively. Assuming complete interconnections between the layers, the weight from neuron j in layer $h-1$ to neuron i in layer h can be represented as w_{ij}^h. The resultant output O_i^h for neuron i in layer h is evaluated as:

$$O_i^h = \lambda_i^h(S_i^h) \tag{7.2}$$

where

$$S_i^h = \sum_j w_{ij}^h O_j^{h-1} \tag{7.3}$$

Typically, the transfer functions λ_i^h remain consistent across most neurons, with certain variations, especially in the output layer.

Training operates on a set of M examples, consisting in input-target couples (I, T), on which the objective revolves around reducing a cost function $\mathcal{L}$, which measures the discrepancy between predictions and provided targets. While the primary focus is on supervised learning, it is noteworthy that several neural network architectures commonly labeled as "unsupervised" (such as autoencoders, neural autoregressive distribution estimators, and generative adversarial networks) do incorporate output targets, fitting them within the aforementioned schema.

7.2.1 The Learning Channel

In a theoretical framework, for the described neural network architecture, training ideally entails determining the weights w_{ij}^h to achieve the minimum of the loss function $\mathcal{L}$. The criteria for these weights at any critical point can be mathematically expressed as:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^h} = 0. \tag{7.4}$$

In a generalized context, the optimal weights are functions of the input data, the target values, and the other existing weights within the network. Thus, the training process can be regarded as a mechanism that transfers the intrinsic information present in the training dataset into the neural network's weight parameters. Within the confines of a feedforward architecture, an effective neural network learning algorithm should possess the capacity to convey specific information about the targets T to the internal weights w_{ij}^h by utilizing an informative channel. Importantly, there isn't a foundational requirement for this channel to transmit information in a step-wise manner, as happens for processes such as backpropagation. Consequently, $I_{ij}^h(T)$ could potentially interact directly with a designated deep layer h.

On the other hand, for optimal training to be achieved, the learning rules governing the deep weights must encompass all necessary information to reach a critical point of $\mathcal{L}$. It's imperative that these learning rules remain local [361], implying they should only involve variables that are spatially and temporally local. A typical local learning rule for any deep layer might thus be expressed as:

$$\Delta w_{ij}^h = F(O_i^h, O_j^{h-1}, w_{ij}^h) \tag{7.5}$$

which is sequentially applied across layers, starting from the input layer. This might be followed by a supervised learning rule depicted as:

$$\Delta w_{ij}^L = F(T_i, O_i^L, O_j^{L-1}, w_{ij}^L) \tag{7.6}$$

operating under the assumption that the last layer treats the targets as localized variables.

A natural inquiry is how effectively this approach can learn input-output functions from examples, especially when comparing it to methods like back-propagation. One notable example of such local learning methods is Hebbian learning [362]. However, in the given setup, crucial information about the targets might not propagate to the deeper layers. Consequently, finding solutions to the critical equations under this formulation often remains a challenge, thereby limiting the ability to learn intricate functions [361].

Therefore, to inject information about targets in the local learning rule and to ensure that the learning process remains localized, representing the target-related information transmitted from the output layer to the weight w^h_{ij} for its update as $I^h_{ij}(T)$, the application of a specific local learning rule can be extended to:

$$\Delta w^h_{ij} = F(I^h_{ij}, O^h_i, O^{h-1}_j, w^h_{ij}). \tag{7.7}$$

Given this context, in principle the information should also depend on the impact of the targeting neuron on the network. So, in a feedforward architecture applied on a supervised learning task, it should have the form $I^h_{ij}(T, O^L, w^l_{rs}$ for $l > h, \lambda'(S^l_r)$ for $l \geq h)$. Additionally, classic backpropagation demonstrates that the same information can be relayed to all synapses connected to a particular neuron: this suggests the feasibility of learning using a more streamlined type of information, represented as:

$$I^h_i(T, O^L, w^l_{rs} \text{ where } l \geq h, \lambda'(S^l_r) \text{ for } l > h)$$

Based on this expression, the local learning rule can be written as:

$$\Delta w^h_{ij} = F(O^h_i, O^{h-1}_j, w^h_{ij}, I^h_i(T, O^L, w^l_{rs} \text{ for } l \geq h, \lambda'(S^l_r) \text{ for } l \geq h)) \tag{7.8}$$

Moreover, if learning rules can be expressed with a multiplicative form, this functional relation can be expressed as:

$$\Delta w^h_{ij} = \eta I^h_i(T, O^L, w^l_{rs} \text{ for } l \geq h, \lambda'(S^l_r) \text{ for } l \geq h) O^{h-1}_j \tag{7.9}$$

related to a combination of the presynaptic activity and a form of backpropagated error information, where conventional backpropagation being a specific case. In relation to this scenario, applying the chain rule, it becomes evident that the propagated error adheres to the following recurrence relationship:

$$B_i^h = \frac{\partial \mathcal{L}}{\partial S_i^h} = (\lambda_i^h)' \sum t B_t^{h+1} w_{ti}^{h+1} \tag{7.10}$$

Along with the boundary condition:

$$B_i^L = \frac{\partial \mathcal{L}}{\partial S_i^L} = T_i - O_i^L \tag{7.11}$$

Consequently, the classical backpropagation rule for learning is expressed as:

$$\Delta w_{ij}^h = -\eta \frac{\partial \mathcal{L}}{\partial w_{ij}^h} = \eta B_i^h O_j^{h-1} \tag{7.12}$$

where η is the learning rate, O_j^{h-1} is the presynaptic activity, and B_i^h is the backpropagated error. In essence, errors are transmitted in a predominantly linear manner in the reverse direction using the transposed matrices from the forward pass: this results in weight symmetry and in the fact that, as each layer is crossed, the errors are multiplied by the derivative of the corresponding forward activations. Moreover, it should be noticed that, in this case, all the information about output and targets is contained in the term $T - O^L$, so:

$$I_i^h(T - O^L, \lambda'(S_r^l) \text{ for } l > h) \tag{7.13}$$

also considering that standard backpropagation uses information about the upper weights both through the output O^L which appears in the error terms $T - O^L$ and through the backpropagation process itself.

While this form of error information dominates in neural network training due to its origin in the backpropagation approach, other methods can be explored as alternatives. Within the context of this framework, it can be deduced that, although the local learning rule necessitates incorporating information pertaining to the target, it is not imperative for this information to be confined to the terminal layer of a neural architecture. Rather, it may be furnished locally through appropriate strategies.

7.2.2 The Forward Forward Algorithm

Driven by the considerations around the implausibility of backpropagation as biological learning mechanism, Geoffrey Hinton introduced an alternative training process in December 2022: the Forward-Forward (FF) algorithm [363].

Drawing inspiration from Boltzmann machines [364] and Noise Contrastive Estimation [365], the Forward-Forward method is a layered, greedy learning approach: the idea is to replace the forward and backward passes of backpropagation by two forward passes that operate in exactly the same way as each other, but on different data and with opposite objectives. The positive pass operates on real data and adjusts the weights to increase the goodness in every hidden layer. The negative pass operates on "negative data" and adjusts the weights to decrease the goodness in every hidden layer.

As measure of the goodness function, the sum of squared neural activities is considered, but in general there are many others that can be applied. The objective of the learning process is to achieve a goodness value that is significantly above a certain threshold for real data and significantly below that threshold for negative data. For example, in the context of a supervised task, the goal is to accurately classify input vectors as positive or negative data, with the probability of an input vector being classified as positive determined by applying the logistic function, σ, to the goodness value minus a certain threshold:

$$\mathcal{G}_h^+ = \sigma\left(\sum_j (O_j^h)^2 - \theta\right) \tag{7.14}$$

Given this, the local loss for the layer h, with $h = 1, ..., L$ can be written as:

$$\mathcal{L}_h = \mathcal{L}_h^+ + \mathcal{L}_h^- \tag{7.15}$$

where

$$\begin{aligned}\mathcal{L}_h^+ &= \sum_i \log(1 + e^{-||O_i^{+,h}||+\theta}) \\ \mathcal{L}_h^- &= \sum_i \log(1 + e^{||O_i^{-,h}||-\theta})\end{aligned} \tag{7.16}$$

and so the update on the weights becomes:

$$\Delta w_{ij}^h = -\eta \frac{\partial \mathcal{L}_h}{\partial w_{ij}^h} \tag{7.17}$$

It is important to note that the Forward-Forward algorithm does not require the use of the chain rule, as the goodness is computed locally. This eliminates the need for BP in the weight update process. So, in this framework, the information about all the upper derivatives is not needed and, if a gradient descent approach is utilized, the learning rule can be written as:

$$\Delta w_{ij}^h = F(O_i^h, O_j^{h-1}, w_{ij}^h, I_i^h(T, \lambda'(S_r^l) \text{ for } l = h)) \tag{7.18}$$

where information about upper weights is not present since the learning is layerwise. Additionally, this formulation guarantees that if the derivatives for a particular layer cannot be computed, an alternative weight update method can be implemented.

7.3 The Locally-Backpropagated Forward Forward Strategy

Based on Hinton's ideas, various studies have been carried out to optimize the aforementioned approach. For example, aiming to develop a model that emulates the cognitive functioning of the human brain, capable of simultaneous information processing and prediction generation, [366] proposed the Predictive Forward-Forward (PFF) pro-

cess: this method combines aspects of feedforward and predictive coding into a robust stochastic neural system. Additionally, [367] introduces an event-driven adaptation of the FF approach, tailoring the feedforward and PFF learning algorithms for spiking neural systems. To tackle the challenge of generating biologically realistic negative data, [368] propose the PEPITA algorithm [369] as a special case of the FF with top-down feedback. In PEPITA, the negative input data for the second forward pass is constructed through a top-down feedback process, integrating error information with the positive image. According to [370], feedforward neural networks have a weakness in the independent training of layers, which limits optimization. To enhance collaboration between layers, they propose a method that not only considers the local goodness during the optimization of a layer but also takes into account the global goodness of the network. In the field of medical image classification, a recent application of FF neural networks has been presented by [371]. They propose a multistage pretraining strategy with FF within a contrastive learning framework, called Forward-Forward Contrastive Learning (FFCL), to be performed before the BP process.
Given the notable characteristics of the Forward-Forward (FF) algorithm – especially its ability to bypass derivative computations through potential “black box” component without a differential model, while still achieving a computational cost comparable with the Backpropagation (BP) method [363] – an explorative study on possibilities of FF’s variants has been conducted. In particular, this study focuses on integrating a localized Backpropagation process to form a hybrid learning strategy that converges faster than the pure FF approach while preserving the ability to avoid backward computations in non-differentiable regions of the network.

7.3.1 Methodology

The proposed methodology combines the Backpropagation (BP) and Forward-Forward (FF) algorithms to leverage the advantages of both techniques. BP, is adept at training large neural networks but relies heavily on an accurate representation of the forward computation to compute gradients. On the other hand, FF operates without needing a comprehensive understanding of the forward computation, making it more appropriate for situations with ambiguous forward details. By integrating BP and FF, this approach aims to address the limitations of each method, creating a more versatile strategy for situations where individual methods may not be optimal. This study focuses on the feasibility of implementing Backpropagation in specific regions of a neural network governed by the FF framework.

From a technical point of view, and following the notation of this chapter, a neural network $\mathcal{N}$ has been partitioned in k sub-network:

$$\mathcal{N} = \bigcup_{k \in K} \mathcal{N}_k \tag{7.19}$$

To the output layer of each sub-network, the loss is computed as reported in (7.16), and this quantity is used to correct the weights of the k-th sub-net layers trough a BP process. On the generic layer h of the sub-net $\mathcal{N}_k$

$$\Delta w_{ij}^{k,h} = -\eta \frac{\partial \mathcal{L}_k}{\partial w_{ij}^{k,h}} = \eta B_i^{k,h} O_j^{k,h-1} \tag{7.20}$$

where

$$B_i^{k,h} = \frac{\partial \mathcal{L}_k}{\partial S_i^{k,h}} = (\lambda_i^{k,h})' \sum_t B_t^{k,h+1} w_{si}^{k,h+1} \tag{7.21}$$

and

$$B_i^{k,L_k} = \frac{\partial \mathcal{L}_k}{\partial S_i^L} = \mathcal{L}_{L_k}^{+,k} + \mathcal{L}_{L_k}^{-,k} \tag{7.22}$$

with L_k indicating the last layer of $\mathcal{N}_k$. In this context, it is worth noticing that for $k = 1, h = 1$ the input layer of the net $\mathcal{N}$ is indicated, while $\forall k \in \{1, ..., K-1\}$,

$$S_i^{k+1,1} = \sum_j w_{ij}^{k+1,1} O^{k,L_k} \tag{7.23}$$

In this framework, Backpropagation can be employed within individual sub-networks, while the sub-networks, denoted as $\mathcal{N}_k$, adhere to a FF training approach. Pertaining to the discussed method, it's crucial to highlight that within a FF-trained network, every layer not only produces an output that acts as a normalized input for the next layer but also directly influences the prediction mechanism. Specifically, each layer yields an embedding of its input while undertaking feature extraction, also ensuring the linkage between the generated embedding and the goodness function. To uphold these traits and thereby retain the network's generalization ability, a stochastic partitioning tactic is implemented during the training: while the total number of blocks within the network is set in advance, the size of each one of these segments is adjusted randomly after a designated number of iterations. Such a methodology ensures that, during training, all layers consistently undergo the optimization criteria of the goodness function, mitigating potential overfitting risks associated with bypassing evaluations for certain layers.

Furthermore, an intrinsic aspect of the Forward-Forward (FF) trained network becomes evident: within such a network, each layer's learning necessitates the neural activity from the preceding layer, subsequently generating an output vector that aligns with the goodness function. A typical layer h is informed of the activity in earlier layers but remains unaware of subsequent ones. Given the absence of Backpropagation, alterations in the output O_i^{h-1} directly influence the optimization of layer h, whereas the reverse does not hold true. This dynamic ensures that the network's initial layers will be optimized more rapidly, in terms of epochs, primarily because their inputs stabilize earlier. For clarity, consider the network's first layer: it consistently receives data as input and locally calibrates its weights to satisfy the goodness function. Although changes in this layer's output affect the training of successive layers, the layer's own adjustments are solely based on its static input and the goodness function. Consequently, the training duration for this layer is anticipated to conclude earlier compared to its successor, which must recalibrate its weights in response to the evolving output of the first layer. Nonetheless, once the first layer's optimization phase stabilizes –

potentially observed as a plateau in the loss function over the training iterations – its weights cease to fluctuate, effectively rendering its output "fixed", in a sense. Given this stabilization, the network's second layer emulates the initial training conditions of the first layer, and this logic propagates consecutively through the network's remaining layers. Hence, it might be deducted that during the progression of training, once the optimization of a given layer (starting with the first) attains a stable state, such a layer could potentially be excluded from the subsequent learning iterations. This

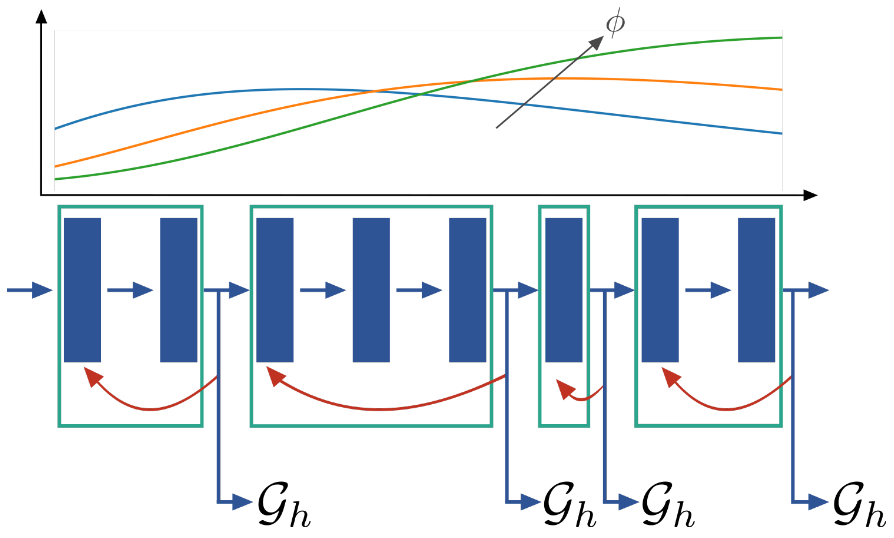


Figure 7.1: Sketched representation of the locally backpropagated Forward-Forward Learning procedure. The Network is partitioned in blocks; at the output of each block the goodness function $\mathcal{G}_h$ is measured and the weights of the layers constituting the blocks are adjusted according to the BP procedure (red arrows). The output of the generic block (blue arrow) is used as input for subsequent blocks. After a global pre-train, not all the blocks are "trained" in each epoch, but they are considered for the weight optimization with a probability density function resembling a $\chi^2(\phi)$ distribution, whose degree of freedom is increased each time a steady state of the weight adjustment on the first trainable block is recognized.

reasoning can also be applied in the case under consideration: in neural network models partitioned into distinct sub-networks, Backpropagation is employed to adjust the internal variables of each segment. Within this structure, any discrepancies in the output O^{k,L_k} relative to the objective function are back-propagated within the sub-network k. It is thus impractical to halt the training on subsequent layers from the first one. Furthermore, the dynamics associated with block sizes – which undergo randomized aggregation after designated epochs – warrant attention. To leverage the insights derived from this structure and further improve computational efficiency and convergence, a probabilistic learning strategy is introduced: following an all-encompassing preliminary training phase that engages all layers, a subsequent fine-tuning phase assigns a selection probability p_i to each layer, drawn from a $\chi^2(\phi)$ distribution where ϕ denotes degrees of freedom. For each epoch, the probability of integrating the k-th block in the learning process is defined as $p^k = \sum_i p_i^k$. Thus, in each iteration, a specific subset of blocks is selected for the optimization phase. Moreover, the performance of the objective function at the output of the initial sub-network is continuously monitored. If this sub-network's learning achieves a plateau, the first available training block is fixed, and the parameter ϕ is incremented, reallocating probability in favor of the later architectural layers. Sequentially, the primary blocks are stabilized, and their training probability diminishes with increasing ϕ values. This methodology is adopted for identifying any periods of stagnation during the training of successive sub-networks. The methodology outlined earlier, sketched in the Algorithm 2, takes into account the distinct attributes of the case in question, including the fluctuating sizes of the blocks, and also leverages a beneficial aspect of local learning intrinsic to the FF algorithm. This approach enhances computational efficiency during training while maintaining convergence results, and sustains the network's generalization capabilities, all while addressing the considerations previously mentioned.

7.3.2 Experimental Framework

The proposed method has been assessed using the widely recognized MNIST dataset of handwritten digits [23]. This dataset comprises $50,000$ grayscale images, each of size 28×28 pixels, for training, and an additional $10,000$ images in the test set for evaluating error rates. We opted for the MNIST dataset due to its compatibility with simpler neural network models, positioning it as an ideal benchmark for gauging novel learning strategies.

Algorithm 2 Random Learning Process

Input: $E, E_R, \tilde{E}, \phi_0, \mathcal{N}, T, block\_ratio, patience$

Parameters: E: max number of epochs
E_R: number of epochs where the blocks are preserved,
$\tilde{E}$: number of epochs of initial learning phase,
ϕ_0: initial degree of freedom for the χ^2 distribution,
$\mathcal{N}$: number of blocks,
T: threshold on loss variation,
$block\_ratio$: percentage of minimum number of selected blocks,
P: patience

```
 1: $\phi \leftarrow \phi_0$
 2: $N_b \leftarrow \mathcal{N}$
 3: Block initialisation
 4: $fixed\_blocks \leftarrow$ First block
 5: for $e$ in $E$ do
 6:     if mod $(e, E_R) = 0$ then
 7:         Randomly generate $N_b$ blocks with layers not in $fixed\_blocks$.
 8:     end if
 9:     if $e < \tilde{E}$ then
10:         Train all the blocks
11:     else
12:         Generate $\mathbf{p} = [p_1, ..., p_L]$, vector of probabilities associated to the network layers according to $\chi^2(\phi)$.
13:         Compute the probabilities for the blocks, given by the sum of layers' probabilities in the blocks.
14:         Choose $\lfloor block\_ratio \cdot N_b \rfloor \leq n \leq N_b$ blocks with higher probabilities.
15:         Perform the weights optimization on the chosen blocks.
16:         Evaluate the loss variation $\Delta\mathcal{L}$ on the first not fixed block.
17:         if $\Delta\mathcal{L} \approx 0$ for $P$ consecutive epochs then
18:             Add new fixed block to $fixed\_blocks$
19:             $\phi \leftarrow \phi + 1$
20:             $N_b \leftarrow N_b - 1$
21:         end if
22:     end if
23: end for
```

Network Configuration

The considered network for the experiments is a sequential neural model composed of 20 linear layers, each containing 500 neurons. Such a configuration was chosen to facilitate the evaluation of the proposed methodology while maintaining the flexibility to select an optimal number of blocks of varying dimensions. It also allowed for monitoring convergence patterns, particularly as the model approached steady-state conditions.

The approach is grounded in the FF algorithm: the initial layer's input is a union of the flattened image and its corresponding target vector. This vector's dimension is dictated by the distinct labels within the dataset and encapsulates the probability distribution of the designated class for a given image. Notably, the input size is determined to be $28 \times 28 + 10$, totaling 794, which in turn establishes the size of the network's input layer.

For training purposes, the block cost function encompasses two inputs, both derived from identical images but represented with different labels: the “positive” data integrates the flattened image with its correct one-hot encoded label, whereas the “negative” data pairs the same image with an incorrect one-hot encoded label representation. Since the only difference between negative and positive datasets is the label, the FF algorithm is conditioned to disregard image attributes that are uncorrelated with the target. Activation functions within the network are instantiated as $ReLU$, and the epoch limit is capped at 100.

Training and Evaluation Procedure

Prior to initiating the training, the network is partitioned into sub-networks at random; the total block count is predetermined at 8. This choice of the value is arbitrary, but rooted in the need of ensuring the formation of both large and compact blocks. Although varying this parameter affects convergence speed, the efficacy of the methodology remains consistent. This parameter essentially gauges the resemblance of our method to either the BP or FF approaches. It is worth noticing that, when the block count mirrors the total layer count ($N_{blocks} = N_L$), the strategy aligns strictly with FF, whereas a singular block implies a BP strategy spanning the entire network.

In order to maintain the independence of these blocks and deter intra-dependence, blocks are re-organized at regular intervals of every $E_R = 10$ epochs. This process involves sampling the $N_{blocks} - 2$ extremes for the sub-networks based on a uniform distribution spanning the range $[N_f + 1, N_L - 1]$. Here, both the N_f^{th} and the final layer, N_L, are consistently utilized in each iteration, with N_f denoting the number of fixed layers.

In neural networks trained via backpropagation, the training of each layer is contingent upon the output from its preceding layer. A layer, if not updated or retrained, is incapable of processing data that diverges from its original training dataset. Additionally, while each layer optimizes its output, it doesn't necessarily gauge the alignment of its results with the expected targets. This layer-to-layer dependency is consistent even within individual blocks, where only the terminal layer's output within a block serves as a prediction aligned to the target. Therefore, regularly modifying the block structure can still preserve the primary benefits characteristic of FF networks. During the initial $\tilde{E} = 10$ epochs, all blocks within the model undergo optimization. Post this phase, the selection of a specific sub-network for training in each epoch is drawn from a $\chi^2(\phi)$ distribution, with a starting parameter $\phi = \phi_0 = 5$.

The previously mentioned steady-state assessment is carried out with a patience level set to $P = 10$ epochs. This evaluation procedure updates both the χ^2 and block parameters, augmenting the ϕ value, reducing the count of unfixed blocks by one, and fixing the initial trainable block. The θ parameter, found in the logistic and cost functions as denoted by Eqs. (7.14, 7.16), is assigned a value of 10. This parameter is recognized as another hyper-parameter in the network, akin to parameters like learning rate, layer count, layer size, and so on. Following the training phase, and in alignment with the FF approach, classification outcomes on the test digits are derived by assessing the output from all layers of the network in response to the input. This input is formulated from the flattened test images combined with a one-hot encoded representation of a generic label l. Specifically, this evaluation process is reiterated n times, where n signifies the total number of potential labels. For the MNIST dataset, n is set to 10, allowing l to vary from 0 to 9. The outputs across the network are consolidated, and the class prediction is determined based on the highest confidence value corresponding to label $\hat{l}$, which is, in essence, the index of the maximum value in the averaged output vector.

7.3.3 Results

Figure 7.2 displays the epoch-by-epoch accuracy of three distinct models: Backpropagation (BP), Forward-Forward (FF), and the proposed strategy. From this illustration, it becomes evident that our approach achieves convergence more rapidly than the traditional Forward-Forward method. A deeper dive into these accuracy trajectories reveals that while the block modifications initially decelerate the training process, they enhance convergence rates in subsequent phases. It's worth noting that all three model configurations employed the Adam optimizer with a learning rate set at $1e-4$.

Table 7.1 presents the time required for each epoch across the three models. Owing to the block sampling dictated by the χ^2 distribution, the strategy's per-epoch computational times show greater variance compared to the other two methods. However, also when considering periods where all blocks are activated – like the initial learning phase where every block is engaged – our technique surpasses the efficiency of the standard FF. Based on the data presented in both the figure and table, it can be inferred that the proposed methodology demonstrates improved convergence in comparison to FF. Furthermore, it exhibits enhanced computational efficiency, reducing the average time per epoch by approximately half relative to FF.

	Time for epoch (s)
Backpropagation	1.45 s ± 0.07 s
Forward Forward	4.6 s ± 0.07 s
Our strategy	2.5 s ± 0.6 s

Table 7.1: LBPFF, BP and FF computational times (in terms of s/epoch).

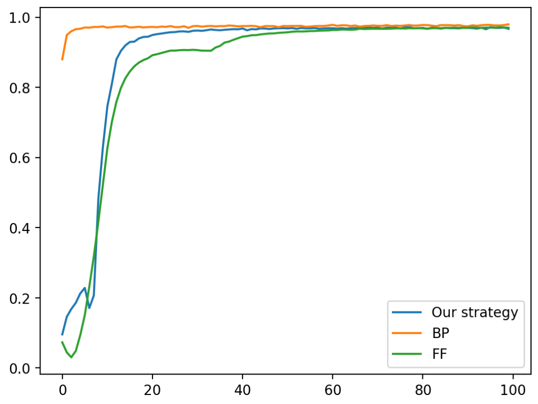
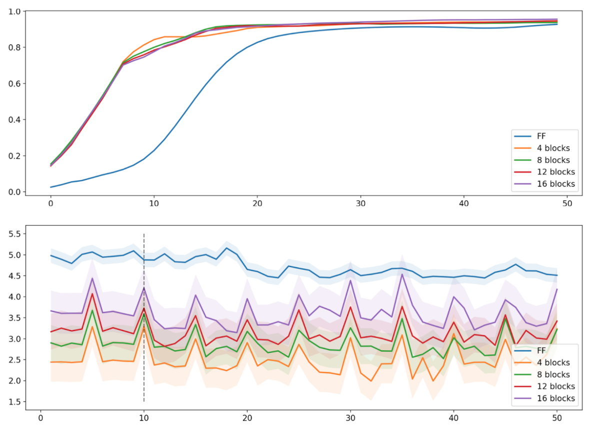
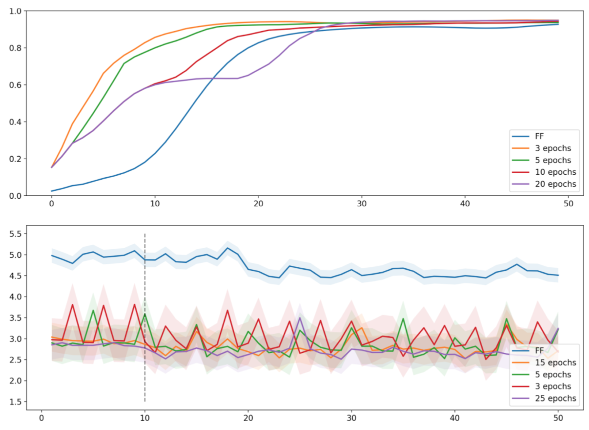


Figure 7.2: The accuracy plot of BP, FF, LBPFF.

A comprehensive analysis was performed to understand the influence of the primary parameters of the proposed strategy on the convergence process. With the epoch time for the random reorganization of blocks fixed at 5, Figure 7.3a illustrates that the convergence is relatively insensitive to the number of blocks into which the network is partitioned. However, this parameter substantially affects the time per epoch. Specifically, as the number of sub-networks increases, the time per epoch rises but remains notably below the one associated with the pure Forward Forward algorithm. In Figure 7.3b, the behavior of these metrics is shown when varying the number of epochs for random re-organization, while keeping the block count fixed at 8. Here, the time per epoch remains relatively consistent, with some variability due to the re-organization phases. Yet, the convergence process is considerably influenced by this parameter. This phenomenon arises because the accuracy measurement, which encompasses all layers, reveals that if a block remains fixed for an extended epoch duration, Backpropagation "saturates" the optimization of the output layer of this block, while it does not optimize the hidden layers of the same block in line with the goodness function. This effect manifests in the figure as plateaus in accuracy when the parameter reaches 20 epochs, and it's anticipated that this behavior would become even more pronounced with fewer blocks.



(a)



(b)

Figure 7.3: Convergence behaviors and computational times in relation to the main parameters of the LBPFF strategy: the blocks number in which the network is partitioned (a) and the number of epochs for the random reorganization phase (b)

7.4 Addressing Systems Heterogeneity with Locally Backpropagated Forward-Forward Learning

Federated Learning (FL) has emerged as a promising approach for training machine learning (ML) models across multiple decentralized entities, known as clients, while preserving data privacy. However, the diverse computational capabilities and resources of these clients pose significant challenges in achieving efficient and reliable convergence. In particular, Federated learning is an approach where multiple clients, like mobile devices, work together to solve a ML problem, under the guidance of a central server, while keeping their training data private and decentralized. Instead of sharing the actual data, only updates needed for immediate aggregation are used to achieve the learning goal [372]. FL's decentralized approach holds significant value in privacy-sensitive applications, especially when data is stored at the edge, by addressing the risks and costs commonly linked to traditional, centralized ML methods. However, implementing FL comes with its own set of challenges. One major concern is ensuring efficient communication among devices. Additionally, there may be differences in data across clients, and variations in computational resources and capabilities among devices [373, 374].

In the typical FL paradigm, the server aggregates parameters from clients' local models to learn a single global model. However, for this knowledge exchange to be feasible, all clients must have the same model structure and size. As a result, the computational capability of FL models becomes limited by the client with the weakest hardware configuration [375]. This means that clients with more powerful hardware cannot utilize their computing power to the fullest, leading to significant delays in the convergence of both local and global models. Additionally, the underutilization of computing power can seriously hinder the overall convergence of FL models. Only recently, researchers have started to address the system heterogeneity issue focusing on heterogeneity-aware FL. To address the challenges associated with system heterogeneity, various approaches have been proposed, such as client selection and split learning methodologies. Among the first, FedCS is an algorithm [376] which utilizes a "greedy" approach to select clients, aiming to maximize the aggregation of updates during limited time intervals. FedAR [377] adopts a trust-based client selection approach: each participating client is assigned a trust score that reflects its performance during the FL process; then, clients that frequently introduce erroneous models or respond slowly are identified as unreliable and consequently excluded from the selection process, ensuring that only reliable clients contribute to the training process, enhancing the overall performance and efficiency of the FL system. FedSAE [378] proposes to customize the workload assigned to clients based on their past training experience, by adopting an active learning approach to select clients. The server intelligently identifies the most relevant or promising clients for the training process, thereby maximizing the efficiency of the entire FL system.

Among the split learning (SL) based methodologies, SplitFed [379] leverages training parallelization from FL and the splitting of the model into separate portions, thereby eliminating the need for clients with strong computational capabilities. Under the same assumptions, the Cluster-based Parallel Split Learning (CPSL) framework [380] facilitates parallel federated training in split learning. The Hybrid Split and Federated Learning (HSFL) scheme [381] divides the participating clients into two groups: one performs FL, while the other performs SL. The clients for each group change at each round, and the server selects them based on their current computational and network resources.

However, in the context of FL on heterogeneous devices, a major challenge arises: Back-propagation, commonly used in traditional FL systems, faces challenges when used with edge devices due to their limited resources and possible lack of reliability. Given these resource constraints, edge devices typically employ computationally affordable models that are quantized and pruned [382], thus making BP challenging. In literature, [383] raised the question of whether FL truly necessitates BP, and they proposed an alternative solution called BAFFLE, whose key feature lies in its exclusive reliance on forward propagation. However, the authors themselves acknowledge that there are still several challenging aspects that need further investigation. Based on this, and following the promising results obtained for the Locally Backpropagated Forward Forward, a novel approach relying on this novel methodology approach has been proposed in the field of Federated Learning on heterogeneous devices: SHELOB-FFL.

7.4.1 Problem Setting

In FL, each participant owns a private local dataset D_c, which is drawn from a specific distribution $P_c(x, y)$, with x and y representing the input features and the associated class labels, respectively. Typically, clients share a model $\mathcal{F}(w; x)$ with identical architecture and hyperparameters. This model is defined by learnable parameters w, neglecting the biases without losing of generality, and input features x. The objective function of FedAvg [384], which is the main aggregation algorithm in FL, can be expressed as follows:

$$\arg\min_{w} \sum_{c=1}^{C} \frac{|D_c|}{C} \mathcal{L}(\mathcal{F}(w; x), y), \tag{7.24}$$

where w represents the global model's parameters, C denotes the number of clients, N is the total number of instances across all clients, $\mathcal{F}$ is the shared model, and $\mathcal{L}$ is a general definition of the loss function addressing a learning task, such as the cross-entropy.

In a real-world FL scenario, however, each client may represent a different device with its distinct user behavior patterns, such as a mobile phone or a geo-localized sensor, resulting in a statistical and/or system heterogeneous environment. In settings with statistical heterogeneity, the distribution P_c varies among clients, resulting in diverse input/output spaces for x and y. On the other hand, in settings with model heterogeneity, the architecture and hyperparameters, represented by $\mathcal{F}_c$, differ across clients, indicating the utilization of distinct model configurations. The objective of the training procedure for the c-th client is to minimize the loss, as described by the following formulation:

$$\underset{w_1,w_2,\ldots,w_M}{\arg\min} \sum_{c=1}^{C} \frac{|D_c|}{N} \mathcal{L}(\mathcal{F}_c(w_c; x), y). \tag{7.25}$$

Existing methods often struggle to effectively handle the aforementioned heterogeneous settings. Specifically, the varying model architecture $\mathcal{F}_c$ leads to differences in the format and size of the local model parameters w_i. Consequently, straightforwardly averaging w_c doesn't effectively optimize the global model parameters w. Thus, alternative weight aggregation strategies, as extensively discussed in the related work section, become necessary. However, the employed solutions often encounter issues in handling non-IID data distribution among clients or pose privacy concerns, as previously mentioned.

SHELOB-FFL

The methodology introduced in this study is designed to address challenges arising from non-IID data and the varying computational capabilities of clients participating in a FL process, achieved through a locally backpropagated block-wise FF network trained according a FF procedure. The central concept revolves around enabling each client to train a distinct segment of the same network, aligning with their computational capacity. These individually trained segments are then aggregated into a unified model, employing weight aggregation strategies rooted in the FL domain. The process of dividing a neural network into discrete blocks mimic the procedure described in the previous Section 7.3.1.

In the context of FL, the block-wise architecture can be harnessed by allocating to each client only a specific portion of the complete network, in alignment with their available computational resources, with no implications on the local training processes. This capability stems from a fundamental aspect of the LBPFF training strategy: the k^{th}

sub-network is aware of the activity in preceding blocks of the network, yet remains unaware of the subsequent ones due to the absence of BP between the blocks.

At the outset of a FL procedure, the weights of the global model are initialized with random values. Following this, individual clients are furnished with partial replicas of the overarching model. Given a set of clients denoted by the index $c \in 1, ..., C$, each client receives a "partial" model designated as $\mathcal{N}^c$, as follows:

$$\mathcal{N}^c = \bigcup_{k \in \{1,...,K^c\}} \mathcal{N}_k^c \tag{7.26}$$

where $K^c \leq K$ is determined based on predefined criteria, such as the computational capabilities of the specific client. Once the local training phase starts, each model $\mathcal{N}^c$ is optimized concerning its local dataset $\mathcal{D}_c$, in accordance with the procedure delineated by Eqs. (7.20)-(7.23). After a predetermined number of epochs, namely at the conclusion of the initial round, each client communicates the updated weights to the server, which holds the comprehensive global model. Importantly, given that the generic local model comprises the first K^c blocks of the global model, the communication phase on the client side exclusively involves the pertinent parameters. In the scenario where the update rule for the global model involves, for the sake of simplicity, a simple average of the local weights, the updated k^{th} block of the global model $\bar{\mathcal{N}}$ is computed as:

$$\bar{\mathcal{N}}_k = \bar{\mathcal{N}}_k(\bar{w}_k, I_k), \quad \text{where} \quad \bar{w}_k = \frac{1}{C_k} \sum_{c=1}^{C_k} w_k^c. \tag{7.27}$$

Here, I_k denotes the input of the k^{th} block, $\bar{w}_k$ represents its parameters, and C_k designates the clients participating in the training of the said block. Once the updates on the global model have been obtained, they are redistributed to the clients, and a further training round is initiated to refine the weights throughout the individual client models. These steps are then repeated until the global model has converged.

In the context of the comprehensive procedure, a pertinent observation emerge due to the peculiar training methodology. In alignment with the proposed paradigm, recalling the fact that each block generates an output that serves both as a normalized input for subsequent sub-networks and also directly contributes to the predictive process it becomes clear that the local model architectures must be partial replicas of the global model commencing from its initial block. This implies that the last blocks of the global model might undergo training from only a limited number of clients or, in the most extreme scenario, could potentially remain untrained entirely. To address these challenges, a strategy termed "freezing" has been developed: within the context of the generic communication round t, on each client, at the end of the local training stage and for the first trainable block, the discrepancy between the global weights at the round $t-1$ and the weights obtained after the training procedure is measured. Then, the variation over the rounds of this quantity is observed. This, formalized through the L_2-norm discrepancy measure, can be expressed as:

$$\frac{\partial \mathcal{H}}{\partial t}, \quad \text{where} \quad \mathcal{H} = \|w_{\hat{k}}^{c,t} - \bar{w}_{\hat{k}}^{t-1}\|_2 \tag{7.28}$$

with $w_{\hat{k}}^{c,t}$ representing the parameters of the first trainable block $\hat{k}$ at the conclusion of training stage t for the local client c, while $\bar{w}_{\hat{k}}^{t-1}$ corresponds to the same parameters extracted from global model following the averaging operation performed at round $t-1$. Leveraging the aforementioned sequential nature of the optimization procedure, as this quantity approaches zero across all clients, or a chosen fraction of their number, the analyzed block is deemed to have reached a steady state. Consequently, this block is removed from the training procedure for all clients.

$$\text{if} \quad \frac{\partial \mathcal{H}}{\partial t} \to 0 \implies \hat{k} = \hat{k} + 1 \tag{7.29}$$

This trigger a sliding effect on the clients' model architectures, altering them in the following manner:

$$\mathcal{N}^c = \bigcup_{k \in \{\hat{k}, \ldots, K^c + (\hat{k})\}} \mathcal{N}^c_k \tag{7.30}$$

where $\hat{k}$ initiates from one and undergoes incremental adjustments constrained by the condition $K^c + \hat{k} \leq K$. The previously delineated strategy ensures that, as successive

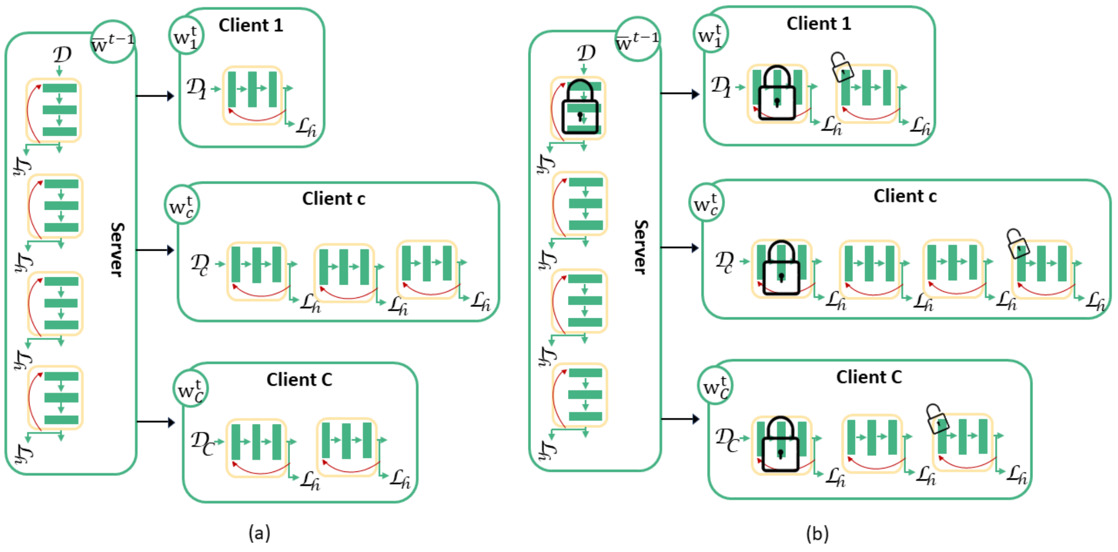


Figure 7.4: By partitioning the main model into fixed blocks, clients allocate resources based on their capabilities at each communication round t, as shown in (a). As clients reach weight convergence on a block, i.e. when $||w_k^t - \overline{w}^{t-1}|| \approx 0$, they freeze it and proceed to the next one. When the majority of clients has frozen the weights on a given block, the server locks that too, fostering global model progress over local optima. Subsequently, each client unlocks an additional block for training (b), and the process carries on.

rounds unfold, all the constituent blocks within the global model undergo the training process across distinct clients, a key consideration given the disparate datasets associated with these clients. This becomes particularly crucial when the data distribution across clients exhibits notable imbalance. Of course, on the generic client, the frozen block remains indispensable due to the inherent feed-forward nature of the methodology. Specifically, the block denoted as $\hat{k}$ and its subsequent counterparts cannot undergo training without leveraging the embedding derived from the preceding block $\hat{k} - 1$. Nonetheless, in light of the absence of an optimization procedure for such a sub-network, resource-intensive computations are unnecessary. As a result, this characteristic maintains the adaptability of the proposed approach, accommodating heterogeneous devices while minimizing computational burdens.

Algorithm 3 SHELOB-FFL.

```
Input: ($C_n$, $T$, $E$, $\eta$, $N_b$, $Tol$, $P$)
Parameters: $C_n$: number of clients,
$T$: max number of communication rounds,
$E$: number of local epochs,
$\eta$: learning rate,
$K$: number of blocks,
$K^c$: number of blocks for the client $c$,
$\epsilon$: threshold on parameters variation,
$P$: patience
OUTPUT: Global model $\overline{\mathcal{N}}$
 1: $start\_block \leftarrow 1$
 2: $\overline{\mathcal{N}} \leftarrow$ create_model($1, K$)
 3: $\overline{w} \leftarrow$ get_parameters($\overline{\mathcal{N}}$)
 4: $\mathcal{N}_{freezed} \leftarrow$ empty
 5: for $c$ in $C_n$ do
 6:     ▷ Assign to each client c the number of blocks to train and create the models
 7:     $\mathcal{N}^c \leftarrow$ create_model($1, K^c$)
 8: end for
 9: for $t = 1$ to $T$ do
10:     for $c = 1$ to $C_n$ do
11:         ▷ Train each client for E epochs and obtain the corresponding parameters
12:         $w^c \leftarrow$ train_client($\mathcal{N}^c$, $E$, $\mathcal{N}_{freezed}$, $\eta$)
13:         $l_2norm^c_t \leftarrow \| w^c - \overline{w} \|_2$
14:     end for
15:     ▷ Obtain the new parameters for the global model through weight aggregation
16:     for $k = 1$ to $K$ do
17:         ▷ Select $w^c$ s.t. $c \in \mathcal{C}$ and c train block k
18:         $\{w^c_k\} \leftarrow$ select_from_clients($w$, $k$)
19:         $\overline{w}_k \leftarrow$ aggregation($\{w^c_k\}$)
20:     end for
21:     if $|l_2norm^c_t - l_2norm^c_{t-1}| < \epsilon$ for the majority of clients for $P$ rounds then
22:         $start\_block \leftarrow start\_block + 1$
23:         if $start\_block = K$ then
24:             break
25:         else
26:             $\mathcal{N}_{freezed} \leftarrow$ create_model_from($\{\overline{w}_k\}$)
27:             /* with $k = 1, ..., start\_block$ */
28:             for $c = 1$ to $C_n$ do
29:                 $\hat{K}^c \leftarrow \min(K^c, K^c + start\_block)$
30:                 $\mathcal{N}^c \leftarrow$ create_model_from($\{w^c_k\}$)
31:                 /* with $k = start\_block, ..., \hat{K}^c$ */
32:             end for
33:         end if
34:     end if
35: end for
36: $\overline{\mathcal{N}} \leftarrow$ set_parameters($\overline{w}$)
```

7.4.2 Experiments

The methodology outlined was executed through the parallel training of multiple clients. The maximum number of communication rounds was established at 200, with each client undergoing training for 5 epochs in every round. A learning rate of $5e-5$ was applied to all clients, with a batch size of 256.

The primary network shared by the server and clients consists of 7 blocks, which, for the following results, are distributed among the clients based on the arrangement illustrated in Fig. 7.7 (a). Specifically, the allocation proceeds as follows: the first client handles one block at a time, the second manages two blocks, the third accommodates three blocks, and so forth, up to the seventh client (indexed as 6), which is capable of hosting all 7 blocks. Subsequently, the pattern recommences from the eighth client onward. This block assignment follows a rule determined by the remainder of the division of the client ID by the number of blocks (i.e., 7), with an additional increment of 1. This approach aimed at ensuring a uniform distribution of blocks across the clients for our experiments.

Within each block, there are three dense layers with dimensions corresponding to the flattened image dimensions combined with one-hot encoded labels, i.e. 794. With reference to the local freezing criterion as presented in Eq. (7.28), its operational implementation is reported in the pseudo-algorithm. From a pragmatic perspective, this criterion adheres to the rationale underlying an early stopping mechanism. Specifically, if the magnitude of the difference $|\mathcal{H}^t - \mathcal{H}^{t-1}|$ falls within the threshold of $\epsilon = 1e-3$ with a patience of 15 rounds for an individual client, a local freezing indicator is activated. For global freezing at the server level, a consensus threshold of 2/3 majority agreement was deemed necessary. This particular choice was made in order to circumvent the potential influence of local oscillations of the $\mathcal{H}$ metric, particularly in scenarios characterized by notably imbalanced data distributions. This strategy was implemented to ensure that these oscillations do not impede the freezing of blocks and thereby avert an imbalanced training regimen of the overall architecture.

Regarding the approach employed to derive global weights from local contributions, it is important to emphasize that the uniqueness of the proposed methodology lies in the capability to train network segments across diverse clients. This does not neces-

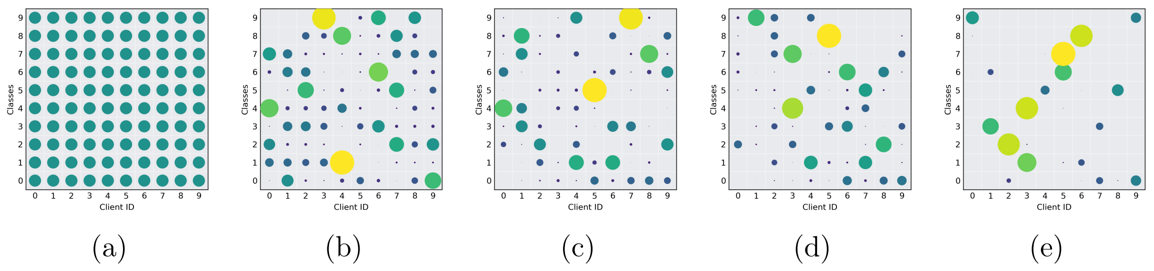


Figure 7.5: Distribution of data among 10 clients as the parameter of the Dirichlet distribution β varies. In a) the distribution is IID, while from b) to e), the β parameter is equal to 1, 0.5, 0.3, and 0.1 respectively.

sitate specialized techniques apart from the acknowledgment that block parameters, originating from various clients, will be transmitted to the server as distinct contributions. Within this context, the forthcoming experiments delve into the exploration of two prominent techniques commonly utilized in the realm of FL: FedAvg [384] and FedProx [374]. In particular, when the second has been considered, the regularization weight was fixed to 0.1.

Non-IID data simulation

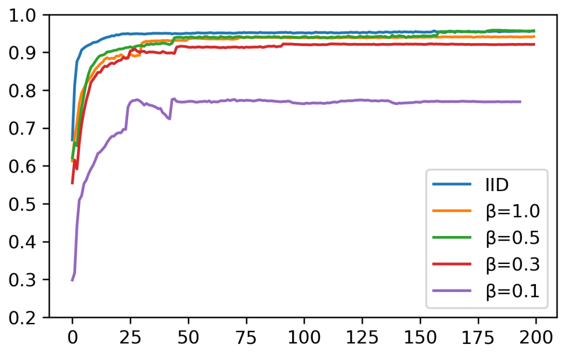
To introduce non-IID-ness in our simulation, similar approaches as used in prior studies [385, 386] have been adopted. To simulate label skewness, a proportion of samples for each label based on the Dirichlet distribution [387] is assigned to each party. For this experiment, β, the concentration parameter of the Dirichlet distribution, is set to be in the range $0.1, 0.3, 0.5, 1.0$, with higher values indicating a more balanced partition. Also a comparison with the IID scenario is provided. Additionally, we simulate quantity skew, where clients possess different amounts of data for the labels. In our experiments, we employ two well-known image datasets, MNIST and Fashion MNIST (FMNIST), both consisting of $70,000$ grayscale images, with each image being 28×28 pixels in size. Figure 7.5 illustrates the distribution configurations utilized in our experiments for both datasets. Specifically, it displays the variations in the distribution as the Dirichlet distribution parameter β changes, considering the scenario with 10 clients.

	MNIST					**FMNIST**				
β	0.1	0.3	0.5	1.0	IID	0.1	0.3	0.5	1.0	IID
FedAvg										
Accuracy	0.7695	0.9206	0.9574	0.9406	0.9627	0.6447	0.8209	0.8456	0.8608	0.8622
Macro F1	0.7449	0.9186	0.9568	0.9397	0.9623	0.6172	0.8137	0.8429	0.8593	0.8616
FedProx										
Accuracy	0.7825	0.9211	0.9316	0.9312	0.9421	0.6744	0.8233	0.8380	0.8434	0.8469
Macro F1	0.7616	0.9195	0.9301	0.9301	0.9414	0.6414	0.8170	0.8335	0.8418	0.8466

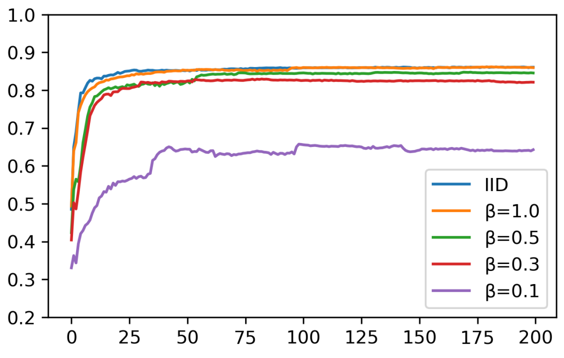
Table 7.2: Accuracy and macro F1 score on MNIST (left) and FMNIST (right) dataset, with 10 clients and varying the values of β for the Dirichlet distribution.

Experimental Results

Table 7.2 show the performance evaluation of FedAvg and FedProx, on MNIST and FMNIST, respectively. The experiments involve 10 clients and investigate the impact of varying values of β for the Dirichlet distribution on the accuracy and macro F1 score. Both algorithms exhibit similar trends in response to varying β values. In particular, they benefit from the improved balance in data distribution as β increases, leading to enhanced accuracy and macro F1 scores. Despite the behaviours' similarity of the two techniques, a nuanced distinction emerges: FedProx displays a minor advantage over FedAvg, especially when dealing with smaller β values. This favorable performance is attributable to FedProx's ability to alleviate the impact of skewed data distribution via its incorporated regularization mechanism. The results are also depicted graphically in Fig. 7.2 where it is possible to observe the increasing trend where higher values of β correspond to greater accuracy. Finally, it is worth noticing the performance drop in the case of $\beta = 0.1$, probably due to the total absence of a large part of labels across the various clients: in fact, while for $\beta = 0.3$ each client still has examples for more than half of the labels, selecting $\beta = 0.1$ the extreme unbalanced distribution of data heavily impacts on the overall convergence.



(a)



(b)

Figure 7.6: Test accuracy on MNIST (a) and on FMNIST (b) datasets for different data distributions across the clients, as reported in the legend. On the x-axis the number of rounds is reported.

7.4.3 Ablation Study

We additionally carry out ablation study to explore essential attributes of our SHELOB-FFL model. This study examines the impact of varying the number of clients on the overall convergence and explores the outcomes resulting from different block assignment distributions to clients.

Influence of the number of clients

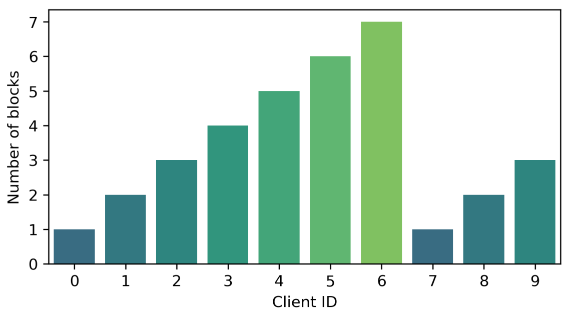
To examine the influence of the number of clients on the overall convergence within our context, we maintain a constant value of 0.3 for the β parameter in the Dirichlet distribution. This choice allows us to showcase evidence within a non-IID scenario that is moderately impactful but not overly severe. We then proceed to vary the number of clients, selecting quantities of 5, 10, 20, and 50 for exploration. Results are shown in Tab. 7.3. When only a single client is involved, i.e. a centralized scenario, we observe the peak of accuracy and macro F1 score across both the MNIST and FMNIST datasets. As the number of clients increases to a modest count of 5 and 10, the algorithm still manages to maintain commendable accuracy and macro F1 scores. Yet, as we extend our exploration to scenarios where 20 or 50 clients are engaged, a discernible shift emerges. The accuracy and macro F1 scores begin to experience a decline. The observed trade-off between the number of clients and achievable performance in FL scenarios underscores the complexity of real-world distributed data settings. As the number of clients increases, the FL process encounters a broader spectrum of data distributions and characteristics. While a larger client pool can lead to a more representative and diverse training dataset, this diversity can hinder the convergence and overall performance.

No. of Clients	1	5	10	20	50
MNIST					
Accuracy	0.9697	0.8965	0.9206	0.8558	0.8921
Macro F1	0.9695	0.8901	0.9185	0.8472	0.8898
FMNIST					
Accuracy	0.8925	0.8053	0.8209	0.8121	0.7957
Macro F1	0.8922	0.8033	0.8137	0.8084	0.7844

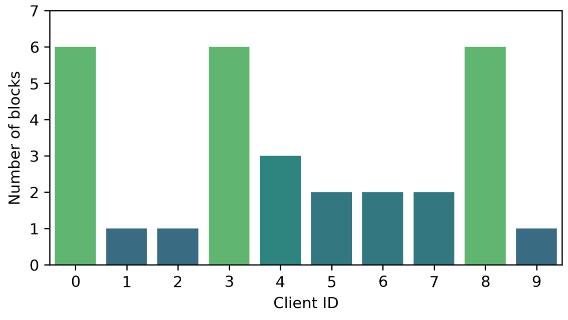
Table 7.3: Accuracy and macro F1 score on MNIST and FMNIST datasets with FedAvg algorithm, $\beta = 0.3$ and varying the number of clients.

Effects of block assignment distribution

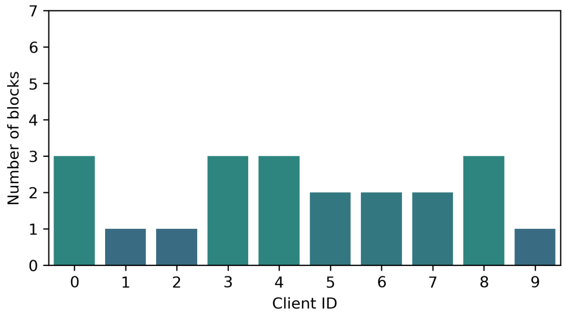
To investigate the effects stemming from diverse block assignment distributions among clients, we conducted experiments by maintaining β at 0.3 and the number of clients at 10. We proceeded to explore different distributions for the allocation of blocks to the clients (see Fig. 7.7b - 7.7d). The results are reported in Table 7.4.
The optimal results are notably achieved through the block distribution in scenario (d), wherein each client undergoes training for the entire network, aligning with the conventional FL approach. Conversely, even within an extremely challenging context such as that of low-performance devices, represented by scenario (c), the decrement in accuracy remains constrained. Scenario (a), employed in the experimental setup, closely approximates the performance of scenario d, while concurrently exploring all conceivable network allocations in terms of local models. The distinct advantage of scenario (a) lies in the existence of clients capable, through the implemented freezing strategy, of training all blocks within the global network in relation to their individual datasets. This capability enables them to foster a comprehensive awareness of the global data



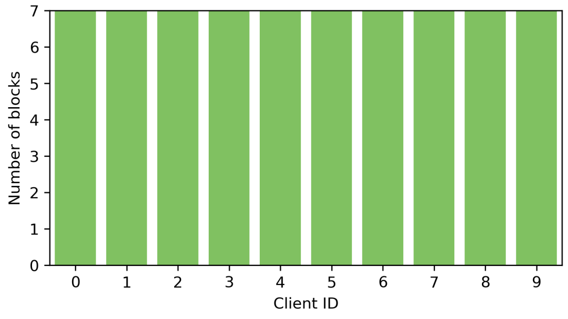
(a)



(b)



(c)



(d)

Figure 7.7: Distribution of the blocks among the clients. The primary distribution (a) was employed during the experiments. Additionally, random distributions were explored in cases (b) and (c), while a scenario (d) was examined where every client possessed exactly 7 blocks (as classic FL approach). These variations were investigated to analyze the impact of block assignment distributions.

distribution. This contrasts with scenario (c), where despite the involvement of three clients participating in the training of nearly the entire network, the inclusion of numerous low-power clients unable to train all designated blocks within the allotted 200 rounds contributes to a decline in model accuracy by nearly two percentage points.

Type of distribution	a)	b)	c)	d)
MNIST				
Accuracy	0.9211	0.9036	0.8876	0.9253
Macro F1	0.9195	0.9003	0.8830	0.9237
FMNIST				
Accuracy	0.8233	0.8177	0.8128	0.8285
Macro F1	0.8170	0.8112	0.8043	0.8222

Table 7.4: Accuracy and macro F1 score on MNIST and FMNIST datasets, with $\beta = 0.3$ and 10 clients.

7.5 Discussions on Hybrid Neural Network Training

The implausibility of Backpropagation as a model of how the biological cortex learns has driven the research towards new models for neural network training. Among these, Hinton's recent Forward Forward algorithm emerges as a promising contender, given its versatility across different problems and its computational comparability with backpropagation. This approach offers computational efficiency comparable to backpropagation across diverse problems. By leveraging target information at the layer level, it eliminates the need to retain neural activity for backward modifications. Consequently, components with unknown differential models can be trained without derailing the entire process. The layer-specific learning also facilitates partitioning a neural architecture for asynchronous training. However, in practice, Backpropagation's efficacy in training neural networks remains unmatched. This is largely attributed to its ability in fine-tuning weights following the gradient direction, securing its position as the predominant learning strategy in the realm of deep learning. Within this context, the idea of combining the Forward Forward algorithm and the Backpropagation through approaches that can exploit the benefits of both the learning strategies opens up new avenues for research. Such a hybrid approach could harness the local learning of the Forward Forward algorithm while still benefiting from the gradient-based weight tuning of Backpropagation. Following this, a Locally Backpropagated Forward Forward strategy has been introduced. The preliminary experimental results shown underscore that the suggested methodology successfully embodies the essence of FF's layer-wise learning and the principle of incremental training, as well as the efficiency of Backpropagation. This underlines the viability and potential prowess of our hybrid technique.

From a practical standpoint, this strategy's inherent benefits could be exploited across a spectrum of domains. For instance, Federated Learning, may benefit from the capability to train model sections without prior knowledge of their complete specifications, bypassing the demand for intricate aggregation mechanisms. In particular, in the context of heterogeneous devices, the idea of a splitting a neural network has been tested to accommodate variations in computational power and available resources among clients. Exploiting inherent characteristics of the Locally Backpropagated Forward Forward training strategy, a Federated learning procedure has been developed, which assigns variable portions of the network to different clients based on their computational power, allowing them to optimize their subnetworks independently. This prevents communication delays and memory issues, while enabling each client's full participation in the learning process.

By leveraging this approach, the modular structure of the neural architecture in FL on heterogeneous devices has been exploited: the procedure assigns variable portions of a network to different clients based on their computational power, allowing them to optimize their subnetworks independently. This prevents communication delays and memory issues, while enabling each client's full participation in the learning process. The feasibility of the proposed strategy in both IID and non-IID scenarios on various datasets has been also investigated, demonstrating also in this case the possibilities that this kind of hybrid approaches open in the research contexts.

Chapter 8

Conclusions

As the dawn of artificial intelligence emerged, few could have predicted the pivotal role that neural networks would come to play. From their humble beginnings as rudimentary models inspired by biological neural systems, they have now become foundational to vast extents of modern research and application.. This exploration of neural networks, from their detailed components to their mathematical foundations, highlights not only the depth of the subject but also the continuous human effort to understand and innovate in the field.

8.1 Outcomes of Foundamental Research

In the earlier chapters of this work, we systematically navigated the expansive domain of neural networks. We began with an examination of their historical development and the primary building blocks that constitute them. A comprehensive understanding of foundational concepts, such as the computational units, their organization into layers, and how these layers interconnect to form an architecture, is essential. Additionally, grasping how the connections within the network are dynamically adapted to ensure the model meets specific requirements and respond to specific tasks paves the way for more advanced discussions and applications. In particular, in Chapter 5, it is showcased how these foundational concepts can be harnessed. Specifically, this chapter demonstrates how the standard building blocks of neural networks can be employed to devise a novel framework for time-series forecasting. The methodology particularly leverages the inherent characteristics of various layer types. Recurrent layers, with their innate ability to recognize temporal patterns, are used in tandem with convolutional layers, which excel at detecting horizontal relationships across features. Dense layers, known for their data aggregation capabilities, are integrated into the mix. These properties are harnessed together to design an intelligent neural combiner for ensemble predictions and time-series variables. Furthermore, to broaden the technique's applicability to multivariate time-series, an additional neural structure was integrated into the framework: a recurrent autoencoder. Capitalizing on this architecture's capability to generate a denoised and compressed feature space, the neural structure was incorporated during a preprocessing stage of the workflow. This enhancement further bolsters the framework's flexibility and robustness. This research has led to the publication of several scientific papers, including:

- Piccialli, F., Giampaolo, F., Salvi, A., & Cuomo, S. (2021). A robust ensemble technique in forecasting workload of local healthcare departments. Neurocomputing, 444, 69-78.

- Piccialli, F., Giampaolo, F., Prezioso, E., Crisci, D., & Cuomo, S. (2021). Predictive analytics for smart parking: A deep learning approach in forecasting of iot data. ACM Transactions on Internet Technology (TOIT), 21(3), 1-21.

- Piccialli, F., Giampaolo, F., Prezioso, E., Camacho, D., & Acampora, G. (2021). Artificial intelligence and healthcare: Forecasting of medical bookings through multi-source time-series fusion. Information Fusion, 74, 1-16.

- Giampaolo, F., Gatta, F., Prezioso, E., Cuomo, S., Zhou, M., Fortino, G., & Piccialli, F. (2023). ENCODE - Ensemble neural combination for optimal dimensionality encoding in time-series forecasting. Information Fusion, 100, 101918.

that contain and extend what has been discussed in Chapter 5. Furthermore, this project resulted in the granting of a patent and the establishment of a university spin-off.

8.2 Outcomes of Advanced Research

As we progressed, the focus shifted to delve into more contemporary and advanced topics, particularly concerning learning dynamics. The training of a neural network is a multifaced process in which many components intricately intersect. Training a neural network is a multifaceted process where numerous components intricately intertwine. Yet, understanding this intricate procedure unveils methodological opportunities that could potentially tackle unresolved challenges in the literature. Inspired by the works outlined in [357, 361], Chapter 7 delved into understanding how the information needed for a task should be incorporated within the learning rule for the model's parameters and a new training strategy proposed by Hinton [363] served as a foundational basis for the development of an original learning paradigm: by merging the features of the Forward Forward algorithm – which is grounded in the concept of local layer-wise learning via a neuron activity-dependent goodness function – with the robustness provided by backpropagation's weight adjustments along gradient directions, a 'Locally Backpropagated Forward Forward' learning strategy has been proposed. This idea, which hinges on a mixed training procedure applied to blocks of a partitioned feedforward network, led to a scientific publication at the International Joint Conference on Neural Networks:

- Giampaolo, F., Izzo, S., Prezioso, E., & Piccialli, F. (2023, June). Investigating Random Variations of the Forward-Forward Algorithm for Training Neural Networks. In 2023 International Joint Conference on Neural Networks (IJCNN) (pp. 1-7). IEEE.

Additionally, the concept of segmenting a neural architecture and training its parts through this new paradigm has been applied to Federated Learning. This approach tackles a key challenge in this field: accommodating variations in computational power and resources among clients, in the context distributed training on heterogeneous devices.

8.3 Final Thoughts

Throughout the discussion, a range of critical themes and insights has emerged, and, like any scientific domain, challenges have also arisen: the scalability of certain architectures, the interpretability of deep neural networks, and the search for more efficient learning algorithms, just to mention few examples, remain pressing concerns. However, one of the evident takeaways is the significant impact and relevance of neural networks in modern technology. And as technology continues to advance, the role of neural networks becomes increasingly central, emphasizing their foundational status in the broader landscape of computational methodologies. In this context, the true power and potential of neural networks don't merely lie in their existing capabilities but in the myriad ways they can be adapted, innovated upon and enhanced. Whether it's the creation of innovative architectures, the evolution of learning procedures, or the modification and introduction of neural units and loss functions, every component of a neural network presents an opportunity for innovation. A deep understanding of the basic mechanisms and details of the learning processes allows researchers to identify more areas where they can introduce new methodologies. This highlights how essential it is to thoroughly understand neural networks, both for their current use and for their potential in future computational research.

Bibliography

[1] J. Schmidhuber, *Deep learning in neural networks: An overview*, Neural networks **61**, 85–117 (2015).

[2] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature **521**, 436–444 (2015).

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, MIT press (2016).

[4] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, Advances in neural information processing systems **27** (2014).

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, Advances in neural information processing systems , 5868–5877 (2017).

[6] W. S. McCulloch and W. Pitts, *A logical calculus of the ideas immanent in nervous activity*, The bulletin of mathematical biophysics **5**, 115–133 (1943).

[7] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and retrieval in the brain*, Psychological Review **65**, 386–408 (1958).

[8] M. Minsky and S. Papert, *Perceptrons: An introduction to computational geometry*, MIT press (1969).

[9] B. Widrow and M. E. Hoff, *Adaptive switching circuits*, IRE WESCON Convention Record **4**, 96–104 (1960).

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, Nature **323**, 533–536 (1986).

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial networks*, Communications of the ACM **63**, 139–144 (2020).

[12] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. *How transferable are features in deep neural networks?* In *Advances in neural information processing systems*, pages 3320–3328, (2014).

[13] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, Proceedings of the thirteenth international conference on artificial intelligence and statistics , 249–256 (2010).

[14] Y. Bengio, *Practical recommendations for gradient-based training of deep architectures*, Neural networks: Tricks of the trade , 437–478 (2012).

[15] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, Proceedings of the IEEE conference on computer vision and pattern recognition , 770–778 (2016).

[16] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980 (2014).

[17] S. Barocas and A. D. Selbst, *Big data's disparate impact*, California Law Review **104**, 671 (2016).

[18] J. Buolamwini and T. Gebru. *Gender shades: Intersectional accuracy disparities in commercial gender classification.* In *Proceedings of Machine Learning Research*, volume 81, pages 1–15, (2018).

[19] S. J. Russell and P. Norvig, *Artificial intelligence: A modern approach*, Prentice Hall (2009).

[20] B. Goertzel, *Artificial general intelligence: Concept, state of the art, and future prospects*, Journal of Artificial General Intelligence **5**, 1–48 (2014).

[21] D. H. Hubel and T. N. Wiesel, *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex*, The Journal of physiology **160**, 106 (1962).

[22] K. Fukushima, *Neocognitron: A hierarchical neural network capable of visual pattern recognition*, Neural Networks **1**, 119–130 (1988).

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE **86**, 2278–2324 (1998).

[24] J. C. Myburgh, C. Mouton, and M. H. Davel. *Tracking translation invariance in CNNs.* In *Southern African Conference for Artificial Intelligence Research*, pages 282–295. Springer, (2020).

[25] J. L. Elman, *Finding Structure in Time*, Cognitive Science **14**, 179–211 (1990).

[26] M. I. Jordan, *Serial Order: A Parallel Distributed Processing Approach*, Advances in Psychology **121**, 471–495 (1997).

[27] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural computation **9**, 1735–1780 (1997).

[28] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation.* In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, (2014). Association for Computational Linguistics.

[29] J. Park and I. W. Sandberg, *Universal approximation using radial-basis-function networks*, Neural computation **3**, 246–257 (1991).

[30] S. Haykin, *Neural networks: A comprehensive foundation*, Pearson Education (1999).

[31] W. Maass, *Networks of spiking neurons: The third generation of neural network models*, Neural Networks **10**, 1659–1671 (1997).

[32] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, *Event-driven random back-propagation: Enabling neuromorphic deep learning machines*, Frontiers in neuroscience **11**, 324 (2017).

[33] P. Baldi and P. Sadowski. *Understanding dropout.* In *Advances in neural information processing systems*, pages 2814–2822, (2013).

[34] M. Schuld, I. Sinayskiy, and F. Petruccione, *The quest for a Quantum Neural Network*, Quantum Information Processing **13**, 2567–2586 (2014).

[35] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximators*, Neural networks **2**, 359–366 (1989).

[36] S. Hochreiter, *The vanishing gradient problem during learning recurrent neural nets and problem solutions*, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **6**, 107–116 (1998).

[37] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Efficient BackProp*, Neural networks: Tricks of the trade , 9–48 (2012).

[38] V. Nair and G. E. Hinton. *Rectified linear units improve restricted boltzmann machines.* In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, (2010).

[39] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, *Dying relu and initialization: Theory and numerical examples*, arXiv preprint arXiv:1903.06733 (2019).

[40] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, Proceedings of the IEEE international conference on computer vision , 1026–1034 (2015).

[41] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al. *Rectifier nonlinearities improve neural network acoustic models.* In *Proc. icml*, volume 30, page 3. Atlanta, GA, (2013).

[42] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. *Fast and accurate deep network learning by exponential linear units (elus).* In *4th International Conference on Learning Representations, ICLR 2016*, (2016).

[43] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, arXiv preprint arXiv:1810.04805 (2018).

[44] D. Hendrycks and K. Gimpel, *Gaussian error linear units (gelus)*, arXiv preprint arXiv:1606.08415 (2016).

[45] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., *Language Models are Few-Shot Learners*, Advances in Neural Information Processing Systems **33** (2020).

[46] P. Ramachandran, B. Zoph, and Q. V. Le, *Searching for activation functions*, arXiv preprint arXiv:1710.05941 (2017).

[47] A. Radford, L. Metz, and S. Chintala. *Unsupervised representation learning with deep convolutional generative adversarial networks.* In *arXiv preprint arXiv:1511.06434*, (2015).

[48] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, arXiv preprint arXiv:1704.04861 (2017).

[49] F. Chollet. *Xception: Deep learning with depthwise separable convolutions.* In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, (2017).

[50] R. Pascanu, T. Mikolov, and Y. Bengio. *On the difficulty of training recurrent neural networks.* In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 1310–1318, (2013).

[51] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling*, arXiv preprint arXiv:1412.3555 (2014).

[52] R. Jozefowicz, W. Zaremba, and I. Sutskever, *An empirical exploration of recurrent network architectures*, Proceedings of the 32nd International Conference on Machine Learning (ICML-15) , 2342–2350 (2015).

[53] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, *Convolutional LSTM network: A machine learning approach for precipitation nowcasting*, Advances in neural information processing systems **28** (2015).

[54] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. *Set transformer: A framework for attention-based permutation-invariant neural networks.* In *International Conference on Machine Learning*, pages 3744–3753. PMLR, (2019).

[55] S. Ioffe and C. Szegedy. *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 448–456, (2015).

[56] J. L. Ba, R. Kiros, and G. Hinton, *Layer normalization*, arXiv preprint arXiv:1607.06450 (2016).

[57] T. Salimans and D. P. Kingma, *Weight normalization: A simple reparam-*

eterization to accelerate training of deep neural networks, arXiv preprint arXiv:1602.07868 (2016).

[58] Y. Wu and K. He. *Group normalization.* In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, (2018).

[59] G. Klambauer, T. Unterthiner, S. Hochreiter, et al., *Self-normalizing neural networks*, arXiv preprint arXiv:1706.02515 (2017).

[60] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, *Dynamic neural networks: A survey*, IEEE Transactions on Pattern Analysis and Machine Intelligence **44**, 7436–7456 (2021).

[61] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, NY (2006).

[62] P. J. Huber, *Robust statistics*, John Wiley & Sons (2004).

[63] E. L. Lehmann and G. Casella, *Theory of point estimation*, Springer Science & Business Media (2006).

[64] F. Chollet. *Keras.* `https://github.com/fchollet/keras`, (2015).

[65] S. Kullback and R. A. Leibler, *On information and sufficiency*, The annals of mathematical statistics **22**, 79–86 (1951).

[66] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley & Sons, Hoboken, NJ, 2nd edition (2006).

[67] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, arXiv preprint arXiv:1312.6114 (2013).

[68] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. *Focal Loss for Dense Object Detection.* In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988. IEEE, (2017).

[69] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, *Signature verification using a" siamese" time delay neural network*, Advances in neural information processing systems **6** (1993).

[70] S. Chopra, R. Hadsell, and Y. LeCun. *Learning a similarity metric discriminatively, with application to face verification.* In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 539–546. IEEE, (2005).

[71] E. Hoffer and N. Ailon, *Deep metric learning using triplet network*, International Workshop on Similarity-Based Pattern Recognition , 84–92 (2015).

[72] F. Schroff, D. Kalenichenko, and J. Philbin. *FaceNet: A unified embedding for face recognition and clustering.* In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, (2015).

[73] A. Hermans, L. Beyer, and B. Leibe, *In defense of the triplet loss for person re-identification*, arXiv preprint arXiv:1703.07737 (2017).

[74] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. *Distributed representations of words and phrases and their compositionality.* In *Advances in Neural Information Processing Systems*, pages 3111–3119, (2013).

[75] B. Singh, H. Li, A. Sharma, and L. S. Davis. *R-fcn-3000 at 30fps: Decoupling detection and classification.* In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1081–1090, (2018).

[76] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu, *CosFace: Large Margin Cosine Loss for Deep Face Recognition*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition , 5265–5274 (2018).

[77] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese. *Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regres-*

sion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 658–666, (2019).

[78] K. He, G. Gkioxari, P. Dollár, and R. Girshick. *Mask R-CNN.* In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2961–2969, (2017).

[79] M. A. Rahman and Y. Wang. *Optimizing intersection-over-union in deep neural networks for image segmentation.* In *International symposium on visual computing*, pages 234–244. Springer, (2016).

[80] S. S. M. Salehi, D. Erdogmus, and A. Gholipour, *Tversky loss function for image segmentation using 3D fully convolutional deep networks*, arXiv preprint arXiv:1706.05721 (2017).

[81] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press (2004).

[82] A. K. Jain, R. P. Duin, and J. Mao, *Statistical Pattern Recognition: A Review*, IEEE Transactions on Pattern Analysis and Machine Intelligence **22**, 4–37 (2000).

[83] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, et al. *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*, (2001).

[84] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. *On the importance of initialization and momentum in deep learning.* In *International conference on machine learning*, pages 1139–1147. PMLR, (2013).

[85] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization.* In *Advances in neural information processing systems*, pages 2933–2941, (2014).

[86] S. Ruder, *An overview of gradient descent optimization algorithms*, arXiv preprint arXiv:1609.04747 (2016).

[87] L. Bottou. *Large-scale machine learning with stochastic gradient descent.* In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, (2010).

[88] J. Duchi, E. Hazan, and Y. Singer. *Adaptive subgradient methods for online learning and stochastic optimization.* In *Journal of Machine Learning Research*, volume 12, pages 2121–2159, (2011).

[89] T. Tieleman, G. Hinton, et al., *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural networks for machine learning **4**, 26–31 (2012).

[90] J. Bergstra and Y. Bengio. *Random search for hyper-parameter optimization.* In *Journal of Machine Learning Research*, volume 13, pages 281–305, (2012).

[91] J. Snoek, H. Larochelle, and R. P. Adams. *Practical Bayesian optimization of machine learning algorithms.* In *Advances in neural information processing systems*, pages 2951–2959, (2012).

[92] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting*, The Journal of Machine Learning Research **15**, 1929–1958 (2014).

[93] L. Prechelt. *Early stopping-but when?* In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, (2002).

[94] C. Shorten and T. M. Khoshgoftaar, *A survey on Image Data Augmentation for Deep Learning*, Journal of Big Data **6**, 60 (2019).

[95] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, *Spectral normalization for generative adversarial networks*, arXiv preprint arXiv:1802.05957 (2018).

[96] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer Science & Business Media (2009).

[97] *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, Journal of Computational physics **378**, 686–707 (2019).

[98] I. S. Alex Krizhevsky and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, Advances in neural information processing systems (2012).

[99] O. Parkhi, A. Vedaldi, and A. Zisserman. *Deep face recognition.* In *BMVC 2015-Proceedings of the British Machine Vision Conference 2015.* British Machine Vision Association, (2015).

[100] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. van Ginneken, and C. I. Sánchez, *A survey on deep learning in medical image analysis*, Medical image analysis **42**, 60–88 (2017).

[101] Y. Kim, *Convolutional neural networks for sentence classification*, arXiv preprint arXiv:1408.5882 (2014).

[102] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al., *End to end learning for self-driving cars*, arXiv preprint arXiv:1604.07316 (2016).

[103] J. Donahue, L. Anne Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell, *Long-term recurrent convolutional networks for visual recognition and description*, IEEE Transactions on Pattern Analysis and Machine Intelligence **39**, 677–691 (2017).

[104] D. Tang, B. Qin, and T. Liu. *Document Modeling with Gated Recurrent Neural Network for Sentiment Classification.* In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1422–1432, (2015).

[105] A. Graves, A.-r. Mohamed, and G. Hinton. *Speech recognition with deep recurrent neural networks.* In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, (2013).

[106] Y. Liang, S. Ke, J. Zhang, X. Yi, and Y. Zheng. *Geoman: Multi-level attention networks for geo-sensory time series prediction.* In *IJCAI*, volume 2018, pages 3428–3434, (2018).

[107] J.-P. Briot, G. Hadjeres, and F. Pachet, *Deep Learning Techniques for Music Generation - A Survey*, arXiv preprint arXiv:1709.01620 (2017).

[108] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, arXiv preprint arXiv:1409.0473 (2014).

[109] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, *WaveNet: A generative model for raw audio*, arXiv preprint arXiv:1609.03499 (2016).

[110] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. *Show, attend and tell: Neural image caption generation with visual attention.* In *International conference on machine learning*, pages 2048–2057, (2015).

[111] G. E. Hinton and R. R. Salakhutdinov, *Reducing the dimensionality of data with neural networks*, Science **313**, 504–507 (2006).

[112] M. Sakurada and T. Yairi. *Anomaly detection using autoencoders with nonlinear dimensionality reduction.* In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, page 4, (2014).

[113] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, *Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion*, Journal of Machine Learning Research **11**, 3371–3408 (2010).

[114] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. *Greedy layer-wise training of deep networks.* In *Advances in neural information processing systems*, pages 153–160, (2007).

[115] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, *Why does unsupervised pre-training help deep learning?*, Journal of Machine Learning Research **11**, 625–660 (2010).

[116] C. Doersch, *Tutorial on variational autoencoders*, arXiv preprint arXiv:1606.05908 (2016).

[117] D. P. Kingma, S. Mohamed, D. Jimenez Rezende, and M. Welling, *Semi-supervised learning with deep generative models*, Advances in neural information processing systems **27** (2014).

[118] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. *InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets.* In *Advances in neural information processing systems*, pages 2172–2180, (2016).

[119] Y. Burda, R. Grosse, and R. Salakhutdinov, *Importance weighted autoencoders*, arXiv preprint arXiv:1509.00519 (2015).

[120] J. Lucas, G. Tucker, R. Grosse, and M. Norouzi, *Understanding posterior collapse in generative latent variable models*, (2019).

[121] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. *Unpaired image-to-image trans-*

lation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, (2017).

[122] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. *Photo-realistic single image super-resolution using a generative adversarial network.* In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, (2017).

[123] M. Mirza and S. Osindero, *Conditional generative adversarial nets*, arXiv preprint arXiv:1411.1784 (2014).

[124] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, *Improved techniques for training gans*, Advances in neural information processing systems **29** (2016).

[125] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, *Unrolled generative adversarial networks*, arXiv preprint arXiv:1611.02163 (2016).

[126] B. Ghorbani, O. Firat, M. Freitag, A. Bapna, M. Krikun, X. Garcia, C. Chelba, and C. Cherry, *Scaling laws for neural machine translation*, arXiv preprint arXiv:2109.07740 (2021).

[127] Y. Liu, *Fine-tune BERT for extractive summarization*, arXiv preprint arXiv:1903.10318 (2019).

[128] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, *Exploring the limits of transfer learning with a unified text-to-text transformer*, The Journal of Machine Learning Research **21**, 5485–5551 (2020).

[129] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., *Language models are unsupervised multitask learners*, OpenAI blog **1**, 9 (2019).

[130] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al., *An image is*

worth 16x16 words: Transformers for image recognition at scale, arXiv preprint arXiv:2010.11929 (2020).

[131] Z. Zhang, Y. Chen, and V. Saligrama. *Efficient training of very deep neural networks for supervised hashing.* In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1487–1495, (2016).

[132] S. Bai, J. Z. Kolter, and V. Koltun, *Deep equilibrium models*, Advances in neural information processing systems **32** (2019).

[133] H. Jaeger, *The "echo state" approach to analysing and training recurrent neural networks-with an erratum note*, Bonn, Germany: German National Research Center for Information Technology GMD Technical Report **148**, 13 (2001).

[134] G. Hinton, S. Sabour, and N. Frosst. *Matrix capsules with EM routing.* In *International Conference on Learning Representations*, (2018).

[135] S. Sabour, N. Frosst, and G. E. Hinton. *Dynamic routing between capsules.* In *Advances in neural information processing systems*, volume 30, (2017).

[136] A. Graves, G. Wayne, and I. Danihelka, *Neural Turing Machines*, arXiv preprint arXiv:1410.5401 (2014).

[137] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, *Neural Ordinary Differential Equations*, Advances in Neural Information Processing Systems **31** (2018).

[138] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, *The graph neural network model*, IEEE transactions on neural networks **20**, 61–80 (2008).

[139] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, *Spectral networks and locally connected networks on graphs*, arXiv preprint arXiv:1312.6203 (2013).

[140] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, arXiv preprint arXiv:1609.02907 (2016).

[141] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, *Graph Attention Networks*, arXiv preprint arXiv:1710.10903 (2017).

[142] D. Mesquita, A. Souza, and S. Kaski, *Rethinking pooling in graph neural networks*, Advances in Neural Information Processing Systems **33**, 2220–2231 (2020).

[143] X. Wang and M. Zhang. *How powerful are spectral graph neural networks.* In *International Conference on Machine Learning*, pages 23341–23362. PMLR, (2022).

[144] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al., *Relational inductive biases, deep learning, and graph networks*, arXiv preprint arXiv:1806.01261 (2018).

[145] Q. Li, Z. Han, and X.-M. Wu, *Deeper insights into graph convolutional networks for semi-supervised learning*, Thirty-Second AAAI Conference on Artificial Intelligence (2018).

[146] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, *Improving language understanding by generative pre-training*, Openai blog **1** (2018).

[147] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., *Mastering the game of Go with deep neural networks and tree search*, Nature **529**, 484–489 (2016).

[148] P. J. Brockwell and R. A. Davis, *Introduction to time series and forecasting*, Springer (2002).

[149] H. Lütkepohl, *New introduction to multiple time series analysis*, Springer Science & Business Media (2005).

[150] G. E. Box and G. M. Jenkins, *Time series analysis. Forecasting and control*, Holden-Day Series in Time Series Analysis (1976).

[151] G. Zhang, B. E. Patuwo, and M. Y. Hu, *Forecasting with artificial neural networks:: The state of the art*, International journal of forecasting **14**, 35–62 (1998).

[152] C. C. Holt, *Forecasting seasonals and trends by exponentially weighted moving averages*, O.N.R. Research Memorandum **52** (1957).

[153] P. R. Winters, *Forecasting sales by exponentially weighted moving averages*, Management Science **6**, 324–342 (1960).

[154] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*, John Wiley & Sons (2015).

[155] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*, OTexts (2018).

[156] R. E. Kalman, *A new approach to linear filtering and prediction problems*, Journal of Basic Engineering **82**, 35–45 (1960).

[157] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning, *STL: A Seasonal-Trend Decomposition Procedure Based on Loess*, Journal of Official Statistics **6**, 3–73 (1990).

[158] G. M. Jenkins and M. Priestley, *The spectral analysis of time-series*, Journal of the Royal Statistical Society: Series B (Methodological) **19**, 1–12 (1957).

[159] P. Jing, Y. Su, X. Jin, and C. Zhang, *High-order temporal correlation model learning for time-series prediction*, IEEE transactions on cybernetics **49**, 2385–2397 (2018).

[160] Q. Shi, J. Yin, J. Cai, A. Cichocki, T. Yokota, L. Chen, M. Yuan, and J. Zeng. *Block Hankel tensor ARIMA for multiple short time series forecasting.* In *Pro-*

ceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 5758–5766, (2020).

[161] S. J. Taylor and B. Letham, *Forecasting at scale*, The American Statistician **72**, 37–45 (2018).

[162] K.-J. Kim, *Financial time series forecasting using support vector machines*, Neurocomputing **55**, 307–319 (2003).

[163] K. Amasyali and N. M. El-Gohary, *A review of data-driven building energy consumption prediction studies*, Renewable and Sustainable Energy Reviews **81**, 1192–1205 (2018).

[164] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, *DeepAR: Probabilistic forecasting with autoregressive recurrent networks*, International Journal of Forecasting **36**, 1181–1191 (2020).

[165] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, *N-BEATS: Neural basis expansion analysis for interpretable time series forecasting*, arXiv preprint arXiv:1905.10437 (2019).

[166] G. P. Zhang, *Time series forecasting using a hybrid ARIMA and neural network model*, Neurocomputing **50**, 159–175 (2003).

[167] X. Wang and M. Meng, *A Hybrid Neural Network and ARIMA Model for Energy Consumption Forcasting.*, J. Comput. **7**, 1184–1190 (2012).

[168] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski, *Deep state space models for time series forecasting*, Advances in neural information processing systems **31**, 7785–7794 (2018).

[169] H.-F. Yu, N. Rao, and I. S. Dhillon. *Temporal Regularized Matrix Factorization for High-dimensional Time Series Prediction.* In *NIPS*, pages 847–855, (2016).

[170] D. Salinas, M. Bohlke-Schneider, L. Callot, R. Medico, and J. Gasthaus, *High-dimensional multivariate forecasting with low-rank gaussian copula processes*, Advances in neural information processing systems **32** (2019).

[171] R. Sen, H.-F. Yu, and I. S. Dhillon, *Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting*, Advances in neural information processing systems **32** (2019).

[172] G. Lai, W.-C. Chang, Y. Yang, and H. Liu. *Modeling long-and short-term temporal patterns with deep neural networks.* In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pages 95–104, (2018).

[173] S. Smyl, *A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting*, International Journal of Forecasting **36**, 75–85 (2020).

[174] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang. *Connecting the dots: Multivariate time series forecasting with graph neural networks.* In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 753–763, (2020).

[175] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister, *Temporal fusion transformers for interpretable multi-horizon time series forecasting*, International Journal of Forecasting **37**, 1748–1764 (2021).

[176] H. Wu, J. Xu, J. Wang, and M. Long, *Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting*, Advances in Neural Information Processing Systems **34**, 22419–22430 (2021).

[177] D. Opitz and R. Maclin, *Popular ensemble methods: An empirical study*, Journal of artificial intelligence research **11**, 169–198 (1999).

[178] R. Polikar, *Ensemble based systems in decision making*, IEEE Circuits and Systems Magazine **6**, 21–45 (2006).

[179] D. H. Wolpert and W. G. Macready, *No free lunch theorems for optimization*, IEEE transactions on evolutionary computation **1**, 67–82 (1997).

[180] S. Lee, N. Ybarra, K. Jeyaseelan, S. Faria, N. Kopek, P. Brisebois, J. D. Bradley, C. Robinson, J. Seuntjens, and I. El Naqa, *Bayesian network ensemble as a multivariate strategy to predict radiation pneumonitis risk*, Medical physics **42**, 2421–2430 (2015).

[181] A. Galicia, R. Talavera-Llames, A. Troncoso, I. Koprinska, and F. Martínez-Álvarez, *Multi-step forecasting for big data time series based on ensemble learning*, Knowledge-Based Systems **163**, 830–841 (2019).

[182] D. Shaub, *Fast and accurate yearly time series forecasting with forecast combinations*, International Journal of Forecasting **36**, 116–120 (2020).

[183] G. Xie, Y. Qian, and S. Wang, *A decomposition-ensemble approach for tourism forecasting*, Annals of Tourism Research **81**, 102891 (2020).

[184] S. Al-Dahidi, P. Baraldi, E. Zio, and E. Legnani. *A dynamic weighting ensemble approach for wind energy production prediction.* In *2017 2nd International Conference on System Reliability and Safety (ICSRS)*, pages 296–302. IEEE, (2017).

[185] M. H. D. M. Ribeiro and L. dos Santos Coelho, *Ensemble approach based on bagging, boosting and stacking for short-term prediction in agribusiness time series*, Applied Soft Computing **86**, 105837 (2020).

[186] A. Chitra and S. Uma, *An ensemble model of multiple classifiers for time series prediction*, International Journal of Computer Theory and Engineering **2**, 454 (2010).

[187] W. Shen, V. Babushkin, Z. Aung, and W. L. Woon, *An ensemble model for day-ahead electricity demand time series forecasting*, , 51–62 (2013).

[188] A. Vlasenko, N. Vlasenko, O. Vynokurova, Y. Bodyanskiy, and D. Peleshko, *A novel ensemble neuro-fuzzy model for financial time series forecasting*, Data **4**, 126 (2019).

[189] P. Musikawan, K. Sunat, and Y. Kongsorot, *Wind power forecasting using a heterogeneous ensemble of decomposition-based NNRW techniques*, ECTI Transactions on Computer and Information Technology **14**, 122–138 (2020).

[190] S. Kaushik, A. Choudhury, N. Dasgupta, S. Natarajan, L. A. Pickett, and V. Dutt. *Ensemble of multi-headed machine learning architectures for time-series forecasting of healthcare expenditures.* In P. Johri, J. K. Verma, and S. Paul, editors, *Applications of Machine Learning*, pages 199–216, Singapore, (2020). Springer Singapore.

[191] S. Krstanovic and H. Paulheim. *Ensembles of recurrent neural networks for robust time series forecasting.* In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 34–46. Springer, (2017).

[192] J. Chen, G.-Q. Zeng, W. Zhou, W. Du, and K.-D. Lu, *Wind speed forecasting using nonlinear-learning ensemble of deep learning time series prediction and extremal optimization*, Energy Conversion and Management **165**, 681–695 (2018).

[193] M. Tan, S. Yuan, S. Li, Y. Su, H. Li, and F. He, *Ultra-short-term industrial power demand forecasting using LSTM based hybrid ensemble learning*, IEEE Transactions on Power Systems **35**, 2937–2948 (2019).

[194] M. Kück, S. F. Crone, and M. Freitag. *Meta-learning with neural networks and landmarking for forecasting model selection: an empirical evaluation of different*

feature sets applied to industry data. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 1499–1506. IEEE, (2016).

[195] J. S. Armstrong, *Combining forecasts*, Springer (2001).

[196] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, *The M4 competition: Results, findings, conclusion and way forward*, International Journal of Forecasting **34**, 802–808 (2018).

[197] M. Mohammadi and A. Al-Fuqaha, *Enabling cognitive smart cities using big data and machine learning: Approaches and challenges*, IEEE Communications Magazine **56**, 94–101 (2018).

[198] C. Stracener, Q. Samelson, J. Mackie, and M. Ihaza, *The Internet of Things grows artificial intelligence and data sciences*, IT Professional **21**, 55–62 (2019).

[199] T. Lin, H. Rivano, and F. Le Mouël, *A survey of smart parking solutions*, IEEE Transactions on Intelligent Transportation Systems **18**, 3229–3253 (2017).

[200] A. Khanna and R. Anand. *IoT based smart parking system.* In *2016 international conference on internet of things and applications (IOTA)*, pages 266–270. IEEE, (2016).

[201] G. Amato, F. Carrara, F. Falchi, C. Gennaro, C. Meghini, and C. Vairo, *Deep learning for decentralized parking lot occupancy detection*, Expert Systems with Applications **72**, 327–334 (2017).

[202] A. Camero, J. Toutouh, D. H. Stolfi, and E. Alba. *Evolutionary deep learning for car park occupancy prediction in smart cities.* In *Proceedings of the International Conference on Learning and Intelligent Optimization*, pages 386–401. Springer, (2018).

[203] I. Aydin, M. Karakose, and E. Karakose. *A navigation and reservation based smart parking platform using genetic optimization for smart cities.* In *2017 5th*

International Istanbul Smart Grid and Cities Congress and Fair (ICSG), pages 120–124, (2017).

[204] E. I. Vlahogianni, K. Kepaptsoglou, V. Tsetsos, and M. G. Karlaftis, *A Real-Time Parking Prediction System for Smart Cities*, Journal of Intelligent Transportation Systems **20**, 192–204 (2016).

[205] L. Breiman, *Random forests*, Machine learning **45**, 5–32 (2001).

[206] L. Breiman, *Bagging predictors*, Machine learning **24**, 123–140 (1996).

[207] R. Tibshirani, *Regression shrinkage and selection via the lasso*, Journal of the Royal Statistical Society Series B: Statistical Methodology **58**, 267–288 (1996).

[208] A. E. Hoerl and R. W. Kennard, *Ridge regression: Biased estimation for nonorthogonal problems*, Technometrics **12**, 55–67 (1970).

[209] J. H. Holland, *Outline for a logical theory of adaptive systems*, J. ACM **9**, 297–314 (1962).

[210] J. J. Grefenstette. *Genetic algorithms and machine learning.* In *Proceedings of the sixth annual conference on Computational learning theory*, pages 3–4, (1993).

[211] S. Varma and R. Simon, *Bias in error estimation when using cross-validation for model selection*, BMC Bioinformatics **7**, 91 (2006).

[212] C. Bergmeir and J. M. Benítez, *On the use of cross-validation for time series predictor evaluation*, Information Sciences **191**, 192–213 (2012).

[213] M. Malik, S. Abdallah, and M. Ala'raj, *Data mining and predictive analytics applications for the delivery of healthcare services: a systematic literature review*, Annals of Operations Research **270**, 287–312 (2018).

[214] S. L. Harris, J. H. May, and L. G. Vargas, *Predictive analytics model for healthcare planning and scheduling*, European Journal of Operational Research **253**, 121 – 131 (2016).

[215] A. Nelson, D. Herron, G. Rees, and P. Nachev, *Predicting scheduled hospital attendance with artificial intelligence*, npj Digital Medicine **2**, 26 (2019).

[216] A. Alahmar, E. Mohammed, and R. Benlamri. *Application of Data Mining Techniques to Predict the Length of Stay of Hospitalized Patients with Diabetes.* In *2018 4th International Conference on Big Data Innovations and Applications (Innovate-Data)*, pages 38–43, (2018).

[217] H. J. Jiang, C. A. Russo, and M. L. Barrett, *Nationwide Frequency and Costs of Potentially Preventable Hospitalizations, 2006: Statistical Brief #72*, Agency for Healthcare Research and Quality (US), Rockville (MD) (2006).

[218] R. J. Linn, D. L. Hall, and J. Llinas. *Survey of multisensor data fusion systems.* In V. Libby, editor, *Data Structures and Target Classification*, volume 1470, pages 13 – 29. International Society for Optics and Photonics, SPIE, (1991).

[219] G. Kelly. *Data fusion: from primary metrology to process measurement.* In *IMTC/99. Proceedings of the 16th IEEE Instrumentation and Measurement Technology Conference (Cat. No.99CH36309)*, volume 3, pages 1325–1329 vol.3, (1999).

[220] Y. Liang, Z. Gao, J. Gao, R. Wang, and H. Zhao, *Data fusion combined with echo state network for multivariate time series prediction in complex electromechanical system*, Computational and Applied Mathematics **37**, 5920–5934 (2018).

[221] Y. Kim, P. Wang, Y. Zhu, and L. Mihaylova. *A Capsule Network for Traffic Speed Prediction in Complex Road Networks.* In *2018 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, pages 1–6, (2018).

[222] F. Rodrigues, I. Markou, and F. C. Pereira, *Combining time-series and textual data for taxi demand prediction in event areas: A deep learning approach*, Information Fusion **49**, 120–129 (2019).

[223] M. Z. Uddin, M. M. Hassan, A. Alsanad, and C. Savaglio, *A body sensor data fusion and deep recurrent neural network-based behavior recognition approach for robust healthcare*, Information Fusion **55**, 105 – 115 (2020).

[224] T. Li, S. Fong, K. K. Wong, Y. Wu, X.-s. Yang, and X. Li, *Fusing wearable and remote sensing data streams by fast incremental learning with swarm decision table for human activity recognition*, Information Fusion **60**, 41–64 (2020).

[225] C. Habib, A. Makhoul, R. Darazi, and R. Couturier, *Health risk assessment and decision-making for patient monitoring and decision-support using Wireless Body Sensor Networks*, Information Fusion **47**, 10–22 (2019).

[226] D. Zhang, C. Yin, J. Zeng, X. Yuan, and P. Zhang, *Combining structured and unstructured data for predictive models: a deep learning approach*, BMC medical informatics and decision making **20**, 1–11 (2020).

[227] F. Ali, S. El-Sappagh, S. R. Islam, D. Kwak, A. Ali, M. Imran, and K.-S. Kwak, *A smart healthcare monitoring system for heart disease prediction based on ensemble deep learning and feature fusion*, Information Fusion **63**, 208–222 (2020).

[228] Y. Zheng and X. Hu, *Healthcare predictive analytics for disease progression: a longitudinal data fusion approach*, Journal of Intelligent Information Systems **55**, 351–369 (2020).

[229] H. Li and Y. Fan. *Early Prediction Of Alzheimer's Disease Dementia Based On Baseline Hippocampal MRI and 1-Year Follow-Up Cognitive Measures Using Deep Recurrent Neural Networks.* In *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, pages 368–371, (2019).

[230] Y. Yoo, L. W. Tang, T. Brosch, D. K. B. Li, L. Metz, A. Traboulsee, and R. Tam. *Deep Learning of Brain Lesion Patterns for Predicting Future Disease Activity in Patients with Early Symptoms of Multiple Sclerosis.* In *Deep Learning and Data Labeling for Medical Applications*, pages 86–94. Springer International Publishing, (2016).

[231] S. Purwar, R. K. Tripathi, R. Ranjan, and R. Saxena, *Detection of microcytic hypochromia using cbc and blood film features extracted from convolution neural network by different classifiers*, Multimedia Tools and Applications **79**, 4573–4595 (2020).

[232] B. Majeed, J. Peng, A. Li, Y. Lin, and R. I. Delgado, *Forecasting the demand of mobile clinic services at vulnerable communities based on integrated multi-source data*, IISE Transactions on Healthcare Systems Engineering **0**, 1–15 (2020).

[233] P. Rangarajan, S. K. Mody, and M. Marathe, *Forecasting dengue and influenza incidences using a sparse representation of Google trends, electronic health records, and time series data*, PLoS computational biology **15**, e1007518 (2019).

[234] S. Pei, S. Kandula, W. Yang, and J. Shaman, *Forecasting the spatial transmission of influenza in the United States*, Proceedings of the National Academy of Sciences **115**, 2752–2757 (2018).

[235] T. Rekatsinas, S. Ghosh, S. R. Mekaru, E. O. Nsoesie, J. S. Brownstein, L. Getoor, and N. Ramakrishnan. *Sourceseer: Forecasting rare disease outbreaks using multiple data sources.* In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 379–387. SIAM, (2015).

[236] D. A. Dickey and W. A. Fuller, *Distribution of the Estimators for Autoregressive Time Series with a Unit Root*, Journal of the American Statistical Association **74**, 427–431 (1979).

[237] M. Shcherbakov, A. Brebels, N. Shcherbakova, A. Tyukov, T. Janovsky, and V. Kamaev, *A survey of forecast error measures*, World Applied Sciences Journal **24**, 171–176 (2013).

[238] G. Zhang, *Time series forecasting using a hybrid ARIMA and neural network model*, Neurocomputing **50**, 159 – 175 (2003).

[239] F.-M. Tseng, H.-C. Yu, and G.-H. Tzeng, *Combining neural network model with seasonal time series ARIMA model*, Technological Forecasting and Social Change **69**, 71 – 87 (2002).

[240] S. Panigrahi and H. Behera, *A hybrid ETS–ANN model for time series forecasting*, Engineering Applications of Artificial Intelligence **66**, 49 – 59 (2017).

[241] J. Kennedy and R. Eberhart. *Particle Swarm Optimization.* In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948, (1995).

[242] D. Li, L. Luo, W. Zhang, F. Liu, and F. Luo, *A genetic algorithm-based weighted ensemble method for predicting transposon-derived piRNAs*, BMC bioinformatics **17**, 1–11 (2016).

[243] P. Baraldi, F. Mangili, and E. Zio, *A Kalman Filter-Based Ensemble Approach With Application to Turbine Creep Prognostics*, IEEE Transactions on Reliability **61**, 966–977 (2012).

[244] E. Spiliotis, K. Nikolopoulos, and V. Assimakopoulos, *Tales from tails: On the empirical distributions of forecasting errors and their implication to risk*, International Journal of Forecasting **35**, 687–698 (2019).

[245] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, *The M4 Competition: 100,000 time series and 61 forecasting methods*, International Journal of Forecasting **36**, 54–74 (2020).

[246] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, *M5 accuracy competition: Results, findings, and conclusions*, International Journal of Forecasting **38**, 1346–1364 (2022).

[247] J. M. Bates and C. W. J. Granger, *The combination of forecasts*, J. Operat. Res. Soc. **20**, 451–468 (1969).

[248] R. T. Clemen, *Combining forecasts: A review and annotated bibliography*, International journal of forecasting **5**, 559–583 (1989).

[249] R. Adhikari. *A neural network based linear ensemble framework for time series forecasting.* In *Neurocomputing*, volume 157, pages 231–242. Elsevier, (2015).

[250] S. Al-Dahidi, P. Baraldi, E. Zio, and E. Legnani. *A dynamic weighting ensemble approach for wind energy production prediction.* In *2017 2nd International Conference on System Reliability and Safety (ICSRS)*, pages 296–302. IEEE, (2017).

[251] M. Kück, S. F. Crone, and M. Freitag. *Meta-learning with neural networks and landmarking for forecasting model selection an empirical evaluation of different feature sets applied to industry data.* In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 1499–1506. IEEE, (2016).

[252] G. Li, S. Wang, et al., *Sunspots time-series prediction based on complementary ensemble empirical mode decomposition and wavelet neural network*, Mathematical Problems in Engineering **2017** (2017).

[253] W. Liu, W. D. Liu, and J. Gu, *Forecasting oil production using ensemble empirical model decomposition based Long Short-Term Memory neural network*, Journal of Petroleum Science and Engineering **189**, 107013 (2020).

[254] Z. Lu, J. Xia, M. Wang, Q. Nie, and J. Ou, *Short-term traffic flow forecasting via multi-regime modeling and ensemble learning*, Applied Sciences **10**, 356 (2020).

[255] E. Soares, P. C. J. Costa, B. Costa, and D. Leite, *Ensemble of evolving data clouds and fuzzy models for weather time series prediction*, Applied Soft Computing **64**, 445–453 (2018).

[256] L. Wang, Z. Wang, H. Qu, and S. Liu, *Optimal forecast combination based on neural networks for time series forecasting*, Applied Soft Computing **66**, 1–17 (2018).

[257] B. Yang, Z.-J. Gong, and W. Yang. *Stock market index prediction using deep neural network ensemble.* In *2017 36th Chinese Control Conference (CCC)*, pages 3882–3887. IEEE, (2017).

[258] L. Yu, Y. Zhao, and L. Tang, *Ensemble forecasting for complex time series using sparse representation and neural networks*, Journal of Forecasting **36**, 122–138 (2017).

[259] X. Zhang, Y. Li, S. Lu, H. F. Hamann, B.-M. Hodge, and B. Lehman, *A solar time based analog ensemble method for regional solar power forecasting*, IEEE Transactions on Sustainable Energy **10**, 268–279 (2018).

[260] W. Zhao, F. Wang, and D. Niu, *The application of support vector machine in load forecasting*, Journal of Computational Physics **7**, 1615–1622 (2012).

[261] Y. Zhao, J. Li, and L. Yu, *A deep learning ensemble approach for crude oil price forecasting*, Energy Economics **66**, 9–16 (2017).

[262] A. Kausar, M. Ishtiaq, M. A. Jaffar, and A. M. Mirza. *Optimization of ensemble based decision using PSO.* In *Proceedings of the World Congress on Engineering*, volume 1, pages 1–6. IAENG, (2010).

[263] Z. Wu and Y. Chen. *Genetic algorithm based selective neural network ensemble.* In *IJCAI-01: Proceedings of the Seventeenth International Joint Conference on*

Artificial Intelligence, volume 3, pages 1323–1328. Morgan Kaufmann Publishers Inc., (2001).

[264] C. Chen, K. Petty, A. Skabardonis, P. Varaiya, and Z. Jia, *Freeway performance measurement system: mining loop detector data*, Transportation Research Record **1748**, 96–102 (2001).

[265] C. Nadeau and Y. Bengio, *Inference for the generalization error*, Advances in neural information processing systems **12** (1999).

[266] A. M. Saxe, J. L. McClelland, and S. Ganguli, *Exact solutions to the nonlinear dynamics of learning in deep linear neural networks*, arXiv preprint arXiv:1312.6120 (2013).

[267] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, MIT press (2018).

[268] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, *Policy gradient methods for reinforcement learning with function approximation*, Advances in neural information processing systems **12** (1999).

[269] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, arXiv preprint arXiv:1707.06347 (2017).

[270] C. J. Watkins and P. Dayan, *Q-learning*, Machine learning **8**, 279–292 (1992).

[271] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., *Human-level control through deep reinforcement learning*, Nature **518**, 529–533 (2015).

[272] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press (1975).

[273] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* In *Doctoral dissertation, University of Michigan*, (1975).

[274] W. M. Spears, K. A. De Jong, T. Bäck, D. B. Fogel, and H. De Garis. *An overview of evolutionary computation.* In *European conference on machine learning*, pages 442–459. Springer, (1993).

[275] D. B. Fogel, *Evolutionary computation: toward a new philosophy of machine intelligence*, John Wiley & Sons, 3 edition (2006).

[276] M. Mitchell, *An introduction to genetic algorithms*, MIT press (1998).

[277] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, Springer (2003).

[278] T. Back, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford University Press (1996).

[279] C. A. C. Coello, *Evolutionary algorithms for solving multi-objective problems*, Springer (2007).

[280] D. J. Montana, L. Davis, et al. *Training feedforward neural networks using genetic algorithms.* In *IJCAI*, volume 89, pages 762–767, (1989).

[281] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, *Designing neural networks through neuroevolution*, Nature Machine Intelligence **1**, 24–35 (2019).

[282] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, *Regularized Evolution for Image Classifier Architecture Search*, Proceedings of the AAAI Conference on Artificial Intelligence **33**, 4780–4789 (2019).

[283] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press (1999).

[284] R. Eberhart and J. Kennedy. *A new optimizer using particle swarm theory.* In *MHS'95. Proceedings of the sixth international symposium on micro machine and human science*, pages 39–43. Ieee, (1995).

[285] J. Martens, *Deep learning via Hessian-free optimization*, Proceedings of the 27th International Conference on International Conference on Machine Learning , 735–742 (2010).

[286] J. R. Shewchuk et al. *An introduction to the conjugate gradient method without the agonizing pain*, (1994).

[287] B. A. Olshausen and D. J. Field, *Sparse coding with an overcomplete basis set: A strategy employed by V1?*, Vision Research **37**, 3311–3325 (1997).

[288] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by Simulated Annealing*, Science **220**, 671–680 (1983).

[289] P. Tseng, *Convergence of a block coordinate descent method for nondifferentiable minimization*, Journal of Optimization Theory and Applications **109**, 475–494 (2001).

[290] T. Elsken, J. H. Metzen, and F. Hutter, *Neural architecture search: A survey*, The Journal of Machine Learning Research **20**, 1997–2017 (2019).

[291] M. Tan and Q. Le. *Efficientnet: Rethinking model scaling for convolutional neural networks.* In *International conference on machine learning*, pages 6105–6114. PMLR, (2019).

[292] M. Huh, P. Agrawal, and A. A. Efros, *What makes ImageNet good for transfer learning?*, arXiv preprint arXiv:1608.08614 (2016).

[293] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. *DeCAF: A deep convolutional activation feature for generic visual recognition.* In *International conference on machine learning*, pages 647–655, (2014).

[294] S. J. Pan and Q. Yang, *A survey on transfer learning*, IEEE Transactions on knowledge and data engineering **22**, 1345–1359 (2009).

[295] S. Ruder, *An overview of multi-task learning in deep neural networks*, arXiv preprint arXiv:1706.05098 (2017).

[296] E. S. Olivas, J. D. M. Guerrero, M. Martinez-Sober, J. R. Magdalena-Benedito, L. Serrano, et al., *Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques*, IGI global (2009).

[297] A. Y. Ng. *Feature selection, L 1 vs. L 2 regularization, and rotational invariance.* In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, (2004).

[298] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. *A survey on deep transfer learning.* In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III 27*, pages 270–279. Springer, (2018).

[299] C. Finn, P. Abbeel, and S. Levine, *Model-agnostic meta-learning for fast adaptation of deep networks*, Proceedings of the 34th International Conference on Machine Learning-Volume 70 , 1126–1135 (2017).

[300] S. Thrun and L. Pratt. *Learning to learn: Introduction and overview.* In *Learning to learn*, pages 3–17. Springer, (1998).

[301] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al., *Matching networks for one shot learning*, Advances in neural information processing systems **29** (2016).

[302] J. Snell, K. Swersky, and R. S. Zemel. *Prototypical networks for few-shot learning.* In *Advances in Neural Information Processing Systems*, pages 4077–4087, (2017).

[303] B. Zoph and Q. V. Le, *Neural architecture search with reinforcement learning*, arXiv preprint arXiv:1611.01578 (2016).

[304] H. Liu, K. Simonyan, and Y. Yang, *Darts: Differentiable architecture search*, arXiv preprint arXiv:1806.09055 (2018).

[305] J. Vanschoren, *Meta-learning: A survey*, arXiv preprint arXiv:1810.03548 (2018).

[306] A. Nichol, J. Achiam, and J. Schulman, *On First-Order Meta-Learning Algorithms*, arXiv preprint arXiv:1803.02999 (2018).

[307] Z. Li, F. Zhou, F. Chen, and H. Li, *Meta-SGD: Learning to Learn Quickly for Few-Shot Learning*, arXiv preprint arXiv:1707.09835 (2017).

[308] S. Bechtle, A. Molchanov, Y. Chebotar, E. Grefenstette, L. Righetti, G. Sukhatme, and F. Meier. *Meta learning via learned loss.* In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4161–4168. IEEE, (2021).

[309] Y. Bengio. *Deep learning of representations for unsupervised and transfer learning.* In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 17–36. JMLR Workshop and Conference Proceedings, (2012).

[310] E. Triantafillou, T. Zhu, V. Dumoulin, P. Lamblin, U. Evci, K. Xu, R. Goroshin, C. Gelada, K. Swersky, P.-A. Manzagol, et al., *Meta-dataset: A dataset of datasets for learning to learn from few examples*, arXiv preprint arXiv:1903.03096 (2019).

[311] W.-Y. Chen, Y.-C. F. Liu, Z. Kira, Y. Wang, and J.-B. Huang, *A Closer Look at Few-shot Classification*, arXiv preprint arXiv:1904.04232 (2019).

[312] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen. *Efficient off-policy meta-reinforcement learning via probabilistic context variables.* In *International conference on machine learning*, pages 5331–5340. PMLR, (2019).

[313] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. *The loss surfaces of multilayer networks.* In *Artificial intelligence and statistics*, pages 192–204. PMLR, (2015).

[314] R. Ge, F. Huang, C. Jin, and Y. Yuan, *Escaping from saddle points—online stochastic gradient for tensor decomposition*, Conference on Learning Theory , 797–842 (2015).

[315] L. Bottou, F. E. Curtis, and J. Nocedal, *Optimization methods for large-scale machine learning*, Siam Review **60**, 223–311 (2018).

[316] Z. Wang, F. Hutter, M. Zoghi, D. Matheson, and N. De Feitas, *Bayesian optimization in a billion dimensions via random embeddings*, Journal of Artificial Intelligence Research **55**, 361–387 (2016).

[317] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*, Cambridge University Press (2014).

[318] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, *Understanding deep learning (still) requires rethinking generalization*, Communications of the ACM **64**, 107–115 (2021).

[319] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. *Exploring Generalization in Deep Learning.* In *Advances in Neural Information Processing Systems*, (2017).

[320] M. Belkin, D. Hsu, S. Ma, and S. Mandal, *Reconciling modern machine learning practice and the classical bias-variance tradeoff*, Proceedings of the National Academy of Sciences **116**, 15849–15854 (2019).

[321] S. Du, J. Lee, H. Li, L. Wang, and X. Zhai. *Gradient descent finds global minima of deep neural networks.* In *International conference on machine learning*, pages 1675–1685. PMLR, (2019).

[322] I. Panageas, G. Piliouras, and X. Wang, *First-order methods almost always avoid saddle points: The case of vanishing step-sizes*, Advances in Neural Information Processing Systems **32** (2019).

[323] M. Telgarsky. *Benefits of depth in neural networks.* In *Conference on Learning Theory*, pages 1517–1539, (2016).

[324] B. Poole, S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli. *Exponential expressivity in deep neural networks through transient chaos.* In *Advances in neural information processing systems*, pages 3360–3368, (2016).

[325] Z. C. Lipton, *The mythos of model interpretability*, arXiv preprint arXiv:1606.03490 (2016).

[326] F. Doshi-Velez and B. Kim, *Towards a rigorous science of interpretable machine learning*, arXiv preprint arXiv:1702.08608 (2017).

[327] J. D. Olden and D. A. Jackson, *Illuminating the 'black box': a randomization approach for understanding variable contributions in artificial neural networks*, Ecological Modelling (2002).

[328] J. Chen, L. Song, M. J. Wainwright, and M. I. Jordan. *Learning to Explain: An Information-Theoretic Perspective on Model Interpretation.* In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 883–892. JMLR. org, (2018).

[329] M. Sundararajan, A. Taly, and Q. Yan. *Axiomatic Attribution for Deep Networks.* In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3319–3328. JMLR. org, (2017).

[330] H. Peng, F. Long, and C. Ding, *Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy*, IEEE Transactions on pattern analysis and machine intelligence **27**, 1226–1238 (2005).

[331] J. H. Friedman, *Greedy function approximation: a gradient boosting machine*, Annals of statistics , 1189–1232 (2001).

[332] M. D. Zeiler and R. Fergus, *Visualizing and understanding convolutional networks*, European conference on computer vision (2014).

[333] L. v. d. Maaten and G. Hinton, *Visualizing data using t-SNE*, Journal of machine learning research **9**, 2579–2605 (2008).

[334] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. *Visualizing higher-layer features of a deep network.* University of Montreal, (2009).

[335] N. Kriegeskorte, M. Mur, and P. Bandettini, *Representational similarity analysis–connecting the branches of systems neuroscience*, Frontiers in systems neuroscience **2**, 4 (2008).

[336] C. Olah, N. Cammarata, L. Schubert, G. Goh, M. Petrov, and S. Carter, *Zoom in: An introduction to circuits*, Distill **5**, e00024–000 (2020).

[337] D. V. Carvalho, E. M. Pereira, and J. S. Cardoso, *Machine Learning Interpretability: A Survey on Methods and Metrics*, Electronics **8** (2019).

[338] M. T. Ribeiro, S. Singh, and C. Guestrin. *"Why should I trust you?" Explaining the predictions of any classifier.* In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, (2016).

[339] S. M. Lundberg and S.-I. Lee. *A unified approach to interpreting model predictions.* In *Advances in neural information processing systems*, (2017).

[340] H. Chen, J. D. Janizek, S. Lundberg, and S.-I. Lee, *True to the model or true to the data?*, arXiv preprint arXiv:2006.16234 (2020).

[341] S. Wachter, B. Mittelstadt, and C. Russell, *Counterfactual explanations without*

opening the black box: Automated decisions and the GDPR, Harv. JL & Tech. **31**, 841 (2017).

[342] K. B. Obaid, S. Zeebaree, O. M. Ahmed, et al., *Deep learning models based on image classification: a review*, International Journal of Science and Business **4**, 75–81 (2020).

[343] P. Druzhkov and V. Kustikova, *A survey of deep learning methods and software tools for image classification and object detection*, Pattern Recognition and Image Analysis **26**, 9–15 (2016).

[344] D. W. Otter, J. R. Medina, and J. K. Kalita, *A survey of the usages of deep learning for natural language processing*, IEEE transactions on neural networks and learning systems **32**, 604–624 (2020).

[345] D. Khurana, A. Koli, K. Khatter, and S. Singh, *Natural language processing: State of the art, current trends and challenges*, Multimedia tools and applications **82**, 3713–3744 (2023).

[346] J. L. K. E. Fendji, D. C. Tala, B. O. Yenke, and M. Atemkeng, *Automatic speech recognition using limited vocabulary: A survey*, Applied Artificial Intelligence **36**, 2095039 (2022).

[347] P. Karmakar, S. W. Teng, and G. Lu, *Thank you for attention: a survey on attention-based artificial neural networks for automatic speech recognition*, arXiv preprint arXiv:2102.07259 (2021).

[348] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, *Random synaptic feedback weights support error backpropagation for deep learning*, Nature communications **7**, 13276 (2016).

[349] B. A. Richards and T. P. Lillicrap, *Dendritic solutions to the credit assignment problem*, Current opinion in neurobiology **54**, 28–36 (2019).

[350] J. Guerguiev, T. P. Lillicrap, and B. A. Richards, *Towards deep learning with segregated dendrites*, ELife **6**, e22901 (2017).

[351] B. Scellier and Y. Bengio, *Equilibrium propagation: Bridging the gap between energy-based models and backpropagation*, Frontiers in computational neuroscience **11**, 24 (2017).

[352] D.-H. Lee, S. Zhang, A. Fischer, and Y. Bengio. *Difference target propagation.* In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part I 15*, pages 498–515. Springer, (2015).

[353] M. Jaderberg, W. M. Czarnecki, S. Osindero, O. Vinyals, A. Graves, D. Silver, and K. Kavukcuoglu. *Decoupled neural interfaces using synthetic gradients.* In *International conference on machine learning*, pages 1627–1635. PMLR, (2017).

[354] A. Gotmare, V. Thomas, J. M. Brea, and M. Jaggi. *Decoupling backpropagation using constrained optimization methods.* In *Proceedings of the 35th International Conference on Machine Learning*, number CONF, (2018).

[355] J. Zeng, T. T.-K. Lau, S. Lin, and Y. Yao. *Global convergence of block coordinate descent in deep learning.* In *International conference on machine learning*, pages 7313–7323. PMLR, (2019).

[356] A. Nøkland, *Direct feedback alignment provides learning in deep neural networks*, Advances in neural information processing systems **29** (2016).

[357] P. Baldi, P. Sadowski, and Z. Lu, *Learning in the machine: Random backpropagation and the deep learning channel*, Artificial intelligence **260**, 1–35 (2018).

[358] S. Duan, S. Yu, Y. Chen, and J. C. Principe, *On kernel method–based connectionist models and supervised deep learning without backpropagation*, Neural computation **32**, 97–135 (2020).

[359] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, *A hypercube-based encoding for evolving large-scale neural networks*, Artificial life **15**, 185–212 (2009).

[360] A. K. Malik, R. Gao, M. Ganaie, M. Tanveer, and P. N. Suganthan, *Random vector functional link network: recent developments, applications, and future directions*, Applied Soft Computing , 110377 (2023).

[361] P. Baldi and P. Sadowski, *A theory of local learning, the learning channel, and the optimality of backpropagation*, Neural Networks **83**, 51–74 (2016).

[362] D. O. Hebb, *The organization of behavior: A neurophychological study*, Wiley Interscience, New York (1949).

[363] G. Hinton, *The forward-forward algorithm: Some preliminary investigations*, arXiv preprint arXiv:2212.13345 (2022).

[364] G. E. Hinton, T. J. Sejnowski, et al., *Learning and relearning in Boltzmann machines*, Parallel distributed processing: Explorations in the microstructure of cognition **1**, 2 (1986).

[365] M. U. Gutmann and A. Hyvärinen. *Noise-contrastive estimation: A new estimation principle for unnormalized statistical models.* In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304, (2010).

[366] A. Ororbia and A. Mali, *The predictive forward-forward algorithm*, arXiv preprint arXiv:2301.01452 (2023).

[367] A. Ororbia, *Learning Spiking Neural Systems with the Event-Driven Forward-Forward Process*, arXiv preprint arXiv:2303.18187 (2023).

[368] R. F. Srinivasan, F. Mignacco, M. Sorbaro, M. Refinetti, A. Cooper, G. Kreiman, and G. Dellaferrera, *Forward Learning with Top-Down Feedback: Empirical and Analytical Characterization*, arXiv preprint arXiv:2302.05440 (2023).

[369] G. Dellaferrera and G. Kreiman. *Error-driven input modulation: solving the credit assignment problem without a backward pass.* In *International Conference on Machine Learning*, pages 4937–4955. PMLR, (2022).

[370] G. Lorberbom, I. Gat, Y. Adi, A. Schwing, and T. Hazan, *Layer Collaboration in the Forward-Forward Algorithm*, arXiv preprint arXiv:2305.12393 (2023).

[371] M. A. Ahamed, J. Chen, and A.-A.-Z. Imran, *Forward-Forward Contrastive Learning*, arXiv preprint arXiv:2305.02927 (2023).

[372] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al., *Advances and open problems in federated learning*, Foundations and Trends® in Machine Learning **14**, 1–210 (2021).

[373] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, B. McMahan, et al., *Towards federated learning at scale: System design*, Proceedings of machine learning and systems **1**, 374–388 (2019).

[374] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, *Federated optimization in heterogeneous networks*, Proceedings of Machine learning and systems **2**, 429–450 (2020).

[375] M. R. Sprague, A. Jalalirad, M. Scavuzzo, C. Capota, M. Neun, L. Do, and M. Kopp. *Asynchronous federated learning for geospatial applications.* In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 21–28. Springer, (2018).

[376] T. Nishio and R. Yonetani. *Client selection for federated learning with heterogeneous resources in mobile edge.* In *ICC 2019-2019 IEEE international conference on communications (ICC)*, pages 1–7. IEEE, (2019).

[377] A. Imteaj and M. H. Amini. *Fedar: Activity and resource-aware federated learning model for distributed mobile robots.* In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1153–1160. IEEE, (2020).

[378] L. Li, D. Liu, M. Duan, Y. Zhang, A. Ren, X. Chen, Y. Tan, and C. Wang, *Federated learning with workload-aware client scheduling in heterogeneous systems*, Neural Networks **154**, 560–573 (2022).

[379] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun. *Splitfed: When federated learning meets split learning.* In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8485–8493, (2022).

[380] W. Wu, M. Li, K. Qu, C. Zhou, X. Shen, W. Zhuang, X. Li, and W. Shi, *Split learning over wireless networks: Parallel design and resource management*, IEEE Journal on Selected Areas in Communications **41**, 1051–1066 (2023).

[381] X. Liu, Y. Deng, and T. Mahmoodi. *Energy efficient user scheduling for hybrid split and federated learning in wireless UAV networks.* In *ICC 2022-IEEE International Conference on Communications*, pages 1–6. IEEE, (2022).

[382] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han. *Haq: Hardware-aware automated quantization with mixed precision.* In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8612–8620, (2019).

[383] H. Feng, T. Pang, C. Du, W. Chen, S. Yan, and M. Lin, *Does Federated Learning Really Need Backpropagation?*, arXiv preprint arXiv:2301.12195 (2023).

[384] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. *Communication-efficient learning of deep networks from decentralized data.* In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, (2017).

[385] M. Yurochkin, M. Agarwal, S. Ghosh, K. Greenewald, N. Hoang, and Y. Khazaeni. *Bayesian nonparametric federated learning of neural networks*. In *International conference on machine learning*, pages 7252–7261. PMLR, (2019).

[386] Q. Li, B. He, and D. Song, *Practical one-shot federated learning for cross-silo setting*, arXiv preprint arXiv:2010.01017 (2020).

[387] J. Huang, *Maximum likelihood estimation of Dirichlet distribution parameters*, CMU Technique report **18** (2005).