Università degli Studi di Napoli Federico II

Dipartimento di Matematica e Applicazioni
"Renato Caccioppoli"

Improving classification models with context knowledge and variable activation functions

Author: Andrea Apicella

Supervisor: Paola Festa
Co-Supervisor: Francesco Isgrò

A thesis submitted for the degree of

Doctor of Philosophy

Contents

Chapter 1

Introduction

One of the main problem addressed by the computer science community is the automatic classification problem. For the human being the task of classifying is in most cases trivial, just think of the the huge amount of information that every moment is sent to the brain and processed unconsciously. The brain, despite the high load of information, can still provide answers in a very short time. Currently, machine learning is still far from reaching similar results, although research in this area is still proceeding.

The attempt to reproduce in an artificial way the abilities of the brain has remote origins; Leibniz with the ideas of the *characteristica universalis* and of the *calculus ratiocinator* ([Leibniz, 1890] or, for example, [Peckhaus, 2010]), that is an idealized language in which any other language can be expressed and at the same time can be replicated by a machine (point of view proposed in [Wiener, 1948]) can be seen as the foundation of modern AI. Other relevant works are [Boole, 1854, Mill, 1884], which help to bring the logic closer to the maths and the AI.

The first pattern recognition algorithm was suggested in [Fisher, 1936], highlighting how the problem of classification could be faced in the search for decision surfaces. A few years later, in [McCulloch and Pitts, 1943] was shown how some logical predicates can be translated into neural activities, but it is in the 1950s that A. Turing, in [Turing, 1950], discussed about the existence of a machine that can learn, and, always, in those years, pioneering studies on machine learning began to appear, such as the Perceptron in [Rosenblatt, 1957] or ADALINE in [Widrow and Hoff, 1960], which were the first attempts to provide a mathematical model of the biological neuron; based on these models, the modern neural networks will be designed. The first computer program based on machine learning is to be attributed to A. Samuel [Samuel, 1959], which, in addition to introducing for the first time the term "machine learning", implemented a version of the game of checkers

in which the moves to be made were computed using the knowledge of past rounds. Machine learning thus became an integral part of the study of Artificial Intelligence. In the 1960s, new classification methods were also proposed, such as the first Support Vector Machines [Vapnik and Lerner, 1963], which will be refined in and extended in [Boser et al., 1992, Cortes and Vapnik, 1995], and the Nearest Neighbor algorithm [Cover and Hart, 1967]. In [Minsky and Papert, 1969] the limits of the first neural networks are discussed, showing how these models were not universal classifiers, because there are decision surfaces that can not be modeled using these early architecture (XOR problem). However, these limits will be overcome by using different architectures and adequate learning rules that will be proposed in the 1970s [Linnainmaa, 1970, Linnainmaa, 1976, Rumelhart et al., 1986]. As early as the mid-50s, there was a decline in interest from the AI community in the machine learning techniques, preferring knowledge-based methods [Jackson, 1998]. These systems used knowledge bases on a given domain providing solutions through the use of rules; among these we mention DENDRAL [Lindsay et al., 1980], which formulates hypotheses on the structure of organic compounds, and Hearsay [Balzer et al., 1980] which analyzes spoken language.

In recent years we have seen a renewed interest of the scientific community in machine learning techniques, thanks to factors as the big amount of data available for machine learning; at the end of 1980s UCI Machine Learning Repository [Dheeru and Karra Taniskidou, 2017], a collection of datasets for studies in machine learning, was introduced and made available to the community; recently, others important dataset were created, as for example the MNIST dataset [LeCun and Cortes, 2010], Cifar10/100 [Krizhevsky and Hinton, 2009] and ImageNet [Deng et al., 2009].

Other factors, as hardware previously used for different purposes (e.g. GPU) and the discovery of new elements able to improve the convergence of the learning algorithms, have allowed to improve the performance of the current supervised classifiers. An important work was made in [Krizhevsky et al., 2012], where a Convolutional Neural Network reached the State of Art in the ImageNet Large Scale Visual Recognition Challenge in 2012.

More specifically, a real change in the orientation of the research has been achieved thanks to the rediscovery of Neural Networks due to the use of new techniques based on:

- external training procedure (e.g. pre-training methods, [Erhan et al., 2010]);

- new activation functions (e.g. ReLU, [Nair and Hinton, 2010]).

These innovations have improved the performance of neural networks both in terms of speed of learning and in terms of accuracy of results.

The innovations introduced in the Neural Networks have rekindled the interest not only in this particular branch of machine learning, but in everything related to artificial intelligence.

In this work, the problem of improving automatic classification performance is addressed, searching for methods to boost the performance of a classification system in two different ways:

- the first one which is suitable for neural network classifiers, and that focus on the possibility to learn appropriate activation functions from data [Apicella et al., 2018c, Apicella et al., 2018b];

- the second one which is suitable for any kind of probabilistic classifier, using knowledge coming from the real world formalized in a probabilistic ontology [Apicella et al., 2017b, Apicella et al., 2017a, Apicella et al., 2018a].

More specifically, the first method is about the *adaptable* activation functions, that are activation functions which are adjusted during the learning phase using the training data and that allow the network to exploit the data better respect to classic activation function with fixed-shape; the current studies on new activation functions and the attention of the scientific community on this subject show that it is an open topic in machine learning research; ReLU [Nair and Hinton, 2010], the newest ELU [Clevert et al., 2015] and all the others functions belonging to the rectifier family are a recent progress that indicate that there is still much to be said on this subject. On a closer look, a part of the scientific community is moving toward the learnable activation functions, that are activation functions whose shape is learned by data. In this work, we made a careful survey on the state of the art of this type of functions, that, in our knowledge, has never been made, and then we propose a new a simple and efficient learnable activation function.

Differently from the first method that is specific for neural network classifiers, the second one provides two frameworks for integrating an external knowledge base with the results given by a generic classifier so as to improve its results "correcting" some errors. The most commonly used tools for encoding a-priori information are standard ontologies.

A probabilistic ontology can be made available for the considered domain, but it could also be built or enriched by using entities and relations extracted from a document related to the image. For example, a picture could have been extracted from a technical report

or a book where the text gives information that is related to the considered images. This world knowledge, formalized in a probabilistic ontology, together with the output of the classifier, is fed to a probabilistic model [Bishop, 2006], with the goal of improving the performance of the single classifiers.

This method does not directly involve the classifier structure, that is it can be defined *model-agnostic*, making it also suitable for classifiers different from Neural Networks, having only as constrain that they provide a probability distribution over the possible classes for every given input, or at least a set of scores on which to build it. The framework described in this work has two main aspects of novelty. The first one is that, to the best of our knowledge, a probabilistic ontology has never been proposed for a computer vision problem. The integration of a probabilistic model with a probabilistic ontology presents a second element of novelty.

Outline

This work is organized in the following way: chapter 2 shows a brief introduction to the classification problem in the supervised learning framework, showing some of the most important classifier techniques present in literature; the work is then divided into two parts:

- The first part is dedicated to the learnable activation functions; after doing a survey of the most important studies for the learnable activation function proposed in literature, a new learnable activation function is proposed; this part is divided as follows:
  - in Chapter 3 Neural Networks are introduced, without wanting to be a detailed treatment on the subject, as it would be outside the scope of this thesis, and referring the reader to more detailed references as for example [Bishop, 2006, Hagan et al., 1996, Haykin, 1994];
  - in Chapter 4 a survey on the learnable activation functions is made; the most important study on this topic are discussed and described, also proposing a taxonomy in which to divide them;
  - in Chapter 5 the proposed model is described together with the experiments made and the results obtained, both on Full-Connected Neural Networks and on Convolutional Neural Networks;

  - in Chapter 8 concludes the work with the final remarks.

- The second part is dedicated to a method based on the integration of a knowledge base with a classifier, is organized as follows:
  - in Chapter 6 is defined the term "ontology", showing how the term is difficult to define and how various studies have been conducted to obtain its complete formalization, also discussing the State of Art in the use of ontologies for the classification task;
  - in Chapter 7 are proposed two different methods to merge the ontology knowledge with the results given by a classifier, together with experiments made and the results obtained.
  - Chapter 8 concludes this works summing up the work done and the results obtained.

Notation

unless otherwise indicated, we will use the following notation from now: we will indicate sets with words starting with capital letters and with lowercase letters the elements that they contain (for example, $D = \{a, b, c, d\}$); in some cases, we will indicate with the elements of a set will be indicated all with the same letter and distinguished by a number at the apex in parenthesis (for example, $D = \{x^{(1)}, x^{(2)}, x^{(3)}\}$) capital letters will also be used to indicate matrix (for example, $W \in \mathbb{R}^{2\times 3}$); a tuple of numbers arranged by column will be indicated with a lowercase letter with an arrow above and the components that compose it will be indicated with the same letter with a number to the subscript (for example, $\vec{v} = (v_1, v_2, v_3)^T$). The same notations could also be used to distinguish between values of a variable in different time moments (for example, $g_k, g_{k+1}, \dots$).

Chapter 2

The classification Problem

Introduction

This chapter wants to give an overview on the classification problem and some methods to deal with it. After giving an introduction on the classification problem in Section 2.1, the remain of the present chapter describes some of the most common machine learning techniques dedicated to the classification problem.

2.1 Empirical Risk Minimization for Supervised Learning

Given a set of pairs $D = \{(\vec{x}^{(1)}, y^{(1)}), (\vec{x}^{(2)}, y^{(2)}), \ldots, (\vec{x}^{(n)}, y^{(n)})\} \subseteq X \times Y$ where X is the input space (usually $X \subseteq \mathbb{R}^d$ with d given) and Y (usually $Y \subseteq \mathbb{Z}$ for classification problem and $Y \subseteq \mathbb{R}$ for regression problems) is the desired output space, a supervised learning problem (see [Vapnik, 1992, Cord and Cunningham, 2008, Scholkopf and Smola, 2001]) consists to infer a mapping function $f : X \to Y$ (which is called a *classifier*, if the output is discrete as in classification problems, or *regression function* if the output is continuous) s.t. $f(\vec{x})$ is good approximation of y, also if $(\vec{x}, y) \notin D$. Usually, in classification problems, we assume that the $(\vec{x}^{(i)}, y^{(i)})$ are sampled by an unknown distribution $P(\vec{x}, y)$ and $\forall(\vec{x}^{(i)}, y^{(i)}) \in Tr$, $\vec{x}^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{N}$; to this aim, a *loss* (or *cost*) function $L(f(\vec{x}), y)$ which measures how the prediction given by $f(\vec{x})$ are different from the real y is used. using the loss function L, is possible to define the *Risk* (or *Generalization error*) as

$$R(f) = \int L\big(f(\vec{x}), y\big) P(\vec{x}, y) d\vec{x} dy \tag{2.1}$$

The objective of a supervised learning algorithm is to find the function f that minimizes the $R(f)$ value, i.e. the algorithm looks for a function $f^* = \arg\min_f R(f)$; in practice, this is not possible because the real distribution $P(\vec{x}, y)$ is unknown.

So, in real cases an approximation of the risk, i.e. the *empirical Risk*, is used; this is given by:

$$R_{emp}(f) = \frac{1}{n} \sum_{i=1}^{n} L\big(f(\vec{x}^{(i)}), y^{(i)}\big). \tag{2.2}$$

This can be seen as the Risk when considering the empirical distribution of the data as a surrogate for the true distribution. For the law of large numbers (see, for example, [Rosenthal, 2006]), as the number of elements in Tr goes to infinity, for every fixed function f, the empirical Risk $R_{emp}(f)$ converge to the Risk $R(f)$; however, this doesn't imply that the minimized function $f^m = \arg\min_f R_{emp}(f)$ will lead to a risk value as good as the best risk $R(f^*)$: there are cases of learned function where the empirical risk is zero on the training data Tr, but is greater on another set of data sampled from the same distribution; this phenomena is said *over-fitting*. So, it is always a good practice to have another set on which to test the function learned (that is called *test set*).

A possible method to deal with this problem is to restrict the search domain for the f^m function; to this aim, instead of $R_{emp}(f)$, it is minimized a constrained variation:

$$R_{erg}(f) = R_{emp}(f) + \lambda\Omega(f) \tag{2.3}$$

where $\Omega(f)$ is a regularization term which introduces a penalty on some kind of functions, and $\lambda > 0$ is a trade-off parameter between the empirical risk minimization and the introduced penalty. The risk function actually used is called the *objective function* $\mathscr{L}$.

2.2 Model selection

Almost all classification methods can suffer of two main problems:

1. over-fit: a model results to over-fit the training set when it gives high performance on the training data and poor performance on new data. As stated in [Picard and Cook, 1984]: "when a model is chosen because of qualities exhibited by a particular set of data, predictions of future observations that arise in a similar fashion will almost certainly not be as good as might naively be expected". For this reason, it is important to have methods to assess how a chosen model will generalize to unseen data.

2. free-parameters choice: many pattern recognition methods need one or more free parameters (or *hyper-parameters*), as for example, the learning rate value in gradient descent algorithms; their values can change sensibly the model performances. For these reasons is necessary to find a good set of hyper-parameters for the problem, i.e. a set of hyper-parameters that gives a model with good performance not only on training data but also on new observations.

Therefore, methods to select a proper model for a classification problem are necessary. The cross-validation techniques (see for example [Mosteller and Tukey, 1968, Arlot et al., 2010]) attempt to address these problems, helping to understand how a given model is good in terms of generalization and then if it can be chosen or not. The main step of a cross-validation method is to split the available data into two subset and taking one subset as *training Set* Tr, that is used materially to train the model, and the remaining data as **test set** Te, on which the learned model is tested; this simple form of validation is said *hold-out validation* [P. Devroye and Wagner, 1979] and it has, as main drawback, a potential high variance, because the evaluation may be significantly different depending on how the division is made. A possible solution can be averaging over several splits, yielding a *cross-validation* estimate. Based on this, there are different method to perform cross-validation, as:

- ***leave-p-out cross validation*** [Shao, 1993]: p points are chosen as Te while the remaining points form the Tr set and an hold-out validation is made for every possible (Tr, Te) data split based on p points and then averaged;

- ***leave-one-out cross validation*** [Stone, 1974, Geisser, 1975]: as leave-p-out, but with $p = 1$;

- ***k-fold cross validation*** [Geisser, 1975]: a k-partition P of the data is made; a set from P is taken as Te, and the union of remaining set in P is taken as Tr; The process is then repeated for every possible pair (Tr, Te) in the partition and then averaged;

- ***Monte Carlo cross validation*** [Picard and Cook, 1984]: the dataset is randomly divided into training and validation data .The process is repeated more times for different random pairs (Tr, Te) and then averaged.

Algorithm 1: k-fold cross validation procedure

Input: Dataset D, number of folds k, hyper-parameters values $\{p_1, p_2, \ldots, p_n\}$ with $p_i = \{$ possible values for i-th hyper-parameter $\}$ with $1 \leq i \leq n$

$foldResults = 0 \in \mathbb{R}^k$;
split D in a $k-$partition $P^k(D)$;
forall $1 \leq i \leq k$ **do**
    $testSet \leftarrow P_i^k(D)$;
    $R \leftarrow P^k(D) \setminus \{TestSet\}$;
    split R in a $2-$partition $P^2(R)$;
    $trainSet \leftarrow P_1^2(R)$;
    $valSet \leftarrow P_2^2(R)$;
    $bestParams \leftarrow \emptyset$;
    $bestResults \leftarrow \emptyset$;
    forall $H \in p_1 \times p_2 \times \cdots \times p_n$ **do**
        $model \leftarrow MakeModel(trainSet, H)$;
        $results \leftarrow Evaluate(model, valSet)$;
        if $results$ ***better than*** $bestResults$ **then**
            $bestParams \leftarrow H$;
        end
    end
    $model \leftarrow MakeModel(trainSet \cup valSet, bestParams)$;
    $foldResults_i \leftarrow Evaluate(model, testSet)$;
end
return $Average(foldResults)$;

Furthermore, to find a good set of hyper-parameters, many cross validation algorithms consider a third subset of examples called *validation set* [Ripley, 2007], disjoint from the training and the test sets that is used only to tune the hyper-parameters of a classifier. The procedure using a k-fold cross-validation scheme is summarized in Algorithm 1.

2.3 Classification types

Classification problems can be divided into two main classes:

- binary (or two-class) classification: every input $\vec{x}$ belongs to one of two possible classes $\{C_1, C_2\}$;
- multi-class classification: every input $\vec{x}$ belongs to one of K possible classes $\{C_1, C_2, \ldots, C_K\}$, with $K > 2$.

For binary classification, the two possible classes are usually indicated through a variable $y \in \{0, 1\}$ (or $y \in \{-1, 1\}$) where $y = 0$ ($y = -1$) indicates that the input $\vec{x}$ belongs to the class C_1 and $y = 1$ in the second class C_2, while, for multi-class classification problems, the variable is usually defined as $y \in \{1, 2, \ldots, K\}$ (or $y \in \{0, 1, \ldots, K-1\}$); anyway, in multi-class case, can be convenient to code the input class, instead that using a variable y, using a vector $\vec{t} \in \{0, 1\}^K$ with the constraint that the component $y_i = 1$ if and on only if the input $\vec{x}$ belongs to the class C_i, otherwise $y_i = 0$.

2.4 Linear classifiers

A linear classifier is one of the most popular classification framework. The main characteristic of a linear classifier is that it assigns input vectors to classes through linear decision boundaries. Formally, considering an input $\vec{x} = (x_1, x_2, \ldots, x_d)^T \in \mathbb{R}^d$, a linear classifier uses a set of parameters $\vec{w} = (w_1, w_2, \ldots, w_d)^T$ to estimate the class of $\vec{x}$ as:

$$\hat{\vec{y}} = f(s(\vec{x}))$$

with $s(\vec{x}) = \vec{w}^T\vec{x} + w_0$ a *decision surface*, and f a discriminative function that assigns the $\vec{x}$ to a class using $s(x)$. If desired, it is possible to include the term w_0 in the vector $\vec{w}$ expanding the vector $\vec{x}$ with a constant 1, i.e. $\vec{w} = (w_0, w_1, w_2, \ldots, w_d)^T \in \mathbb{R}^{d+1}$, $\vec{x} = (1, x_1, x_2, \ldots, x_d)^T \in \mathbb{R}^{d+1}$ obtaining the form

$$s(\vec{x}) = \vec{w}^T\vec{x}.$$

For a binary classification problem with classes C_1, C_2, the discriminant function is usually the sign$(\cdot)$ function, that is, the input can be considered belonging to the C_1 class if $s(\vec{x}) \geq 0$, C_2 class otherwise.

For the multi-class problems,it is possible to consider three possible approaches:

- *One vs All*: a different binary classifier is trained for each class and each classifier predicts whether the instance $\vec{x}$ belongs to the target class or not; this approach can lead to ambiguity regions, for example if multiple classifiers say that the input belongs to its class;

- *All vs All*: a different binary classifier is trained for each possible pair of classes obtaining $K(K-1)/2$ classifiers; each point can be classified based on majority vote between classifier; also in this case there could be ambiguous decision regions;

- *One vs All with score*: as the *One vs All* approach, but instead of considering just the final binary prediction of each classifier, this approach considers the score associated with the prediction (in linear case, it is the distance from the decision boundary given by $s(x)$ value) and take the maximum value.

Usually, the *One vs All with score* is the most used approach.

A linear model makes the strong assumption that the output $\hat{y}$ can be modeled through a linear function on $\vec{w}$, that is the data are *linearly separable*, which implies that they can be separated exactly by a linear decision surfaces, an assumption that can be flawed especially in complex models. However, this kind of model results to be fast, so that it can be used in situations where speed is particularly important even at the cost of obtaining less accurate results. The remainder of this Section discusses some of the most common linear model in literature.

2.4.1 Least Square method

One of the most popular method to find the parameters $\vec{w}$ of a linear classifier is *Least Square* method. Considering a binary classification problem, LS method requires a set $D = \{(\vec{x}^{(1)}, y^{(1)}), (\vec{x}^{(2)}, y^{(2)}) \dots, (\vec{x}^{(n)}, y^{(n)})\}$ of data from which to fit the model parameters, with $\vec{x}^{(i)} \in \mathbb{R}^{d+1}$ and $y^{(i)} \in \{-1, +1\}$ as described in section 2.4. Given a parameter vector $\vec{w} \in \mathbb{R}^{d+1}$, it is possible to estimate $s(x)^{(i)} = \vec{w}^T \vec{x}^{(i)}$, and the the *residual* as:

$$r^{(i)}(\vec{w}) = (\vec{w}^T \vec{x}^{(i)} - y^{(i)})^2.$$

The residual sum of squares is given by:

$$RSS(\vec{w}) = \sum_{i=1}^{n} r^{(i)}(\vec{w}).$$

Knowing that D can be written in matrix form as a matrix $X \in \mathbb{R}^{n \times (d+1)}$ and a vector $\vec{y}^T \in \{-1, +1\}^n$. In matrix form, RSS becomes

$$RSS(\vec{w}) = (\vec{y} - X\vec{w})^T (\vec{y} - X\vec{w}).$$

The least square method search for the vector $\vec{w}^*$ which minimizes the $RSS(\vec{w})$ function, that is

$$\vec{w}^* = \arg\min_{\vec{w}} \; RSS(\vec{w})$$

the minimum can be found searching for the point where the gradient respect to $\vec{w}$ becomes zero, i.e. $\nabla RSS(\vec{w}) = 0$; the solution can be found analytically, $\nabla RSS(w) = 2X^T(\vec{y} - X\vec{w}) = 0$ if $\vec{w}^* = (X^TX)^{-1}X^T\vec{y}$.

In a multi-class problem, the dataset can be considered as $D = \{(\vec{x}^{(1)}, \vec{y}^{(1)}), (\vec{x}^{(2)}, \vec{y}^{(2)}) \ldots, (\vec{x}^{(n)}, \vec{y}^{(n)})\}$ where $\vec{y}^{(i)} \in \{0,1\}^K$ is the class variable coded as described in section 2.3. In matrix form, the dataset D can be coded into two matrices, a first matrix $X \in \mathbb{R}^{n\times(d+1)}$ and a second matrix $Y \in \{0,1\}^{n\times K}$. The model parameters can be represented by a matrix $W \in \mathbb{R}^{(d+1)\times K}$ and the sum of residual can be written as:

$$RSS(W) = \mathrm{Tr}\big((XW - Y)^T(XW - Y)\big)$$

and the estimated W^* can be computed as:

$$W^* = (X^TX)^{-1}X^TY.$$

One of the main drawbacks of the Least Square method is that it doesn't result very robust to outliers [Vecchia and Splett, 1994].

2.4.2 Rosenblatt's Perceptron

In [Rosenblatt, 1957] the Perceptron is presented, one of the best known examples of linear model for binary classification. In this framework, the data has to be expressed as a set $D = \{(\vec{x}^{(1)}, y^{(1)}), (\vec{x}^{(2)}, y^{(2)}) \ldots, (\vec{x}^{(n)}, y^{(n)})\}$ which requires that $y^{(i)} \in \{-1, 1\}$. The Perceptron linear model is expressed as:

$$\hat{y} = f\big(\vec{w}^T\phi(\vec{x})\big)$$

where $\phi(\vec{x})$ is a fixed non-linear transformation of the input $\vec{x}$, and f the step function, i.e.

$$f(a) = \begin{cases} -1 \text{ if } a < 0 \\ +1 \text{ otherwise.} \end{cases}$$

So, f maps all values above 0 threshold to the first class and all the other values to the second class. The Perceptron method is based on the observation that all the correct classifications respect the constraint $\vec{w}^T\phi(x)y > 0$; the target can be defined as to minimize the following quantity:

$$E(\vec{w}) = -\sum_{i\in M} \vec{w}^T\phi(\vec{x})y,$$

where M is the set of misclassified points in D. This error function can be minimized using the Gradient Descent (henceforth GD) algorithm (see section 3.3 for details). Briefly, the GD algorithm brings the parameters along the opposite gradient direction that is $\vec{w} \leftarrow \vec{w} - \nabla E(\vec{w})$. It has been proven [Rosenblatt, 1961] that, if this process is repeated for a finite number of steps and the data are linearly separable, the algorithm will converge to a minimum (Perceptron convergence theorem); in the opposite case, the algorithm will not converge. However, also if the data are linearly separable, the number of steps could be very large. Furthermore, the Perceptron is difficult to generalize for multi-class classification problems.

The solution given by a Perceptron classifier depends on the initial values of parameters $\vec{w}$ and on the order in which the data points are presented to the learning algorithm. Furthermore, the solutions can be easily subject to over-fitting.

2.4.3 Linear Support Vector Machines

Differently from the Perceptron, an SVM tries to find the solution with the smallest generalization error. To achieve this result, the SVMs use the concept of *margin*, that is the smallest distance between the decision surface and any input point. Considering a binary classification problem with class labels $y^{(i)} \in \{-1, 1\}$, for any point $x^{(i)}$ and a decision boundary as defined in section 2.4.2 but with external bias term, i.e. $s(\vec{x}) = \vec{w}^T \phi(\vec{x}) + w_0$, the distance from a surface identified by a parameters vector $\vec{w}$ can be computed as $d(\vec{w}, \vec{x}) = \frac{y^{(i)} s(\vec{x}^{(i)})}{||\vec{w}||}$. The margin is thus defined as the distance between the decision boundary and the nearest point in D (at least one for each class); so, a SVM wants to resolve the following problem:

$$\vec{w}^*, w_0^* = \arg\max_{\vec{w}, w_0} \Big(\frac{1}{||\vec{w}||} \min_{1 \leq i \leq n} \big(y^{(i)} s(\vec{x}^{(i)}) \big) \Big)$$

This problem can be simplified considering that scaling the vector $\vec{w}$, which is orthogonal to the hyperplane, or w_0, does not change the distance from the surface, so it is possible to constrain $y^{(i)} s(\vec{x}^{(i)}) = 1$ for all the points closest to surface; these points are said *support vectors*. A consequence of this is that, for all points in D, $y^{(i)} s(\vec{x}^{(i)}) \geq 1$. Using this constraint, the problem can be formalized in the following more convenient form:

$$\vec{w}^*, w_0^* = \arg\min_{\vec{w}, w_0} \big(\frac{1}{2} ||\vec{w}||^2 \big) \text{ subject to } y^{(i)} s(\vec{x}^{(i)}) \geq 1, \ \forall \ 1 \leq i \leq n.$$

Logistic Regression

The Logistic Regression is a classification method which returns a probability distribution for a given input point. Considering a binary classification problem with classes $\{C_1, C_2\}$, assuming that the data are sampled from two normal distributions having respectively average μ_1 and μ_2,the same covariance matrix Σ and the same priors, It can be shown (see for example [Flach, 2012]) that

$$\log \frac{P(\vec{x}|C_1)}{P(\vec{x}|C_2)} = \vec{w}^T \vec{x} + q$$

and that

$$\log \frac{P(C_1|\vec{x})}{P(C_2|\vec{x})} = \vec{w}^T \vec{x} + q + \frac{\log P(C_1)}{\log P(C_2)} = \vec{w}^T \vec{x} + w_0$$

that, in binary classification problems, is equivalent to write:

$$\log \frac{P(C_1|\vec{x})}{1 - P(C_1|\vec{x})} = \vec{w}^T \vec{x} + w_0$$

then

$$\frac{P(C_1|\vec{x})}{1 - P(C_1|\vec{x})} = \exp\big(\vec{w}^T \vec{x} + w_0\big) = \exp\big(s(\vec{x})\big)$$

from which is possible to derive (using the notation the notation in which $\vec{w}$ includes w_0 and $\vec{x}$ is expanded with a 1):

$$P(C_1|\vec{x}) = \frac{1}{1 + \exp\big(-s_{\vec{w}}(\vec{x})\big)} = \mathrm{sig}(s_{\vec{w}}(\vec{x}))$$

and

$$P(C_2|\vec{x}) = 1 - P(C_1|\vec{x}) = 1 - \mathrm{sig}(s_{\vec{w}}(\vec{x})) = \mathrm{sig}\big(-s_{\vec{w}}(\vec{x})\big)$$

that is the Sigmoid function. Using as class labels $y \in \{0, 1\}$ where $y = 1$ means that the item belongs to C_1 class and $y = 0$ to the other, it is possible to view the problem to estimate $\vec{w}^*, w_0^*$ as a Maximum Likelihood problem with Bernoulli distribution:

$$\vec{w}^* = \arg\max_{\vec{w}} \prod_{i=1}^{n} P(C_1|\vec{x}^{(i)})^{y^{(i)}} \big(1 - P(C_1|\vec{x}^{(i)})\big)^{1-y^{(i)}}$$

This can be shown as a convex optimization problem, which means that there is only one maximum and then many optimization techniques can be used; for example, if we use as Loss function the negative logarithm of the likelihood:

$$\mathscr{L}(\vec{w}) = -\sum_{i=1}^{n} y^{(i)} \log \mathrm{sig}(s_{\vec{w}}(\vec{x}^{(i)}) \cdot (1 - y^{(i)}) \log(1 - \mathrm{sig}(s_{\vec{w}}(\vec{x}^{(i)})))$$

is possible to use a Gradient Descent algorithm or Newton methods to minimize it. To notice that, if $y \in \{-1, 1\}$, the problem can be written as

$$\vec{w}^* = \arg\max_{\vec{w}} \prod_{i=1}^{n} \text{sig}\big(y^{(i)} \vec{w}^T \vec{x}^{(i)}\big)$$

so the loss function becomes:

$$\mathscr{L}(\vec{w}) = -\sum_{i=1}^{n} \log\big(1 + \exp(y^{(i)} \vec{w}^T \vec{x}^{(i)})\big);$$

it is important to notice that, if the assumption of a equal covariance matrix is not made, the resulting function is no more a linear one, but a quadratic one.

If we consider a multi-class case with K classes, it is possible to consider one class as main reference ("pivot" class, e.g. the K class) and consider $K-1$ logistic regressors:

$$\begin{aligned}
\log \frac{P(C_1|\vec{x})}{P(C_K|\vec{x})} &= \vec{w}^{(1)T}\vec{x} \implies P(C_1|\vec{x}) = P(C_K|\vec{x}) e^{\vec{w}^{(1)}\vec{x}} \\
\log \frac{P(C_2|\vec{x})}{P(C_K|\vec{x})} &= \vec{w}^{(2)T}\vec{x} \implies P(C_2|\vec{x}) = P(C_K|\vec{x}) e^{\vec{w}^{(2)}\vec{x}} \\
&\vdots \\
\log \frac{P(C_{K-1}|\vec{x})}{P(C_K|\vec{x})} &= \vec{w}^{(K-1)T}\vec{x} \Rightarrow P(C_{K-1}|\vec{x}) = P(C_K|\vec{x}) e^{\vec{w}^{(K-1)}\vec{x}}
\end{aligned}$$

where every $\vec{w}^{(i)},\ \forall\ 1 \leq i \leq K-1$ is a different vector. $P(C_K|\vec{x})$ can be computed as:

$$\begin{aligned}
P(C_K|\vec{x}) = 1 - \sum_{t=1}^{K-1} P(C_K|\vec{x}) e^{\vec{w}^{(t)T}\vec{x}} &= 1 - P(C_K|\vec{x}) \sum_{t=1}^{K-1} e^{\vec{w}^{(t)T}\vec{x}} \\
\implies P(C_K|\vec{x}) + P(C_K|\vec{x}) \sum_{t=1}^{K-1} e^{\vec{w}^{(t)}\vec{x}} &= 1 \\
\implies P(C_K|\vec{x})(1 + \sum_{t=1}^{K-1} e^{\vec{w}^{(t)T}\vec{x}}) &= 1 \\
\implies P(C_K|\vec{x}) &= \frac{1}{1 + \sum_{t=1}^{K-1} e^{\vec{w}^{(t)T}\vec{x}}}
\end{aligned}$$

and then the probability of the others class can be computed as

$$P(C_1|\vec{x}) = P(C_K|\vec{x})e^{\vec{w}^{(1)T}\vec{x}} = \frac{e^{\vec{w}^{(1)T}\vec{x}}}{1 + \sum_{t=1}^{K-1} e^{\vec{w}^{(1)T}\vec{x}}}$$

$$P(C_2|\vec{x}) = P(C_K|\vec{x})e^{\vec{w}^{(2)T}\vec{x}} = \frac{e^{\vec{w}^{(2)T}\vec{x}}}{1 + \sum_{t=1}^{K-1} e^{\vec{w}^{(2)T}\vec{x}}}$$

$$\vdots$$

$$P(C_{K-1}|\vec{x}) = P(C_K|\vec{x})e^{\vec{w}^{(K-1)T}\vec{x}} = \frac{e^{\vec{w}^{(K-1)}\vec{x}}}{1 + \sum_{t=1}^{K-1} e^{\vec{w}^{(K-1)T}\vec{x}}}$$

The Maximum Likelihood problem can be written, using the $1-K$ coding scheme for the labels $\vec{y}$ described in section 2.3, as:

$$\vec{w}^{(1)*}, \vec{w}^{(2)*}, \ldots, \vec{w}^{(K)*} = \arg \max_{\vec{w}^{(1)*}, \vec{w}^{(2)*}, \ldots, \vec{w}^{(K)*}} \prod_{i=1}^{n} \prod_{j=1}^{K} P(C_j|\vec{x}^{(i)})^{\vec{y}_j^{(i)}}$$

2.5 Non-linear classifiers

The following section wants to describe some of the most common non-linear classifier present in literature; non-linear classifier are used in situations where linear classifier are not enough to achieve good classification performance. In this section Neural Networks are not mentioned, since, being one of the main subjects of this work, it was decided to dedicate a separate chapter to it, i.e. the Chapter 3.

2.5.1 k-Nearest Neighbors

k-NN algorithm is one of the simplest non-linear classification methods; given a set of data points D and an integer value $k > 0$, k-NN assigns to each new point $\vec{x}$ the most frequent class considering the k points in D nearest to $\vec{x}$ ("majority voting" on k nearest data points). Euclidean distance is usually used as closeness measure, however other distance measures can be used. One of the crucial choices of this algorithm lies in the value of the parameter k; Obviously, the parameter k should be odd to avoid "break even" situations (that is, many classes with the same score). A common way to choose it is to use some parameter selection techniques (e.g., cross-validation, see section 2.2). In case of $k = 1$ the point is simply assigned to the same class as the nearest point.

The main drawback of this technique is the need to keep the entire dataset stored, in addition to the computational cost for the computation of the distances between points. Furthermore, the presence of outliers can influence the classification. One way to improve the performances is to weight the points relying on the proximity to the point to be classified. k-NN is an example of algorithm based on distance-based learning, i.e. a class of learning methods that, instead of learning a function to generalize from the training data, make decisions relying on the data themselves; in other terms, no model is learned because the training instances themselves represent the knowledge.

2.5.2 Kernel methods

If the data are non-linearly separable in a certain space, nothing prevents them to be linearly separable if they are projected into a higher dimensionality space; for example, suppose that a set $D = \{(x_1, x_2)^{(1)}, (x_1, x_2)^{(2)}, \ldots, (x_1, x_2)^{(n)}\}$ of n points is sampled from a two dimensional space and that they form two concentric circles centered at the origin; obviously, they are not linearly separable. But, if these points are non-linearly mapped into an extended space using as third dimension the radius of the circles, (i.e. $D' = \{(x_1, x_2, x_1^2 + x_2^2)^{(1)}, (x_1, x_2, x_1^2 + x_2^2)^{(2)}, \ldots, (x_1, x_2, x_1^2 + x_2^2)^{(n)}\}$) the two groups of points can be linearly separated. This observation is enunciated and demonstrated by Cover [Cover, 1965] in the following statement: "A complex pattern-classification problem, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated."

On the ground of what said above, a possible approach to obtain a linearly separable set could be to map the data into a feature space with a higher dimensionality and to learn a linear model in this new space, that is, to find a function $\phi : X \to S$ with dimensionality of S larger than the dimensionality of X and then to learn a linear model $\hat{y} = f(\vec{w}^T \phi(\vec{x}))$. The main drawback of a similar approach is that to work in a larger space can be expensive in computational terms. However, it may be considered that some linear classifiers, as Perceptron and SVM, do not directly require that the points are mapped in the new space, but they require only the inner product between points. So, it is possible to replace the inner product with a function $k(v, w)$ (known as *kernel* function) that returns the inner product of v and w in some feature space. In other words, a kernel function returns the value of the inner product $< v, w >_S$ in some space S. More formally, a function $k : X \times X \to \mathbb{R}$ is a kernel if:

- k is symmetric;

- $\forall n \in \mathbb{N}$ and $\forall x_1, x_2, \ldots, x_n$ chosen from X, the Gram Matrix K defined by $K_{ij} = k(x_i, x_j)$ is positive semidefinite.

the operation of replacing the inner product with a kernel function is known as the *kernel trick* and any method that uses a kernel trick is said to be a *kernel method*.

2.5.2.1 Kernel Perceptron

Reviewing the Perceptron weights update rule, it's possible to note that, every time an example $x^{(i)}$ is misclassified, the GD algorithm adds the $y^{(i)}\vec{x}^{(i)}$ quantity to the weight vector $\vec{w}$, so the weights can also be computed as:

$$\vec{w} = \sum_{i=1}^{n} \alpha_i y^{(i)} \vec{x}^{(i)}$$

with α_i number of misclassification of the data point $\vec{x}^{(i)}$; in other words, the weight vector can be viewed as a linear combination of the training instances, so the Perceptron rule can be rewritten as $\hat{y}^{(j)} = f(\sum_{i=1}^{n} \alpha_i y^{(i)} \vec{x}^{(i)T} \vec{x}^{(j)})$ and the learning algorithm searchs for $\vec{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ instead that $\vec{w}$. The parameters $\vec{\alpha}$ can be used to have further information about data points D, e.g.:

- $\alpha_i = 0 \implies$ the data point $x^{(i)}$ isn't useful to the training procedure because it will not affect the final results;

- $\alpha_i > 0 \implies$ the data point $x^{(i)}$ has been misclassified during the training at least once.

The Perceptron problem formulated in this way is called *dual*; the product $\vec{x}^{(i)T}\vec{x}^{(j)}$ can be substituted by a kernel $k(\vec{x}^{(i)}, \vec{x}^{(j)})$.

2.5.2.2 Kernel SVM

Linear SVM described in section 2.4.3 can be formulated in the Lagrange dual form (see for example [Rockafellar, 1993]) taking into account Karush-Kuhn-Tucker conditions [Kuhn and Tucker, 2014]; In general, the Lagrange multipliers transform a constrained problem

$$a^*, b^* = \arg\min_{a,b} f(a, b) \text{ subject to } g(a, b) = c$$

into an equivalent form that uses a function

$$\Lambda(a, b, \lambda) = f(a, b) - \lambda(g(a, b) - c).$$

This problem is then solved for $\nabla\Lambda(a, b, \lambda) = 0$, because $\frac{\partial\Lambda}{\partial a,b} = f(a, b) - \lambda g(a, b) = 0$ implies that f and g point to the same direction, and $\frac{\partial\Lambda}{\partial\lambda} = g(a, b) - c = 0$ implies that the original constrain is satisfied. Thanks to the KKT conditions, this method can be generalized to problem with inequality constraints. So, applying the above method to the Linear SVM formulation, we obtain the following Lagrange function:

$$\Lambda(\vec{w}, w_0, \alpha_1, \alpha_2, \ldots, \alpha_n) = \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{n} \alpha_i(y_i(\vec{w}^T\vec{x}^{(i)} + w_0) - 1)$$

setting $\nabla\Lambda = 0$ is equivalent to resolve the following problem:

$$\vec{\alpha}^* = \arg\max_{\vec{\alpha}} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y^{(i)}y^{(j)}\vec{x}^{(i)T}\vec{x}^{(j)} + \sum_{i=1}^{n}\alpha_i, \text{ subject to } \forall\alpha_i > 0 \text{ and } \sum_{i=1}^{n}\alpha_i y^{(i)} = 0$$

with $\vec{\alpha}^* = (\alpha_1, \alpha_2, \ldots, \alpha_n)$. This is the dual formulation for the SVM problem and, as for the Perceptron, it is possible to know some interesting properties using α values:

- $\alpha_i = 0 \implies$ the data point $x^{(i)}$ is not useful for the training;
- $\alpha_i > 0 \implies$ the data point $x^{(i)}$ determines the decision boundary and then it is a *support vector*.

Furthermore, given that $\frac{\partial\Lambda}{\vec{w}} = 0 \iff \vec{w} = \sum_{i=1}^{n}\alpha_i y^{(i)}\vec{x}^{(i)}$, it is possible to determine the decision boundary while w_0 can be recovered by the equation $y^{(i)}(\vec{w}^T\vec{x}^{(i)} + w_0) = 1$ with $x^{(i)}$ support vector.

In the dual formulation, it is possible to see again the use of an inner product $\vec{x}^{(i)T}\vec{x}^{(j)}$, that can be substituted by a kernel function as in the Kernel Perceptron case.

For many years, the kernel SVM have been the state of the art in the classification for many machine learning tasks.

Part I

Learnable Activation Functions

Chapter 3

Neural Networks

"*Deep learning is just a buzzword for neural nets, and neural nets are just a stack of matrix-vector multiplications, interleaved with some non-linearities. No magic there.*"
Ronan Collobert

Introduction

In this chapter an overview of Neural Networks will be made. Due to the vastness of this topic,this chapter wants to be only a general introduction to the subject, referring to other readings (for example [Bishop, 2006, Hagan et al., 1996, Haykin, 1994]) for further information.

In the first part of the present chapter we will make a general description of Neural Networks, describing from which elements it is composed and the main architectures generally used, focusing on Full-Connected Networks and Convolutional Networks as the main theme of the work done. The chapter ends with a discussion of the most used learning algorithms in neural networks, highlighting their limits and crucial points.

3.1 General description

A Neural Network (NN) consists of a number of elements, called *neurons*, arranged together into a structure. Following the schema proposed in [Haykin, 1994], we can briefly define a neuron as a processing unit constituted by:

- a set of external input $\{x_1, x_2, \ldots, x_n\}$;
- a set of real numbers $\{w_1, w_2, \ldots, w_n\}$, called *weights* or *connection strengths*), each one of them associated to an input;
- a threshold value θ called *bias*, with the aim to increase or decrease the input sum;
- an output y; the output of the unit can be modeled as $y = f(\sum_{i=1}^{n} x_i w_i - \theta)$ where $f(\cdot)$ is a function called *activation function*.

The activation function is used for limiting the amplitude of the output neuron; the choice of the activation function could be a critical point for the network performances. In [Cybenko, 1989]it is shown that a non-linear activation function enables the network to approximate arbitrarily complex functions. the most common activation functions used in neural network literature are showed in section 4.2. Another mathematical equivalent formulation of a neuron compacts the bias term θ adding a weight $w_0 = \theta$ and a constant input $x_0 = -1$ transforming the output y as $y = f(\sum_{i=0}^{n} x_i w_i)$ [Bishop, 2006, Hastie et al., 2009].

3.2 Common architectures

Another important characteristic of a Neural Network is the architecture, that is how the neurons are arranged together. The possible architectures are usually divided into two classes: *Recurrent Neural Networks*, which have feedback loops, and *Feed-forward Neural Networks*, without any cycle. In the rest of this chapter, we will focus on Feed-forward Neural Networks as the main objects of the present discussion, referring the interested reader to Recurrent Networks to other readings (see for example [Bishop, 2006, Lipton, 2015])

A Feed-forward Neural Network is organized in *layers*, in which every layer is a set of neurons without any connection between each others; every layer makes independent computations on the data that it receives from the previous layer and passes the results to the next layer, and so on. Finally, the last layer (that can be composed of one or

more neurons) determines the final network output. According to the number of layers, Feed-forward NN can be categorized in two main types:

- *single-layer* Feed-forward networks: networks with a single layer of neurons; every neuron is connected to every input producing a single output; the network results to be a mapping from the d input to n output; in the simplest case, the network has just one neuron (and then a single output y), $d \geq 2$ input $\{x_1, x_2, \ldots, x_d\}$ a set of $\{w_1, w_2, \ldots, w_d\}$ weights and a bias therm θ; the output value can be computed simply as $y = f(\sum_{i=1}^{d} w_i x_i - \theta)$ where f is the activation function (for example the *heavyside* function, see table 4.1); using this setup, the final output of the network can be -1 or $+1$, so that it could be used for binary classification tasks; notorious example of this kind of network are Perceptron [Rosenblatt, 1957] and Adaline [Widrow and Hoff, 1960]. Currently, this kind of network are not considered very interesting because in [Minsky and Papert, 1969] it is shown that a single-layer networks cannot solve problems which are not linearly separable.

- *multi-layer* Feed-forward Neural Networks: all the networks with more than a single layer of neurons. Differently from single-layer Networks, it is shown that every continuous function from input to output can be approximated by a network with enough hidden units, one hidden layer, and proper nonlinear activation functions (see [Cybenko, 1989, Sonoda and Murata, 2017]);

  The possible connections between layers divide the types of feed forward neural networks into 2 different types:

  - *Full-Connected* Network: a FFNN where each neuron of a layer is connected with all the neurons in the next layer (see for example [Bishop, 2006]);
  - textitPartially-Connected Network: there are missing connections between neurons in consecutive layer;
  - *Convolutional* Network: as partially-Connected network, but with some weights shared between units (see [LeCun et al., 1999]).

Figure 3.1 shows an example of two Full Connected architectures. In the remainder of this section, these Full-Connected and Convolutional networks are briefly described. Partially Connected networks are not described as they are equal to convolutional networks without the sharing weights constraint. Given the vastness of the topic, this paragraph

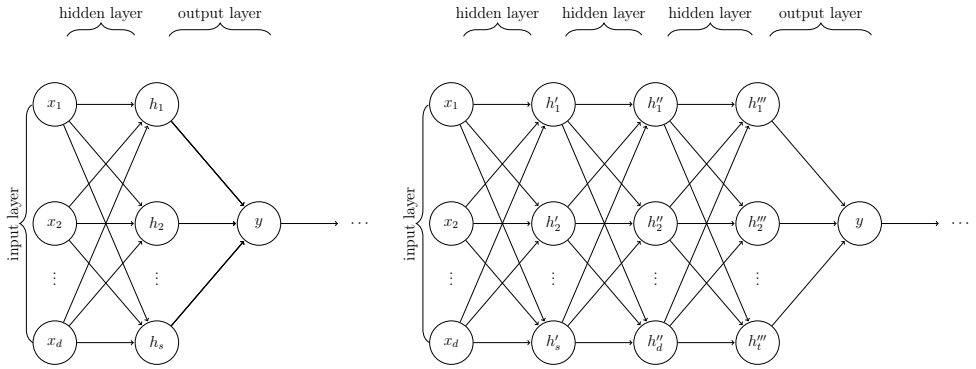


Figure 3.1: Examples of Full-Connected Neural Networks architectures with a single output; on the left, a single-layer FCNN; on the right, a 3-layer FCNN.

only wants to be an introduction to the argument, for a more in-depth discussion, please refer to more appropriate references such as [Bishop, 2006] and [LeCun et al., 2015].

Full-Connected Neural Networks

Neural networks with totally connected levels are historically the most famous and studied networks; they are composed of layers totally connected among them, that is for every level, the output of every neuron is connected to all the neurons of the next level. we will indicate with:

- $\vec{x} \in \mathbb{R}^d$ a generic data point of d components and, for a generic layer l composed by n neurons;
- $W^l \in \mathbb{R}^{n \times d}$ the weights matrix where the generic component $w_{ij}^{(l)}$ is the connection strength between the i-th neuron and the j-th input for the layer l;
- $\vec{b}^{(l)} \in \mathbb{R}^n$ the biases of the n neurons.

The first layer is said the *input layer*, the last layer the *output layer*, and all layers that are placed between the first and the last layers are said the *hidden layers*. Furthermore, an activation function $f^{(l)} : \mathbb{R}^n \to \mathbb{R}^n$ is associated to each leayer. Using this notation, the output of a generic layer $z^{(l)}$ with input $\vec{x}$ can be expressed as $z^{\vec{(l)}} = f^{(l)}(W\vec{x} + \vec{b}^{(l)})$. So, given a generic multilayer feed-forward network composed by L layers with weight matrices $(W^{(1)}, \ldots, W^{(L)})$ and $(\vec{b}^{(1)}, \vec{b}^{(2)}, \ldots, \vec{b}^{(L)})$ biases, given an input point $\vec{x}$, the final output $\vec{y}$ of the network can be computed as described in Algorithm 2. So, the parameters necessary for the construction of a generic FC layer l are:

Algorithm 2: Output computation of a Full-Connected FFNN (forward propagation)

Input: a point $\vec{x}$, weight matrices $(W^{(1)}, \ldots, W^{(L)})$, biases vectors $(b^{(1)}, \ldots, b^{(L)})$, activation functions $(f^{(1)}, \ldots, f^{(L)})$

Output: The final output y

$\vec{z}^{(0)} \leftarrow \vec{x}$

forall $l \leftarrow 1$ *to* L **do**

  $\vec{z}^{(l)} \leftarrow f^l(W^{(l)}\vec{z}^{(l-1)} + \vec{b}^{(l)})$

$\vec{y} \leftarrow \vec{z}^{(L)}$

- the number of neurons $n^{(l)}$;

- the activation function $f^{(l)}$.

Convolutional Neural Networks

Convolutional networks are particularly suitable for signal-related tasks (e.g. image processing, audio processing); essentially, CNN are mainly composed by *convolutional levels* that are based on three key principles:

- sparse interactions (or sparse weights): connections between consecutive levels are reduced, so as to decrease the number of parameters that need to be learned;

- parameter sharing: some connections share the same weight values;

- equivalent representations: similar input features are represented in similar way.

In a convolution level, the input is considered divided into overlapping regions of fixed size called *receptive field* and each neuron is connected only to a single region of the input. The overlap and the distance between consecutive windows is determined by a parameter called *stride*. The weight-sharing mechanism is obtained through the convolution operation [Dumoulin and Visin, 2016]: the weights are stored in a matrix called *filter* which acts locally on each individual input window through the convolution operator; the output of the convolution operation is then used as argument of an activation function; the final output is called a *feature map*. Each convolutive layer can have many filters, and then it can produce many features map; so, every filter can be viewed as a group of neurons that share the same weights that, through a convolution operation, act on different regions of the input. Generally, for every convolutive layer a *pooling* operation follows, which

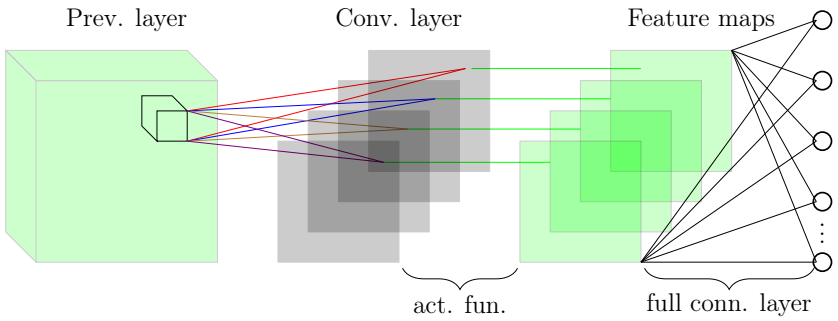


Figure 3.2: Examples of a Convolutional Neural Network architectures with an FC layer as last layer. Every input region is processed by a different filter (grey) and then it is fed to an activation function (e.g., ReLU). The ouput is then send to a pooling layer, or to the next layer that can be another convolutional layer or the last layer, that is usually a Full Connected layer.

is responsible for extracting useful statistics from local input areas. So, the parameters necessary for the construction of a generic convolution layer l are:

- the number of filters $n^{(l)}$;
- the receptive field size $M^{(l)}$;
- the stride $s^{(l)}$
- the activation function $f^{(l)}$.

The last layer of a Convolutive Network is usually a Full Connected Layer or a Global Pooling Layer as proposed in [Lin et al., 2013]. Figure 3.2 shows an example of a convolutional architecture.

3.3 Learning in neural networks

Optimizing a neural network objective function means to find the best set of weights and biases such that the network gives the minimum risk on a given training set and that best generalizes the problem, that is avoiding as much as possible the over-fitting. Although the objective function is not convex, methods based on gradient descent [Robbins and Monro, 1951] still lead to acceptable results; in this paragraph some of these methods are described. Anyway, almost all these learning algorithm in neural networks suffer of three critical issues:

1. *initial value of the parameters*: different weights' initializations can lead the network to converge to a different local minima, so the initial value of the weights can be decisive for the network performances; a classic approach [Krizhevsky et al., 2012] consists in initializing weights using Gaussian noise with mean equal to zero and standard deviation with value 0.01; other methods [Glorot and Bengio, 2010, Glorot et al., 2011, He et al., 2015] propose to estimate the standard deviation using some features of the layers. Other methods were proposed, e.g. Random Walk Initialization [Sussillo and Abbott, 2014]; however, a rule for the right initialization schema to use does not seem to exist, the most appropriate criterion seems to depend on the task to do and on the activation functions used.

2. *hyper-parameters selection*: the choice of the learning algorithm parameters, as for example the learning rate in gradient descent method, can affect the training progress in terms of convergence and velocity; for example a too low learning rate can lead to a very slow learning, while a too large value can lead to a non-convergence situation; the choice of a proper learning rate is still an open topic in the machine learning community. One of most simple way is to use a parameter selection method as cross validation (see section 2.2); another method is to use a recent learning algorithm, as AdaGrad [Duchi et al., 2011], which uses different learning rates computed from the gradients, or techniques like the one proposed in [Smith, 2015] where a range within which the learning rate can be varied is computed cyclically.

3. *gradient computation*: gradient descent methods need to compute many times the first order gradient of the loss function, so it is necessary an efficient method to compute it; a computational acceptable way is to use the *backpropagation* algorithm (see for example [Rumelhart et al., 1986, Bishop, 2006]) which is the current standard de facto for calculating the gradient in algorithms based on gradient descent.

Gradient descent

The most popular algorithm to train a classifier is the *gradient descent*, also known as *steepest descent* [Robbins and Monro, 1951]. The basic idea is, given an initial set of parameters w_0 and a training set Tr, we can iteratively find a better set of parameters $w_t = w_{t-1} - \eta \frac{\partial \mathscr{L}(w_{t-1}, Tr)}{\partial w_{t-1}}$; the negative gradient is the direction in which the objective function $\mathscr{L}$ decrease most rapidly and η (called*learning rate* or *step size*) controls how large is the step during each iteration; the pseudo-code is shown in algorithm 3.

Algorithm 3: Gradient Descent algorithm

Input: a starter set of parameters w_0, a training set Tr, a learning rate η
Output: The final set of parameters w_f,

```
w ← w_0;
cost ← 𝓛(Tr, w);
while stop criteria is not met do
    compute the gradient g;
    w ← w − ηg;
    cost ← 𝓛(Tr, w);
w_f ← w
```

Usually the stop criteria is a maximum number of iterations or a threshold on the objective function.

The main drawback of the gradient descent training algorithm is that it requires many iterations for functions which have long, narrow valley structures. Furthermore, the start point w_0 could affect the learning result; usually, it is chosen random but it could not be the best choice. Gradient descent is one of the most famous learning algorithm for classifiers as neural networks, but it could result too slow, especially when data or the number of parameters is large; there are many variations of the original algorithm which try to overcome these cases.

Mini-batch stochastic gradient descent

Mini-batch Stochastic Gradient Descent (see for example [Bottou et al., 2018]) is the most used GD variation, especially in neural networks training: at each iteration, rather than computing the gradient $g = \nabla\mathscr{L}(w)$, a gradient descent takes a randomly samples $B \subseteq Tr$ from Tr and computes $g = \nabla\mathscr{L}(w)$ using only the B points instead of all the points in Tr. In the extreme case, when $|B| = 1$, the algorithm is also called *Online Gradient Descent*. A full iteration of the algorithm over all the points in the dataset Tr is said an *epoch*.

Gradient descent with momentum

Momentum [Rumelhart et al., 1986, Sutskever et al., 2013, Polyak, 1964, Rutishauser, 1959] is a modification of the GD algorithm based on the observation that the convergence might be improved smoothing out the oscillations of the trajectory in the parameters space; parameters update makes a linear combination between the gradient and the previous

update (momentum) using a momentum term $0 \leq \alpha \leq 1$ to set up the momentum effect; the algorithm is described in Algorithm 4.

An interesting interpretation of view of momentum effect is shown in [Hagan et al., 1996], where the momentum is explained as a kind of low-pass filter on the gradient trajectory.

Algorithm 4: Gradient Descent algorithm with Momentum

Input: a starter set of parameters w_0, a training set Tr, a learning rate η, α
Output: The final set of parameters w_f

```
1 w ← w_0;
2 v ← 0;
3 cost ← 𝓛(Tr, w);
4 while stop criteria is not met do
5 |   compute the gradient g;
6 |   v = αv − ηg;
7 |   w ← w + v;
8 |   cost ← 𝓛(Tr, w);
9 w_f ← w
```

Gradient descent with Nesterov momentum

In [Nesterov, 1983, Bengio et al., 2013] is introduced a variation on GD + momentum which computes the gradient value in the approximated new position instead of in the current position, so that the descent can be faster. The pseudo-code is shown in algorithm 5.

Gradient descent with variable learning rate

The cost function for a classifier could not be a quadratic function, having instead different shape in different area of parameters space. A common way to speed up the convergence is modifying the learning rate during the training; a possible approach in this direction is shown in [Vogl et al., 1988].

Adaptive Gradient algorithms

The benefits given by the variable learning rate may increase if a different learning rate is applied to different parameters, as proposed in AdaGrad and its variations (see [Duchi

Algorithm 5: Gradient descent with Nesterov momentum

Input: a starter set of parameters w_0, a training set Tr, a learning rate η, α
Output: The final set of parameters w_f

$w \leftarrow w_0$;
$v \leftarrow 0$;
$cost \leftarrow \mathscr{L}(Tr, w)$;
while *stop criteria is not met* **do**
    compute the gradient g respect to $w + v$;
    $v = \alpha v - \eta g$;
    $w \leftarrow w + v$;
    $cost \leftarrow \mathscr{L}(Tr, w)$;
$w_f \leftarrow w$

Algorithm 6: Adagrad algorithm

Input: a starter set of d parameters w_0, a training set Tr, a learning rate η, α
Output: The final set of d parameters w_f

$w \leftarrow w_0$;
$hist \leftarrow 0 \in \mathbb{R}^d$;
$cost \leftarrow \mathscr{L}(Tr, w)$;
while *stop criteria is not met* **do**
    compute the gradient g;
    foreach g_i **do**
        $hist_i = hist_i + g_i^2$;
        $w_i = w_i - \frac{\eta}{\sqrt{hist_i}} g_i$
    $cost \leftarrow \mathscr{L}(Tr, w)$;
$w_f \leftarrow w$

et al., 2011]); with AdaGrad algorithm, the parameters that during the learning received little updates will have larger learning rate respect to parameters with smaller updates. A pseudo-code of this algorithm is presented in Algorithm 6. The main drawback of this algorithm is that, during the training, the learning rate can go to zero because of the accumulation of the squared gradients in the denominator.

To deal the vanishing learning rate of AdaGrad algorithm, there are at least two variation, RMSprop and AdaDelta [Ruder, 2016, Zeiler, 2012] which restricts the window of accumulated past gradients to a moving average on fixed size window. Other algorithms as ADAM, AMSGrad [Kingma and Ba, 2014, Reddi et al., 2018] try to merge together RMSProp/AdaDelta approach with momentum, while NADAM [Dozat, 2016] merge it with

Algorithm 7: Newton-Raphson algorithm

Input: a starter set of parameters w_0, a training set Tr
Output: The final set of parameters w_f

$w \leftarrow w_0$;
$cost \leftarrow \mathscr{L}(Tr, w)$;
while *stop criteria is not met* **do**
    compute the gradient g;
    compute the Hessian matrix H;
    compute the inverse Hessian matrix H^{-1};
    $w \leftarrow w_{t-1} - H^{-1}g$;
    $cost \leftarrow \mathscr{L}(Tr, w)$;
$w_f \leftarrow w$

Nesterov.

Conjugate-gradient method

Conjugate-gradient is a method based on gradient descent and Newton-Raphson method (see for a more accurate description [Galántai, 2000]). Newton's method uses the Hessian matrix $H = \nabla^2\mathscr{L}(w)$ to find a better training direction respect to gradient descent; in an iterative way, it constructs a quadratic approximation of the objective function which matches the first and second derivative values. The algorithm then minimizes the approximate quadratic function instead of the original objective function.

A quadratic approximation of an objective function $\mathscr{L}$, using a starting point w_0, can be obtained using the Taylor expansion $E = \mathscr{L}(w_0) + \nabla\mathscr{L}(w_0)\cdot(w - w_0) + \frac{1}{2}\cdot(w - w_0)^2 \cdot H(w_0)$; setting $\nabla\mathscr{L}(w_0) = 0$, the iterative expression has the form $w_t = w_{t-1} - H^{-1}\nabla\mathscr{L}(w_i)$. The algorithm is shown in Algorithm 7.

One of the main drawback of this algorithm is that, if the Hessian Matrix is not positive definite, the movement can be toward a maximum instead of a minimum; furthermore, the computation of the Hessian Matrix and its inverse is often complicated or intractable, requiring a lot of computation.

The conjugate gradient algorithm tries to overcome these problem, showing generally a faster convergence than gradient descent algorithm. Conjugate-gradient method is based on the definition of *conjugate* vectors of a matrix, i.e. a set of vectors $\vec{v}^{(1)}, \vec{v}^{(2)}, \ldots, \vec{v}^{(k)}$ is mutually *conjugate* with respect to a symmetric positive definite matrix A if and only if $\forall i \neq j,\ \vec{v}^{(i)}A\vec{v}^{(j)} = 0$; The matrix A can be the Hessian Matrix, but it is hard to

Algorithm 8: A conjugate-gradient algorithm

Input: a starter set of parameters w_0, a training set Tr
Output: The final set of parameters w_f

compute the gradient g_0;
$p \leftarrow -g_0$; $w \leftarrow w_0$;
$cost \leftarrow \mathscr{L}(Tr, w)$;
while *stop criteria is not met* **do**
    compute the gradient g_t;
    $\beta = \frac{g_t^T g_t}{g_{t-1}^T g_{t-1}}$;
    $p \leftarrow -g_t + \beta p$;
    search η s.t. minimize $E(Tr, w_t + \eta p)$; ▷ Line Search
    $w \leftarrow w_t + \eta p$;
    $cost \leftarrow \mathscr{L}(Tr, w)$;
$w_f \leftarrow w$

compute; furthermore, we can found a set of *conjugate* directions without to compute directly the Hessian matrix(see [Luenberger and Ye, 2015, Hagan et al., 1996]) but instead of constructing a set of vectors in a iterative way. Briefly, defining the change of the gradient between two iterations as $\Delta g_k = g_{k+1} - g_k$, in each iteration a vector $p^{(k)}$ orthogonal to the set $\{p^{(1)}, p^{(2)}, \ldots, p^{(k-1)}\}$ is built; this vector has the form $p^{(k)} = -g_k + \beta_k p^{(k)}$ where β_k can be computed in various ways, as $\beta_k = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}}$ (given by Fletcher and Reeves, for alternative ways, see for example [Hagan et al., 1996]). A possible pseudo-code of Conjugate-gradient method is shown in 8.

Each iteration of the conjugate gradient algorithms requires a Line Search step that is computationally expensive, since it requires to compute the network response to all training points many times for each search. In [Møller, 1993] is proposed a method to avoid the time-consuming Line Search. However, there are other approaches that tries to overcome the computational problems of Newton's method, for example the Quasi-Newton methods (see for example [Nocedal and Wright, 2006]) which build an approximation to the inverse Hessian at each iteration of the algorithm using only information on the first derivatives of the objective function.

Chapter 4

Activation Function in Neural Networks

Introduction

This chapter wants to give an overview on the most used functions present in the literature proposing a possible taxonomy to classify them. The chapter is structured as follows: in Section 4.1 a taxonomy for the activation functions is proposed, dividing the activation functions in fixed-shape family, which are briefly discussed in Section 4.2, and learnable-shape family, which are discussed in Section 4.3 in more detail. Finally, in Table 4.2 the best accuracy values found in literature on some datasets are reported; it is important to keep in mind that the values shown are only indicative, as they do not take into account the network architecture actually used.

4.1 A taxonomy of activation functions

The activation function plays a central role for the expressiveness of a neural network; the introduction of new activation functions has contributed to renew the interest of the scientific community for neural networks; the use of ReLU, Leaky ReLU parametric ReLU, and similar activation functions in neural networks has been shown to improve significantly the network performances, thanks to properties as the no saturation feature that helps to avoid typical learning problems as vanishing gradient. So, individuating new functions that can potentially improve the results is still an open field of research.

In this regard, we propose a possible taxonomy of the main activation functions presented in the literature. A diagram of the proposed taxonomy of activation functions is given in Figure 4.1.

For what has been said so far, a first division of the activation function can be made on the possibility of being able to modify their shape based on the training data. So, we can divide the activation functions currently present in the literature into two main categories:

- **fixed shape activation functions**: in this category fall all the activation functions the behaviour of which is fixed a priori as, for example all the classic activation functions used in neural network literature, such as sigmoid, tanh, ReLU etc. However, since the introduction of rectified functions (as the ReLU) can be considered a turning point in literature, resulting in one of the factors that gave new life to the study of neural networks, we can further divide this kind of functions in:
  - rectified-based function: all the function belonging to the rectifier family, as ReLU, LReLU, etc.
  - classic activation function: all the function that are not in rectifier family, as the sigmoid, tanh, step functions.
- **learnable-shape activation functions**: in this category we put all the activation functions the behaviour of which is learned from a set of data; the idea behind this kind of functions is to search the best function shape using knowledge given by training data. The studies currently present in the literature model the proposed functions mainly in two ways:
  - using sub-networks: the activation function is modelled through one or more sub-networks which determine the final shape of the function; in many cases, this kind of functions does not require changes in the learning algorithm, because the function parameters are fully integrated into the network, and therefore can be learned like any other network weight;
  - using sub-functions: the activation function is modelled through a set of explicit sub-functions which are combined together; in many cases, this kind of functions requires changes in the learning algorithm to handle the parameters that determine its shape. However, we can divide these functions into other three subcategories, that are:
    * Quasi-fixed: their shape is given by a slight change learned from the data of a basic fixed-shape function;
    * Interpolation based: their shape is computed using classical interpolation techniques;

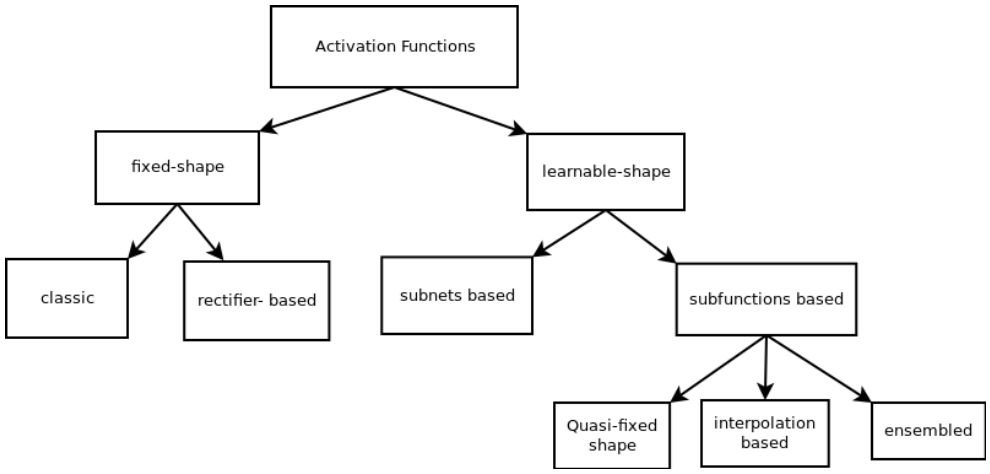


Figure 4.1: A proposed taxonomy of the activation functions present in the Neural Networks literature.

* Ensembled: their shape is the combination of many functions, also very different from each other.

4.2 Fixed-shape activation functions

With Fixed-shape activation functions we refer to all activation functions in which there is no learnable parameter, i.e. its values are only dependent on the activation value given in input.

As far as this work is concerned about learnable activation function, in this section a description regarding the main fixed-shape activation functions used in neural networks is made; This is motivated by the fact that many learnable activation functions in literature are proposed as combination or variation of fixed-activation function.

This section is divided into two parts, the first one that talks about the classic activation functions not based on rectifiers used primarily in the past, the second one that talks about ReLU and its possible improvements given by changing its shape in a slight ways so limit its drawbacks using just new hyper-parameters.

Classic activation functions

A list of the most common activation functions in feed forward neural network literature is given in table 4.1. For many years, bounded activation function as sigmoid [Cybenko, 1988]

Name	Expression	Range
Identity	$\mathrm{id}(x) = x$	$(-\infty, +\infty)$
Step (Heavyside)	$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$	$\{0, 1\}$
Bipolar	$f(x) = \begin{cases} -1 & \text{if } x < 0 \\ +1 & \text{otherwise} \end{cases}$	$\{-1, 1\}$
Sigmoid	$\mathrm{sig}(x) = \frac{1}{1+e^{-x}}$	$(0, 1)$
Bipolar sigmoid	$f(x) = \frac{1-e^{-x}}{1+e^{-x}}$	$(-1, 1)$
Tanh	$\tanh(x) = \tanh(x)$	$(-1, 1)$
Hard tanh	$f(x) = \max\big(-1, \min(1, x)\big)$	$[-1, 1]$
Absolute	$f(x) = \lvert x \rvert$	$(0, +\infty)$
Cosine	$f(x) = \cos(x)$	$[-1, 1]$

Table 4.1: Commonly used activation functions

or hyperbolic tangent [Chen, 1990] have been the most famous activation function for neural networks; unfortunately, these functions suffer of the *vanishing gradient problem* (see [Bengio et al., 1994]) compromised learning in networks with many levels, although they have excellent results especially in shallow network architecture (see for example [Glorot et al., 2011, Pedamonti, 2018]). In [Cybenko, 1989, Hornik et al., 1989] is shown that any continuous function can be approximated by a feed-forward network with a single hidden layer containing a finite number of neurons on compact subsets, under the assumptions that the activation function are non-constant, bounded and monotonically-increasing continuous function. This theorem proved that activation functions like the identity function or any other linear function, used for example in early networks as ADALINE or MADALINE [Widrow and Hoff, 1960, Widrow and Lehr, 1990], are not suitable to approximate any continuous function. In [Pinkus, 1999, Sonoda and Murata, 2017] is shown that the previous assumptions on the activation functions were too strong, showing that also neural networks with unbounded but non-polynomial activation functions (e.g. ReLU, [Nair and Hinton, 2010]) respect the universal approximation conditions. Furthermore, to use unbounded activation function seems to attenuate the vanishing gradient problem (see [Nair and Hinton, 2010]); this interesting property opened new frontiers in research on classifiers based on neural networks and on machine learning in general.

Rectifier-based activation functions

Over the last years, many different activation function have been proposed, most of which are inspired by the success obtained by ReLU and therefore fundamentally of the same type, with small variations compared to the original work of [Glorot et al., 2011]. This kind of activation functions are the standard *de facto* in modern neural network design, surpassing other classic functions as $Sigmoid$ and $Tanh$ used in the past literature. One of the first studies that showed the performance improvements was [Glorot et al., 2011], where DNNs equipped with ReLU activation functions perform better respect to networks with sigmoid units. The main advantage of using rectified activation functions is that they avoid the vanishing gradient problem [Bengio et al., 1994], which has been one of the main problems for learning deep networks for many years. This event has opened new frontiers in research, focusing the scientific community attention in seeking new activation functions in order to further improve performance.

- **ReLU**: the introduction of ReLU within the neural networks has been a real turning point within the scientific community allowing to reach new and promising results in the field of machine learning. Introduced for the first time by [Hahnloser et al., 2000], but it was with the works done by [Nair and Hinton, 2010],[Glorot et al., 2011] that has gained popularity thanks to the obtained results, becoming the new de facto standard in place of the activation functions used previously. Defined as $\text{ReLU}(a) = \max(0, a)$, these function has important positive aspects has:

  - to alleviate vanishing gradient: being not bounded in at least one direction, ReLU function doesn't allow saturation conditions generally present with other functions as sigmoid or tanh that, as the number of layers of the network increases, leads to the vanishing gradient problem;
  - sparse coding: using ReLU, the percentage of neurons that are active at the same time is very low; the benefits of sparsity can be seen in [Glorot et al., 2011] and can be resumed in a better dimensionality of the representation and in a greater invariance to slight input changes.

  However, ReLU function is not free from defects, as the non-differentiability at zero or the "dying" ReLU problem [Maas et al., 2013], i.e. when a large negative unit bias is learned causing the output of the ReLU to be always zero regardless of the input.

- **Leaky ReLU** and its randomized version: One of the first rectified-based activation function was LReLU, proposed by [Maas et al., 2013]. More in details, a ReLU function is defined as $\text{ReLU}(a) = \max(0, a)$; this function can originate at least two problems:

  - gradient-based learning algorithm will not change the weights of units with potential in the negative semi-axis because the gradient is 0 whenever the unit is not active;
  - assuming that the lack of activation of some neurons does not affect the convergence, learning could still be slow because of the gradient equal to 0 for non-positive input values.

  LReLU was proposed in an attempt to alleviate potential problems given by ReLU in the event of too low activations, it is defined as:

$$\text{LReLU}(a) = \begin{cases} a & \text{if } a > 0 \\ 0.01 \cdot a & \text{otherwise.} \end{cases}$$

  A Leaky Rectifier activation function allows learning to continue giving a small gradient when the unit is saturated and not active. Anyway, empirical results show that Leaky rectifiers perform nearly identically to standard rectifier DNNs, resulting in a negligible impact on performances. A randomized version, (Randomized Leaky ReLU), where the weight value for a is sampled by an uniform distribution $U(l, u)$ with $l < u$ and $l, u \in [0, 1)$ was proposed in [Xu et al., 2015].

- **Truncated Rectified**: [Memisevic et al., 2014], tackle the problem by focusing on a particular type of DNN, i.e. autoencoders, starting from the observation that this type of network tends to have large negative bias to achieve sparsity but that can make difficult to learn non-trivial manifold as a side effect. From this observation, the authors propose the Truncated Rectified as activation function which can be defined as:

$$\text{TRec}_t(a) = \begin{cases} a & \text{if } a > t \\ 0 & \text{otherwise.} \end{cases}$$

  to note that the t point is in $TRec$ is a non-continuity point, unlike ReLU in which the threshold point (which is 0) is only a non-differentiable point. Authors use TRec only during training, and then replace it with ReLU during testing.

- **Softplus**: Introduced by [Dugas et al., 2000], the *softplus* function can be seen as a smooth approximation of ReLU function. It is defined as

$$\text{softplus}(a) = \log\left(1 + \exp(a)\right)$$

the smoother form and the lack of points of non-differentiation could suggest a better behavior and an easier training as an activation function; however, experimental results made in [Glorot et al., 2011] tend to contradict that hypothesis, suggesting that ReLU properties can help supervised training better than softplus functions.

- **Exponential Linear Unit**: Introduced by [Clevert et al., 2015], ELU introduce is an activation function with negative values that keeps the identity for positive arguments. It is defined as:

$$\text{ELU}(a) = \begin{cases} a & \text{if } a > 0 \\ \alpha \cdot (\exp(a) - 1) & \text{otherwise.} \end{cases}$$

with α hyper-parameter that controls the value for negative inputs. The negative values given by ELU units pushes the mean of the activations closer to zero, giving a faster learning (as showed in [Clevert et al., 2015]), at the cost of an extra hyper-parameter that requires to be set.

- **Sigmoid-weighted Linear Unit**: Originally proposed in [Elfwing et al., 2018], Sigmoid-weighted Linear Unit is a sigmoid function weighted by its input, i.e.:

$$\text{SiLU}(a) = a \cdot \text{sig}(a)$$

In the same study, another activation function is proposed, that is the derivative of SiLU uses as activation function, i.e.:

$$\text{dSiLU}(a) = \text{sig}(a)\big(1 + a(1 - \cdot\text{sig}(a))\big)$$

These function are tried on Reinforcement Learning Task, but further results are given in [Ramachandran et al., 2017] where the same function is tested with the name of Swish-1.

4.3 Learnable-shape Activation functions

The idea of using learnable activation functions is not new in neural networks. Many studies were published on the subject as early as the 1990s, for example [Piazza et al., 1992, Piazza et al., 1993, Guarnieri, 1995, Chen and Chang, 1996]. In recent years, the renewed interest in the field of neural networks has led the scientific community to consider again the hypothesis that the learnable activation functions can improve the performance of neural networks with respect to fixed-shape functions. One of the main reasons for this renewed interest is the availability of more powerful computing resources combined with a large amount of data that can be processed. This has made it possible to train models of increasingly complex neural networks with good results. In this section we describe and analyze the main methods in literature related to the activation functions that can be learned. We divide them into groups based on their main characteristics. Specifically, we have isolated the following families of learnable activation functions:

- **Sub-network based**: activation functions which are fitted by a sub-network inside the main network. The key property of this kind of functions is that they don't require any change (or a minimal) in the network learning algorithm, because their learnable parameters are integral parts of the general network.
- **sub-functions based**: activation functions which shape is modelled using one or more explicit sub-functions in different ways; we divide these activation functions in the following sub-categories:
  - **Quasi-fixed shape activation functions**: all the activation functions whose form is bound to be a simple variation of a basic function such as sigmoid or tanh;
  - **Interpolation-based Activation Functions**: all the activation functions which are approximated using classical interpolation techniques;
  - **Activation functions based on ensemble methods**: all the activation functions which are obtained by some combination of many functions.

In the following of this section we will describe each of these categories.

4.3.1 Sub-network based functions

With Sub-network based functions we refer to all the activation functions which are originally modelled as network that can be integrated into the network without modification or addition to the learning algorithm; in fact, the sub-network structure allows the function to be trained in a "transparent" way for the rest of the network and for the chosen training algorithm.

Maxout unit

[Goodfellow et al., 2013] was one of the first modern study that proposed a more general learnable activation function. The name *Maxout* was given by the fact that output is the max of a set of linear functions. Maxout units are more than simply activation function, since they use multiple activations for every neuron instead of a single activation given by $a = \vec{w}\vec{x} + b$ as for classical neurons. More precisely, a Maxout unit defines a vector of activations $\vec{a} = (a_1, a_2, \ldots, a_k)$ with $\forall i \in \{1, k\},\ a_i = \vec{w}^{(i)T}\vec{x} + b^{(i)}$ with $\vec{x} \in \mathbb{R}^d$ output given by the previous layer, $\{\vec{w}^{(1)} \in \mathbb{R}^d, \vec{w}^{(2)} \in \mathbb{R}^d, \ldots, \vec{w}^{(k)} \in \mathbb{R}^d\}$, $\{b^{(1)} \in \mathbb{R}^m, b^{(2)} \in \mathbb{R}^m, \ldots, b^{(k)} \in \mathbb{R}^m\}$ parameters to be learned. and then returns their maximum, i.e.

$$\text{Maxout}(\vec{a}) = \max_{1 \leq j \leq k} \{a_i\}.$$

In other words, Maxout units take the maximum value over a subspace of k trainable linear functions of the same input $\vec{x}$, obtaining a piece-wise linear approximator capable of approximating any convex functions. The same model can be defined putting all the $\vec{w}^{(i)}$ vectors as column of a single matrix $W \in \mathbb{R}^{d \times k}$ and all the $b^{(i)}$ scalars in a single vector $\vec{b} \in \mathbb{R}^k$, obtaining $\vec{a} = W^T\vec{x} + \vec{b}$.

To notice that, setting $k = 2$ and $\vec{w}^{(1)} = 0, b^{(1)} = 0$, Maxout becomes

$$\begin{aligned}\text{Maxout}(\vec{a}) = &\max(\vec{w}^{(1)T}\vec{x} + b^{(1)}, \vec{w}^{(2)T}\vec{x} + b^{(2)}) = \\ &\max(0, \vec{w}^{(2)T}\vec{x} + b^{(2)}) = \text{ReLU}(\vec{w}^{(2)T}\vec{x} + b^{(2)}),\end{aligned}$$

In a similar way, Maxout unit can be made equivalent to Leaky ReLU, so Maxout can be viewed as a generalization of classic rectifier units. Being Maxout constituted by a set of feed-forward sub-networks, its parameters can learned together with the whole network using classical SGD approach. By running a cross-validation experiment the authors of [Goodfellow et al., 2013] found that Maxout offers a clear improvement over ReLU units in terms of classification errors. Despite the performance, this approach requires many new weights respect to a classic network based on ReLU, namely k times the number of parameters for every single neuron, greatly increasing the cost of the learning process.

Multi-layer Maxout

Authors of [Sun et al., 2018] generalize the Maxout-based networks using a function composition approach that they call Multi-layer Maxout Network (MMN) that further increase the number of parameters. To limit the computational costs introduced, the authors themselves propose to replace just a portion of activation functions in traditional DNN with MMNs as a trade-off scheme between the accuracy and computing resources.

Probabilistic Maxout

In [Springenberg and Riedmiller, 2013] is described Probout, a stochastic generalization of the maxout unit trying to improve its invariance, which replace the maximum operation in Maxout with a probabilistic sampling procedure, i.e.

$$\text{Probout}(\vec{a}) = a_i, \text{ with } i \sim \text{Multinomial}(p_1, p_2, \ldots, p_k)$$

where $p_i = \frac{\exp(\lambda a_i)}{\sum_{j=1}^{k} \exp(\lambda a_j)}$ with λ hyperparameter. The maximum operation substitution in Maxout arises from the observation that to use other operation could be useful to improve the performances. To notice that the Probout function reduces to the Maxout for $\lambda \to +\infty$.

Network In Network

[Lin et al., 2013] proposed a learnable activation function designed for convolution networks: they substitute the activation functions of a classic convolution layer with a full-connected multilayer perceptron. For instance, given a CNN, a MLP sub-net with l layers is applied to every x_{ij}, where with x_{ij} we refer to the input patch centered on the point (i, j). This operation results in a set of functions $f^{(i)}$ of the shape:

$$\begin{aligned} f^{(1)}_{i,j,c_1} &= \max(W^{(1)}_{c_1} x_{ij} + \vec{b}_{c_1}, 0), \\ &\vdots \\ f^{(l)}_{i,j,c_l} &= \max(W^{(l)}_{c_l} f^{l-1}_{ij} + \vec{b}_{c_l}, 0). \end{aligned}$$

where c_i are the input channels and l the chosen number of layers for the MLP. So, this network can be viewed as an MLP with ReLU units. So, the NIN output is a map where every point is the output of last layer function $f^{(l)}$. Despite the good performances obtained, this methods requires a lot of new parameters to learn especially if l is large.

Convolution In Convolution

Based on [Lin et al., 2013], authors of [Pang et al., 2016] proposed Convolution in Convolution which uses a sparse MLP instead of the classic full-connected MLP as activation function; NIN and CIC seem to move away from the concept of variable activation function of a single neuron because the MLP could have common connections with other nodes of the previous layers, in other words there are no constraints on the fact that the final layer of the MLP can have more than one output.

Batch-Normalized Maxout Network in Network

In [Chang and Chen, 2015] are combined together two different learnable architecture, i.e. Maxout and NIN, with Batch Normalization [Ioffe and Szegedy, 2015]; the general architecture replaces the ReLU functions present in NIN with Maxout to avoid the zero saturation followed by Batch Normalization to avoid the problems connected with changes in data distribution [Ioffe and Szegedy, 2015]. For instance, given a CNN, a MIN sub-net with l layers and Maxout sub-nets composed by $\{k_1, k_2, \dots k_l\}$ activations (see Section 4.3.1 is applied to every x_{ij}, where with x_{ij} we refer to the input patch centered on the point (i, j). So, the final function si given by:

$$\begin{aligned} f^{(1)}_{i,j,c_1} &= BN(W^{(1)}_{c_1} x_{ij} + \vec{b}_{c_1}), \\ &\vdots \\ f^{(l)}_{i,j,c_l} &= \max_{1 \le m \le k_l} \{BN(W^{(l)}_{k_m} f^{l-1}_{ij} + \vec{b}_{c_l})\}. \end{aligned}$$

4.3.2 Quasi-fixed activation functions

With Quasi-fixed activation functions we refer to all the function with a shape very similar to a given basis fixed-shape function, but whose diversity from the latter comes from a set of trainable parameters. the addition of these parameters therefore requires changes, even minimal ones, in the learning algorithm, for example, in the case of using gradient-based methods with backpropagation, the partial derivatives of these new parameters are needed.

Adjustable Generalized Sigmoid

A first attempt to have a learnable activation function was given in [Hu, 1992]; the proposed activation functions was a generalization of classic Sigmoid $\text{sig}(a) = \frac{1}{1+\exp(-a)}$ which adds

two learnable parameters α, β to rule the function shape, i.e.:

$$\mathrm{AGSig}_{\alpha,\beta}(a) = \frac{\alpha}{1 + \exp(-\beta a)}$$

Both values are adjustable together using classic gradient descent algorithm.

Sigmoidal selector

In [Singh and Chandra, 2003] is proposed the following class of sigmoidal functions:

$$\mathrm{S}_k(a) = \Big(\frac{1}{1 + \exp(-a)}\Big)^k$$

parameterized by a value $k \in (0, +\infty)$. A neural network can use any of them and have the universal approximation property [Haykin, 1994], by a previous choice of k parameter. In [Chandra and Singh, 2004] the k parameter is learned (and so the effective function is selected) by the gradient descent algorithm + backpropagation together with the other network parameters.

Adjustable Generalized Hyperbolic Tangent

Proposed in [Chen and Chang, 1996], this activation function generalizes the classic hyperbolic tangent function $\tanh(a) = \frac{1-e^{-2a}}{1+e^{-2a}}$ introducing two learnable parameters α, β:

$$\mathrm{AGTanh}_{\alpha,\beta}(a) = \frac{\alpha\big(1 - \exp(-\beta a)\big)}{1 + \exp(-\beta a)}$$

in this function, α adjusts the saturation level while β controls the slope. This two parameters are learned using classic gradient descent algorithm combined with back-propagation together with the weights between neurons, initializing all the values in a random way. In [Yamada and Yabuta, 1992a, Yamada and Yabuta, 1992b] a similar activation function was proposed, with the main difference that it used just one learnable parameter which controls the slope.

Parametric ReLU

[He et al., 2015] introduced another ReLU-like function but able to partially learn its shape in a linear way from the training set transforming the negative part using a parameter w; the function Parametric ReLU, can be defined as:

$$\mathrm{PReLU}_\alpha(a) = \begin{cases} a & \text{if } a > 0 \\ \alpha \cdot a & \text{otherwise.} \end{cases}$$

The additional α parameters are learned jointly with the whole model using classical gradient based methods with back propagation without weight decay to avoid pushing α to zero during the training. The gradient of the loss function L respect to the new parameter α is given by $\frac{\partial L}{\partial \alpha} = \min(0, a)\frac{\partial L}{\partial \text{PReLU}_\alpha(a)}$ which the term $\frac{\partial L}{\partial \text{PReLU}_\alpha(a)}$ is the gradient propagated by the successive level of the network. The additional parameter appears to be a negligible addition to the total number of network parameters if a α-sharing policy is used. Empirical experiments show that the magnitude of α rarely is larger than 1. although no constraints on the range are applied. However, the resulting function remains basically a modified version of the ReLU function that can change its shape just in the negative part.

Parametric ELU

[Trottier et al., 2017] tries to eliminate the need to manually set the α parameter of ELU unit by proposing an alternative version based on two parameters that can be learned, i.e.

$$\text{PELU}_{\beta,\gamma}(a) = \begin{cases} \frac{\beta}{\gamma}a & \text{if } a \geq 0 \\ \beta \cdot (\exp(\frac{a}{\gamma}) - 1) & \text{otherwise.} \end{cases}$$

where $\beta, \gamma > 0$ are parameters that control the function shape learning together with the others network parameters using any optimization gradient-based method. The partial derivatives of PELU function respect to β and γ result to be

$$\frac{\partial \text{PELU(a)}}{\partial \beta} = \begin{cases} \frac{a}{\gamma} & \text{if } a \geq 0 \\ (e^{\frac{a}{\gamma}} - 1) & \text{otherwise} \end{cases}, \frac{\partial \text{PELU(a)}}{\partial \gamma} = \begin{cases} (-\frac{\beta a}{\gamma^2}) & \text{if } a \geq 0 \\ (-\frac{\beta}{\gamma^2})e^{\frac{a}{\gamma}} & \text{otherwise} \end{cases}$$

4.3.3 Interpolation-based Activation Functions

In this section we report a set of activation functions whose shape is computed using classical interpolation techniques. These techniques may need some additional input, depending on the technique used (for example a set of sampled points from a start function).

Spline Activation Function

In [Guarnieri, 1995] was introduced the use of spline based activation functions whose shape can be learned by data using a set of Q representative points. This method has therefore been improved by [Vecci et al., 1998, Scardapane et al., 2016]. More in detail, this technique try to find a cubic spline to model the activation function initializing the control points from a sigmoid (as in [Guarnieri, 1995]) or from another function (e.g. hyperbolic tangent,

as in [Scardapane et al., 2016]) assuring universal approximation capability. the resulting function is given by:

$$\text{SAF}(a) = \vec{u}^T B \vec{q}_{i:i+P}$$

where:

- i is the index of the closest knot;
- $\vec{q}$ is the knots vector, with $\vec{q}_{i:i+P} := (q_i, q_{i+1}, \ldots, q_{i+P})^T$, so the output is computed by spline interpolation over the closest knot and its P right-most neighbors. supposing that the knots are uniformly spaced, i.e. $\vec{q}_{i+1} = \vec{q}_i + \Delta t$, for a fixed $\Delta t \in \mathbb{R}$, the normalized abscissa value can be computed as $u = \frac{a}{\Delta t} - \lfloor \frac{a}{\Delta t} \rfloor$;
- $\vec{u}^T = (u^P, u^{P-1}, \ldots, u, 1) \in \mathbb{R}^{P+1}$ is the reference vector;
- $B \in \mathbb{R}^{(P+1)\times(P+1)}$ is the basis spline matrix. Different bases makes different interpolation schemes; in [Vecci et al., 1998] is used the Catmull-Rom matrix [Smith, 1983].

The $\vec{q}$ values are then adapted during the learning, adding a regularization term on $\vec{q}$ to prevent the over-fitting. The regularization term results to be a very critic issue: while the authors of [Vecci et al., 1998] act on the Δx value, in [Scardapane et al., 2016] is proposed to penalize changes in $\vec{q}$ respect to a "good" set of values, as for example the initial control points values, sampled from a standard NN activation function.

Look-up Table Units

Based on a similar principle, [Wang et al., 2018] introduce Look-up Table Unit based on spline; in this work, the activation function is controlled by a look-up table containing a set of anchor-points that determine the function shape. The look-up table idea is not new in learnable activation function field; a first work was made in [Piazza et al., 1993], where a generic adaptive look-up table was addressed by the neuron activation and learned by data. The main difference with [Wang et al., 2018] is the look-up table structure which returns the result of a spline interpolation instead of the raw number in the table. More in detail, defining the set of anchor point as $A = \{(q_1, u_1), (q_2, u_2), \ldots, (q_m, q_m)\}$ with $q_i = q_1 + i \cdot \Delta t$ and u_i the learnable parameters, [Wang et al., 2018] proposes two different methods to generate the activation function; the first one by interpolation, i.e.:

$$\text{LuTU}(a) = \frac{1}{t} u_i (q_{i+1} - a) + u_{i+1}(a - q_i), \text{ if } q_i \leq a \leq q_{i+1}$$

and the second one using cosine smoothing, i.e.:

$$\text{LuTU}(a) = \sum_{i}^{m} u_i \cdot r(a - q_i, \alpha t)$$

where $\alpha \in \mathbb{N}$ and

$$r(w, \tau) = \begin{cases} \frac{1}{2\tau}(1 + \cos(\frac{\pi}{\tau} w)) & \text{if } -\tau \leq w \leq \tau \\ 0 & \text{otherwise.} \end{cases}$$

the method based on cosine smoothing was proposed to address the gradient instability suffered by the interpolation method. This kind of approaches require to set additional hyper-parameters like the function input domain, the number of anchor point and the space between, the α value in [Wang et al., 2018] second approach or the spline type for [Vecci et al., 1998] and [Scardapane et al., 2016] approach so, beyond a strong mathematical formulation, these approaches seem to require to set up many hyper-parameters manually or through a model selection process.

4.3.4 Activation functions based on ensemble methods

With ensemble methods we refer to some technique to merge together different functions. Basically, each of these techniques provides:

- a set of basis functions, which can contains fixed-shape functions or trainable functions or together;
- a merge model, that is the method by which the basis functions are combined together.

obviously, based on the choices made in the two cases, the learning algorithm may need to be modified.

Harmon & Klabjan Activation Ensembles

Some studies try to define activation functions using different available activation functions rather than creating an entirely new function. For example, authors in [Harmon and Klabjan, 2017] allow the network to chose the best activation function from a predefined set $F = \{f^{(1)}, f^{(2)}, \ldots, f^{(k)}\}$ or some combination of those. Differently from Maxout, the activation functions are combined together instead of simply taking the maximum value.

The activation function proposed by [Harmon and Klabjan, 2017] works on single mini-batch, i.e. its input is tuple $\vec{a}^{(u)} = (a_1^{(u)}, a_2^{(u)}, \ldots, a_B^{(u)})$ which every element $a_i^{(u)},\ 1 \leq i \leq B$ is the activation of the unit u on the i-th element of mini-batch. The proposed activation function is based on a sum of normalized functions weighted by a set of of learned weights; the resulting activation function $\Phi(a)$ for an input activation $\vec{a}_b^{(u)}$ has the form:

$$\Phi^{(u)}(a_b^{(u)}) = \sum_{j=1}^{k} \alpha_u^j \big(\eta^{(j)} g^j(a_b^{(u)} + \delta^{(j)})\big)$$

where α_u^j a weight value for the u-th unit and the j-th function, $\eta^{(j)}$ and $\delta^{(j)}$ are inserted to allow the network choosing to leave the activation at its original state if the performance of one are particularly good and

$$g^j(a_b^{(u)}) = \frac{f^j(a_b^{(u)}) - \min\limits_i f^j(a_i^{(u)})}{\max\limits_i f^j(a_i^{(u)}) - \min\limits_i f^j(a_i^{(u)}) + \epsilon}$$

where ϵ is a small number. Authors emphasize that, during the experiments, many neurons favored the ReLU function since respective α value was greater in magnitude respect to the others. The learning of α value was done formalizing the problem as a optimization problem with the additional constraint that α values must be non-zero and sum to one to limit the magnitude. So, the approach proposed by [Harmon and Klabjan, 2017] seems to require the additional computational cost to solve the defined optimization problem together with a standard network learning procedure.

Adaptive Piecewise Linear Units

In [Agostinelli et al., 2014] the activation functions are modeled as a sum of hinge-shaped functions that results in a different piece-wise linear activation function for every unit u:

$$\text{APL}_u(a) = \max(0, a) + \sum_{i=1}^{k} w_{u,k} \max(0, -a + b_{u,k})$$

with k hyper-parameter and $w_{u,k}, b_{u,k}$ variables learned during the network training. The total overhead in terms of number of parameter to learn respect to a classic NN with n units is $2 \cdot k \cdot n$, so the number of parameters increases with the number of hidden units and, for a large enough input, the learned function tends to have a ReLU-like behavior, reducing the expressiveness of the learned activation functions. In the experiments reported in [Agostinelli et al., 2014] the value of k was determined using a validation while the w and b parameters were regularized with an L_2 penalty, so that the optimizer can not choose very large values for the parameters that would lead to numerical instability.

S-Shaped ReLU

Taking inspiration by Webner-Fechner law [Fechner, 1966] and Stevens law [Stevens, 1957], the authors of [Jin et al., 2016] designed an activation S-shape function composed by three linear function:

$$\text{SReLU}(a) = \begin{cases} b_1 + w_1(a - b_1) & \text{if } a \leq b_1 \\ a & \text{if } b_1 < a < b_2 \\ b_2 + w_2(a - b_2) & \text{if } a \geq b_2 \end{cases}$$

with b_1, w_1, b_2, w_2 parameters that can be learn together to the others. The partial derivative of the SReLU function respect the introduced parameters are:

$$\frac{\partial \text{SReLU(a)}}{\partial b_1} = \begin{cases} 1 - w_1 & \text{if } a \leq b_1 \\ 0 & \text{otherwise} \end{cases}, \frac{\partial \text{SReLU(a)}}{\partial w_1} = \begin{cases} a - b_1 & \text{if } a \leq b_1 \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial \text{SReLU(a)}}{\partial b_2} = \begin{cases} 1 - w_2 & \text{if } a \geq b_2 \\ 0 & \text{otherwise} \end{cases}, \frac{\partial \text{SReLU(a)}}{\partial w_2} = \begin{cases} w_2 - b_2 & \text{if } a \geq b_2 \\ 0 & \text{otherwise} \end{cases}$$

Also in this case, the weight decay it not usable because it tends to pull the parameters to zero. SReLU can learn both convex and non-convex functions, differently from others as Maxout that can learn just convex function. Furthermore, ReLU can approximate also ReLU when $b_2 \geq 0, w_2 = 1, b_1 = 0, w_1 = 0$ or LReLU/PReLU when $b_2 \geq 0, w_2 = 1, b_1 = 0, w_1 > 0$.

Adaptive Activation functions

In [Qian et al., 2018] are presented different ways to obtain mixture of eLU and ReLU, in a probabilistic and hierarchical context, able to obtain an activation function learned by data. They propose the following activation function:

- Mixed activation: $\Phi_M(a) = p \cdot \text{LReLU}(a) + (1 - p) \cdot \text{ELU}(a)$ with $p \in [0, 1]$ learned from data;
- Gated activation: $\Phi_G(a) = \text{sig}(\beta a) \cdot \text{LReLU}(a) + (1 - \text{sig}(\beta a)) \cdot \text{ELU}(a)$ with $\text{sig}(\cdot)$ the sigmoid function and β learned from data;
- Hierarchical activation: this function is composed by a three-levels little neural network, where each input unit u is connected to n units, every pair of nodes are combined together in similar manner as in gated activation (with the substitution of ELU function with PELU and ReLU with PReLU) and the last layer takes the maximum

of middle level unit; so, the different function output can be formalized as:

$$\begin{cases} \phi_l^{(1)}(a) = \text{PReLU}(a) \text{ and } \phi_r^{(1)}(a) = \text{PELU}(a) & \text{for the first level} \\ \phi_i^{(2)}(a) = \text{sig}(\beta a) \cdot \phi_l^{(1)}(a) + (1 - \text{sig}(\beta a)) \cdot \phi_r^{(1)}(a) & \text{for the } i\text{-th unit of the second level} \\ \phi^{(3)}(a) = \max_i \phi_i^{(2)}(a) & \text{for the third level} \end{cases}$$

and the final activation function is $\Phi_H(a) = \phi_l^{(3)}(a)$.

Swish

In [Ramachandran et al., 2017] a search technique for activation function is proposed. Given a set of basis activation function, a set of new activation function is constructed by repeatedly composing them. For each candidate activation function, a network with which uses generated function is trained on some task to evaluate the performance. Between all the tested functions, the best one results to be:

$$\text{Swish}(a) = a \cdot \text{sig}(\alpha \cdot a)$$

with α trainable parameter. To note that for $\alpha \to +\infty$, Swish becomes like ReLU, while, if $\alpha = 1$, the becomes equal to SiLU [Elfwing et al., 2018]. The derivative can be computed as:

$$\frac{\partial \text{Swish}(a)}{\partial a} = \alpha \cdot a \cdot \text{sig}(a) + \text{sig}(\alpha \cdot a)(1 - \alpha \cdot a \cdot \text{sig}(\alpha \cdot a))$$

4.3.5 Other studies

In [Piazza et al., 1992] was proposed an attempt to model activation functions using a polynomial activation function with adaptable coefficients; for a given degree $k \in \mathbb{N}$, the relative polynomial function is

$$\text{AP}(a) = \sum_{i=0}^{k} \alpha_i \cdot a^i$$

with $(\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_k)$ parameters to learn. These function parameters can be learned together with net parameters using gradient descent with back-propagation. The gradient respect to the new parameters can be computed trivially as $\frac{\partial \text{AP}(a)}{\partial a} = \sum_{i=1}^{k} i \cdot \alpha_i \cdot a^{i-1}$. However, it must be taken into account that networks with polynomial activation function are not universal approximator, as shown in [Stinchcombe and White, 1990].

[Eisenach et al., 2016] tries to approximate an activation function using a Fourier expansions; this study uses a 2-stage Stochastic Gradient Descend (SGD) algorithm to learn

the parameters of the activation functions and of the network. [Urban et al., 2017] tries to learn the activation function using Gaussian processes. [Goh and Mandic, 2003] tries to learn the amplitude of the activation functions but in Recurrent Neural Networks [Cardot and Romuald, 2011]. [Ertuğrul, 2018] and [Li et al., 2013] proposed two learnable activation functions using linear regression and subnet-based approach respectively; however, these studies are done on network models different from the classic feed-forward neural networks (see for example [Chen et al., 2018, Huang, 2015]).

4.4 Conclusions

In this chapter, we have discussed about the most common activation functions present in the literature, proposing a possible taxonomy to classify them. We have divided the possible functions into two main categories: fixed shape and learnable shape, focusing especially on the latter as the main topic of the work done. However, it must be kept in mind that the activation functions are not the only entities that determine the performances of a neural network. Other parameters, as the number of neurons together with the way they are arranged, or the weights initialization protocol can be decisive for the performance of the network, together with the parameters to be set for the learning algorithm. Moreover, even if the experiments are conducted with the same datasets, these may have been pre-processed in different ways (e.g., ZCA [Bell and Sejnowski, 1997] or data-augmentation [Perez and Wang, 2017]) which can in turn condition the results. So, it can be really difficult to make a comparison between the activation functions, since the experiments made in the literature are often conducted on very different setup. Table 4.2 reports the accuracies obtained by some works present in the literature. As an indication, we report just the best values obtained, without taking care of the different architectures used.

				Accuracy %	
		SVHN	**MNIST**	**CIFAR10**	**CIFAR100**
Fixed shape	*sigmoid*		97.9 [Pedamonti, 2018];		
	tanh		98.21 [Eisenach et al., 2016]		
	softplus			94.9 [Ramachandran et al., 2017]	83.7 [Ramachandran et al., 2017]
	ReLU		98.0 [Pedamonti, 2018]; 99.53 [Jin et al., 2016]; 99.16 [Eisenach et al., 2016];	87.55 [Xu et al., 2015]; 92.27 [Jin et al., 2016]; 94.59 [Trottier et al., 2017]; 91.51 [Qian et al., 2018]; 84.8 [Eisenach et al., 2016]; 95.3 [Ramachandran et al., 2017]	57.1 [Xu et al., 2015]; 67.25 [Jin et al., 2016]; 75.45 [Trottier et al., 2017]; 64.42 [Qian et al., 2018] 83.7 [Ramachandran et al., 2017]
	LReLU		98.2 [Pedamonti, 2018]; 99.58 [Jin et al., 2016];	88.8 [Xu et al., 2015]; 92.3 [Jin et al., 2016]; 92.32 [Qian et al., 2018]; 95.6 [Ramachandran et al., 2017]	59.58 [Xu et al., 2015]; 67.3 [Jin et al., 2016]; 64.72 [Qian et al., 2018]; 83.3 [Ramachandran et al., 2017]
	RReLU			88.81 [Xu et al., 2015]	59.75 [Xu et al., 2015]
	ELU		98.3 [Pedamonti, 2018];	93.45 [Clevert et al., 2015]; 94.01 [Trottier et al., 2017]; 92.16 [Qian et al., 2018]; 94.4 [Ramachandran et al., 2017]	75.72 [Clevert et al., 2015]; 74.92 [Trottier et al., 2017]; 64.06 [Qian et al., 2018]; 80.6 [Ramachandran et al., 2017]
	AGSig (Swish-1)			95.5 [Ramachandran et al., 2017]	83.8 [Ramachandran et al., 2017]
Quasi fixed	PReLU		99.59 [Jin et al., 2016]	88.21 [Xu et al., 2015]; 92.32 [Jin et al., 2016]; 94.64 [Trottier et al., 2017]; 95.1 [Ramachandran et al., 2017]	58.37 [Xu et al., 2015]; 67.33 [Jin et al., 2016]; 74.5 [Trottier et al., 2017]; 81.5 [Ramachandran et al., 2017]
	PELU			94.64 [Trottier et al., 2017]	75.45 citetrottier2017
Subnets based	*Maxout*	97.53 [Goodfellow et al., 2013]	99.55 [Goodfellow et al., 2013]	90.62 [Goodfellow et al., 2013];	61.43 [Goodfellow et al., 2013]
	Probout	97.61 [Springenberg and Riedmiller, 2013]		88.65 [Springenberg and Riedmiller, 2013]	61.86 [Springenberg and Riedmiller, 2013]
	MMN			92,34 [Sun et al., 2018]	66.76 [Sun et al., 2018]
	NIN	97.65 [Lin et al., 2013]	99.53 [Lin et al., 2013]	91.19 [Lin et al., 2013]	64.32 [Lin et al., 2013]
	CIC			91.54 [Pang et al., 2016]	68.6 [Pang et al., 2016]
	MIN	98,19 [Chang and Chen, 2015]	99.76 [Chang and Chen, 2015]	92.15 [Chang and Chen, 2015]	71.14 [Chang and Chen, 2015]
Interpolated	LuTU			94.22 [Wang et al., 2018]	
Ensembled	Gated Act.			92.65 [Qian et al., 2018]	65.75 [Qian et al., 2018]
	Mixed Act.			92.6 [Qian et al., 2018]	65.44 [Qian et al., 2018]
	hierarc. Act.			92,99 [Qian et al., 2018]	66.23 [Qian et al., 2018]
	Harmon Klabjan		99.40 [Harmon and Klabjan, 2017]		74.20 [Harmon and Klabjan, 2017]
	SReLU		99.65 [Jin et al., 2016]	93.02 [Jin et al., 2016]	70.09 [Jin et al., 2016]
	APL			92.49 [Agostinelli et al., 2014]	69.17 [Agostinelli et al., 2014]
	NPF		99.31 [Eisenach et al., 2016]	86.44 [Eisenach et al., 2016]	
	Swish			95.5 [Ramachandran et al., 2017]	83.9 [Ramachandran et al., 2017]

Table 4.2: Accuracy on some datasets of the most common activation functions in literature; here we report the best values without taking into account the differences between the architectures used.

Chapter 5

Variable Activation Functions based on a simple and efficient Neural Network architecture

Introduction

Trained activation functions is a active topic in neural network research. Despite a lot of works and promising results, currently is still difficult to find a simple and handful way to learn an activation function without introducing new parameters or different learning techniques. The actual State-of-Art in neural networks activation functions has been discussed in chapter 4. The introduction of new activation functions has contributed to renew the interest of scientific community for neural networks; the use of ReLU and similar fixed-shape functions, such as Leaky ReLU and parametric ReLU, as activation function in neural networks has been shown to improve significantly the network performances thanks to properties as no saturation and sparse coding. Individuating alternative functions that can potentially still improve the results is an open field of research. From this point of view, a number of recent papers compare neural architectures with different activation functions as in [Liu and Yao, 1996, Yao, 1999, Pedamonti, 2018]. The idea of the learnable activation function is instead to search the "best" activation functions using knowledge given by training data. In this work, we propose a simple way to obtain a set of trained activations function simply using local sub-nets with few neurons. Using the taxonomy that we defined in chapter 4, our approach can be considered belonging to the sub-networks based functions family. In particular, we built upon the possibility to obtain adaptable activation functions in terms of sub-networks with just one hidden layer. In a nutshell, each neuron with a non-linear activation function g can be *substituted* with a neuron with

an *Identity* activation function sending its output to one-hidden layer sub-network with just one output neuron.

5.1 Proposed model: Variable Activation Function Sub-network

In general, in a MLFF network the output of a neuron i belonging to the l-th layer is obtained by a two-step computation (see [Bishop, 2006], Section 4): the first step computes the input of a neuron i as $a_i^l = \vec{w}_i^l \vec{x}^{l-1} + b_i^l$ where $\vec{w}_i^l$ are the weight connections between the neuron i and the previous layer, $\vec{x}^{l-1}$ are the output of the neurons belonging to the previous layer (or network input values), $\vec{w}^l$ are the connection weights going from the neurons of the previous layer $l-1$ to the neuron i, and b_i^l is the bias associated to the neuron i . The output of the neuron i is then computed in a second step transforming the input a_i^l using a fixed activation function f, obtaining $z_i^l = f(a_i^l)$.

The key idea of our approach is to modify the second step of the computation by transforming the input a_i^l using a small one-hidden layer sub-network with m hidden neurons and just one input and one output neuron. Let us call it *Variable Activation Function* (VAF) sub-network. So, a VAF for a neuron i can be described as a network composed by:

- an hidden layer, composed by $m > 1$ neurons directly connected to the neuron i by a set of $\alpha_h,\ \forall 1 \leq h \leq m$ weights;
- a set of fixed basis functions $f_1, f_2, \ldots, f_m$;
- an output layer composed by a single neuron connected to all the neurons of the hidden layers by a set of $\beta_h,\ \forall 1 \leq h \leq m$ weights.

computation of VAF sub-network can be described as follows: VAF is fed with an input a, the m neurons of the hidden layer compute m outputs as: $y_h = f(\alpha_h a + \alpha_{0h})$ with $h = 1, 2, \cdots, m$, the output node computes $z = \sum_h \beta_h y_h + \beta_0$. α_h and α_{0h} are weights and biases of the hidden layer of the subnetwork, respectively. β_h and β_0 are weights and bias of the output layer of VAF sub-network, respectively. In this way the output of the neuron i can be expressed as :

$$\text{VAF}(a) = \vec{\beta}^T \phi\big(a \cdot \vec{\alpha} + \vec{\alpha_0}\big) + \beta_0$$

with $\vec{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m)^T$, $\vec{\alpha_0} = (\alpha_{01}, \alpha_{02}, \ldots, \alpha_{0m})^T$, $\vec{\beta} = (\beta_1, \beta_2, \ldots, \beta_m)^T$, and $\phi(\vec{x}) = \big(f_1(x_1), f_2(x_2), \ldots, f_k(x_k)\big)^T$. $\vec{\alpha}, \vec{\alpha_0}, \vec{\beta}, \beta_0$ are parameters that can be learned by data during

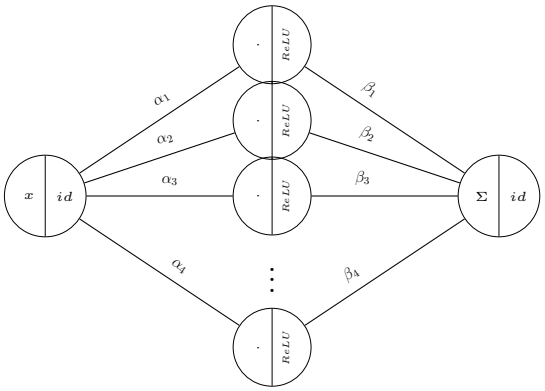


Figure 5.1: A VAF unit.

the training process.
A general schema of a VAF unit is shown in figure 5.1. This schema enables one to approximate arbitrarily well any activation function provided that:

- the number m of VAF's hidden neurons is sufficiently large;
- all the activation functions f_i of VAF's hidden layer are not-polynomial functions.

The first condition is given by [Hornik et al., 1989, Hornik, 1991] where it was shown that a multilayer feed-forward networks can approximate any continuous function provided that sufficiently many hidden units are available and that the activation functions are continuous, bounded and non-constant. This result was generalized in [Leshno et al., 1993], where it is shown that a standard multilayer feed-forward network can approximate any continuous function to any degree of accuracy if and only if the network's activation function is not polynomial. With these conditions, a VAF activation function can substitute any other network activation function without loss in generality and having as only overhead an increase in the number of networks parameters equal to $N \cdot (3k+1)$ with N total number of the hidden neurons of the network. The number of required parameters can drop to $L \cdot (3k+1)$ with L number of hidden layers if we adopt the shared weights principle, i.e. the functions on the same layer share the same VAF weights. With this design choice, we reduce the number of parameters by making the reasonable assumption that if one function is good for a neuron, then it should also be good for the other neurons of the same layer. This assumption can also be motivated, instead of under the profile of the sub-networks weights, in terms of activation function of a classic neural networks used in the neural network literature, where activation functions belonging to same layer exhibit

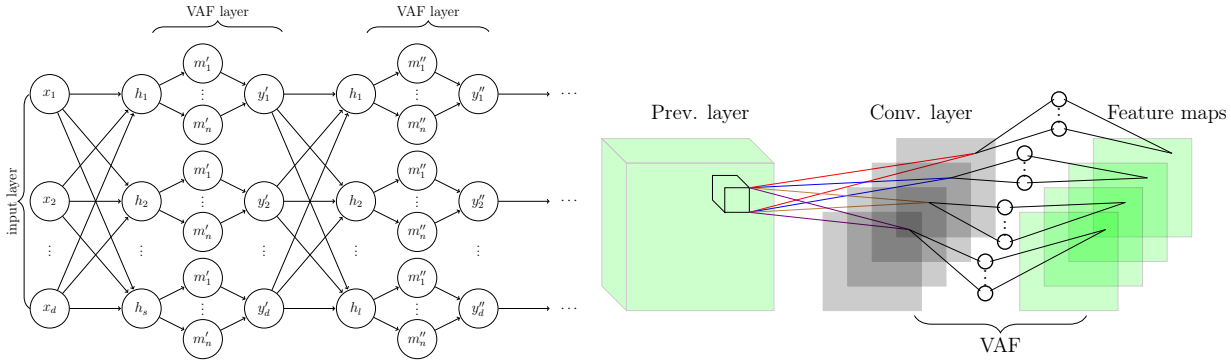


Figure 5.2: VAF units in a Full-Connected network (left) and in a Convolutional Neural Network (right).

the same activation function. For the same reason we can assume that, for any level, all the basis function $f_i(\cdot)$ are equals to each other. Summing up, for every network level the only hyper-parameters to set are:

- the number of hidden nodes m;
- basis function $f(\cdot)$.

It is also to be highlighted that in our approach we have a neural sub-network architecture which is still a MLFF network without adding external structures, so the only new hyper-parameter is the number of VAF hidden units m. The training procedure can be can be left unaltered (e.g. Stochastic Gradient Descent) since the new unit, being a sparse sub-net with weights shared similarly to a convolutional network, fits perfectly into any network architecture. Figure 5.2 shows how a VAF network can be integrated into a common multilayer full-connected neural network (on the left) and in a convolutional neural network (on the right).

5.2 VAF learning

As above discussed, our neural architecture with VAF is a MLFF network, consequently it can be trained using any learning algorithm dedicated to MLFF network (see section 3.3). However, in case of considering the same VAF acting uniformly for all neurons of a layer, then there is the constrain that the weights of VAF networks should be considered *shared weights*. From an implementing point of view this corresponds to consider VAF network as a function convolving with the a_i^l values [Lin et al., 2013]. The weight values of the VAF,

being few and connected to each unit, they influence the behavior of the entire network, therefore their behavior must be particularly taken into consideration during the training phase, in particular, the initial value of the VAF weights can be decisive. Training of neural networks usually starts initializing the weights and biases in a random way [Bishop, 2006] or using any initialization rule as for example [Glorot and Bengio, 2010]. Although this approach can also be followed in this case, one can choose different solutions for the VAF weights. In particular, a possible alternative is to select the initial VAF's weights so that at the start of learning process the VAF networks approximate a pre-fixed function such as, for example, a classic activation function as ReLU or Sigmoid or the f basis function associated to the other hidden layers of the network. In this way hypothetically the function would start from a notoriously already valid form in which the training process should only modify it just enough to improve the performance of the network based on the training data. However, it should be borne in mind that this choice would risk negatively affecting the solution generated by the learning process, given that the resulting VAF could be too similar to the initial function.

5.3 Comparison with existing models

According to our knowledge, the approach currently present in the literature that is closest to that proposed by us is Network in Network (see section 4.3.1 and [Lin et al., 2013]), with some substantial differences:

- hypothetically, the number of nodes in the last layer of a NIN unit could be greater than one, thus losing the one-to-one correspondence between the neuron input and the output. The lack of this constraint causes the loss of "function" approximator for the sub-network, which is instead preserved in our approach by constraining the VAF last layer to be composed by a single node.

- NIN sub-network provides the ability to have multiple hidden levels, resulting in a new hyper-parameter (the number of levels) to be set and a greater number of weights to learn. We think that this solution can be excessive, since the universality theorem [Hornik et al., 1989, Hornik, 1991], together with the work carried out by [Leshno et al., 1993], guarantees us that a single level is sufficient to approximate any function, especially if this function has a form not too complex as expected.

- the work proposed by [Lin et al., 2013] is tested only on convolutional network, we instead consider also classic full-connected feed-forward networks.

5.4 Experiments and results

To empirically evaluate our approach we performed two different series of experiments. In the first one, we consider standard MLFF networks, and in the second one convolutional MLFF networks.

5.4.1 Model validation

A feed-forward neural network usually has many hyper-parameters to be set, some closely related to the design of the network, as the number of neurons, the number of levels, the activation function to be used, and others related to learning process as the learning rate and momentum, weight decay. A set of acceptable parameters can be found using any model validation procedure (for example cross validation 2.2); while this method could be done with network with a low parameters number, it can be unfeasible with networks with a great number of parameters since the validation procedure is exponential on the number of hyper-parameters to be found. So, instead of using this kind of model selection procedure for all the model's hyper-parameters, is often preferred to fix some of these to fixed values that other studies have shown to be valid even if on unequal but similar problems. In our case, in CNN experiments, we use a network architecture that is very similar to [Lin et al., 2013] and learning hyper-parameters setup used in [He and Sun, 2015]. To evaluate our approach we use K-Fold Cross-Validation (KFCV) procedure (see section 2.2 and algorithm 1). The optimal K value is between 5 and 10, because the statistical performance does not increase a lot for larger values of K [Hastie et al., 2009]. Thus, in our experiments we use K-FCV procedure with $K = 10$ to evaluate different neural network models and learning approaches, unless it is not specified differently.

5.4.2 Selecting VAF hidden units

As above discussed, an important hyper-parameter to setup is the number k of hidden nodes of the VAF units; instead of increasing the computational load of the cross validation procedures adding another hyper-parameter, we decide to find an acceptable k value making a separate 5-CV run. We evaluate the accuracy given by a single-layer CNN with a different number of filters and different values of k. We choose a subset of 7000 images

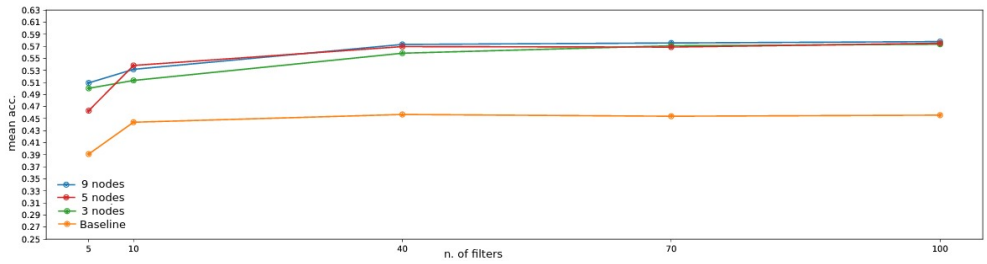


Figure 5.3: The mean accuracy of single-layer CNN networks equipped with VAF units on MNIST dataset with 5-fold CV. On the x axis we have the number of filter of the CNN; each plot is a different value of k, that is the number of hidden nodes for the VAF units.

Name	Istances	Input Dim.	N. classes	Task	Neural Network Arch.	Ref.
Liver	345	7	2	Classif.	MLFF	[Dheeru and Karra Taniskidou, 2017]
Wine	178	13	3	Classif.	MLFF	[Dheeru and Karra Taniskidou, 2017]
Statlog Image Segmentation	2310	19	7	Classif.	MLFF	[Dheeru and Karra Taniskidou, 2017]
Statlog Landsat Satellite	6435	36	7	Classif.	MLFF	[Dheeru and Karra Taniskidou, 2017]
Cardiotocography	2126	22	3	Classif.	MLFF	[Dheeru and Karra Taniskidou, 2017]
Seismic bumps	2584	18	2	Classif.	MLFF	[Sikora and Wróbel, 2010]
Dermatology	336	35	3	Classif.	MLFF	[Dheeru and Karra Taniskidou, 2017]
Diabetic retinopathy debrecen	1151	19	2	Classif.	MLFF	[Antal and Hajdu, 2014]
QSAR biodegradation	1055	41	2	Classif.	MLFF	[Mansouri et al., 2013]
Climate model simulation	540	18	2	Classif.	MLFF	[Lucas et al., 2013]
MNIST	70000	28×28	10	Classif.	CNN	[LeCun and Cortes, 2010]
Fashion MNIST	70000	28×28	10	Classif.	CNN	[Xiao et al., 2017]
Cifar10	60000	$32 \times 32 \times 3$	10	Classif.	CNN	[Krizhevsky and Hinton, 2009]

Table 5.1: Properties of used datasets and neural network architectures applied with.

taken from the MNIST dataset [LeCun and Cortes, 2010]. Results are showed in figure 5.3. The baseline is a CNN network equipped classic ReLU activation function. We can notice two interesting results: the first one is that using VAF unit in a simple single-layer CNN seems to significantly improve the performance, the second one is that increasing the number of VAF hidden nodes beyond a certain limit does not bring significant benefits; the best compromise for the k value seems to be between 3 and 5, since taking a higher number increases the complexity of the model without bringing significant improvements.

5.4.3 First experimental scenario: Full Connected Feed-Forward Neural Networks

In this experimental scenario we focus on evaluating the impact of both VAF sub-networks as adaptable activation functions and VAF initialization, using standard MLFF networks. In particular, we consider standard MLFF networks with 1 or 2 hidden layers with ReLU

Model number	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
Standard	net_{10}	net_{25}	net_{50}	net_{100}	$net_{25,10}$	$net_{50,10}$	$net_{100,10}$	$net_{50,25}$	$net_{100,25}$	$net_{100,50}$
VAF	$vnet^3_{10}$	$vnet^3_{25}$	$vnet^3_{50}$	$vnet^3_{100}$	$vnet^3_{25,10}$	$vnet^3_{50,10}$	$vnet^3_{100,10}$	$vnet^3_{50,25}$	$vnet^3_{100,25}$	$vnet^3_{100,50}$

Table 5.2: Neural network architectures used in the first experimental scenario. See text for further details.

activation function. To validate our network, we use 10 public datasets, 5 for binary classification and 5 for multiclass classification, with different number of points and features. They are all reported in the first part of table 5.1.

The number of hidden neurons of the network changes in the set $\{10, 25, 50, 100\}$, but for neural networks with 2 hidden layers we only selected models with a number of hidden neurons belonging to the first layer greater than the number of hidden neurons of the second layer. ReLU was selected as basis function f for the hidden nodes of VAF sub-networks. Thus, for each dataset we made 4 network models with 1-hidden layer and 6 models with 2-hidden layers. Let us call net_{m_1} and net_{m_1,m_2} the 1-hidden and 2-hidden layer networks, respectively, with $m_1, m_2 \in \{10, 25, 50, 100\}$. On the basis of what was discussed in Section 5.1, to each network net_{m_1} (net_{m_1,m_2}) it is possible to associate a neural network $vnet^m_{m_1}$ ($vnet^m_{m_1,m_2}$) equipped with VAF sub-networks, m is the number of hidden nodes of VAF sub-networks. In Table 5.2 we report the neural network architectures used in this series of experiments. Neural network architectures were sorted in ascendant way according to their complexity. Networks were trained according to an usual learning approach as described in chapter 3. In particular, we used a batch approach, RProp [Riedmiller and Braun, 1992], with "small" datasets, i.e, when the number of examples was less than $5 \cdot 10^3$, otherwise we used a mini-batch approach, RMSProp [Tieleman and Hinton, 2012]. Moreover, networks with VAF sub-networks were trained using two different strategies:

- a random initialization;
- an initialization such that they approximate the basis function (in our case, ReLU).

All the network models, i.e., net_{m_1},net_{m_1,m_2},$vnet^m_{m_1}$ and $vnet^m_{m_1,m_2}$, were compared in a 10-fold cross validation schema (see Algorithm 1). Note that Learning Rate (LR) in RMSProp has been varied in the range $[0.0001, 0.1]$ considering 10 equispaced values, while in RProp η^+ was selected equal to 1.01 and η^- equal to 0.5. In Table 5.3 are summarized the parameters of this series of empirical evaluations.

m_1, m_2	# VAF hidd. units	VAF init.	Learning approaches	# max. epochs	# folds
{10, 25, 50, 100}	3	{Random, ReLU}	{RMSProp, RProp}	300	10

Table 5.3: Parameters of the first experimental scenario. See text for further details.

# filters for layer	# VAF hidd. units	VAF init.	Learning approaches	# max. epochs	# folds
192	5	{Random, ReLU}	SGD	{100,300}	10

Table 5.4: Parameters of the second experimental scenario. See text for further details.

5.4.4 Second experimental scenario: Convolutional Neural Networks

To experimentally evaluate the impact of VAF on Convolutional Neural Networks (CNN), we consider standard CNN networks with 2 and 3 convolutional layers and 3 different dataset: MNIST, Fashion MNIST and Cifar10 (see Table 5.1 for further details); let us call $cnet_2$ and $cnet_3$ respectively the 2-convolutional layer and 3-convolutional layer networks with classic activation function (in our case, ReLU); as stated in section 5.1, to each $cnet$, it is possible to associate a neural network $vcnet^m$ equipped with VAF sub-networks equipped with m hidden units. Due to the greater complexity of the data compared to those used in the first scenario, for MNIST and Fashion-MNIST dataset we use 2-layer CNN networks and for Cifar10 3-layer CNN networks. All the experiments were done using a 10-Fold Cross Validation schema as described in 1. As also discussed in Section 5.2 and 5.4.3, a key aspect is how the VAF network is initialized, thus also in this case we choose to initialize the weights of the VAF sub-networks either randomly or to approximate a ReLU function. Networks were trained using Stochastic Gradient Descent (SGD) method with mini-batching. Properties of the used CNN architectures and learning process are summarized in table 5.4.

5.5 Results

In this section, we show the results obtained in the experiments carried out; we show the performance difference between the different approaches and some activation functions that have been obtained. Performances are measured in terms of averaged accuracy and standard deviation, the best results are reported in bold style.

5.5.1 First experimental scenario: Full connected feed-forward NN

Performance results are showed in Table 5.5; in general, Neural networks with VAF outperforms neural networks without VAF except in two cases where neural networks without VAF outperforms neural networks with VAF. Anyway, standard deviations remain comparable or lower than those without VAF. Using one or the other of the two initialization schemes does not seem to give. in general, significant differences.

	standard Relu	VAF Init random	VAF Init ReLU
	Accuracy: mean + St.Dev	Accuracy: mean + St.Dev	Accuracy: mean + St.Dev
Liver	0.6203 ± 0.0474 ($net_{25,10}$)	0.6290 ± 0.0378 ($vnet^3_{100,10}$)	$\mathbf{0.6348 \pm 0.0375}$($vnet^3_{25}$)
Wine	0.8879 ± 0.0516 (net_{10})	$\mathbf{0.9552 \pm 0.0371}$ ($vnet^3_{50}$)	0.9162 ± 0.0434 ($vnet^3_{10}$)
Image segmentation	$\mathbf{0.9463 \pm 0.0128}$ (net_{25})	0.9351 ± 0.0179 ($vnet^3_{50}$)	0.9381 ± 0.0079 ($vnet^3_{50}$)
Satellite image	0.8821 ± 0.0101 ($net_{100,50}$)	0.8856 ± 0.0028 ($vnet^3_{100}$)	$\mathbf{0.8875 \pm 0.0080}$ ($vnet^3_{100,25}$)
CTG	0.8979 ± 0.0263 (net_{100})	$\mathbf{0.9040 \pm 0.0073}$ ($vnet^3_{100}$)	0.8984 ± 0.0261 ($vnet^3_{50,25}$)
Seismic bumps	$\mathbf{0.9346 \pm 0.0009}$ (net_{10})	0.9234 ± 0.0074 ($vnet^3_{10}$)	0.9342 ± 0.0001 ($vnet^3_{10}$)
Dermatology	0.9749 ± 0.0116 ($net_{50,25}$)	0.9692 ± 0.0182 ($vnet^3_{10}$)	$\mathbf{0.9750 \pm 0.0248}$ ($vnet^3_{100}$)
Diabetic	0.7254 ± 0.0290 (net_{100})	0.7315 ± 0.0238 ($vnet^3_{10}$)	$\mathbf{0.7333 \pm 0.0231}$ ($vnet^3_{50}$)
Biodegradation	0.8635 ± 0.0336 (net_{10})	$\mathbf{0.8673 \pm 0.0225}$ ($vnet^3_{100,10}$)	0.8569 ± 0.0108 ($vnet^3_{50}$)
Climate simulation	0.9500 ± 0.0140 ($net_{50,25}$)	0.9519 ± 0.0211 ($vnet^3_{10}$)	$\mathbf{0.9556 \pm 0.0240}$ ($vnet^3_{100}$)

Table 5.5: Obtained accuracies in classification datasets by a K-Fold Cross-validation evaluation. In bold the best results. The best neural architecture is in round brackets.

5.5.2 Second experimental scenario: Convolutional NN

In Table 5.6 are showed mean and standard deviations of accuracy for the three classification datasets Cifar10, MNIST and Fashion MNIST using a 10-fold cross-validation approach. The best results are reported in bold style. One can note that VAF approach outperforms always the standard approach, especially using random initialization scheme. Also in this experimental scenario the standard deviations obtained by networks with VAF remain comparable or lower than those without VAF subnetworks. Especially in Cifar10 case, we obtain a considerable improvement. In Figure 5.7, 5.10 are shown some examples of learned activation functions respectively in $vcnn_2$ and $vcnn_3$; in this scenario, the initialization of the VAF hidden units seems to condition the performance more clearly, especially for more complex dataset as Cifar10, where the performance improvements is more evident. Analyzing the shape of the resulting activation function, it seems that, in case of initialization as ReLU, the initial shape remains mostly unchanged, giving a resulting function that looks like a PReLU/Leaky ReLU. A more interesting behavior is given by random initialization, where every VAF unit seems to exhibit greater changes respect

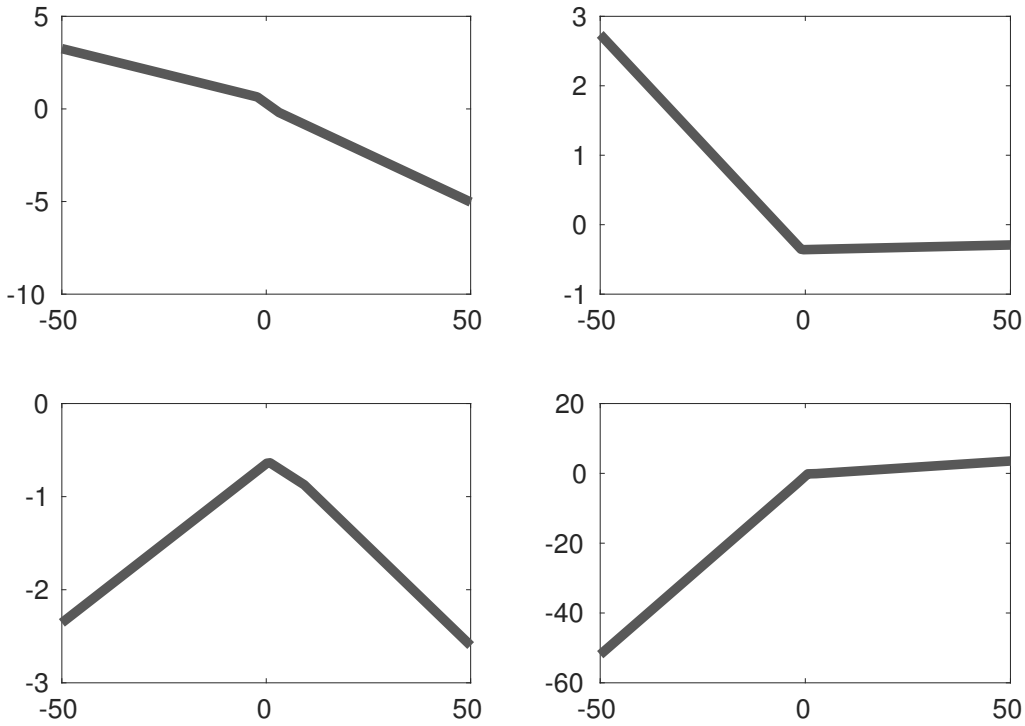


Figure 5.4: Plots of some VAF behaviors on Full-Connected NN at the end of the learning process.

to the initial function. This greater variability given by random initialization respect to ReLU initialization seems to give an improvement in accuracy results as shown in Table 5.6.

	standard ReLU Acc. + St.Dev	VAF Init random Acc. + St.Dev	VAF Init ReLU Acc. + St.Dev
Cifar10	0.857 ± 0.002 ($cnet_3^5$)	$\mathbf{0.875 \pm 0.003}$ ($vcnet_3^5$)	0.860 ± 0.002 ($vcnet_3^5$)
MNIST	0.991 ± 0.001 ($cnet_2^5$)	$\mathbf{0.994 \pm 0.001}$ ($vcnet_2^5$)	0.993 ± 0.002 ($vcnet_2^5$)
Fashion MNIST	0.923 ± 0.001 ($cnet_2^5$)	$\mathbf{0.935 \pm 0.002}$ ($vcnet_2^5$)	0.934 ± 0.001 ($vcnet_2^5$)

Table 5.6: Results on convolutional networks with 10-fold cross Validation

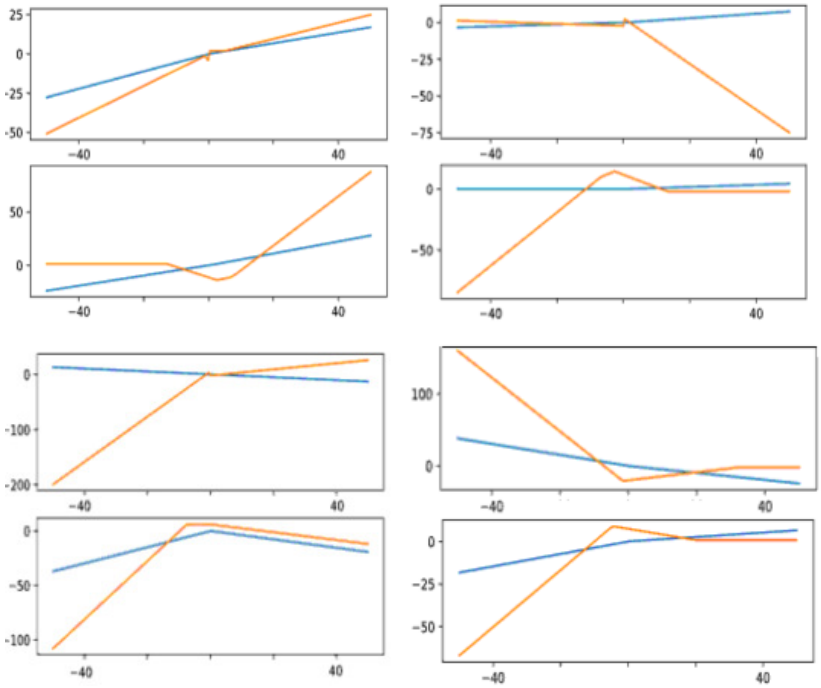
5.6 Conclusions

In this work, we proposed a simple and direct way to obtain adaptable activation functions in a feed-forward neural networks. In particular, we proposed to modify a feed-forward neu-

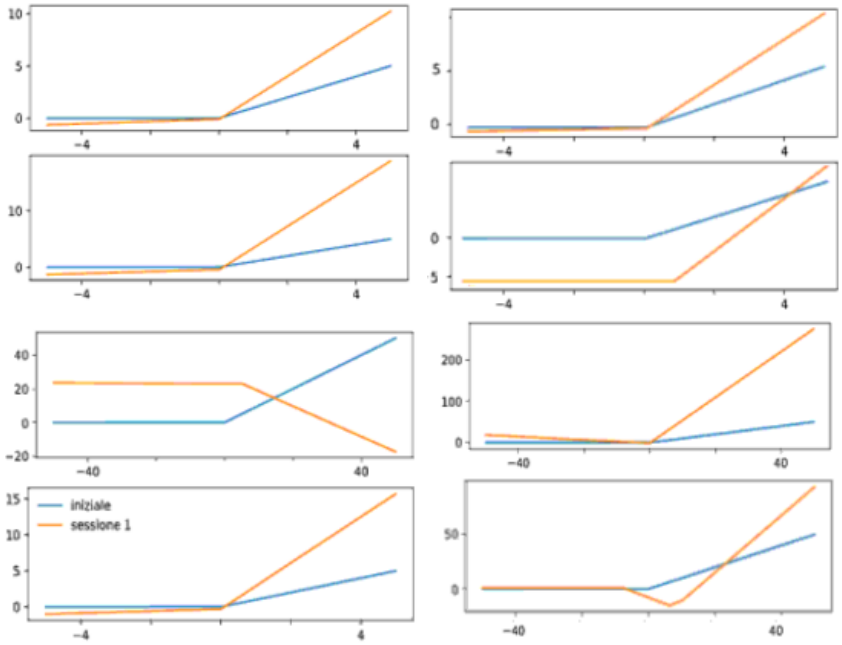
ral network without adaptable activation function by adding Variable Activation Functions (VAF) in terms of one-hidden layer subnetworks. The resulting network is again a feed-forward neural network. The proposed architecture doesn't need of many more parameters than networks using not adaptable activation functions as ReLU and the learning process follows standard approaches. Importantly, VAF subnetworks can approximate arbitrarily well any activation functions provided that the number of hidden neurons is sufficiently large. We experimentally evaluated our architecture in two different series of experiments. In the first series, we considered full-connected Multi-Layered Neural Network (MLFF) networks. More specifically, we selected 10 networks with 1 or 2 hidden layers. A correspondent network with VAF subnetworks was built for each of these 10 networks. We obtained a total of 20 different neural architectures. These neural architectures were evaluated and compared using a K-Fold Cross-Validation (KFCV) procedure on 10 different datasets (see Table 5.1). The results show that the approach with VAF subnetworks is uniformly more performing than that without VAF networks. In particular, our approach overcomes that without VAF networks in the 80% of the datasets. Only on three datasets our approach had worse results.

In the second series, we considered Convolutional Neural Networks with 2 and 3 layers and correspondent networks with VAF units and we evaluate them using 3 image dataset for classification; also in this case the VAF subnetworks seem more performing respect to network with fixed-shape units.

In conclusion, VAF units have been tested using traditional DNN and CNN networks with various dataset and seem to give better results compared with networks with similar design but with traditional ReLU functions. We showed that is possible to obtain encouraging results without the need to use complex designs or particular initialization schemes.

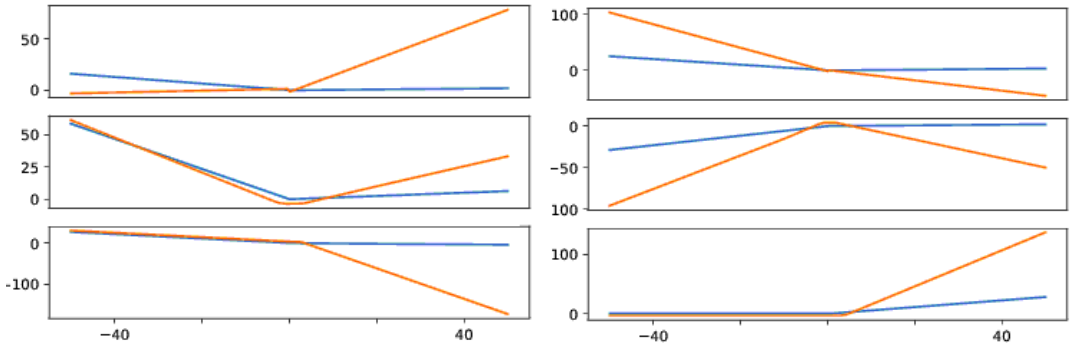


Random init.

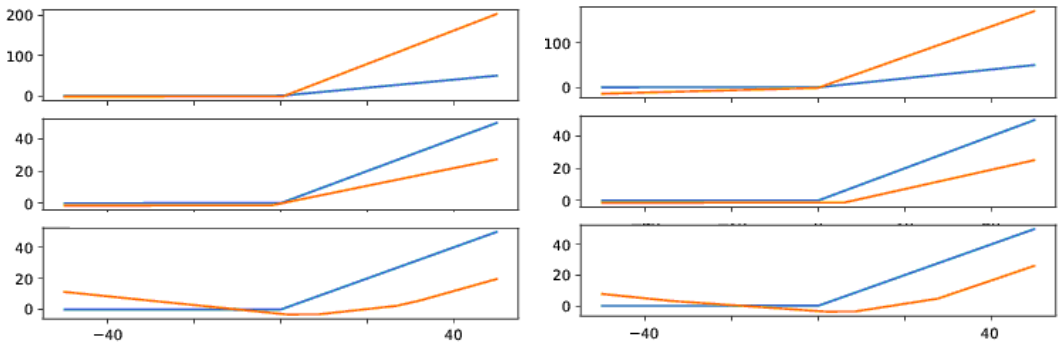


ReLU init.

Figure 5.7: Examples of changes in a VAF in a 2 layer conv. network using random (on the left) and ReLU initialization (on the right). The blue line is the start function, the orange line is the learned function.



Random init.



ReLU init.

Figure 5.10: Examples of a resulting VAF in a 3 layer conv. using random (top) and ReLU initialization (bottom). The blue line is the start function, the orange line is the learned function.

Part II

Integration of Context Information through Probabilistic Ontological Knowledge into Image Classification

Chapter 6

Problem definition and related works

Introduction

This chapter wants to give an introduction to the ontologies; in the first part, the term "ontology" will be discussed, showing how it is difficult to formalize. Various definitions have been proposed over the years, each of these adding or modifying the previous ones trying to give an ever more precise meaning to the term. It will therefore be shown how, from the concept of ontology, we have passed to the concept of probabilistic ontology, and of how this is defined. The chapter concludes with a brief state of the art on the use of ontologies in pattern recognition and how these can be used in the representation of the *context*, that is, in the representation of relationships between items in the same domain, focusing in the image domain.

6.1 Problem definition

The topic of this part is the problem of recognising the content of a digital image. This is a particularly important problem due to the very large number of images now available on the Internet, and for producing an automatic description of the content of the images. This research topic has received increasing attention, as shown by the references in Section 6.3, and well performing systems using deep networks have been proposed. We consider a method for exploiting context information in the image for improving the performance of a classifier. Classifiers for recognising the content of natural images are based usually on information extracted only from images, and can be, in the most general case, prone to errors. The approach taken in this paper attempts to integrate some domain knowledge in the loop. The framework presented here aims at integrating the output a classier/detector,

considered as the probability of a particular object to be present in a definite part of the input image, with an encoded domain knowledge. The most used tool for encoding a-priori information are standard ontologies, however, they do fail when dealing with real world uncertainty. For this reason we preferred to include in our framework a Probabilistic Ontology (henceforth PO) [Ding and Peng, 2004], which associates probabilities to the coded information, and then provides an adequate solution to the issue of coding the context information necessary to correctly understand the content of an image. Such information is then combined with the classifier output to correct possible classification errors on the basis of surrounding objects.

The aim of this work is to boost the performance of a system for the recognition/identification of classes of objects in natural images introducing in the loop knowledge coming from the real world, expressed in terms of probability of a set of spatial relations between the objects in the images. A probabilistic ontology can be made available for the considered domain, but it could also be built or enriched by using entities and relations extracted from a document related to the image. For example, the picture could have been extracted from a technical report or a book, where the text gives information which are related to the considered images. We wish to stress the fact that we are not thinking of a text directly commenting or describing the image, but of a text which is completed and illustrated by the image. In this case, both the classes of objects which can appear in the image and the relations connecting them could be mentioned in the text and could therefore be automatically extracted [Bach and Badaskar, 2007]. A probability can then be associated to them on the basis of the reliability of the extraction or the frequency of the item in the text.

The objective of the our system is to obtain a set of keywords that can be used to describe the content of an image. The system takes an image as input and produces a set of hypotheses on the presence of some objects in the image. Some of this hypotheses are likely to be wrong. As an example let us consider the case of the reflection of a building on the water, beneath a boat; it is likely that a simple classifier will label that reflection as building, while the boat can be labelled correctly. Our opinion is that the spatial relation between the two image segments, together with the external knowledge that an image segment beneath a boat and surrounded by water is more likely to be water than a building, can be used to correct the misclassification. This world knowledge, formalised in a probabilistic ontology, together with the output of the classifier, is fed to a probabilistic model [Bishop, 2006], with the goal to improve the performance of the single classifiers.

The framework described in this work has two main aspects of novelty. The first one is that, at the best of our knowledge, a probabilistic ontology has never been proposed for a computer vision problem. The integration of a probabilistic model with a probabilistic ontology presents a second element of novelty.

6.2 Ontologies

The term "ontology" has remote origins; born in the philosophical field (as discussed in, for example, [Devaux and Lamanna, 2009, Guarino, 1995]), it comes to the present day in the information technology field as a knowledge description; one of the first definitions of "ontology" in computer science was given in [Gruber, 1995]:

Definition 1. *"An ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence. For Artificial Intelligence (AI) systems, what "exists" is that which can be represented. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly".*

A more formal description of an ontology is given by in [Guarino, 1995], where the previous definition is examined and discussed, focusing on the meaning of term *conceptualization*, which is fundamental for the ontology concept. In [Genesereth and Nilsson, 1987] a *conceptualization* is defined as

Definition 2. *"Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly."*

and is explained as an equivalent of an *extensional relational structure* that we report as defined in [Guarino et al., 2009]:

Definition 3. ***An extensional relation structure*** *is a tuple* (D, R) *where:*

- D *is a set called the universe of discourse*
- R *is a set of relations on* D*.*

In [Guarino et al., 2009] is pointed out that the above definition depends too much on a specific state of the world, while a *conceptualization* should be about *concepts*, *i.e.,* it should

not change if the world changes, while an extensional specification of a conceptualization would require listing the extensions of every (conceptual) relation for all possible worlds. For this purpose, the authors suggest to specify a conceptualization in an *intensional way*, using suitable axioms in a given first-order logic language L. More formally, the authors define a conceptualization as an *intensional relation structure*:

Definition 4. *An **intensional relation structure** is a triple $C = (D, W, R)$ with:*

- *D is the universe of the discourse;*
- *W the set of the possible worlds;*
- *R a set of intensional relation in the domain $< D, W >$ where an intensional relation r_n of arity n is a total function $r_n : W \rightarrow 2^{D^n}$ where 2^{D^n} is the set of all n-ary extensional relation on D.*

Finally, using the above definition, [Guarino et al., 2009] defines *ontological commitment* used to define an *ontology*:

Definition 5. *Let L be a first-order logical language with vocabulary V and $C = (D, W, R)$ an intensional relation structure; an **ontological commitment** for L is a tuple $K = (C, I)$ where I is a total function $I : V \rightarrow D \cup R$ that maps every symbol of V to either an element of D or to an intensional relation in R.*

Definition 6. *Let L be a first-order logical language with vocabulary V and $C = (D, W, R)$ an intensional relation structure and K an ontological commitment; an **ontology** O_K for C with vocabulary V and ontological commitment K is a logical theory consisting of a set of formulas of L, designed so that the sets of its models approximates as well as possible the set of intended models of L according to K.*

So, an ontology can be viewed as just a logical theory designed in order to capture the intended models corresponding to a certain conceptualization and to exclude the unintended ones, in other terms "it is an approximate specification of a conceptualization: the better intended models will be captured and non-intended models will be excluded" [Guarino et al., 2009].

Another important property of an ontology, as described in [Sir et al., 2015], is the *Open Word Assumption* (OWA,[Reiter, 1987]), which highlights even more the difference between ontologies and common databases, which are based instead on *Closed Word Assumption* (CWA, [Genesereth and Nilsson, 1987]); while CWA is used by systems that

assume to have complete information about a given domain, an OWA system is based on the assumption that the contained information are incomplete, so the desired information have to be inferred or deducted.
In [Noy and Mcguinness, 2001] are summarized a list of typical reason for the development of ontologies:

- To share common understanding of the information structure between people or software;
- To enable reuse of the domain knowledge;
- To make the domain assumptions;
- To separate the domain knowledge from the operational knowledge;
- To analyze the domain knowledge.

Ontologies can not cope properly with uncertain information when dealing with real world problems. To overcome this problem, over the last years some tools have been designed for adding probabilities to the information contained in ontologies, obtaining *probabilistic ontologies* [Ding and Peng, 2004]. A definition of probabilistic ontology is given in [Costa, 2005]:

Definition 7. *A probabilistic ontology is an explicit, formal knowledge representation that expresses knowledge about a domain of application. This includes:*

1. *Types of entities that exists in the domain;*
2. *Properties of those entities;*
3. *Relationships among entities;*
4. *Processes and events that happen with those entities;*
5. *Statistical regularities that characterize the domain;*
6. *Inconclusive, ambiguous, incomplete, unreliable, and dissonant knowledge;*
7. *Uncertainty about all the above forms of knowledge;*

where the term entity refers to any concept (real of fictitious, concrete or abstract) that can be described and reasoned about within the domain of application.

Among the tools proposed one of the most important is probably PrOWL [Costa, 2005]; PrOWL, based on Multi Entity Bayesian Networks (MEBN) [Laskey, 2008] (an extension to Bayesian [Ben-Gal, 2008] network to first order logic's expressive power), allows to define probabilistic ontologies capable to encode a priori knowledge for real world applications and make inferences based on ontologies defined.

6.3 Related Work

The importance of context is known from the dawn of pattern recognition; in [Toussaint, 1978] is made a clear dissertation on this topic, pointing out how the same entity can show different properties in different contexts, showing some basic visual example. An example of the importance of relations between object In the human brain is shown in [Bar and Ullman, 1996] where is stated that "proper spatial relations among the features of a scene decrease response times and error rates in the recognition of individual features"; this consideration is shown through a set of psycho-physical experiments. In pattern recognition, the work done in [Schneiderman and Kanade, 1998] uses a probabilistic model for object recognition task using local appearance. For instance, given an object O and an image I, the authors model the posterior probability $P(O|I)$ in a functional form applying a set of simplifications and assumptions to the general form of the posterior probability function, i.e. $P(O|Region) = \frac{P(Region|O)P(O)}{P(Region)}$; applying the Bayes theorem and making statistical independence assumptions, is then possible to write a decision rule in the form:

$$\begin{cases} \frac{P(Region|O)}{P(Region|\overline{O})} \geq \lambda & \text{the object is present in the region} \\ \frac{P(Region|O)}{P(Region|\overline{O})} < \lambda & \text{the object is not present in the region} \end{cases}$$

with $\lambda = \frac{P(\overline{O})}{P(O)}$. This approach could be generalized for a large set of object, taking into account that assumption as statistical independence makes this approach not good to represent many relationships as for example brightness distribution across the object larger than a subregion. For instance, authors tries this approach for face detection.

In a similar way, [Schmid, 1999] tried to recognize a query image which can be a part of a model image in a probabilistic way; briefly, given a query image Q and a set of n model images $M = \{m_1, m_2, \ldots, m_n\}$, they model every possible set of matches between a query image and model images as an *Hypothesis*, so the query image can be represented by the set $H = \{h_1, h_2, \ldots, h_k\}$ of all possible *Hypothesis* and searching the most similar model image can be view as to find the model image m^* with the highest probability $P(m^*|H)$; assuming independence between hypothesis in H and using Bayes rule,

$\Pr(m^*|H) = \frac{\sum_{i=1}^{k} \Pr(H_i,|m^*) \Pr(m^*)}{\sum_{j=1}^{n} \sum_{i=1}^{k} \Pr(H_i,|m_n) P(m^*)}$; the probabilities $P(H_i, |m_n)$ are computed as combination of correspondences quality, spatial and global coherence, as defined in [Schmid, 1999]. A few years later, another study in this direction, but for image automatic annotation, was proposed in [Zhang et al., 2005], where a probabilistic semantic model in which the visual features and the textual words are connected via a hidden layer is proposed. More recently in the context of 3D object recognition, a system that builds a probabilistic model for each object based on the distribution of its views was proposed in [Wang et al., 2013].

In [Spyns et al., 2002] the differences and similarities between ontologies and data models (as databases) are discussed, and a new ontology engineering framework (DOGMA framework), which wants to resolve the conflict between the generality of the knowledge, as a fundamental asset of an ontology, and the high number of domain rules that are needed for effective interoperability, is introduced. The authors of DOGMA decompose an ontology into an "ontology base", which consists of sets of intuitively "plausible" domain fact types, and a layer of "ontological commitments", where each commitment holds a set of domain rules to mediate between the ontology base and its applications.

[Breen et al., 2002a, Breen et al., 2002b] was, in our knowledge, one of the first studies that introduces ontologies in a classification framework; the proposed system combines the use of ontologies and neural networks as object identifiers to improve the precision in the classification of an image based on its content using the actual objects within an image to discover useful relationships classifying the entire image.

The study presented in [Mezaris et al., 2003] is a method to combine image processing, ontology-based knowledge and machine learning techniques together to make an object-based image retrieval system; in this work, the ontology results as a vocabulary used to make a qualitative description of an image region obtained by an initial segmentation process. So, for every semantic object, is necessary to supply a description using the intermediate-level descriptors provided by the ontology. These intermediate-level features are mapped to a set of low-levels features calculated for each region that summarize properties as color, position and shape. The intermediate descriptor are used to product a first output which is subsequently refined with a relevance-feedback system (useful for excluding obviously undesirable regions) and an Support Vector Machine to product the final query output. In [Hollink et al., 2003] was proposed a tool to make semantic search in image dataset with the support of four common ontologies.

[Wu et al., 2004] an ontology-based learning strategy to improve the accuracy of individual classifiers considering the possible influence relations between concepts in an ontology hierarchy was proposed. Moreover, an ontology driven classification system for protein classification was proposed in [Wolstencroft et al., 2006].

The authors of [Chang and Huang, 2008] propose a document classifier system based on Ontology and the Naive-Bayes Classifier.

[Fan et al., 2007, Fan et al., 2008] proposed a scheme for achieving automatic multi-level image annotation incorporating concept ontology in large-scale image collections. In [Hudelot et al., 2008] was proposed an interesting work dedicated to the representation of uncertain using a fuzzy representations of spatial concepts in the image domain, i.e. relations such as "intersects", "in the interior of", "exterior to" and many others were defined from fuzzy set theoretical concepts [Dubois, 1980]; these relations are then used to enrich a generic spatial ontology in order to guide image interpretation.

in [Zhang et al., 2014] a hierarchical Bayesian network is introduced in a weakly supervised segmentation model; in particular the system learns the semantic associations between sets of spatially neighbouring pixels, defined as the probability these sets to share the same semantic label.

In the context of image retrieval, [Sarwar et al., 2013] proposes SIRNS (Semantic Image Retrieval of Natural Scenes), an ontology based image retrieval framework from a corpus of natural scene images while, in the context of information Retrieval, [Alicante et al., 2014] introduced a probabilistic model based on Graphical Models to integrate ontological constraints with the output of a statistical classifier, with the goal of constructing an index for a more semantically oriented search engine for collection of documents. [Abdollahpour et al., 2015] improves Bag-of-Visual-Words method [Sivic and Zisserman, 2003, Csurka et al., 2004] using an ontology to achieve higher classification accuracy.

Finally [Eweiwi et al., 2015], in the context of action recognition, presents a generative model that allowsfor characterising joint distributions of regions of interest, local image features, and human actions.

Chapter 7

Integrating a priori knowledge

Introduction

This chapter wants to describe the approach proposed to improve the classification results using the knowledge given by an ontology. Using the general architecture described in Section 7.1, two different framework to combine together the knowledge given by the ontology with the results given by a generic probabilistic classifier are designed. It is worth to point out that the proposed frameworks are *model-agnostic*, that is they don't require to know any information about the structure and the internal state of the used classifier. The only constraint required by our frameworks is that the used classifiers return a probability distribution of the possible classes for any input. The chapter is structured as follow: Section 7.1 describes the proposed system architecture; 7.2 describes our experimental setup; in Section 7.3 are shown the results obtained.

7.1 Proposed architecture

The proposed model is composed by the following components:

- a probabilistic classifier, which returns the probability for a detected object to belong to a class;
- a probabilistic ontology, which contains the knowledge about object relations in a probabilistic way;
- the probabilistic model, which merges the ontological knowledge with the class probability given by the classifier.

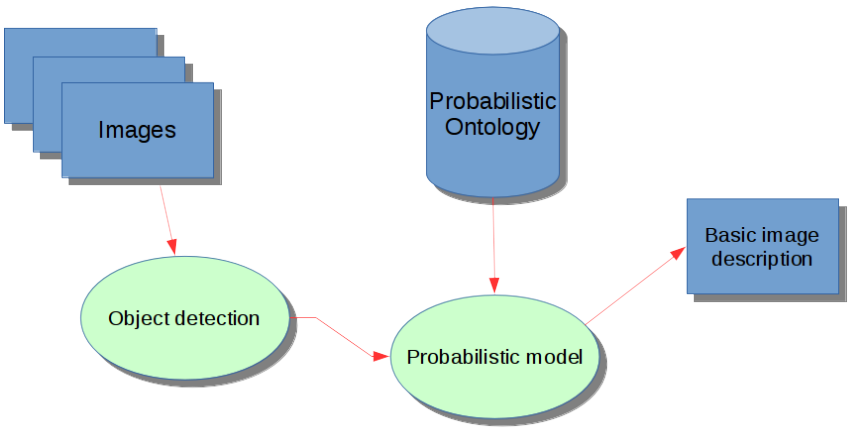


Figure 7.1: Scheme of the proposed framework.

We start from the hypothesis to have a classifier that does not always return reliable results, for example due to a few training data available, and then we try to improve the results of the classification through the use of the domain knowledge contained in an ontology.

The classifier

In a classification problem, given an input $x \in X$, with X input space, a probabilistic classifier Γ returns an estimated class for x using an estimated probability distribution on the set C of all possible classes, i.e. $\Pr_{\Gamma}(c|x)$, $\forall c \in C$. The simplest measure of goodness of a classifier is the accuracy, that is the percentage of examples correctly classified in a given set (generally called test set). Formally, given a classifier Γ, a test set $S = \{(x^{(1)}, c^{(1)}), (x^{(2)}, c^{(2)}), \ldots, (x^{(n)}, c^{(n)})\} \subseteq X$, and a classifier $\Gamma : X \to C$, the accuracy A is given by:

$$A = \frac{|\{\Gamma(x^{(i)}) = c^{(i)} \text{ s.t. } (x^{(i)}, c^{(i)}) \in S\}|}{|S|}$$

with $\Gamma(x^{(i)}) = \arg\max_{c \in C} \Pr_{\Gamma}(c|x)$.

The accuracy doesn't take into account how much every wrong classification is far from the correct one, since the predicted class is the one with the maximum probability, without taking into account the other class probabilities that can be viewed as other source of information. For example, when two or more class probabilities are very close to each other, the right class could be not the first one but the second one in terms of probability, and this could be a particularly serious issue when the right class and the wrong class selected by the classifier are very close to each other in terms of probability's

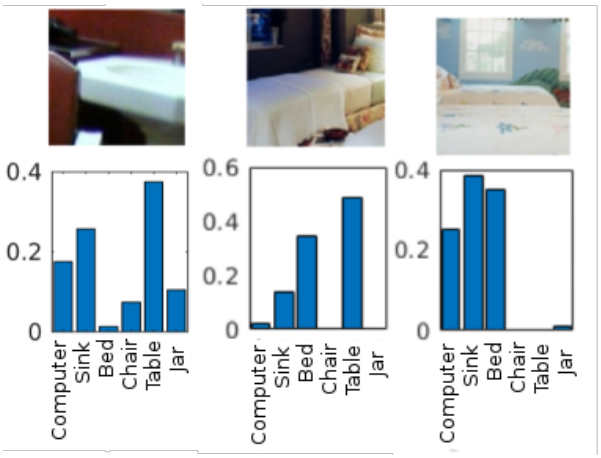


Figure 7.2: Examples of probability distributions for a classifier.

values. Figure 7.2 shows an example: on the left, an image classifier gives the probability distribution under six class: computer, sink, bed, chair Table, Jar; the most probable class for the classifier is the table, while the real class (sink) is in second-most probable class; the center and the right figures show a similar situation with the right class (bed) that is still the second one in terms of probability.

So, if we define a function $\rho_\Gamma(x, c) : X \times \{1, \ldots, |C|\} \to C$ which returns, for every $x \in X$, the K-th most probable class under the distribution $\Pr_\Gamma$, we can define a classifier Γ a K-best classifier with accuracy A if it gives a mean accuracy of A and, for every misclassified item $\overline{x}$, the real class $\overline{c}$ is in $\{\ \rho_\Gamma(\overline{x}, 2), \ldots, \rho_\Gamma(\overline{x}, K)\}$.

In other terms, in a K-best classifier we take care not only on the accuracy A, but also if the right class is in the first K classes or not, so it is possible to deduce that the learned model is not too wrong and that it can be improved without training it from scratch. Furthermore, we can use the distance between probabilities as a measure of certainty of a classifier: if for an input x there is a value preponderant over the others, a classifier can be considered *Certain* about its result, but if there are many values close to the maximum, the classifier can be considered *Doubtful*.

In other words, we want to take into account when the classifier has more than one possible plausible choice for a given input x; in the worst case, all the classes have the same probability, that is we have an uniform distribution where every class has a probability of value $\frac{1}{|C|}$, with $|C|$ number of classes; so, given an input x and a $|C|$-class classifier Γ, we can say that:

- the classifier Γ is *Certain* about $x^{(i)}$ if exists a class probability $c' \in C$ that is

positively "far enough" from the uniform probability, i.e. $\exists c' \in C$ s.t. $|\frac{1}{|C|} - \Pr_{\Gamma}(c^{(i)} = c')| > 1 - \delta$ with $0 \leq \delta \leq 1$ *degree of reliability* of the classifier Γ;

- the classifier Γ is $Doubtful$ about $x^{(i)}$ when the classifier is not $Certain$, i.e. all the classes are close to the uniform probability.

When the classifier is $Certain$, we can be sure enough of the answer, but when it is $Doubtful$ we may need external help.

The Probabilistic Ontology

The scheme of the chosen Probabilistic Ontology contains a set of spatial relations between the classes of the items. The probabilities are estimated from a training set of images where the objects have been manually labelled, and spatial relations are constructed analyzing pairs of regions of interest.

The probabilistic models

We want to improve a probabilistic classifier Γ using knowledge in an Ontology Ω, so we propose two strategies for merge the probabilities given by Γ together with the probabilities given by Ω; the first one can be suitable for a classifier where the only performance measure considered is the classic accuracy, while in the second one we consider different top-K classifiers (see section 7.1).

First Framework

In this scenario, we consider a generic classifier without taking care about any goodness measures about the learned classifier. We gain the problem learning a log-linear model that bind together the classification given by the probability with the classification given by the ontology; we can model the probability that the real classes $c^{(i)}, c^{(j)}$ of two input $x^{(i)}, x^{(j)}$ that are in a relation $r(x^{(i)}, x^{(j)}) \in R$ are respectively c_1 and c_2 as:

$$\begin{aligned}&\Pr\left(c^{(i)} = c_1, c^{(j)} = c_2 | x^{(i)}, x^{(j)}, r(x^{(i)}, x^{(j)})\right) = \\ &= \frac{1}{Z} \exp\left(v_{c_1} \Pr_{\Gamma}(c^{(i)} = c_1 | x^{(i)}) + v_{c_2} \Pr_{\Gamma}(c^{(j)} = c_2 | x^{(j)}) + w_{r,c_1,c_2} \Pr_{\Omega}(r, c_1, c_2)\right)\end{aligned}$$

with $\{v_c : c \in C\}, \{w_{r,c_1,c_2} : c_1, c_2 \in C \wedge r(c_1, c_2)\}$ parameters to learn; the first ones are class parameters, the second one are relation parameters; in total, we have $|C|$ class parameters

and $|R| \cdot |C|^2$ relation parameters to estimate using a maximum likelihood approach. So, given an input $x^{(i)}$, the final class can be estimated computing a score for every class c:

$$\text{Score}(c, x^{(i)}) = \max_{x^{(h)}:\exists r \in R, r(x^{(i)}, x^{(h)})} \sum_{c_2 \in C} \sum_{r \in R} \Pr\left(c^{(i)} = c, c^{(h)} = c_2 | x^{(i)}, x^{(h)}, r(x^{(i)}, x^{(h)})\right) \quad (7.1)$$

and we find the final estimated class $\hat{c}$ for the input x maximizing on it:

$$\hat{c} = \arg\max_{c \in C}(\text{Score}(c, x)) \quad (7.2)$$

Second framework

In this scenario, we consider doubtful classifiers, i.e. with more classes close to the maximum probability values. Remembering that, as stated in Section 7.1, we define a classifier Γ *Certain* about an input $x^{(i)}$ if $\exists c' \in C$ t.c. $|\frac{1}{|C|} - \Pr_{\Gamma}(c^{(i)} = c')| > 1 - \delta$, we model the probability that the real classes $c^{(i)}, c^{(j)}$ of two input $x^{(i)}, x^{(j)}$ that are in a relation $r(x^{(i)}, x^{(j)}) \in R$ are respectively c_1 and c_2 as:

$$\Pr(c^{(i)} = c_1, c^{(j)} = c_2 | r(x^{(i)}, x^{(j)})) =$$
$$= \begin{cases} \Pr_{\Gamma}(c^{(i)} = c_1, c^{(j)} = c_2 | x^{(i)}, x^{(j)}) & \text{if } |\frac{1}{|C|^2} - \Pr_{\Gamma}(c^{(i)} = c_1, c^{(j)} = c_2 | x^{(i)}, x^{(j)})| > (1-\delta)^2 \\ \Pr_{\Gamma}(c^{(i)} = c_1, c^{(j)} = c_2 | x^{(i)}, x^{(j)})(1 + \epsilon G(\Delta P)) & \end{cases}$$

where:

- $\Pr_{\Gamma}(c^{(i)} = c_1, c^{(j)} = c_2 | x^{(i)}, x^{(j)})$ is the probability given by the classifier Γ that $x^{(i)}$ belongs to class $c^{(i)}$ and $x^{(j)}$ belongs to class $c^{(j)}$; assuming that $x^{(i)}$ and $x^{(j)}$ are statistically independent, $\Pr_{\Gamma}(c^{(i)} = c_1, c^{(j)} = c_2 | x^{(i)}, x^{(j)}) = \Pr_{\Gamma}(c^{(i)} = c_1 | x^{(i)}) \Pr_{\Gamma}(c^{(j)} = c_2 | x^{(j)})$;
- $\frac{1}{|C|^2}$ is the uniform probability of two points assuming that they are statistically independent;
- $0 \leq \delta \leq 1$ is the degree of reliability of the Classifier;
- $\epsilon = 1 - \delta$ is the is degree of reliability in the ontology Ω; it shows how much increase or decrease the probability relying on the Ontology response; we assume that a decreasing confidence in the classifier increases the confidence in ontology;

- $\Delta P = \Pr_{\Omega}(c_1, c_2|r) - \Pr_{\Gamma}(c^{(i)} = c_1, c^{(j)} = c_2|x^{(i)}, x^{(j)})$ is the difference between the probability given by the Ontology and the probability given by the Classifier (assuming that $x^{(i)}, x^{(j)}$ are statistically independents);

- $G(x)$ is a function that takes into account the difference ΔP between the probabilities given by Γ and Ω; for simplicity, we choice $G(x) = \text{sign}(x)$, so that $G(\Delta P)$ gives us just the sign of its argument, leaving the amount of how much increase or decrease the potential to the confidence factor ϵ (e.g. if the probability given by Ω is greater than the probability given by Γ, we increase the potential of a percentage ϵ, decrease otherwise).

The score and the final class is then computed using Equations 7.1 and 7.2.

7.2 Experimental assessment

The main objective of the experiments described in this Section is to assess whether the model proposed really improves the performance of a classifier. To this end we measured the classification performance of our model against the classifier performance.

7.2.1 Model validation

The dataset selected for this experimental assessment is a subset of the *MIT-Indoor* [Quattoni and Torralba, 2009], where interesting objects have been manually segmented and labelled; this approach gives us a reliable ground-truth for estimating the performance of our combination model. The data set includes $1,700$ images that have been manually segmented. These images were taken at common indoor locations, such as kitchens, bedrooms, libraries, gyms and so on. The data set has been partitioned in three subsets: the first two, each one containing 30% of the whole data set, are used for training the probabilistic ontology (D_{PO}) and the first combination model (D_{CM}) proposed in this work, and the remaining 40% go into the D_{Test} subset that is used to assess the performance of the system. In our experiments the three subset have been selected randomly at each run of the algorithm. Each run has been repeated several times in order to avoid experiment bias due to lucky or unlucky data splits. From our point of view it is particularly important that the probabilistic ontology and the combination model are trained on different data, as this is what it is very likely to happen in real cases.

The data-set contains a large set of object classes, some of them with very few objects. In order to avoid the impact that small classes might have on the construction of the probabilistic ontology we preferred to take only the six with the largest number of items.

Used classifiers

In order to make this experiment as general as possible we initially decided not to use an existing system for the detection/classification task, but preferred to design a simulated top-K classifier, for which we were able to set a desired accuracy (see, for instance, [Zouari et al., 2004]) together with K value. To this end we designed a strategy that is detailed by the pseudo-code in Algorithm 9. Briefly, we assign the highest probability value to the gold class with a probability given by the desired accuracy, while in the other case the gold class has a probability in the first K value in descending order.

In this way it is possible to have an idea of the impact that the ontological information has on the performance, and to describe the dependence of the system performance on the classification accuracy. Next, we tried in a real case if the chosen approach led to significant improvements. Once verified that our approach gave acceptable results in the simulated classification environment, we moved to an experimental environment that uses a real classifier. For instance, we used a Neural Network designed as described in [Krizhevsky et al., 2012], well known in the literature thanks to the good performances achieved on difficult data-sets like ImageNet [Deng et al., 2009].

Ontology construction

The ontology used was build searching a set of binary relation as described in Section 7.1 on the $D_{PO} \subset D$ subset. We choose the following two symmetrical relations:

- intersect(x, y): two objects x and y intersect if exists at least a point of x contained in y;
- closeness(x, y, T) : the object x is close the object y if $\frac{Area(x)+Area(y)}{ConvexHull(x,y)} > T$ where $Area(\cdot)$ is a function that returns the area of the argument and $ConvexHull(\cdot, \cdot)$ returns the convex hull of the arguments.

the final relations set R is then composed by the following three relation:

$$R = \{\text{intersect}(x, y), \text{near}(x, y), \text{veryNear}(x, y)\}$$

with $\text{near}(x, y) = \text{closeness}(x, y, T_1)$ and $\text{veryNear}(x, y) = \text{closeness}(x, y, T_2)$ using $T_1 = 0.5$ and $T_2 = 0.8$. The probability that two classes are in a given relation is estimated by the frequency of such event in the data set. Formally, denoting by D a set of image segments used for computing the probabilities, and with R the set of relations considered, with C the set of classes of objects, the probability that an object of class $c_1 \in C$ is in relation $r \in R$ with an object of class $c_2 \in C$ is computed as as:

$$\Pr_{\Omega}(r, c_1, c_2) = \frac{|D_r(c_1, c_2)|}{\sum_{c_x, c_y \in C} |D_r(c_x, c_y)|} \tag{7.3}$$

where $D_r(c_1, c_2) \subseteq D$ is the set of regions of classes respectively c_1 and c_2 that satisfy the relation r. for simplicity, we use binary relation but the structure is easily generalizable to relations with different cardinality.

We use Protégé [Musen, 2015] for formalizing the schema of the ontology, and we use Pronto [Klinov and Parsia, 2008, Klinov and Parsia, 2013] as a reasoner for Probabilistic Ontologies.

7.2.2 First experimental scenario

In this scenario, we tried our first Framework described in Section 7.1 using a simulated multi-class top K-classifier with a given accuracy. For the first experimental framework, we don't need the K-best properties, so we fix $K = |C|$ i.e. the only parameter is the desired accuracy A. The tests were performed with simulated classifiers with different accuracy; for instance, we try it with accuracies in set $\{30\%, 40\%, 50\%, 60\%, 70\%, 80\%, 90\%\}$ and K in $\{2, 3, 4, 5\}$. As stated in Section 7.1, this approach needs a set of parameters to be learned. If we consider the number of used relations $|R| = 3$ and classes $|C| = 6$, we need 114 parameters. To validate our results, we repeat the experiments for 20 times and plot the average accuracy.

7.2.3 Second experimental scenario

In this scenario, we tries our second Framework described in Section 7.1 using a simulated multi-class top K-classifier with a given accuracy. The main advantage lies in not needing a training phase for the probabilistic model, at the price of stronger a priori conditions on the classifier. For this reason this approach has been tested on different simulated top-K classifiers with different accuracy values. For instance, we try it with accuracies in set

$\{30\%, 40\%, 50\%, 60\%, 70\%, 80\%, 90\%\}$ and K in $\{2, 3, 4, 5\}$. To validate our results, we repeat the experiments for 10 times and plot the average accuracy.

7.2.4 Third experimental scenario

In this experiment, we try our second approach using a real classifier. For instance, we use a neural network designed as described in [Krizhevsky et al., 2012]. To avoid retraining the network from scratch, we decided to use a freely available model already trained on ImageNet data [Deng et al., 2009] and then only make a fine-tuning phase with our training data. This kind of approach is very common in literature as a type of *transfer learning* [Pan and Yang, 2010] and it allows to obtain acceptable performances by using models already trained on data that are usually not available or that require a large computational time to be learned.

7.3 Results

7.3.1 First experimental scenario

The system accuracy of the first approach proposed in Section 7.1 are depicted in Figure 7.3 and compared with the accuracy of the simulated classifier applied alone (main diagonal). We see that the proposed model outperforms the baseline for low classifier accuracy, while deteriorate when the classifier accuracy improves. This trend is in line with our expectations, since it highlights how the contribution given by an external agent is useful in case of low performance of the basic system, while it can become irrelevant (or even counter-productive as in the case of the model under examination) in the case in which the classifier already has good performance.

7.3.2 Second experimental scenario

The system accuracy of the second approach proposed in Section 7.1 on simulated classifier are depicted in Figure 7.4; every figure shows a line for every top-K classifier+ontology with $K \in \{2, 3, 4, 5\}$. We see that the proposed model outperforms the baseline of low K-best classifier accuracy with low K, while deteriorates when the classifier accuracy improves and with high values of K. The explanation of this behavior can be given by the fact that as K increases, the goodness of the trained model decreases, making the contribution of the ontology vain if not counterproductive.

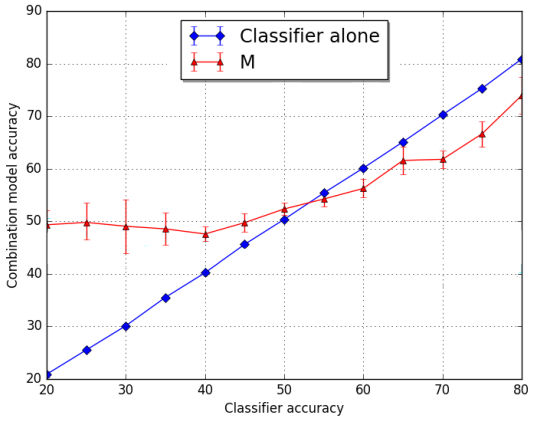


Figure 7.3: Results given by the first framework proposed (red line)

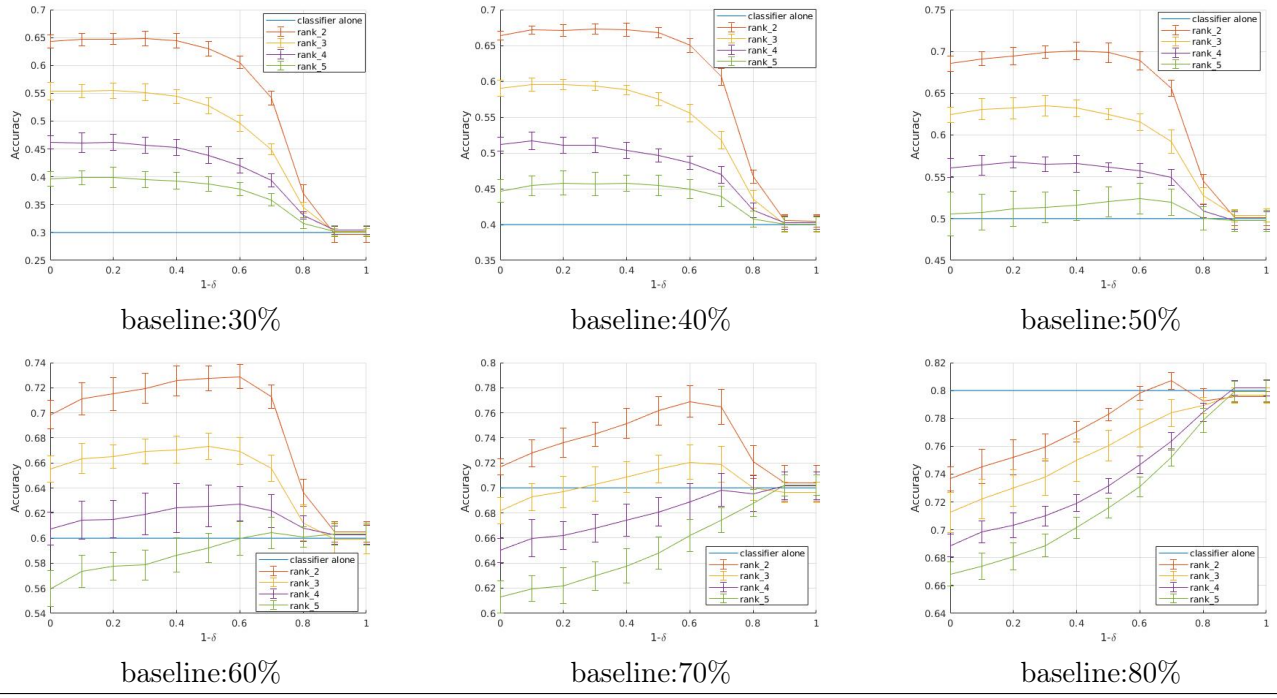
7.3.3 Third experimental scenario

The system accuracy of the approach proposed in Section 7.1 using a real classifier with different δ values are depicted in Figure 7.5; is possible to see that, with acceptable values of trust in the ontology, we have an improvement in the classification base accuracy, confirming what has already been verified with the simulated classifiers. Even in this case, giving too much confidence to the ontology, the performance gets worse. This could be due to the relationships used, perhaps too simple for the problem in exam. However, the improvements obtained seem encouraging showing the validity of the proposed method and we expect that, when more sophisticated ontologies will be available containing information from large data sets, the integration will give better results.

7.4 Conclusions

This work proposes two probabilistic models for integrating probabilities coming from a probabilistic ontology, representing a domain knowledge, with the probabilities produced by some sort of statistical classifier. The two models have been experimentally evaluated and both of them showed performance that may encourage to push forward for an use in real systems, as showed in the third experimental scenario.

In order to obtain a clear idea of the performance of the integration module, we removed the effects of most of the external factors. To this end, we conducted our experiments using images that have been manually segmented and labelled, and used a simulated classifier designed in such a way that we could control its accuracy.



baseline:30% baseline:40% baseline:50%

baseline:60% baseline:70% baseline:80%

Figure 7.4: Accuracy obtained in the second experimental framework

A prototype of a fragment of a probabilistic ontology has been designed and populated using three binary relations which can be automatically detected in input images. The probabilities corresponding to each relations have been estimated from their frequencies in the ontology training set. When more sophisticated ontologies will be available containing information from large data sets, we expect the integration to give even better results.

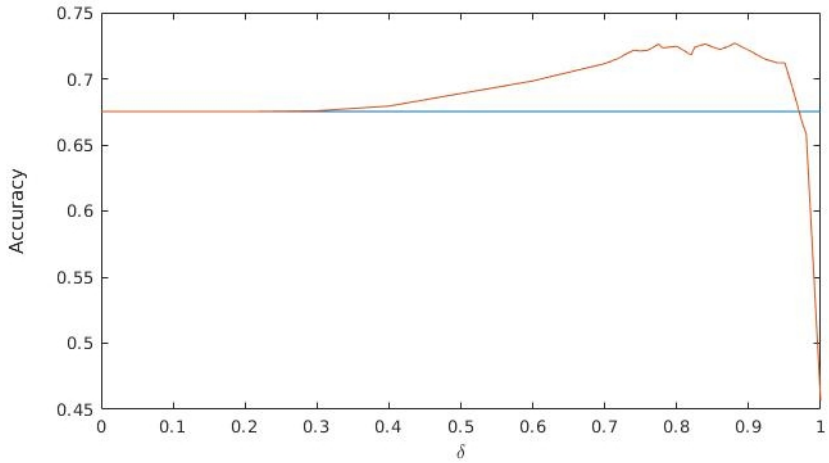


Figure 7.5: Accuracy of the second framework proposed (red line) applied to a real classifier with various values of δ; is possible to see that, with high value of δ (i.e. acceptable values of trust in the ontology) we have an improvement in the classification base accuracy (blue line)

Algorithm 9: K- best Classifier simulator

Input:

K : desidered K value;

A : desidered accuracy;

C : classes set;

c^* : the gold class index of the input to classify.

Output:

a probability distribution with the gold class c^* in the first K values.

Let $vectorProb[|C|]$;
Let $vectorOutput[|C|]$;
forall $0 \leq i < |C|$ **do**
    $vectorProb[i] \sim U(0,1)$;
$vectorProb \leftarrow Sort(vectorProb,' DescentOrder')$;
if $U(0,1) < A$ **then**
    $VectorOutput[c^*] \leftarrow VectorProb[0]$;
    $VectorProb[0] \leftarrow -1$;
else
    Let $r \sim \lfloor U(1,K) \rfloor$;
    $Output[c^*] \leftarrow vectorProb[r]$;
forall $0 \leq i < |C|$ **do**
    if $i == c^*$ **then**
        continue;
    Let $t \sim U(0, |C| - 1)$;
    while $VectorProb[t] == -1$ **do**
        $t \sim U(0, |C| - 1)$;
    $vectorOutput[i] \leftarrow VectorProb[t]$;
$normalize(vectorOutput)$;
return $vectorOutput$;

Chapter 8

Conclusions and final considerations

8.1 Achievements

This thesis has proposed two different methods to improve automatic classification performances in terms of accuracy, the first one suitable for specific type of classifier, that is a Feed-Forward Neural network, the second one suitable per each probabilistic classifier, that is a classifier whose output is a probability distribution over the possible classes.

In the first part of this work (Chapters 3,4 and 5), a novel type of trainable activation function is presented, together with survey of the other works in this area. The main contributions of this part are two, detailed below.

The first one (Chapter 4) proposes a new taxonomy to categorize the trainable activation functions proposed in the literature based on their main features, together with a description of these.

The second one (Chapter 5) presents a simple trainable activation function for Feed-Forward Neural Networks which can be build in terms of one-hidden layer sub-networks, so that it does not require any modification to the learning algorithm. Experiments show that the use of this new activation function improves the accuracy of the neural networks in many cases respect to fixed-shape activation functions.

In the second part of this work (Chapters 6 and 7), two novel frameworks to improve the accuracy of a classifiers using the support of an external knowledge base are proposed. Several tests on both synthetic and real classifiers show that the proposed approaches can improve the classification performances.

8.2 Final considerations

The introduction of rectified activation functions has certainly helped to renew the interest in topics such as neural networks, however the extensive literature on this subject (Chapter 4) is showing how these functions are not said to be the best choice, and how a "customized" function based on data can, in many cases, help performance.

The activation function proposed in this work (Chapter 5) is easy to implement and, at the same time, provides promising results compared to those obtained on functions usually used in literature.

In any case, the introduction of the new parameters necessary for a trainable activation function leads the model to be more complex, thus leading to the question of when to use this particular class of functions; obviously, this depends on the degree of accuracy that the real application requires.

The second part of the this work shows how a generic classifier can achieve better performance through the use of external media, as a basis of knowledge. The use of a simulated environment allowed us to get away from the classifier actually used, giving us the possibility to generalize our results for any probabilistic classifier. In this work the knowledge base has been implemented as a fragment of a probabilistic ontology, that has been built by using three relation types related with reciprocal position of the objects in the images. They can be automatically recognised directly in the input images, while the corresponding probabilities have been estimated from their frequencies.

We have therefore provided a possible framework to integrate the information contained in the ontology with the outputs of a classifier (Chapter 7). The strategies proved to perform in an acceptable way to a system only based on the statistical classifier and could be used in an actual system. The work carried out therefore opens up new scenarios for the revaluation of classifiers not based on neural networks, and how these can be improved even without knowing their internal structure in detail.

Bibliography

[Abdollahpour et al., 2015] Abdollahpour, Z., Samani, Z. R., and Moghaddam, M. E. (2015). Image classification using ontology based improved visual words. In *2015 23rd Iranian Conference on Electrical Engineering*, pages 694–698.

[Agostinelli et al., 2014] Agostinelli, F., Hoffman, M., Sadowski, P. J., and Baldi, P. (2014). Learning activation functions to improve deep neural networks. *CoRR*, abs/1412.6830.

[Alicante et al., 2014] Alicante, A., Benerecetti, M., Corazza, A., and Silvestri, S. (2014). A distributed information extraction system integrating ontological knowledge and probabilistic classifiers. In *2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 420–425.

[Antal and Hajdu, 2014] Antal, B. and Hajdu, A. (2014). An ensemble-based system for automatic screening of diabetic retinopathy. *CoRR*, abs/1410.8576.

[Apicella et al., 2017a] Apicella, A., Corazza, A., Isgrò, F., and Vettigli, G. (2017a). Exploiting context information for image description. In Battiato, S., Gallo, G., Schettini, R., and Stanco, F., editors, *Image Analysis and Processing - ICIAP 2017*, pages 320–331, Cham. Springer International Publishing.

[Apicella et al., 2017b] Apicella, A., Corazza, A., Isgro, F., and Vettigli, G. (2017b). Integrating a priori probabilistic knowledge into classification for image description. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2017 IEEE 26th International Conference on*, pages 197–199. IEEE.

[Apicella et al., 2018a] Apicella, A., Corazza, A., Isgrò, F., and Vettigli, G. (2018a). Integration of context information through probabilistic ontological knowledge into image classification. *Information*, 9(10):252.

[Apicella et al., 2018b] Apicella, A., Isgrò, F., and Prevete, R. (in preparation, 2018b). Trainable activation functions: a survey. *Neural Computation*.

[Apicella et al., 2018c] Apicella, A., Isgrò, F., and Prevete, R. (submitted, 2018c). Variable activation functions based on a simple and efficient neural network architecture. *Neural Networks*.

[Arlot et al., 2010] Arlot, S., Celisse, A., et al. (2010). A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79.

[Bach and Badaskar, 2007] Bach, N. and Badaskar, S. (2007). A review of relation extraction. Technical report, Language Technologies Institute, Carnegie Mellon University.

[Balzer et al., 1980] Balzer, R., Erman, L., London, P., and Williams, C. (1980). Hearsay-iii: a domain-independent framework for expert systems. In *Proceedings of the First AAAI Conference on Artificial Intelligence*, pages 108–110. AAAI Press.

[Bar and Ullman, 1996] Bar, M. and Ullman, S. (1996). Spatial context in recognition. *Perception*, 25(3):343–352.

[Bell and Sejnowski, 1997] Bell, A. J. and Sejnowski, T. J. (1997). The "independent components" of natural scenes are edge filters. *Vision research*, 37(23):3327–3338.

[Ben-Gal, 2008] Ben-Gal, I. (2008). Bayesian networks. *Encyclopedia of statistics in quality and reliability*, 1.

[Bengio et al., 2013] Bengio, Y., Boulanger-Lewandowski, N., and Pascanu, R. (2013). Advances in optimizing recurrent networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8624–8628.

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[Bishop, 2006] Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer.

[Boole, 1854] Boole, G. (1854). *An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities*. Dover Publications.

[Boser et al., 1992] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM.

[Bottou et al., 2018] Bottou, L., Curtis, F. E., and Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311.

[Breen et al., 2002a] Breen, C., Khan, L., Kumar, A., and Wang, L. (2002a). Ontology-based image classification using neural networks. In *Internet Multimedia Management Systems III*, volume 4862, pages 198–209. International Society for Optics and Photonics.

[Breen et al., 2002b] Breen, C., Khan, L., and Ponnusamy, A. (2002b). Image classification using neural networks and ontologies. In *Proceedings. 13th International Workshop on Database and Expert Systems Applications*, pages 98–102.

[Cardot and Romuald, 2011] Cardot, H. and Romuald, B. (2011). *Recurrent Neural Networks for Temporal Data Processing*. IntechOpen.

[Chandra and Singh, 2004] Chandra, P. and Singh, Y. (2004). An activation function adapting training algorithm for sigmoidal feedforward networks. *Neurocomputing*, 61:429–437.

[Chang and Chen, 2015] Chang, J.-R. and Chen, Y.-S. (2015). Batch-normalized maxout network in network. *arXiv preprint arXiv:1511.02583*.

[Chang and Huang, 2008] Chang, Y.-H. and Huang, H.-Y. (2008). An automatic document classifier system based on naive bayes classifier and ontology. In *Machine Learning and Cybernetics, 2008 International Conference on*, volume 6, pages 3144–3149. IEEE.

[Chen and Chang, 1996] Chen, C.-T. and Chang, W.-D. (1996). A feedforward neural network with function shape autotuning. *Neural networks*, 9(4):627–641.

[Chen, 1990] Chen, F.-C. (1990). Back-propagation neural networks for nonlinear self-tuning adaptive control. *IEEE control systems Magazine*, 10(3):44–48.

[Chen et al., 2018] Chen, Y., Song, S., Li, S., Yang, L., and Wu, C. (2018). Domain space transfer extreme learning machine for domain adaptation. *IEEE Transactions on Cybernetics*.

[Clevert et al., 2015] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289.

[Cord and Cunningham, 2008] Cord, M. and Cunningham, P. (2008). *Machine learning techniques for multimedia: case studies on organization and retrieval*. Springer Science & Business Media.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[Costa, 2005] Costa, P. C. G. D. (2005). *Bayesian Semantics for the Semantic Web*. PhD thesis, George Mason University, Fairfax, VA, USA. AAI3179141.

[Cover and Hart, 1967] Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.

[Cover, 1965] Cover, T. M. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*, 3(EC-14):326–334.

[Csurka et al., 2004] Csurka, G., Dance, C. R., Fan, L., Willamowski, J., and Bray, C. (2004). Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22.

[Cybenko, 1988] Cybenko, G. (1988). Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *CVPR09*.

[Devaux and Lamanna, 2009] Devaux, M. and Lamanna, M. (2009). The rise and early history of the term ontology (1606-1730). *Quaestio*, 9:173–208.

[Dheeru and Karra Taniskidou, 2017] Dheeru, D. and Karra Taniskidou, E. (2017). UCI machine learning repository.

[Ding and Peng, 2004] Ding, Z. and Peng, Y. (2004). A probabilistic extension to ontology language owl. In *Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 4 - Volume 4*, HICSS '04, pages 40111.1–.

[Dozat, 2016] Dozat, T. (2016). Incorporating nesterov momentum into adam. In *ICLR*.

[Dubois, 1980] Dubois, D. J. (1980). *Fuzzy sets and systems: theory and applications*, volume 144. Academic press.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

[Dugas et al., 2000] Dugas, C., Bengio, Y., BÃ©lisle, F., Nadeau, C., and Garcia, R. (2000). Incorporating second-order functional knowledge for better option pricing. In *NIPS*, pages 472–478.

[Dumoulin and Visin, 2016] Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

[Eisenach et al., 2016] Eisenach, C., Wang, Z., and Liu, H. (2016). Nonparametrically learning activation functions in deep neural nets. In *online available*.

[Elfwing et al., 2018] Elfwing, S., Uchibe, E., and Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*.

[Erhan et al., 2010] Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.*, 11:625–660.

[Ertuğrul, 2018] Ertuğrul, Ö. F. (2018). A novel type of activation function in artificial neural networks: Trained activation function. *Neural Networks*, 99:148–157.

[Eweiwi et al., 2015] Eweiwi, A., Cheema, M. S., and Bauckhage, C. (2015). Action recognition in still images by learning spatial interest regions from videos. *Pattern Recognition Letters*, 51:8 – 15.

[Fan et al., 2008] Fan, J., Gao, Y., and Luo, H. (2008). Integrating concept ontology and multitask learning to achieve more effective classifier training for multilevel image annotation. *IEEE Transactions on Image Processing*, 17(3):407–426.

[Fan et al., 2007] Fan, J., Luo, H., Gao, Y., and Jain, R. (2007). Incorporating concept ontology for hierarchical video classification, annotation, and visualization. *IEEE Transactions on Multimedia*, 9(5):939–957.

[Fechner, 1966] Fechner, G. (1966). *Elements of psychophysics*. New York, Holt, Rinehart and Winston.

[Fisher, 1936] Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188.

[Flach, 2012] Flach, P. (2012). *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press.

[Galántai, 2000] Galántai, A. (2000). The theory of newton's method. *Journal of Computational and Applied Mathematics*, 124(1):25 – 44. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

[Geisser, 1975] Geisser, S. (1975). The predictive sample reuse method with applications. *Journal of the American statistical Association*, 70(350):320–328.

[Genesereth and Nilsson, 1987] Genesereth, M. R. and Nilsson, N. J. (1987). *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, volume 9, pages 249–256.

[Glorot et al., 2011] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323.

[Goh and Mandic, 2003] Goh, S. L. and Mandic, D. P. (2003). Recurrent neural networks with trainable amplitude of activation functions. *Neural Networks*, 16(8):1095 – 1100.

[Goodfellow et al., 2013] Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1319–1327.

[Gruber, 1995] Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(5):907 – 928.

[Guarino, 1995] Guarino, N. (1995). Formal ontology, conceptual analysis and knowledge representation. *International journal of human-computer studies*, 43(5-6):625–640.

[Guarino et al., 2009] Guarino, N., Oberle, D., and Staab, S. (2009). What is an ontology? In *Handbook on ontologies*, pages 1–17. Springer.

[Guarnieri, 1995] Guarnieri, S. (1995). Multilayer neural networks with adaptive spline-based activation functions. *Proceedings of Word Congress on Neural Networks WCNN'95, Washington, DC, July*, pages 17–21.

[Hagan et al., 1996] Hagan, M. T., Demuth, H. B., and Beale, M. (1996). *Neural Network Design*. PWS Publishing Co., Boston, MA, USA.

[Hahnloser et al., 2000] Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., and Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947.

[Harmon and Klabjan, 2017] Harmon, M. and Klabjan, D. (2017). Activation ensembles for deep neural networks. *arXiv preprint arXiv:1702.07790*.

[Hastie et al., 2009] Hastie, T. J., Tibshirani, R. J., and Friedman, J. J. H. (2009). *The elements of statistical learning: data mining, inference and prediction*. Springer Series in Statistics. Springer-Verlag, second edition.

[Haykin, 1994] Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

[He and Sun, 2015] He, K. and Sun, J. (2015). Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5353–5360.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

[Hollink et al., 2003] Hollink, L., Schreiber, G., Wielemaker, J., and Wielinga, B. (2003). Semantic annotation of image collections. In *In Workshop on Knowledge Markup and Semantic Annotation, KCAP'03, 2003*, pages 0–3.

[Hornik, 1991] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366.

[Hu, 1992] Hu, Z., S. H. (1992). The study of neural network adaptive control systems. *Control and Decision*, 7(2):361–366.

[Huang, 2015] Huang, G.-B. (2015). What are extreme learning machines? filling the gap between frank rosenblatt's dream and john von neumann's puzzle. *Cognitive Computation*, 7(3):263–278.

[Hudelot et al., 2008] Hudelot, C., Atif, J., and Bloch, I. (2008). Fuzzy spatial relation ontology for image interpretation. *Fuzzy Sets and Systems*, 159(15):1929 – 1951.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[Jackson, 1998] Jackson, P. (1998). *Introduction to expert systems*. Addison-Wesley Longman Publishing Co., Inc.

[Jin et al., 2016] Jin, X., Xu, C., Feng, J., Wei, Y., Xiong, J., and Yan, S. (2016). Deep learning with s-shaped rectified linear activation units. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 1737–1743. AAAI Press.

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

[Klinov and Parsia, 2008] Klinov, P. and Parsia, B. (2008). Pronto: Probabilistic ontological modeling in the semantic web. In *Proceedings of the 2007 International Conference on Posters and Demonstrations-Volume 401*, pages 82–83. CEUR-WS. org.

[Klinov and Parsia, 2013] Klinov, P. and Parsia, B. (2013). *Uncertainty Reasoning for the Semantic Web II: International Workshops URSW 2008-2010 Held at ISWC and UniDL 2010 Held at FLoC, Revised Selected Papers*, chapter Pronto: A Practical Probabilistic Description Logic Reasoner, pages 59–79. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Krizhevsky and Hinton, 2009] Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. In *Computer Science Department, University of Toronto, Tech. Rep*, volume 1.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA. Curran Associates Inc.

[Kuhn and Tucker, 2014] Kuhn, H. W. and Tucker, A. W. (2014). Nonlinear programming. In *Traces and emergence of nonlinear programming*, pages 247–258. Springer.

[Laskey, 2008] Laskey, K. B. (2008). Mebn: A language for first-order bayesian knowledge bases. *Artificial Intelligence*, 172(2):140 – 178.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

[LeCun and Cortes, 2010] LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/.

[LeCun et al., 1999] LeCun, Y., Haffner, P., Bottou, L., and Bengio, Y. (1999). Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, pages 319–, London, UK, UK. Springer-Verlag.

[Leibniz, 1890] Leibniz, G. (1890). Dissertatio de arte combinatoria. *Leipzig (1666)*.

[Leshno et al., 1993] Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867.

[Li et al., 2013] Li, B., Li, Y., and Rong, X. (2013). The extreme learning machine learning algorithm with tunable activation function. *Neural Computing and Applications*, 22(3-4):531–539.

[Lin et al., 2013] Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *CoRR*, abs/1312.4400.

[Lindsay et al., 1980] Lindsay, R. K., Buchanan, B. G., Feigenbaum, E. A., and Lederberg, J. (1980). Applications of artificial intelligence for organic chemistry: the dendral project. *New York*.

[Linnainmaa, 1970] Linnainmaa, S. (1970). The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's thesis, Univ. Helsinki.

[Linnainmaa, 1976] Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160.

[Lipton, 2015] Lipton, Z. C. (2015). A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019.

[Liu and Yao, 1996] Liu, Y. and Yao, X. (1996). Evolutionary design of artificial neural networks with different nodes. In *International Conference on Evolutionary Computation*, pages 670–675.

[Lucas et al., 2013] Lucas, D. D., Klein, R., Tannahill, J., Ivanova, D., Brandon, S., Domyancic, D., and Zhang, Y. (2013). Failure analysis of parameter-induced simulation crashes in climate models. *Geoscientific Model Development*, 6(4):1157–1171.

[Luenberger and Ye, 2015] Luenberger, D. G. and Ye, Y. (2015). *Linear and Nonlinear Programming*. Springer Publishing Company, Incorporated.

[Maas et al., 2013] Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*.

[Mansouri et al., 2013] Mansouri, K., Ringsted, T., Ballabio, D., Todeschini, R., and Consonni, V. (2013). Quantitative structure-activity relationship models for ready biodegradability of chemicals. *Journal of Chemical Information and Modeling*, 53(4):867–878.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[Memisevic et al., 2014] Memisevic, R., Konda, K. R., and Krueger, D. (2014). Zero-bias autoencoders and the benefits of co-adapting features. *CoRR*, abs/1402.3337.

[Mezaris et al., 2003] Mezaris, V., Kompatsiaris, I., and Strintzis, M. G. (2003). An ontology approach to object-based image retrieval. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, volume 2, pages II–511–14 vol.3.

[Mill, 1884] Mill, J. S. (1884). *A system of logic, ratiocinative and inductive: Being a connected view of the principles of evidence and the methods of scientific investigation*, volume 1. Longmans, green, and Company.

[Minsky and Papert, 1969] Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA.

[Møller, 1993] Møller, M. F. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–533.

[Mosteller and Tukey, 1968] Mosteller, F. and Tukey, J. (1968). Data analysis, including statistics. In Lindzey, G. and Aronson, E., editors, *Revised Handbook of Social Psychology*, volume 2, pages 80–203. Addison Wesley.

[Musen, 2015] Musen, M. A. (2015). The protégé project: a look back and a look forward. *AI Matters*, 1(4):4–12.

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

[Nesterov, 1983] Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. Akad. Nauk SSSR*, volume 269, pages 543–547.

[Nocedal and Wright, 2006] Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer, New York, NY, USA, second edition.

[Noy and Mcguinness, 2001] Noy, N. F. and Mcguinness, D. L. (2001). Ontology development 101: A guide to creating your first ontology. Technical report, Stanford knowledge systems laboratory, Stanford medical informatics.

[P. Devroye and Wagner, 1979] P. Devroye, L. and Wagner, T. (1979). Distribution-free performance bounds for potential function rules. In *Information Theory, IEEE Transactions on*, volume IT-25, pages 601 – 604.

[Pan and Yang, 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359.

[Pang et al., 2016] Pang, Y., Sun, M., Jiang, X., and Li, X. (2016). Convolution in convolution for network in network. *CoRR*, abs/1603.06759.

[Peckhaus, 2010] Peckhaus, V. (2010). Leibniz's influence on 19th century logic. *Stanford Encyclopedia of Philosophy*.

[Pedamonti, 2018] Pedamonti, D. (2018). Comparison of non-linear activation functions for deep neural networks on mnist classification task. *CoRR*, abs/1804.02763.

[Perez and Wang, 2017] Perez, L. and Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*.

[Piazza et al., 1992] Piazza, F., Uncini, A., and Zenobi, M. (1992). Artificial neural networks with adaptive polynomial activation function. In *Proc. of the IJCNN*, volume 2, pages 343–349. Citeseer.

[Piazza et al., 1993] Piazza, F., Uncini, A., and Zenobi, M. (1993). Neural networks with digital lut activation functions. In *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, volume 2, pages 1401–1404 vol.2.

[Picard and Cook, 1984] Picard, R. R. and Cook, R. D. (1984). Cross-validation of regression models. *Journal of the American Statistical Association*, 79(387):575–583.

[Pinkus, 1999] Pinkus, A. (1999). Approximation theory of the mlp model in neural networks. *Acta numerica*, 8:143–195.

[Polyak, 1964] Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

[Qian et al., 2018] Qian, S., Liu, H., Liu, C., Wu, S., and Wong, H.-S. (2018). Adaptive activation functions in convolutional neural networks. *Neurocomputing*, 272:204–212.

[Quattoni and Torralba, 2009] Quattoni, A. and Torralba, A. (2009). Recognizing indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 413–420.

[Ramachandran et al., 2017] Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for activation functions. *CoRR*, abs/1710.05941.

[Reddi et al., 2018] Reddi, S. J., Kale, S., and Kumar, S. (2018). On the convergence of adam and beyond. In *ICLR 2018*.

[Reiter, 1987] Reiter, R. (1987). *Readings in Nonmonotonic Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Riedmiller and Braun, 1992] Riedmiller, M. and Braun, H. (1992). Rprop - a fast adaptive learning algorithm. Technical report, Proc. of ISCIS VII), Universitat.

[Ripley, 2007] Ripley, B. D. (2007). *Pattern recognition and neural networks*. Cambridge university press.

[Robbins and Monro, 1951] Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407.

[Rockafellar, 1993] Rockafellar, R. T. (1993). Lagrange multipliers and optimality. *SIAM review*, 35(2):183–238.

[Rosenblatt, 1957] Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton*. Cornell Aeronautical Laboratory.

[Rosenblatt, 1961] Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, CORNELL AERONAUTICAL LAB INC BUFFALO NY.

[Rosenthal, 2006] Rosenthal, J. S. (2006). *A first look at rigorous probability theory*. World Scientific Publishing Company.

[Ruder, 2016] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533.

[Rutishauser, 1959] Rutishauser, H. (1959). Theory of gradient methods. In *Refined iterative methods for computation of the solution and the eigenvalues of self-adjoint boundary value problems*, pages 24–49. Springer.

[Samuel, 1959] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229.

[Sarwar et al., 2013] Sarwar, S., Qayyum, Z. U., and Majeed, S. (2013). Ontology based image retrieval framework using qualitative semantic image descriptions. *Procedia Computer Science*, 22:285 – 294.

[Scardapane et al., 2016] Scardapane, S., Scarpiniti, M., Comminiello, D., and Uncini, A. (2016). Learning activation functions from data using cubic spline interpolation. *CoRR*, abs/1605.05509.

[Schmid, 1999] Schmid, C. (1999). A structured probabilistic model for recognition. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2, page 490 Vol. 2.

[Schneiderman and Kanade, 1998] Schneiderman, H. and Kanade, T. (1998). Probabilistic modeling of local appearance and spatial relationships for object recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 45–. IEEE Computer Society.

[Scholkopf and Smola, 2001] Scholkopf, B. and Smola, A. J. (2001). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.

[Shao, 1993] Shao, J. (1993). Linear model selection by cross-validation. *Journal of the American statistical Association*, 88(422):486–494.

[Sikora and Wróbel, 2010] Sikora, M. and Wróbel, Ł. (2010). Application of rule induction algorithms for analysis of data collected by seismic hazard monitoring systems in coal mines. *Archives of Mining Sciences*, 55(1):91–114.

[Singh and Chandra, 2003] Singh, Y. and Chandra, P. (2003). A class+ 1 sigmoidal activation functions for ffanns. *Journal of Economic Dynamics and Control*, 28(1):183–187.

[Sir et al., 2015] Sir, M., Bradac, Z., and Fiedler, P. (2015). Ontology versus database. *IFAC-PapersOnLine*, 48(4):220–225.

[Sivic and Zisserman, 2003] Sivic, J. and Zisserman, A. (2003). Video google: A text retrieval approach to object matching in videos. In *Proceedings of the Ninth IEEE International Conference on Computer Vision-Volume 2*, page 1470. IEEE Computer Society.

[Smith, 1983] Smith, A. R. (1983). Spline tutorial notes. *Computer Graphics Project, Lucasfilm Ltd (presented as tutorial notes at the 1983 and 1984 SIGGRAPH)*.

[Smith, 2015] Smith, L. N. (2015). Cyclical Learning Rates for Training Neural Networks. *ArXiv e-prints*.

[Sonoda and Murata, 2017] Sonoda, S. and Murata, N. (2017). Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, 43(2):233 – 268.

[Springenberg and Riedmiller, 2013] Springenberg, J. T. and Riedmiller, M. A. (2013). Improving deep neural networks with probabilistic maxout units. *CoRR*, abs/1312.6116.

[Spyns et al., 2002] Spyns, P., Meersman, R., and Jarrar, M. (2002). Data modelling versus ontology engineering. *ACM SIGMod Record*, 31(4):12–17.

[Stevens, 1957] Stevens, S. S. (1957). On the psychophysical law. *Psychological review*, 64(3):153.

[Stinchcombe and White, 1990] Stinchcombe, M. and White, H. (1990). Approximating and learning unknown mappings using multilayer feedforward networks with bounded weights. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 7–16. IEEE.

[Stone, 1974] Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society. Series B (Methodological)*, pages 111–147.

[Sun et al., 2018] Sun, W., Su, F., and Wang, L. (2018). Improving deep neural networks with multi-layer maxout networks and a novel initialization method. *Neurocomputing*, 278:34 – 40.

[Sussillo and Abbott, 2014] Sussillo, D. and Abbott, L. F. (2014). Random Walk Initialization for Training Very Deep Feedforward Networks. *CoRR*.

[Sutskever et al., 2013] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147.

[Tieleman and Hinton, 2012] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

[Toussaint, 1978] Toussaint, G. (1978). The use of context in pattern recognition. *Pattern Recognition*, 10(3):189–204.

[Trottier et al., 2017] Trottier, L., Gigu, P., Chaib-draa, B., et al. (2017). Parametric exponential linear unit for deep convolutional neural networks. In *Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on*, pages 207–214. IEEE.

[Turing, 1950] Turing, A. M. (1950). I.- computing machinery and intelligence. *Mind*, LIX(236):433–460.

[Urban et al., 2017] Urban, S., Basalla, M., and van der Smagt, P. (2017). Gaussian process neurons learn stochastic activation functions. *CoRR*, abs/1711.11059.

[Vapnik, 1992] Vapnik, V. (1992). Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838.

[Vapnik and Lerner, 1963] Vapnik, V. and Lerner, A. (1963). Pattern Recognition using Generalized Portrait Method. *Automation and Remote Control*, 24.

[Vecchia and Splett, 1994] Vecchia, D. and Splett, J. (1994). Outlier-resistant methods for estimation and model fitting. *ISA Transactions*, 33(4):411 – 420.

[Vecci et al., 1998] Vecci, L., Piazza, F., and Uncini, A. (1998). Learning and approximation capabilities of adaptive spline activation function neural networks. *Neural Networks*, 11(2):259–270.

[Vogl et al., 1988] Vogl, T. P., Mangis, J. K., Rigler, A. K., Zink, W. T., and Alkon, D. L. (1988). Accelerating the convergence of the back-propagation method. *Biological Cybernetics*, 59(4):257–263.

[Wang et al., 2013] Wang, M., Gao, Y., Lu, K., and Rui, Y. (2013). View-based discriminative probabilistic modeling for 3d object retrieval and recognition. *Trans. Img. Proc.*, 22(4):1395–1407.

[Wang et al., 2018] Wang, M., Liu, B., and Foroosh, H. (2018). Look-up table unit activation function for deep convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1225–1233.

[Widrow and Hoff, 1960] Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs.

[Widrow and Lehr, 1990] Widrow, B. and Lehr, M. A. (1990). 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442.

[Wiener, 1948] Wiener, N. (1948). Time, communication, and the nervous system. *Annals of the New York Academy of Sciences*, 50(1):197–220.

[Wolstencroft et al., 2006] Wolstencroft, K., Lord, P., Tabernero, L., Brass, A., and Stevens, R. (2006). Protein classification using ontology classification. *Bioinformatics*, 22(14):e530–e538.

[Wu et al., 2004] Wu, Y., Tseng, B. L., and Smith, J. R. (2004). Ontology-based multi-classification learning for video concept detection. In *Multimedia and Expo, 2004. ICME'04. 2004 IEEE International Conference on*, volume 2, pages 1003–1006. IEEE.

[Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747.

[Xu et al., 2015] Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

[Yamada and Yabuta, 1992a] Yamada, T. and Yabuta, T. (1992a). Neural network controller using autotuning method for nonlinear functions. *IEEE Transactions on Neural Networks*, 3(4):595–601.

[Yamada and Yabuta, 1992b] Yamada, T. and Yabuta, T. (1992b). Remarks on a neural network controller which uses an auto-tuning method for nonlinear functions. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 2, pages 775–780 vol.2.

[Yao, 1999] Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87:1423–1447.

[Zeiler, 2012] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

[Zhang et al., 2014] Zhang, L., Yang, Y., Gao, Y., Yu, Y., Wang, C., and Li, X. (2014). A probabilistic associative model for segmenting weakly supervised images. *IEEE Trans. Image Processing*, 23(9):4150–4159.

[Zhang et al., 2005] Zhang, R., Zhang, Z., Li, M., Ma, W.-Y., and Zhang, H.-J. (2005). A probabilistic semantic model for image annotation and multimodal image retrieval. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 846–851 Vol. 1.

[Zouari et al., 2004] Zouari, H., Heutte, L., and Lecourtier, Y. (2004). Simulating classifier ensembles of fixed diversity for studying plurality voting performance. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 1, pages 232–235 Vol.1.