UNIVERSITAT POLITECNICA DE CATALUNYA

DEPARTAMENT DE LLENGUATGES I SISTEMES INFORMATICS

PROGRAMA DE DOCTORAT EN INTEL·LIGENCIA ARTIFICIAL

TESI DOCTORAL

Heterogeneous Neural Networks: Theory and Applications

June 2000

Memòria presentada per en Lluís A.
Belanche Muñoz per a optar al títol
de Doctor en Informàtica.

Directors: Julio José Valdés Ramos i Renato Alquézar Mancho

Als meus pares, per ser
unes meravelloses persones

A la meva germana, dolça alegria

A Sonia, mi femenina conciencia

Abstract

This work presents a class of functions serving as generalized neuron models to be used in artificial neural networks. They are cast into the common framework of computing a *similarity* function, a flexible definition of a neuron as a pattern recognizer. The similarity endows the model with a clear conceptual view and serves as a unification cover for many of the existing neural models. The framework includes those models traditionally used for the MultiLayer Perceptron (MLP) and most of those used in Radial Basis Function Networks (RBF). These families of models are conceptually unified and their relation is clarified. Further, the possibilities of deriving *new* instances are explored and several neuron models –which are representative of their families– are proposed.

The similarity view naturally leads to further extensions of the models so as to handle heterogeneous information. That is to say, information coming from sources radically different in character, including continuous and discrete numerical quantities (ordinal or nominal) and fuzzy quantities. Missing data are also explicitly considered. A neuron of this class is called an *heterogeneous neuron* and any neural structure making use of these neurons is an **Heterogeneous Neural Network** (HNN), regardless of the specific architecture or learning algorithm. Among them, in this work we concentrate on feed-forward networks, as the initial focus of study. The learning procedures may include a great variety of techniques, basically divided into derivative-based methods (such as the conjugate gradient) and evolutionary ones (such as variants of genetic algorithms).

This thesis explores a number of directions towards the construction of better neuron models within an integrant envelope, which are better adapted to the problems they are meant to solve. The main ideas developed are:

- The integration and joint use of concepts previously separated, such as similarity and heterogeneity, in artificial neuron models. There is also the expectancy in a thereby better understanding of these notions.
- A step forward in the unification of traditionally separated neuron models, particularly those used in MLP and RBF networks.
- The derivation of working instances of the generic model and their validation by extensive experimentation in a number of benchmarks. The practical concern of the approach is highlighted by experiments on real-world problems.

This work also describes how a certain generic class of heterogeneous models leads to a satisfactory performance, which is comparable to, and often better than, that of traditional neural models. This is particularly so in the presence of heterogeneous information and imprecise or incomplete data, and in a wide range of domains, most of them corresponding to real-world problems.

Further, a proof for the universal approximation ability of some of the introduced classes of neuron models is developed. This property has already been proved for standard neuron models. The generic extent of the neuron models potentially included in the framework prevents the existence of a general proof. Instead, dedicated demonstrations are worked out on particular cases or specially interesting models.

Acknowledgements.

Work is due to the efforts of people and I am indebted to many people for their sincere help in getting through with the research work herein reported. I list them in a random order, as they come to mind.

My first thanks and cheers go to Ignasi R. Roda and Quim Comas, members of the Environmental and Chemical Engineering group of the University of Girona. We have shared many good times, even in moments where we were busy in our everyday tasks, and they always found excuses to show their sense of humour. Their skill and availability are very much appreciated.

I am specially thankful to Enrique Romero, whose patience and cordiality endured to the last minute of endless discussions on integrability and other less romantical aspects of mathematics. His comments shed a clarity to some concepts, that cannot be found on textbooks and I am indebted to him for this.

I would also like to have a word of gratitude to Javier Béjar, friend of mine since the old career days, whom I have consulted about hundreds of little matters. He always seems to have an answer or a pointer to one.

As usual in these cases, there was somebody who did their best in ironing the English. Fortunately for them, they only had to check part of the document. I am thankful to Victoria Arranz and to an anonymous (for me) native speaker for their contributions.

My deepest thanks and best wishes go to my two advisors, Julio Valdés and René Alquézar. The opportunity of sharing with them so many moments, both for personal or scientific matters, has shaped my way of being for the good.

A special mention is to Lluís Valentín, my oldest friend who still is, and a very special soul.

Financial support by the Spanish CICYT grants AMB97-0889 and TAP99-0747 is gratefully acknowledged.

Contents

Chapter 1

Introduction

This thesis deals with the development of new neuron models to be used as the basic computing elements in **Artificial Neural Networks** (ANN). These networks are information processing structures evolved as an abstraction of known or assumed principles of how the brain might work. The different computing elements, called *neurons* or *units*, are linked to one another with a certain strength, called *weight*. In their simplest form, each unit computes a *function* of its inputs —which are either the outputs from other units or external signals— influenced by the weights of the links conveying these inputs.

The network is said to *learn* when, as a result of exposure to samples of a target function, the weights of all of its units are adapted to represent the information present in the samples, in an optimal sense to be precised. When exposed to such a *supervised training* process, the network builds an internal representation of the target function by combining certain parameterized base functions (PBF), either forming local models, as in Radial Basis Function networks (RBF), or making up a global model, as in the MultiLayer Perceptron (MLP). In both cases, the network relies upon the representation capacity of the PBF (that is, of the neuron model) as the cornerstone for a good approximation.

Research in ANN currently concerns the development of *learning algorithms* for weight adaptation or, more often, the enhancement of existing ones. New *architectures* (ways of arranging the units in the network) are also introduced from time to time. Classical *neuron models*, although useful and effective, are lessened to a few generic function classes, of which only a handful of instances are used in practice. In any case, biological plausibility is usually put aside or is simply of no importance, the emphasis being put on problem solving.

The strong points of ANN are their appealing capacity to learn from examples, their distributed computation —which helps them tolerate partial failures to a certain extent— and the possibility —so often exploited— to use them as black-box models, with practically no assumption or knowledge about the problem.

This last characteristic is paradoxically one of the major weaknesses, given that in practice this autonomy of functioning acts as a requirement or precept, with no transfer of knowledge from the designer. Therefore, with the exception of very specific architectures, the networks are forced to learn from scratch most of the times.

This drawback is related to its counterpart, namely, the extreme difficulty of *extracting* significant knowledge from a trained network. The network *works* (in the sense that solves a problem to satisfaction) but the weights convey information about the solution as high-dimensional real vectors whose meaning about the problem can be intricate or at best obscure. An additional practical obstacle is the process of finding the most adequate architecture for a given task.

The possibility of expressing prior information in a *procedural* —devising a specific neuron model— or *declarative* way —telling the network about the data types, is then a means to pave the way towards more powerful and expressive neuron models and networks.

1.1 Motivation

1.1.1 Similarity in Artificial Neural Networks

Among the several neuron models in the literature, the majority of them carry out a decomposition of the whole work in two functions: the first function *combines* the inputs with the weights (coming both in vector form) to produce a scalar value that is then non-linearly treated by the second one.

The *combination function* is the core of the unit and is meant to evaluate the likeness of two vectors (input and weight) in n-dimensional Euclidean space. By far, the dominant functions are the scalar product and the Euclidean distance (though other distances are also possible). Since the scalar product is a measure of the degree of colinearity between two vectors, the more coincident these vectors are with respect to their direction and orientation, the higher the response. As for the Euclidean distance, the interpretation for the two vectors corresponds to a unit whose response is higher the lower the norm of their difference.

These two functions represent means to compute a very crude measure of the *similarity* between the two vectors. The problem with this approach is that these measures are *fixed* a priori to compute a very specific *physical* similarity, which regards the input variables (and the weights) as the points in Euclidean space, given by a cartesian product of n-copies of the reals. Therefore, patterns $\vec{x}$ that are similar to a neuron i with weight vector $\vec{w}^i$ —because the inner product $\vec{x} \cdot \vec{w}^i$ is high or $||\vec{x} - \vec{w}^i||$ is low[1]— do not need to be similar at all in the eyes of the user. The learning task consists precisely in this: showing the network what is the actual similarity relation. In consequence, very often this measure may not correspond to the task-dependent similarity we would like the network to perform.

In this vein, a clear shortcoming of the neuron models existent in the literature is the difficulty of adding knowledge about the problem to be solved (or about the data themselves). The neuron model is used as a black-box function which maps input data in *raw form*, from a space of possibly high dimension, to a new space (the *hidden space*, or space spanned by the hidden units) in such a way that the problem is simpler when projected to this new space. The internal workings of the neuron are thus obscure, because the weights have not been set so as to shape a previously defined (and considered adequate) similarity measure, but

[1]These quantities are functionally related for normalized vectors.

rather to adapt a general physical measure to the problem at hand. In any case, the task for such a blind processor is twofold: finding a structure in the data, as well as more convenient relations for the problem *given* the data representation.

All the burden of work devoted to find the appropriate transformations, so that patterns requiring similar responses (for the problem) are indeed similar for the network, is performed, in theory, by the hidden layer(s) in multilayer networks. During the learning process, thus, patterns seen as physically similar have to be told apart and vice versa. On the one hand, there is the task of discovering the relations or structure induced by the chosen coding scheme. On the other, the network has to find ways to accommodate the desired similarity relationship —which is inherent in the examples with which it is being taught— to its fixed similarity measure computation.

In practice, however, several layers may be needed for complex transformations, or a large amount of neurons per layer if we restrict the maximum number of hidden layers, for instance, to one or two, as is common proceeding. An increase in neurons leads to a corresponding growth in the number of free parameters, and these are less likely to be properly constrained by a limited size data set [Bishop, 95]. This gives support to a deeper and more precise formulation of neuron models as similarity computing devices.

1.1.2 Heterogeneity in Artificial Neural Networks

The subject of prior knowledge is strongly related to input representation. Real-world data come from many different sources, described by mixtures of numeric and qualitative variables. These variables include continuous or discrete numerical processes, symbolic information, etc. In particular, qualitative variables might have a different nature. Some are *ordinal* in the usual statistical sense (i.e., with a discrete domain composed by k categories, but totally ordered w.r.t a given relation) or *nominal* (discrete but without an ordering relation). The data also come with their own peculiarities (vagueness, uncertainty, incompleteness), and thus may require completely different treatments.

This *heterogeneity* is traditionally handled by preparing the data using a number of coding methods, so that all variables are to be treated as real quantities. However, this pre-processing is *not* part of the original task and may have deep consequences in the structure of the problem. These consequences could be, for example, a change in input distribution and an increase in dimension, with the corresponding growth in the number of weights the network is forced to learn. This, in turn, increments the training time, eventually leading to more complex error surfaces, etc.

In conclusion, when solving any practical problem, the fact of having the network learn what is known in advance is a wasteful and added effort [Hinton, 89]. Nonetheless, as it has been seen, the ways this information can be encoded in the ANN workings are very limited. The difficulty of the general ANN learning task is therefore exacerbated because the data (and with them, the problem itself) are adapted to the neuron model (and, possibly, to the learning algorithm), and not otherwise. Alternatively, we believe *it is the model* that has to conform to the data, by making use of domain or *prior* knowledge (traditionally known in inductive learning as *background knowledge*). This knowledge can be roughly defined as the information possessed about the problem itself *and* about the nature of its data, besides the raw examples themselves. If this information (which is assumed relevant) is supplied to a neural network or included in its design, it will not need to be discovered.

1.2 Aims and scope

1.2.1 Statement of objectives

In broad terms, our work is devoted to the development of general families of neuron models cast in the framework of similarity computing devices. Moreover, data heterogeneity is explicitly taken into account by devising specific measures that can be used to extend the models, thereby expanding $\mathbb{R}^n$ to generic input spaces. The main aim is to provide a conceptual guidance and to extend the richness and expressiveness of the function computed by artificial neurons.

The resulting general models are referred to as *heterogeneous neurons* and any architecture making use of them as **Heterogeneous Neural Network** or HNN. Among the benefits of using a similarity measure are its bounded nature (preventing arbitrarily large values) and the possibility of capturing higher-order relationships in the data.

A wider statement of the objectives is the following:

- The construction of better neuron models within an integrant framework, and better adapted to the problems they are meant to solve.
- The integration and joint use of concepts previously separated, as similarity and heterogeneity in artificial neuron models. There is also the expectancy in a thereby better understanding of these notions.
- A step forward in the unification of traditionally separated neuron models, particularly those used in MLP and RBF networks.
- The derivation of working instances of the generic model and their validation by experimentation in a number of benchmarks.
- Last but not least, the practical use of these neuron models in real-world problems and applications is one of the main objectives of the work.

Learning algorithms based on derivative information (like the Back-propagation scheme [Rumelhart, Hinton and Williams, 86], see §2.1.6 in this thesis), depend on the differentiability of the function computed by the neural network for their application. Besides this, the fact of having a *smooth* unit is essential for the application of gradient-descent techniques [Sontag, 90], which restricts their applicability.

In our case, there is a need for a generic training procedure, due to the existence of domains other than the real continuum, the presence of missing data and the eventual use of a non-differentiable similarity function. The one chosen in this work for the HNN is based on *Evolutionary Algorithms* (EA) [Bäck, 96]. Traditionally, only *Genetic Algorithms* (GA) [Goldberg, 89], [Davis, 91] have been extensively used for this task. In our work, both a GA and a *Breeder Genetic Algorithm* (BGA) [Mühlenbein and Schlierkamp-Voosen, 93] are used. A further objective is the investigation of these methods as neural network trainers and their enhancement in a number of ways.

1.2.2 Scope of the Thesis

The work is bounded by the following decisions:

1. We have limited the practical extent of the experiments to supervised learning in feed-forward architectures, due to the richness and variety of neuron models to be studied and in order to confine the work within definite bounds.

2. In relation to this, it should be pointed out that the experiments show only a limited attempt to perform a model selection process (looking for a model of *optimal* complexity), because the primary aim has been to compare the *relative* performance of the different models tested, by having them compared in different architectures and experimental settings. Therefore, an exploration on the architectural space in search of the best architecture for every problem has not been performed. In fact, such a principled search is a subject of research in itself [Moody, 94]. Hence, the results on standard benchmarks should not be taken as state-of-the-art competitors (although they are not out of tune) but rather as developed examples of application.

3. We have considered the inclusion of similarity measures over the most commonly encountered data types (such as real, nominal, ordinal, sets and forms of fuzzy information). Nonetheless, other less usual types could also be included. Distances defined on trees, strings or graphs are found in the literature of various fields (see, for example, [Honavar, 92]). The derived similarities could be integrated with no modification.

4. Despite giving the initial impetus to the field, research in the ANN paradigm is no longer guided by biological plausibility. Rather, it is guided by effectiveness and efficiency in learning and representation, and oriented towards problem solving. These are also our motivations.

5. The experiments in this work are confined to a few generic families of new models. Nevertheless, the general framework allows and shows ways to construct many other classes of generic neuron models. In this sense, given the three "design topics" in ANN (neuron model, architecture and learning algorithm) the proposed approach is useful for any conceivable architecture and is independent of the algorithm, in the sense that it does not assume any specific choice for these two settings.

1.3 Contributions

1.3.1 A framework for similarity

A comprehensive framework is proposed where the notion of similarity is characterized in the context of ANN. The concepts of similarity, dissimilarity and distance in a certain space are formalized following their classical definitions [Chandon and Pinson, 81]. From these, conditions for aggregating and composing partial measures are introduced. The obtained aggregation operator can take semantic information into account and missing information is explicitly considered. The framework is comprehensive and flexible, admitting the introduction of non-linearities, before or after the aggregation of partial measures. Any combination is admissible as long as it fulfills the corresponding properties.

A similarity measure for a neuron model can then be constructed in several ways, either designed directly, or from a dissimilarity, or by combination of partial distance-based functions. These functions can be defined in their respective spaces and combined, or else obtained from a global distance in the entire space.

A number of measures to compute the similarity between non-standard data types are compiled, by integration and extension of previously used similarity measures in data analysis. The notion of heterogeneous space is then defined as a cartesian product of single spaces of mixed variables. As a consequence of the definition, heterogeneous similarity measures can be devised in this space, using specific forms for aggregation operators.

The proposed framework is general enough to offer controlled means of designing models that have a certain number of desirable properties. The definition of flexible neuron models in the form of a general framework in which:

1. the computation performed by the model is defined explicitly as a similarity measure;
2. different data sources are directly and specifically treated,

permits a natural extension of existing models, and opens a way for devising new ones under the general cover of similarity computing devices. It also gives a *semantics* to the neuron computation (e.g., a psychological similarity) in contrast to the "syntactic" physical similarity. It is our hypothesis that these models will add to the *practical* computing power by enhancing the expressiveness, by adapting the model to the data.

1.3.2 Similarity-based neuron models

An heterogeneous neuron, or H-neuron, is defined as a mapping from elements in a heterogeneous input space to an also heterogeneous output space. An S-neuron is defined as an H-neuron whose mapping is a similarity measure.

This leads to the development of general neuron models working in heterogeneous spaces. Within this framework, two of the most widely used neuron models (in RBF and MLP networks) are cast as two of the basic ways of constructing a form of similarity measures (in this case, homogeneous). As a particularly useful instance of S-neurons of the *real kind* (that is, models for which the codomain is a subset of the reals), a distinguished generic measure is created based on Gower's similarity index [Gower, 71]. A collection of newly derived models is tested and shown to have a remarkable performance in the presence of heterogeneity and missing values in the data. A comparable performance is displayed in absence of these considerations.

The overall framework carries a number of additional advantages:

1. The definition of a neuron as a similarity measure permits to base the conceptual network similarity computation on principled grounds. As a consequence, it allows a deeper influence on the workings of a neural network, by delimiting the task performed by each of its hidden neurons.

2. It also serves, although this is not a primary objective, as a unification cover for many of the existent neuron models in the literature.

3. It opens the possibility of adding prior knowledge to the design of the neuron model, by exploiting information either on known data singularities or about the problem itself.

1.3.3 Other theoretical and methodological contributions

A proof for the universal approximation ability of some of the introduced families of neuron models is developed. This property has already been proved for standard neuron models (see, for example, [Scarselli and Tsoi, 98] for a review). In our case, the amount and generic extent of the potential neuron models included in the framework prevent a general proof to be attempted. Instead, dedicated demonstrations are worked out on particular cases or specially interesting models.

The Breeder Genetic Algorithm has been investigated in traditional testbed optimization problems and as a neural network trainer. Proposals for its main parameters are made based on extensive experimentation on a difficult benchmark dataset. The algorithm is also enhanced in a number of ways, so as to accept and manipulate heterogeneous variables in its chromosomic material. A modification to be used in standard recombination operators is also introduced, consisting in a simple but effective method to dynamically set one of their parameters.

1.3.4 Practical work

The neuron models derived from the approach have been tested empirically in a variety of situations and experimental conditions. In particular, they have been explored in three general kinds of problems:

- In real-world tasks, using data directly available to the author, and where there was a motivation to apply the ideas developed in the present work.

- In a specific industrial setting, which is the operation and control of WasteWater Treatment Plants (WWTP) [Lean and Hinrichsen, 94]. In addition, a cooperation team has been established with the Environmental and Chemical Engineering group of the University of Girona, whose members signaled several of the current WWTP problems and supplied the necessary amount of data and skill.
- In well-known neural benchmarking databases [Prechelt, 94], displaying various degrees of heterogeneity. These experiments are carried out in a controlled experimental setting. The results are compared to those obtained with standard neuron models or other methods.

As a side outcome, both the GA and BGA can be said to have been extensively tested for the error function optimization problem. They have been found to be competitive when compared to derivative-based methods, not only regarding performance (quality and variability of the results) but also computational time.

1.4 Chronological account

The approach has been developed and validated across a variety of experiments and applications, in successive stages. In [Valdés and García, 97], heterogeneous neuron models and networks were first introduced and shown to be capable of learning from non-trivial data sets. In particular, it is described how a certain class of heterogeneous models leads to an effectiveness that is comparable, and often better, than that of traditional methods, such as k-nearest neighbours. They specially exhibited a remarkable robustness when information progressively degrades due to the increasing presence of missing data.

The basic measure was then enhanced allowing fuzzy sets (in the form of fuzzy numbers) to occur as part of the input and the corresponding weights. The resulting generic model -of which a preliminary working document is [Belanche, 00a]- has been extensively tested and applied to a wide range of domains, most of them corresponding to real-world problems: Medicine ([Belanche and Valdés, 98c], [Belanche, Valdés and Alquézar, 98a]), Environmental Sciences ([Valdés, Belanche and Alquézar, 00], [Belanche and Valdés, 99a]) and Engineering ([Belanche *et al.*, 98b], [Belanche *et al.*, 99c], [Belanche *et al.*, 99b], [Belanche *et al.*, 00]). In all these studies, a general choice of the heterogeneous neuron family of models, based on Gower's similarity index, is shown to enhance performance with respect to the classical neuron models, especially in the presence of heterogeneous information, imprecise or incomplete data.

The initial training method developed was a standard Genetic Algorithm (GA), modified in a number of ways to accept a missing value as a valid allele in the genotype. Since the seminal work of [Montana and Davis, 89], the use of this kind of algorithm for the training of a neural network is a recurrent theme in the literature. However, the difficulty of a truly fair comparison, the lack of thorough studies —except those for specific tasks— and the raised opinions contrary to its general adequacy [Whitley, 95] have prevented a widespread use.

In our studies, we have found that the GA is a fairly acceptable and robust tool for neural network optimization. Nevertheless, a considerable amount of work has been devoted to train the HNN with more classical derivative-based methods, whenever this was possible. In [Nieto, 00], extensive experiments are carried out with several families of distance-based similarity models, in data sets where all the information is real-valued and in the presence of missing information. A general and consistent superiority of these models over scalar product (adapted to handle missing values) is reported. In this case, the training procedure is a combination of conjugate gradient with an annealing schedule [Ackley, 87]. This method has also been the one chosen —besides the GA— to train the classical models in some of the reported results (for example, in [Belanche and Valdés, 98c], [Belanche *et al.*, 98b]).

The basic GA was later replaced by a Breeder Genetic Algorithm (BGA), which is a method in between the GA and Evolution Strategies -see Chapter 6. The BGA is chosen because of its compromise between simplicity and efficiency, and the lack of a coding scheme for the representation of individuals. It has been extensively studied and enhanced [Belanche, 99d], [Belanche, 99e] to pave the way for its use in ANN error minimization [Belanche, 00b]. In addition, its genetic operators have been extended to work with non-standard information, in particular, with recombination and mutation of nominal, ordinal and fuzzy quantities, and with provision for missing values. The heterogeneous BGA is used in the more recent experiments on several widespread benchmarks [Belanche, 00c].

1.5 Overview

The thesis is structured in ten chapters, and four appendices, and is organized as follows:

In Chapter (2), a brief survey on supervised feed-forward neural networks is carried out, emphasizing those points within this vast field that are in contact with our work. To the best of our knowledge, there is no previous work of the kind described here. We thus proceed to trace the touching points with related disciplines and describe connectionist approaches concerning, to a greater or lesser degree, the explicit use of similarity. A mention to fuzzy neural networks is included.

Chapter (3) serves as an informal discussion, where many of the later unfolded ideas are presented. In particular, notes on how current ANN approaches deal with the concepts of similarity and heterogeneity are surveyed and examined with reference to their merits and defects.

Chapter (4) is an important part of the thesis, where the ideas already introduced are set forth and formalized, to slowly construct a generic framework from which working models can be developed. Our attention will be primarily focused on defining the basic concepts within this conceptual framework and explore some elementary properties or advantages.

The section on similarity measures (§4.2) contains the definitions and propositions which constitute the core of the chapter. The basic definitions of *distance* and *similarity* constitute the departure points, and they can be found in classical textbooks on Typological Analysis (our basic reference is [Chandon and Pinson, 81]). More complete descriptions of the various classical measures existent in the literature can also be found therein. The obtained framework is then oriented to be used as a guidance for artificial neural models to be seen as similarity computing devices.

The description of heterogeneous variables is the topic of the next Section (§4.3), ending in the definition of a heterogeneous space, as well as proposals for similarity measures for it. The introduction of a S-neuron (a neuron computing a similarity and working in one such space) is dealt with in Section (§4.4). A worked example is therein developed. Several classes of distance to similarity transforming functions are then displayed, leading to the concepts of model and network sensitivity. Finally, a numerical experiment (§4.5) is carried out to illustrate the different ideas introduced.

Chapter (5) deals with the more theoretical properties of the obtained classes of models. In particular, the universal approximation property is proven or sketched for several subclasses.

In Chapter (6), since there is a need for a generic training procedure, an introduction to Evolutionary Algorithms is carried out, with emphasis on the two used instances: modified versions of a Genetic Algorithm and a Breeder Genetic Algorithm. The latter is specially covered and enhanced, as part of the investigation, for its use as a generic neural network trainer.

The next three Chapters embrace the practical side of the work. To begin with, Chapter (7) focuses on real-world tasks, using data directly available to the author, and where there was a motivation to apply the ideas developed in the work. As a complement, Chapter (9) is devoted to experiments with benchmarking databases, carried out in a controlled experimental setting.

Due to its importance as a social and economical problem, the correct operation and control of WasteWater Treatment Plants was chosen as a case-study application, in cooperation with chemical engineers of the University of Girona. The subject and the undertaken experiments are described in Chapter (8).

The Thesis ends with some reflections and concluding remarks, and a prospect for future work.

The Appendices mainly contain proof for those Propositions and claims of the work. As a general rule, they are developed in the main body whenever they illustrate or give further insight to the exposition. Otherwise they have been relegated to Appendix (B).

Appendix (A) includes additional material on distances, relevant for the discussion but in some sense secondary. Appendix (C) collects some useful properties concerning integrability. Appendix (D) is a general store which deals with all other material. In any case, everything for which a proof is supplied is work and responsibility of the author.

Chapter 2

Related Work

The answer to the theoretical question:
"Can a machine be built capable of doing
what the brain does?" is yes, provided
you specify in a finite and unambiguous way
what the brain does.

Warren S. McCulloch

The class of adaptive systems known as Artificial Neural Networks (ANN) were motivated by the amazing parallel processing capabilities of biological brains (especially the human brain). The main motivation, at least initially, was to re-create these abilities by constructing artificial models of the biological neuron. The actual artificial neurons -as used in the ANN paradigm- have little in common with their biological counterpart. Rather, they are primarily used as *computational devices*, clearly intended to problem solving: optimization, approximation of functions, identification of systems, classification, time-series prediction, and others.

The power of biological neural structures stems from the enormous number of highly interconnected simple units. The simplicity comes from the fact that, once the complex electro-chemical processes are abstracted, the resulting computation turns out to be conceptually very simple. Artificial neurons are modeled to take profit of this by proposing simple computing devices that resemble the abstracted original function. However, in the ANN paradigm only very few elements are connected in most practical situations (on the order of hundreds, to say the most) and their connectivity is low. Therefore, it seems reasonable, once the "biological origin" is so departed, to think in compensating these low numbers by increasing the power of single units, while retaining the conceptual simplicity of seeing a neuron (a computational unit) as a *pattern recognizer*: a device that integrates its incoming sources with its own local information available and outputs a single value expressing how much do they match.

2.1 Artificial Neural Networks

2.1.1 Preliminaries

Artificial Neural Networks [Bishop, 95], [Haykin, 94], [Hertz, Krogh and Palmer, 91], [Hecht-Nielsen, 90] are information processing structures without global or shared memory, where each of the computing elements operates only when all its incoming information is available, a kind of data-flow architectures. Each element is a simple processor with internal and adjustable parameters. The interest in ANN is primarily related to the finding of satisfactory solutions for problems —cast as function approximation tasks— for which there is scarce or null knowledge about the process itself, but a (limited) access to examples of response. They have been widely and most fruitfully used in a variety of applications during the last fifteen years (see [Fiesler and Beale, 97] for a comprehensive review), especially after the boosting seminal works of [Hopfield, 82], [Rumelhart, Hinton and Williams, 86], [Fukushima, 80] and [Kohonen, 88].

The most general form for an ANN is a *directed graph*, where each of the nodes (called *units* or *neurons*) has a certain computing ability and is connected to and from other nodes in the network via labelled edges. The edge label is a real number expressing the strength with which the two involved units are connected. These labels are called *weights*. The *architecture* of a network refers to the number of units, its arrangement and connectivity.

In its basic form, the computation of a unit i is expressed as a function F_i of its input (the *transfer* function), parameterized with its weight vector or local information. The purpose of the transfer function of a given unit is the tesselation of the input space in the most convenient way for the problem at hand. The whole system is thus a collection of interconnected elements, and the transfer function performed by a single one (i.e., the *neuron model*) is the most important fixed characteristic of the system.

There are two basic types of neuron models in the literature used in practice. Both decompose the overall computation of the unit in two stages, as is classically done since the earlier model proposal of McCulloch & Pitts [McCulloch and Pitts, 43], in a characteristic form cast as a composition of two functions:

$$F_i(\vec{x}) = \{g(h(\vec{x}, \vec{w}_i)),\ \vec{w}_i \in \mathbb{R}^n\}, \qquad \vec{x} \in \mathbb{R}^n \tag{2.1}$$

where $\vec{w}_i$ is the weight vector of neuron i, $h : \mathbb{R}^n, \mathbb{R}^n \to \mathbb{R}$ is called the *net input* or *aggregation* function, and $g : \mathbb{R} \to \mathbb{R}$ is called the *activation* function. All neuron parameters are included in its weight vector. Let us denote by S_F the set of such functions.

The choice $h(\vec{x}, \vec{w}_i) = \vec{x} \cdot \vec{w}_i + \theta$, where $\theta \in \mathbb{R}$ is an offset term that may be included in the weights, leads to one of the most widely used neuron models. When neurons of this type are arranged in a feed-forward architecture, the obtained neural network is called MultiLayer Perceptron (MLP) [Rumelhart, Hinton and Williams, 86]. Usually, a smooth non-linear and monotonic function is used as g. Among them, the sigmoidals are a preferred choice.

The selection $h(\vec{x}, \vec{w}_i) = \frac{1}{\theta}||\vec{x} - \vec{w}_i||_2$ (or other distance measure), where $\theta > 0 \in \mathbb{R}$

is a smoothing term, plus an activation g with a monotonically decreasing response from the origin leads to the wide family of localized Radial Basis Function networks (RBF) [Poggio and Girosi, 89]. Localized means that they give a significant response only in a neighbourhood of their centre $\vec{w}_i$. A Gaussian is a usual choice for the g function.

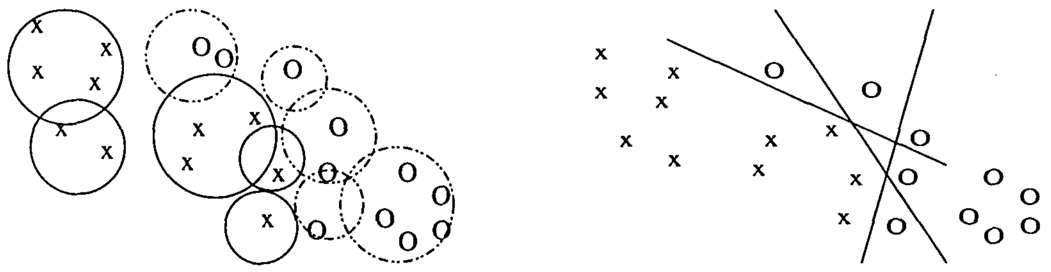


Figure 2.1: A classification problem. Left: separation by spherical RBF units (R-neurons). Right: separation by straight lines (P-neurons) in the MLP.

These two basic neuron models have traditionally been regarded as completely separated, both from a mathematical and a conceptual point of view. To a certain degree, this is true: the local vs. global approximation approaches to a function that they carry out make them apparently quite opposite methods –Fig. (2.1). Besides, though a RBF can be trained like a MLP, with a derivative-based technique, the learning algorithms usually applied are quite different in nature [Orr, 96]. Mathematically, under certain conditions, they can be shown to be related [Dorffner, 95]. These conditions (basically, that *both* vectors are normalized to unit norm) are rarely met. We shall show in Chapter (4) that, under the unifying cover of similarity, the two models can be seen as different instances of the same general model, without the need of such assumptions.

Other types of neurons are obtained by generalizing the previous choices for h to take into account correlations between input variables (components of the input vector). In the first case, the inner product (containing no cross-product terms) can be generalized to a real quadratic form (an homogeneous polynomial of second degree with real coefficients) or even further to higher degrees, leading to so-called higher-order units (or $\Sigma\Pi$ units). A higher-order unit of degree k includes all possible cross-products of at most k input variables, each with its own weight. Conversely, basic Euclidean distances can be generalized to completely weighted distance measures, where all the (quadratic) cross-products are included. However, these full expressions are not commonly used because of the high numbers of parameters they involve. Other models, like product units, replace the weighted arithmetic mean of the scalar product by its geometric counterpart, $h(\vec{x}, \vec{w}^i) = \prod_j x_j^{w_j^i}$. Although more powerful, they generate a much more difficult search space [Durbin and Rumelhart, 89].

A *layer* is defined as a collection of independent units (that is, not connected with one another) sharing the same input, and of the same general shape (same F_i but different $\vec{w}^i$), i.e., of the same neuron model. Multilayer feed-forward networks take the form of directed acyclic graphs obtained by concatenation of a number of layers. All the layers but the last (the output) are labelled as *hidden*. This kind of networks (shown in Fig. (2.2)) compute a parameterized function $\mathcal{F}_{\underline{\vec{w}}}(\vec{x})$ of their input vector $\vec{x}$ by evaluating the layers in order, giving as final outcome the output of the last layer[1]. The vector $\underline{\vec{w}}$ represents the collection of all weight vectors in the network.

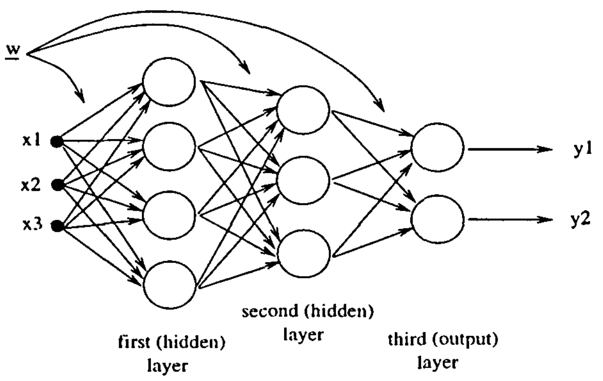


Figure 2.2: A two-hidden-layer ANN example, mapping a three-dimensional input space $\vec{x} = (x_1, x_2, x_3)$ to a two-dimensional output space $(y_1, y_2) = \mathcal{F}_{\underline{\vec{w}}}(\vec{x})$. The network has four and three hidden units in the first and second hidden layers, respectively, and two output neurons. The vector $\underline{\vec{w}}$ represents the collection of all weights in the network.

Output neurons take the form of a scalar product (a linear combination), eventually followed by an activation function σ. For example, assuming a single output neuron, a one-hidden-layer neural network with h_1 hidden units computes a function $\mathcal{F} : \mathbb{R}^n \to \mathbb{R}$ of the form:

$$\mathcal{F}_{\underline{\vec{w}}}(\vec{x}) = \sigma(\sum_{i=1}^{h_1} c_i F_i(\vec{x}) + \theta) \tag{2.2}$$

where $\theta \in \mathbb{R}$ is a possibly null offset term, $c_i \in \mathbb{R}$ and σ can be set as desired, including the choice $\sigma(z) = z$. Such a feed-forward network has $dim(\underline{\vec{w}}) = (n+1)h_1 + h_1 + 1$ parameters to be adjusted.

2.1.2 Neuron models

We consider only the basic models (not including higher-order terms). Both RBF and MLP networks provide parameterized families of functions suitable to function approximation on multidimensional spaces. These two basic neuron models are sometimes described in a somewhat misleading way. A sigmoidal neuron is often explained as putting up an hyperplane that divides its input space in two halves. In other words, the points of equal neuron activation (with fixed weights) are hyperplanes. This behaviour is not caused by the sigmoidal, but by the scalar product. In the same vein, the isoactivation contours for an RBF unit (in case

[1]For simplicity, we are not considering connections between non-adjacent layers and assume otherwise total connectivity. The input layer is *not* counted as a layer.

it is using an unweighted Euclidean norm) are hyperspheres. The radially symmetric and centered response, again, is not caused by the activation function (Gaussian, exponential or otherwise) but by the norm. In both cases, the activation function is acting as a non-linear and monotonic distorsion of its argument -the net input- as computed by the aggregation function.

Definition 2.1 (Isoactivation set) *Given a real function $\sigma : \mathbb{R}^n \to (\sigma_{min}, \sigma_{max})$, define I^α_σ as the set of isoactivation points $I^\alpha_\sigma = \{\vec{x} \in \mathbb{R}^n \mid \sigma(\vec{x}) = \alpha\}$.*

Definition 2.2 (P-neuron) *A neuron model $F_i \in S_F$ of the form:*

$$F_i(\vec{x}) = \{g(\vec{w}^i \cdot \vec{x} + \theta), \vec{w}^i \in \mathbb{R}^n, \theta \in \mathbb{R}\} \tag{2.3}$$

with g a bounded, non-linear, strictly monotonic function for which $\lim_{z\to\infty} g(z) = g_{max} \in \mathbb{R}$ and $\lim_{z\to-\infty} g(z) = g_{min} \in \mathbb{R}$ will be denoted P-neuron (from Perceptron). For these neurons, the sets $I^\alpha_{F_i}$ are (n-1)-dimensional hyperplanes for constant values of α, parallel with one another for different α. This response comes from the use of scalar product as the aggregation function.

In practice, the g are usually the well-behaved sigmoidals (logistic, hyperbolic tangent, etc), though other activation functions are sometimes found in the literature (e.g., sinusoidals). The latter are not included in the above Definition.

Definition 2.3 (R-neuron) *A neuron model $F_i \in S_F$ of the form:*

$$F_i(\vec{x}) = \{g(\frac{1}{\theta}||\vec{x} - \vec{w}^i||_q), \vec{w}^i \in \mathbb{R}^n, \theta > 0 \in R, q \geq 1 \in \mathbb{R}\} \tag{2.4}$$

with g a symmetric function such that $g(|z|)$ is monotonic, and with a maximum g_{max} at $F_i(\vec{w}^i)$ and a (possibly asymptotically reached) $g_{min} = 0$ will be denoted R-neuron (from Radial). For these neurons, the sets $I^\alpha_{F_i}$ are (n-1)-dimensional hypersurfaces (centered at $\vec{w}^i$) for constant values of α (e.g., hypercubes for $q = 1$, hyperspheres for $q = 2$) concentric with one another for different α. The radial response comes from the use of the norm as the aggregation function.

The norm used can be any Minkowskian norm of the form:

$$||\vec{z}||_q = \left(\sum_{i=1}^{n} |z_i|^q\right)^{\frac{1}{q}} \tag{2.5}$$

In practice, typical choices are $q = 2$ and g a Gaussian function.

Definition 2.4 *A feed-forward neural network —which we denote FFNN[c]— consisting of c hidden layers, is a function $\mathcal{F}_{\underline{\vec{w}}} : \mathbb{R}^n \to \mathbb{R}^m$ made up of pieces of the form*

$\vec{y}^{(l)} = (F_1^l(\vec{y}^{(l-1)}), \dots, F_{h_l}^l(\vec{y}^{(l-1)}))$, representing the output of layer l, for $1 \le l \le c+1$. The F^l denote the neuron model of layer l and $h_l \in \mathbb{N}^+$ their number, and each neuron F_i^l has its own parameters $\vec{w}_i^l$ as defined in (2.2) and (2.3), collectively grouped in the network parameters $\underline{\vec{w}}$.

The first output is defined as $\vec{y}^{(0)} = \vec{x}$. For the last (output) layer, $h_{c+1} = m$ and the $F_l^{c+1}, 1 \le l \le h_{c+1}$ are P-neurons or linear units (obtained by setting g to be the identity function in a P-neuron). The final outcome for $\mathcal{F}_{\underline{\vec{w}}}(\vec{x})$ is the value of $\vec{y}^{(c+1)}$.

Definition 2.5 *A single-output feed-forward neural network —denoted 1-FFNN[c]— and consisting of c hidden layers is a function $\mathcal{F}_{\underline{\vec{w}}} : \mathbb{R}^n \to \mathbb{R}$, obtained by setting $m = 1$ in Definition (2.4). For the last (output) layer, $h_{c+1} = 1$ and the single F^{c+1} is a P-neuron or a linear unit. The final outcome for $\mathcal{F}_{\underline{\vec{w}}}(\vec{x})$ is the value of $\vec{y}^{(c+1)}$ (a vector of a single component).*

Definition 2.6 (MLPNN) *A MultiLayer Perceptron Neural Network is a m-FFNN[c] for which $c \ge 1$ and all the F^l are P-neurons, $1 \le l \le c$.*

Definition 2.7 (RBFNN) *A Radial Basis Function Neural Network is a m-FFNN[c] for which $c = 1$ and all the F^c are R-neurons.*

We shall concentrate in this work on one-hidden-layer networks. For convenience, we provide with the following proposition, as a useful and common particular case of the previous definitions:

Proposition 2.1 *A single-output one-hidden-layer neural network 1-FFNN[1] with h_1 hidden units represents a family of functions $\mathcal{F}_{\underline{\vec{w}}} : \mathbb{R}^n \to \mathbb{R}$ of the form:*

$$\mathcal{F}_{\underline{\vec{w}}}(\vec{x}) = \{\sigma(\sum_{i=1}^{h_1} c_i F_i(\vec{x}) + \theta), c_i \in \mathbb{R}, \theta \in \mathbb{R}, \sigma : \mathbb{R} \to \mathbb{R}, F_i \in S_F\} \tag{2.6}$$

where $\theta \in \mathbb{R}$ is a possibly null offset term, $c_i \in \mathbb{R}$ and σ is to be set appropriately, including the choice $\sigma(z) = z$.

Proof: The expression is obtained by setting $c = 1$ in Definition (2.4), and making use of Definition (2.2).

Due to their widespread use, we review here two of the most popular sigmoidals, and show they are tightly related.

A *sigmoidal* function g can be defined as a monotonically increasing function exhibiting smoothness and asymptotic properties. The two more commonly found representatives are: the **logistic**:

$$g_{\beta,\theta}^{\log}(z) = \frac{1}{1 + e^{-\beta(z-\theta)}} \in (0,1) \tag{2.7}$$

and the **hyperbolic tangent:**

$$g^{\tanh}_{\beta,\theta}(z) = \frac{e^{\beta(z-\theta)} - e^{-\beta(z-\theta)}}{e^{\beta(z-\theta)} + e^{-\beta(z-\theta)}} \in (-1,1) \tag{2.8}$$

The offset θ is in practice usually set to zero, because its function is realized by the *bias* term θ, integrated in the aggregation function in (2.3). These two families of functions can be made exactly the same shape (assuming $\theta = 0$) by making the β in (2.7) be twice the value of the β in (2.8). For instance, for $\beta = 1/2$:

$$g^{\tanh}_{\frac{1}{2},0}(z) = g^{\tanh}_{1,0}(\frac{z}{2}) = \frac{1-e^{-z}}{1+e^{-z}} = 2g^{\log}_{1,0}(z) - 1 \tag{2.9}$$

is the bipolar version of $g^{\log}_{1,0}(z) = \frac{1}{1+e^{-z}}$. These functions are normally chosen because of their simple analytic behaviour, especially in what concerns differentiability, of great importance for learning algorithms relying in derivative information. In particular,

$$(g^{\log}_{\beta,0})'(z) = \beta g^{\log}_{\beta,0}(z)[1 - g^{\log}_{\beta,0}(z)] \tag{2.10}$$

The interest in sigmoidal functions also relies in the behaviour of their derivatives. Consider, for example, (2.7) with $\beta = 1.5, \theta = 0$, plotted in Fig. (2.3). The derivative of a sigmoidal is always positive. For $\theta = 0$, all the functions are centered at $z = 0$. In this point, the function has a medium activation, and its derivative is maximum, allowing for maximum weight updates.

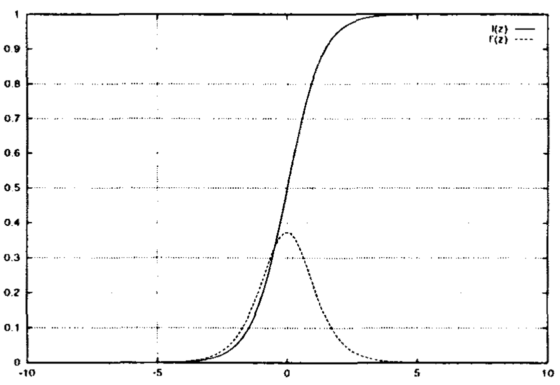


Figure 2.3: The logistic function $l(z)$ and its first derivative $l'(z) > 0$. This function is maximum at the origin, corresponding to a medium activation at $l(0) = \frac{1}{2}$. This point acts as an initial "neutral" value around a quasilinear slope.

2.1.3 Types of Artificial Neural Networks

It is not our intention here to review all the existent families of ANN in the literature. Rather, this Chapter is focused to supervised feed-forward networks, because they are the subject of study in the Thesis. Besides, the field has become so vast that even a complete and clear cut description of all the approaches is a difficult task; we refer the reader to [Fiesler and Beale, 97] for a comprehensive exposition.

One of the basic distinctions that can be made relies on the kind of architecture, basically divided in feed-forward (for which the graph contains no cycles) and recurrent (the rest of situations). A very common architecture, because of its simplicity, among the feed-forward ones, contains no intra-layer connections and all possible inter-layer connections between adjacent layers. More formally, a ***bipartitioned graph*** is a graph G whose nodes V can be partitioned in two disjoint and proper sets V_1 and V_2, $V_1 \cup V_2 = V$, in such a way that no pair of nodes in V_1 is joined by an edge, and the same property holds for V_2. We write then G_{n_1,n_2}, with $n_1 = |V_1|, n_2 = |V_2|$. A bipartitioned graph G_{n_1,n_2} is *complete* if every node in V_1 is connected to every node in V_2. These concepts can be generalized to an arbitrary number of partitions, as follows: A k-partitioned graph $G_{n_1,\ldots,n_k}$ is a graph whose nodes V can be partitioned in k disjoint and proper sets $V_1, \ldots, V_k$, $\bigcup_{i=1}^{k} V_i = V$, in such a way that no pair of nodes in V_i is joined by an edge, for all $1 \leq i \leq k$. In these conditions, a feed-forward fully connected neural network with c hidden layers and h_l units per layer l, $1 \leq l \leq c+1$, takes the form of a directed complete $c+1$-partitioned graph $G_{h_1,\ldots,h_{c+1}}$.

The other fundamental distinction is given by the learning paradigm. A system can be said to *learn* if its performance on a given task improves (w.r.t. some measure) as a result of experience. In ANN, the "experience" is the result of exposure to a training set of data, accompanied with weight modifications. Learning paradigms vary in the amount and nature of the feedback supplied to the network. An essential difference can therefore be made on the type of learning taking place in a ANN. In *supervised* learning the inputs fed to the network are accompanied by the desired response, and the task is to find a regression function that models the underlying function represented by these input/output pairs. In *unsupervised* (also called *self-organized* learning), the task is to model the probability density function of the inputs or to represent the input space in the form of cluster centers with variances.

Considering only these two grounding issues, almost all neural approaches proposed are basically divided in the resulting four groups:

1. Supervised feed-forward networks. These include many of the most well-known and used approaches, as the Perceptron, the MultiLayer Perceptron (MLP) and the families of Radial Basis Function networks (RBF).

2. Supervised recurrent networks. These have also become very popular due to its capacity to accept and model training processes composed of ordered sequences of examples. Among them we find Elman networks [Elman, 90] and Jordan networks [Jordan, 86].

3. Unsupervised feed-forward networks. These include, among others, Counter-Propagation networks [Hecht-Nielsen, 87], Kohonen networks and their variants [Kohonen, 88], and competitive learning [Rumelhart, Hinton and Williams, 86].

4. Unsupervised recurrent networks. These include the well-known Hopfield networks [Hopfield, 82], Boltzmann machines [Hinton, Ackley and Sejnowski, 84], BAM (Bidi-

rectional Associate Memory) machines [Kosko, 92], and a plethora of derivations of the latter.

2.1.4 Learning in Artificial Neural Networks

The main problem tackled in what is known as *supervised learning* is that of regression, the approximation of an (unknown) n-dimensional function $f : X \subset \mathbb{R}^n \rightarrow \mathbb{R}$ by finite superposition of known parameterized *base* functions (PBF) of the style in (2.1). Their combination gives rise to expressions of the form $\mathcal{F}_{\underline{\vec{w}}}(\vec{x})$. This last function is the one realized by the neural network. Usually one is interested in finding parameters $\underline{\vec{w}}^*$ such that $\mathcal{F}_{\underline{\vec{w}}^*}(\vec{x})$ optimizes a cost functional $\mathcal{L}(f, \mathcal{F}_{\underline{\vec{w}}})$ called *loss*, such that

$$\underline{\vec{w}}^* = \underset{\underline{\vec{w}} \in \mathbb{R}^n}{\operatorname{argmin}} \mathcal{L}(f, \mathcal{F}_{\underline{\vec{w}}}) \tag{2.11}$$

Typically, the only information available takes the form of a finite set D of p fixed samples of f, $D = \{< \vec{x}_i, y_i >, f(\vec{x}_i) = y_i\}$, where $\vec{x}_i \in \mathbb{R}^n$ is the stimulus, $y_i \in \mathbb{R}$ is the target, $|D| = p$. The examples are generated taking these $\vec{x}_i$ randomly from a probability density function $p(\vec{x})$, being $p(\vec{x}) = 0$ for $\vec{x} \notin X$. In other words, the $\vec{x}$ are random variables and each $\vec{x}_i$ is a realization of $\vec{x}$ according to $p(\vec{x})$. An estimation of $\mathcal{L}(f, \mathcal{F}_{\underline{\vec{w}}})$ can be hence obtained as $\tilde{\mathcal{L}}(D, \mathcal{F}_{\underline{\vec{w}}})$. This cost measure is called *apparent loss*; it is computed separately for each sample in D,

$$\tilde{\mathcal{L}}(D, \mathcal{F}_{\underline{\vec{w}}}) = \sum_{<\vec{x}_i, y_i> \in D} \lambda(y_i, \mathcal{F}_{\underline{\vec{w}}}(\vec{x}_i)) \tag{2.12}$$

A common form for λ is that of an error function, as the squared-error $\lambda_{\mathrm{SE}}(a, b) = (a-b)^2$. When using the square error, the expression (2.12) can be viewed as the (squared) Euclidean norm in $\mathbb{R}^p$ of the p-dimensional error vector $\vec{e} = (e_1, \ldots, e_p)$, known as the sum-of-squares error, with $e_i = y_i - \mathcal{F}_{\underline{\vec{w}}}(\vec{x}_i)$, as:

$$\tilde{\mathcal{L}}(D, \mathcal{F}_{\underline{\vec{w}}}) = \sum_{<\vec{x}_i, y_i> \in D} (y_i - \mathcal{F}_{\underline{\vec{w}}}(\vec{x}_i))^2 = \vec{e} \cdot \vec{e} = ||\vec{e}||^2 \tag{2.13}$$

The usually reported quantity $\frac{1}{p}||\vec{e}||^2$ is called *mean square error* (MSE), and is a measure of the empirical error (as opposed to the unknown *true* error). We shall denote the error function simply as $E(\underline{\vec{w}}) = \tilde{\mathcal{L}}(D, \mathcal{F}_{\underline{\vec{w}}})$.

When exposed to a training process, the network builds an internal representation of the target function by finding ways to combine the set of base functions $\{F_i(\vec{x})\}_i$. The network relies in the representation capacity of the PBF (that is, of the neuron model) as the cornerstone for a good approximation.

The validity of a solution is mainly determined by an acceptably balanced performance between data in D —that is, a low $\tilde{\mathcal{L}}(D, \mathcal{F}_{\underline{\vec{w}}})$— and *outside* D —a low $\tilde{\mathcal{L}}(D_{\text{out}}, \mathcal{F}_{\underline{\vec{w}}})$, $D_{\text{out}} \subset$

$X \setminus D$, to ensure that f has been correctly estimated from the data. Network models too inflexible or simple or, on the contrary, too flexible will generalize inadequately. This tradeoff is reflected in $\tilde{\mathcal{L}}(D_{\text{out}}, \mathcal{F}_{\underline{\vec{w}}})$, which can be decomposed in two opposing terms called *bias* and *variance* [Geman, Bienenstock and Doursat, 92]. The expectation for the sum-of-squares error function, averaged over the complete ensemble of data sets D is written as [Bishop, 95]:

$$\begin{aligned} E(\underline{\vec{w}}) &= E_D\{(\mathcal{F}_{\underline{\vec{w}}}(\vec{x}) - <y|\vec{x}>)^2\} = \\ &(E_D\{(\mathcal{F}_{\underline{\vec{w}}}(\vec{x}) - <y|\vec{x}>\})^2 + E_D\{(\mathcal{F}_{\underline{\vec{w}}}(\vec{x}) - E_D\{\mathcal{F}_{\underline{\vec{w}}}(\vec{x})\})^2\} \end{aligned} \tag{2.14}$$

where $<y|\vec{x}>$ denotes the conditional average of the target $y = f(\vec{x})$ (which expresses the optimal network mapping), given by:

$$<y|\vec{x}> = \int y\, p(y|\vec{x})\, dy \tag{2.15}$$

The first term in the right hand side of (2.14) is the (squared) bias and the second is the variance. The bias measures the extent to which the average (over all D) of $\mathcal{F}_{\underline{\vec{w}}}(\vec{x})$ differs from the desired target function $<y|\vec{x}>$. The variance measures the sensitivity of $\mathcal{F}_{\underline{\vec{w}}}(\vec{x})$ to the particular choice of D. As expressed above, too inflexible or simple models will have a large bias, while too flexible or complex will have a large variance. These are hence complementary quantities that have to be minimized simultaneously; both can be shown to decrease with increasing availability of larger data sets D.

The expressions in (2.14) are functions of an input vector $\vec{x}$. The average values for bias and variance can be obtained by weighting with the corresponding density $p(\vec{x})$:

$$\begin{aligned} &\int_X E_D\{(\mathcal{F}_{\underline{\vec{w}}}(\vec{x}) - <y|\vec{x}>)^2\} p(\vec{x})\, d\vec{x} = \\ &\quad \int_X (E_D\{(\mathcal{F}_{\underline{\vec{w}}}(\vec{x}) - <y|\vec{x}>\})^2 p(\vec{x})\, d\vec{x} + \int_X E_D\{(\mathcal{F}_{\underline{\vec{w}}}(\vec{x}) - E_D\{\mathcal{F}_{\underline{\vec{w}}}(\vec{x})\})^2\} p(\vec{x})\, d\vec{x} \end{aligned} \tag{2.16}$$

The key conditions for acceptable performance on novel data are given by a training set D as large and representative as possible of the underlying distribution, and a set D_{out} of previously unseen data which should not contain examples exceedingly different from those in D. An important additional consideration is the use of a net with minimal complexity, given by the number of free parameters (that is, the number of components in $\underline{\vec{w}}$).

A useful and many times necessary assumption is that of *smoothness* of the obtained solution, to filter out small input variations (noise) and of benefit since it biases the search space of functions, thereby reducing it. This requirement can be realized in practice in various ways. In regularization theory, the solution is obtained from a variational principle including the loss and prior smoothness information, by defining a *smoothing functional* ϕ in such a way that lower values correspond to smoother functions. A solution of the approximation problem is given by minimization of the following functional [Girosi, Jones and Poggio, 93]:

$$H[\mathcal{F}_{\vec{w}}] = \tilde{\mathcal{L}}(D, \mathcal{F}_{\vec{w}}) + \lambda\phi[\mathcal{F}_{\vec{w}}] \tag{2.17}$$

where λ is a positive scalar controlling the tradeoff between fitness to the data and smoothness of the solution. A common choice is a differential operator $P(f) = f''$ (the second derivative) of which the (squared) Euclidean norm is taken:

$$\phi[\mathcal{F}_{\vec{w}}] = ||P(\mathcal{F}_{\vec{w}})||_2 = \int_{\mathbb{R}} \{\mathcal{F}''_{\vec{w}}(t)\}^2 dt \tag{2.18}$$

The most typical learning problems are well known and —besides determination of the learning parameters— include [Hertz, Krogh and Palmer, 91], [Hinton, 89], [Bishop, 95]:

1. The possibility of getting stuck in *local optima* of the cost function, in which conventional non-linear optimization techniques will stay forever. The incorporation of a global scheme (like multiple restarts or an annealing schedule) is surely to increase the chance of finding a better solution, although the cost can become prohibitedly high. A feed-forward network has multiple equivalent solutions, created by weight permutations and sign flips. A network with a single hidden layer of h_1 units has $s(h_1) = h_1!2^{h_1}$ solutions, so the chances of getting in the basin of attraction of one of them are reasonable high[2]. However, the complexity of the error surface -especially in very high dimensions- makes the possibility of getting trapped a real one. The use of reliable and fast optimization techniques, enhanced with a means to escape from local optima are almost always the best choice. The quality of the solution is then determined by the finite precision of numerical calculations and by the finite number of iterations allowed.

2. Long *training times*, oscillations and network paralysis. These are features highly related to the specific learning algorithm, and relate to bad or too general choices for the parameters of the optimization technique (such as the learning rate in steepest descent). The presence of saddle points -regions where the error surface is very flat- also provoke an extremely slow advance for extensive periods of time. The use of more advanced methods that dynamically set these and other parameters, such as the conjugate gradient (see §2.1.6) can alleviate the problem.

3. *Non-cumulative* learning. It is hard to take an already trained network and re-train it with additional data without losing previously learned knowledge.

4. The *curse of dimensionality*, roughly stated as the fact that the number of examples needed to represent a given function grows exponentially with the number of dimensions.

5. Difficulty of finding a *structure* in the training data, possibly caused by a very high dimension or a distorting preprocessing scheme.

6. Bad *generalization*, which can be due to several causes: the use of poor training data or attempts to extrapolate beyond them, an excessive number of hidden units, too

[2]For a moderate $h_1 = 10$, $s(h_1) \approx 3.7 \cdot 10^9$.

long training processes or the use of a too high learning rate. All of them can lead to an *overfitting* of the training data, in which the ANN adjusts the training set as an *interpolation* task, trying to pass the surface over each training example.

7. Not amenable to *inspection*. It is generally arduous to interpret the knowledge learned, especially in large networks or with a high number of model inputs.

8. *Missing* data. Unlike other methods, like decision trees, missing feature values are difficult and expensive to handle.

2.1.5 Learning algorithms

The view of ANN as systems which adapt their functionality as a result of exposure to information has lead to the development of generic procedures and techniques, also known as *learning rules*. By far the most widely used in the neural network context are those relying in derivative information, herein called derivative-based methods (DBM). Chronologically, the first of these is the Perceptron Learning Rule [Rosenblatt, 62], useful only for single layer feed-forward architectures exposed to the learning of data from a linearly separable problem. None the less, its simplicity and beauty, and the existence of a *convergence theorem* (a rare situation in the ANN paradigm) make it a basic departure point in neural learning algorithms. This algorithm is a particular case of the continuous Widrow-Hoff or *delta* rule [Widrow and Hoff, 60], applicable in general to networks which can be completely continuous (as opposed to the binary-output perceptron) and whose error function is quadratic in the parameters. It has been very successfully used for decades in pattern recognition and signal processing applications.

The first truly useful algorithm for feed-forward multilayer networks is the now golden standard *backpropagation* algorithm [Rumelhart, Hinton and Williams, 86], reportedly proposed first by Werbos [Werbos, 74] and Parker [Parker, 82]. Many efforts have been devoted to enhance the basic algorithm in a number of ways, especially concerning speed and reliability of convergence (see [Haykin, 94] for a review). Other, faster and more powerful schemes exist –though following the same basic idea of back-propagating the errors– such as the Conjugate Gradient Descent [Shewchuck, 94] or the Levenberg-Marquardt method, described in [Bishop, 95]. An excellent review of many of the methods in a common framework is [Fiesler and Beale, 97]. For recurrent networks, a good survey of the algorithms is [Pearlmutter, 90].

2.1.6 Error-minimization methods

Neural networks are classically trained by iteratively setting appropriate values for the network parameters $\underline{\vec{w}}$ so as to minimize an error function $E(\underline{\vec{w}})$, such as that in 2.13. If this error function is quadratic in $\underline{\vec{w}}$, then the solution can be found by solving a linear system of equations, for instance with the Singular Value Decomposition method [Press *et al.*, 92] or iteratively with the mentioned delta rule. This situation usually happens for networks with no hidden units and linear output units, or most commonly as one of the stages of RBF network training (discussed in §2.1.7). In the general case, the minimization is realized by a variant of a *gradient descent* procedure, whose ultimate outcome is a local minimum -a $\underline{\vec{w}}^*$ from which any infinitesimal change makes $E(\underline{\vec{w}}^*)$ increase) which, as discussed in §2.1.4, may not correspond to one of the global minima. Different solutions are likely to be found by starting at different initial states.

The entire learning process can be depicted as a walk across a surface in a very high-dimensional weight space. The optimal weight vector $\underline{\vec{w}}^*$ is in general not to be reached because of the limited number of hidden units and learning set size. The process is also perturbed by roundoff errors.

Iterative minimization methods are based on the following principle. Given $E(\underline{\vec{w}})$ to be minimized, and an initial state $\underline{\vec{w}}^0$, for each iteration, and until a stopping criteria is met, perform the updating step:

$$\underline{\vec{w}}^{i+1} = \underline{\vec{w}}^i + \alpha_i \underline{\vec{u}}^i \tag{2.19}$$

where $\underline{\vec{u}}^i$ is the *minimization direction* (the direction in which to move) and $\alpha_i \in \mathbb{R}$ is the *step size* (how far to make a move in $\underline{\vec{u}}^i$), also known as the *learning rate* in the context of ANN optimization. For future convenience, we define $\Delta\underline{\vec{w}}^i = \underline{\vec{w}}^{i+1} - \underline{\vec{w}}^i$

First-order methods

The *gradient* $\nabla E_{\underline{\vec{w}}}$ of an r-dimensional function is the vector field of first derivatives of $E(\underline{\vec{w}})$ w.r.t. $\underline{\vec{w}}$,

$$\nabla E_{\underline{\vec{w}}} = \left(\frac{\delta E(\underline{\vec{w}})}{\delta w_1}, \ldots, \frac{\delta E(\underline{\vec{w}})}{\delta w_r} \right) \tag{2.20}$$

In our case, $r = dim(\underline{\vec{w}})$. A linear approximation to $E(\underline{\vec{w}})$ in an infinitesimal neighbourhood of an arbitrary point $\underline{\vec{w}}^i$ is given by:

$$E(\underline{\vec{w}}) \approx E(\underline{\vec{w}}^i) + \nabla E_{\underline{\vec{w}}}(\underline{\vec{w}}^i) \cdot (\underline{\vec{w}} - \underline{\vec{w}}^i) \tag{2.21}$$

We write $\nabla E_{\underline{\vec{w}}}(\underline{\vec{w}}^i)$ to refer to the gradient $\nabla E_{\underline{\vec{w}}}$ evaluated at $\underline{\vec{w}}^i$. These are the first two terms of the Taylor expansion of $E(\underline{\vec{w}})$ around $\underline{\vec{w}}^i$. In first-order methods, this local gradient alone determines the minimization direction $\underline{\vec{u}}^i$. They are known as *steepest* or *gradient* descent methods. Since, at any point $\underline{\vec{w}}^i$, the gradient $\nabla E_{\underline{\vec{w}}}(\underline{\vec{w}}^i)$ points in the direction of fastest increase of $E(\underline{\vec{w}})$, an adjustment of $\underline{\vec{w}}^i$ in the negative direction of the local gradient leads to its maximum decrease. Thus, being $\underline{\vec{w}}^i$ the current network weight vector, the direction $\underline{\vec{u}}^i = -\nabla E_{\underline{\vec{w}}}(\underline{\vec{w}}^i)$ is taken.

In conventional steepest descent, the step size α_i is obtained by a *line search* in the direction of $\underline{\vec{u}}^i$. This concerns how far to go along $\underline{\vec{u}}^i$ before a new direction is chosen. To this end, evaluations of $E(\underline{\vec{w}})$ and its derivatives are carried out to locate some local or global minimum, and stop there. Although it is possible to locate the global minimum, the cost can become prohibitely high (tens to hundreds of evaluations of $E(\underline{\vec{w}})$). It has also been argued that perfect line minimization is not desirable because it can lead to getting stuck in local minima of the overall error landscape.

The idea behind line search is to move in the chosen direction $\underline{\vec{u}}^i$ to find the minimum of $E(\underline{\vec{w}})$ along it. For this one-dimensional minimization problem, the simplest approach is to proceed along $\underline{\vec{u}}^i$ in small steps, evaluating $E(\underline{\vec{w}})$ at each sampled point, until it starts to increase. One often used method works out a divide-and-conquer strategy [Fletcher, 80]:

1. Bracket the search by setting three points[3] $\underline{\vec{a}} < \underline{\vec{b}} < \underline{\vec{c}}$ along $\underline{\vec{u}}^i$ such that $E(\underline{\vec{a}}) > E(\underline{\vec{b}})$ and $E(\underline{\vec{b}}) < E(\underline{\vec{c}})$. Since $E(\underline{\vec{w}})$ is continuous, there is a local minima in the line joining $\underline{\vec{a}}$ to $\underline{\vec{c}}$.
2. Fit a parabola (a quadratic polynomial) to $\underline{\vec{a}}, \underline{\vec{b}}, \underline{\vec{c}}$.
3. Compute the minimum $\underline{\vec{\mu}}$ of the parabola in the line joining $\underline{\vec{a}}$ to $\underline{\vec{c}}$. This value is an approximation of the minimum of $E(\underline{\vec{w}})$ in this interval.
4. Set three new points $\underline{\vec{a}}, \underline{\vec{b}}, \underline{\vec{c}}$ out of $\underline{\vec{\mu}}$ and the two points among the old $\underline{\vec{a}}, \underline{\vec{b}}, \underline{\vec{c}}$ having the lowest $E(\underline{\vec{w}})$. Repeat again from 2.

In standard back-propagation, the line search is replaced by a fixed step size α, which has to be carefully chosen. A sufficiently small value is required, so that $-\alpha\nabla E_{\underline{\vec{w}}}(\underline{\vec{u}}^i)$ (the update on $\underline{\vec{w}}^i$) is effectively very small and the expansion (2.21) can be applied. Besides, a too large value might cause to overshoot and quite possibly to lead to divergent oscillations and a complete breakout of the algorithm. On the other hand, very small values translate in a painfully slow minimization. In practice, a trial-and-error process is carried out.

A popular heuristic to enhance the basic method is to use a historic average of previous changes to exploit tendencies and add inertia to the descent in weight space. This is accomplished by adding a so-called *momentum* term $\beta_i \Delta\underline{\vec{w}}^{i-1}$, where $\Delta\underline{\vec{w}}^{i-1}$ is the previous weight update [Rumelhart, Hinton and Williams, 86]. This term greatly helps to avoid or smooth out oscillations in the motion towards a minimum. In practice, like the learning rate, in standard back-propagation it is set to a constant value $\beta \in [0.5, 1]$.

Altogether, for steepest descent, the update equation (2.19) reads:

$$\underline{\vec{w}}^{i+1} = \underline{\vec{w}}^i + \alpha_i \underline{\vec{u}}^i + \beta\Delta\underline{\vec{w}}^{i-1} \tag{2.22}$$

where $\underline{\vec{u}}^i = -\nabla E_{\underline{\vec{w}}}(\underline{\vec{w}}^i)$ and $\Delta\underline{\vec{w}}^{i-1} = \underline{\vec{w}}^i - \underline{\vec{w}}^{i-1}$.

The performance of this method is very sensitive to the chosen values for α_i and β, to the point that different values are required for different problems and even for different stages in the learning process of a given problem [Toolenaere, 90]. This has lead to the development of adaptive schemes to dynamically set these values, since the optimum typically changes during

[3] For ease of reading, the "<" relation in $\mathbb{R}^r$ is taken to be along the chosen direction.

optimization. Generally speaking, the inefficiency of the overall steepest descent method stems from the fact that both $\underline{\vec{u}}^i$ and α_i are somewhat poorly chosen. Unless the first step is chosen leading straight to a minimum, the iterative procedure is very likely to wander with many small steps in zig-zag. A method in which both parameters are well-chosen as part of the method itself is the conjugate gradient.

Second-order methods

First-order approximations ignore the curvature of $E(\underline{\vec{w}})$. This can be fixed by additionally considering the second-order term of the Taylor expansion around some point $\underline{\vec{w}}^i$ in weight space, given by:

$$E(\underline{\vec{w}}) \approx E(\underline{\vec{w}}^i) + \nabla E_{\underline{\vec{w}}}(\underline{\vec{w}}^i) \cdot (\underline{\vec{w}} - \underline{\vec{w}}^i) + \frac{1}{2}(\underline{\vec{w}} - \underline{\vec{w}}^i) \cdot H_{\underline{\vec{w}}}(\underline{\vec{w}}^i) \cdot (\underline{\vec{w}} - \underline{\vec{w}}^i) \tag{2.23}$$

where $H_{\underline{\vec{w}}} = \nabla\nabla E_{\underline{\vec{w}}}$ is the Hessian matrix, or $r \times r$ matrix of cross second derivatives, with components $H_{\underline{\vec{w}}} = (h_{ij})$, $h_{ij} = \frac{\delta^2 E(\underline{\vec{w}})}{\delta w_i \delta w_j}$. Again, $H_{\underline{\vec{w}}}(\underline{\vec{w}}^i)$ indicates the evaluation of $H_{\underline{\vec{w}}}$ in $\underline{\vec{w}}^i$. The Hessian traces the curvature of the error function in weight space, portraying information about how $\nabla E_{\underline{\vec{w}}}$ changes in different directions. Given a direction $\underline{\vec{u}}^i$ from $\underline{\vec{w}}^i$, the product $H_{\underline{\vec{w}}}(\underline{\vec{w}}^i) \cdot \underline{\vec{u}}^i$ is the rate of change of the gradient along $\underline{\vec{u}}^i$ from $\underline{\vec{w}}^i$. Differentiating (2.23) w.r.t. $\underline{\vec{w}}$, a local approximation of the gradient around $\underline{\vec{w}}^i$ is obtained as:

$$\nabla E_{\underline{\vec{w}}} \approx \nabla E_{\underline{\vec{w}}}(\underline{\vec{w}}^i) + H_{\underline{\vec{w}}} \cdot (\underline{\vec{w}} - \underline{\vec{w}}^i) \tag{2.24}$$

For points close to $\underline{\vec{w}}^i$, (2.23) and (2.24) give reasonable approximations to the error function $E(\underline{\vec{w}})$ and to its gradient. This forms the basis of second-order algorithms. Setting (2.24) to zero, and solving for $\underline{\vec{w}}$, we get:

$$\underline{\vec{w}} = -H_{\underline{\vec{w}}}^{-1} \cdot \nabla E_{\underline{\vec{w}}}(\underline{\vec{w}}^i) \tag{2.25}$$

Hence, knowledge of the Hessian and the gradient leads to straight optimization methods if the error function is quadratic. However, calculating $H_{\underline{\vec{w}}}^{-1}$ is computationally prohibitive. This motivates the development of alternative approximation methods.

Conjugate Gradient Descent

The conjugate gradient minimization technique (explained at length in [Shewchuck, 94]) is based on the idea that, when a new direction $\underline{\vec{u}}^{i+1}$ is chosen, it should not spoil previous minimizations in the directions $\underline{\vec{u}}^i, \underline{\vec{u}}^{i-1}, \ldots, \underline{\vec{u}}^1$. This is certainly the case if we simply choose $\underline{\vec{u}}^i = -\underline{\vec{g}}^i$, where $\underline{\vec{g}}^i = \nabla E_{\underline{\vec{w}}}(\underline{\vec{w}}^i)$, as was found in §2.1.6 for steepest descent.

At most points on $E(\underline{\vec{w}})$, the gradient does *not* point directly towards the minimum. Besides, after a line minimization, the new gradient $\underline{\vec{g}}^{i+1}$ is orthogonal to the line search direction, that is, $\underline{\vec{g}}^{i+1} \cdot \underline{\vec{u}}^i = 0$. Thus, successive search directions will also be orthogonal, and the error function minimization will proceed in zig-zag, in a very undecided manner, resulting in a extremely slow advance to a minimum [Bishop, 95].

The solution to this problem lies in determining consecutive search directions $\underline{\vec{u}}^{i+1}$ in such a way that the component of the gradient parallel to $\underline{\vec{u}}^i$, which has just been made to be zero because we minimized in that direction, remains zero, so that consecutive search directions complement each other, and avoid the possibility of undoing the progress done in previous iterations.

Let us assume a line minimization has just been made along $\underline{\vec{u}}^i$, starting from the current weights $\underline{\vec{w}}^i$; we have thus found a new point $\underline{\vec{w}}^{i+1}$ for which it holds:

$$\nabla E_{\underline{\vec{w}}}(\underline{\vec{w}}^{i+1}) \cdot \underline{\vec{u}}^i = 0 \tag{2.26}$$

The next search direction $\underline{\vec{u}}^{i+1}$ is chosen to retain the property that the component of the gradient parallel to $\underline{\vec{u}}^i$, remains zero:

$$\nabla E_{\underline{\vec{w}}}(\underline{\vec{w}}^{i+1} + \alpha_i \underline{\vec{u}}^{i+1}) \cdot \underline{\vec{u}}^i = 0 \tag{2.27}$$

Expanding (2.27) to first order in α_i, and applying to it (2.23) and (2.26), we obtain the condition [Bishop, 95]:

$$\underline{\vec{u}}^{i+1} \cdot H_{\underline{\vec{w}}}(\underline{\vec{w}}^{i+1}) \cdot \underline{\vec{u}}^i = 0 \tag{2.28}$$

If the error surface is quadratic, (2.28) holds regardless of the value of α_i, because the Hessian is constant, and higher-order terms in the previous expansion vanish. Search directions $\underline{\vec{u}}^{i+1}, \underline{\vec{u}}^i$ fulfilling (2.28) are said to be *conjugate*. It can be proven that, in these conditions, it is possible to construct a sequence $\underline{\vec{u}}^1, \ldots, \underline{\vec{u}}^r$ such that $\underline{\vec{u}}^r$ is conjugate to all previous directions, so that the minimum can be located in at most $r = dim(\underline{\vec{w}})$ steps.

The *conjugate gradient* technique departs from the same general expression (2.19), reproduced for convenience:

$$\underline{\vec{w}}^{i+1} = \underline{\vec{w}}^i + \alpha_i \underline{\vec{u}}^i \tag{2.29}$$

The method sets $\underline{\vec{u}}^{i+1} = -\underline{\vec{g}}^{i+1} + \beta_i \underline{\vec{u}}^i$, with $\underline{\vec{u}}^1 = -\underline{\vec{g}}^1$. It turns out that the coefficients β_i can be found without explicit knowledge of the Hessian, as:

$$\beta_i = \frac{\underline{\vec{g}}^{i+1} \cdot (\underline{\vec{g}}^{i+1} - \underline{\vec{g}}^i)}{\underline{\vec{g}}^i \cdot \underline{\vec{g}}^i} \tag{2.30}$$

This is the *Polak-Ribière* updating: there are others. The α_i can be found by line minimization of $\nabla E_{\underline{\vec{w}}}(\underline{\vec{w}}^i + \alpha_i \underline{\vec{u}}^i)$ w.r.t. α_i. As can be seen, all this results in a particular case of steepest descent with momentum, in which the parameters α_i, β_i are determined at each iteration step. For a quadratic error surface $E(\underline{\vec{w}})$, the method finds the minimum after at most $r = dim(\underline{\vec{w}})$ steps, without calculating the Hessian. In practice, $E(\underline{\vec{w}})$ may be far from being quadratic; therefore, the technique needs to be run for many iterations and augmented with a criterion to reset the search vector to the negative gradient direction $\underline{\vec{u}}^{i+1} = -\underline{\vec{g}}^{i+1}$ after every r steps. A detailed description can be found in [Press *et al.*, 92].

With these and other enhancements, such as the *scaled* version [Møller, 93], which takes some account of the non-quadratic nature of the error function, the method is generally believed to be fast and reliable. Also, contrary to steepest descent, it is relatively insensitive to its parameters -the line search for α_i and the variants of computing β_i- if they are set within a reasonable tolerance.

The Back-propagation algorithm

The breakthrough that this algorithm represented in connectionist learning was the possibility to compute the gradient of the network weight parameter $\underline{\vec{w}}$ in a sequential but distributed way. The algorithm is so-called because the components of the gradient concerning weights belonging to output units are computed first, and then propagated backwards (toward the inputs) to compute the rest, in the order marked by the layers.

In its very essence, it is nothing more than the successive application of the chain rule to compute the partial derivatives of a function like (2.13) that depends of a functional expression like that in Definition (2.4). Intuitively, what the algorithm founds is the extent to which the adjustment of one connection will reduce the discrepancy between the actual and correct outputs of the network. This information is of course the partial derivative of the error function $E(\underline{\vec{w}})$ w.r.t. the connection, and therefore the overall gradient vector $\nabla E_{\underline{\vec{w}}}$ is being computed by the algorithm. This information is then used to adjust each connection (each single weight) by any method generally conforming to (2.19).

To derive the algorithm, we first introduce some extra notation. Given $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ the function to be approximated, we depart from a finite training set D of p samples of f, $D = \{<\vec{x}_1, \vec{y}_1>, \ldots, <\vec{x}_p, \vec{y}_p>\}$, $f(\vec{x}_\mu)_k = y_{\mu,k}$. For simplicity, we assume the loss is the square error and define:

$$E(\underline{\vec{w}}) = \frac{1}{2}\sum_{\mu=1}^{p}\sum_{k=1}^{m}(y_{\mu,k} - \zeta_k^{\mu,c+1})^2 \tag{2.31}$$

where $\zeta_k^{\mu,c+1} = \mathcal{F}_{\underline{\vec{w}}}(\vec{x}_\mu)_k$ is the k-th component of the network's response to input pattern $\vec{x}_\mu$ (the network has $c+1$ layers, of which c are hidden). For a given input pattern $\vec{x}_\mu$, we define:

$$E^\mu(\underline{\vec{w}}) = \frac{1}{2}\sum_{k=1}^{m}(y_{\mu,k} - \zeta_k^{\mu,c+1})^2 \tag{2.32}$$

so that $E(\underline{\vec{w}}) = \sum_{\mu=1}^{p} E^\mu(\underline{\vec{w}})$. The computation of a single unit i in layer l, $1 \leq l \leq c+1$

upon presentation of pattern $\vec{x}_\mu$ to the network is expressed $\zeta_i^{\mu,l} = g(\hat{\zeta}_i^{\mu,l})$, with g a smooth function as in Definition (2.2) –as the sigmoidals (2.7) or (2.8)–, and $\hat{\zeta}_i^{\mu,l} = \sum_j w_{ij}^l \zeta_j^{\mu,l-1}$. The first outputs are defined $\zeta_i^{\mu,0} = x_{\mu,i}$. A single weight w_{ij}^l denotes the connection strength from neuron j in layer $l-1$ to neuron i in layer l, $1 \le l \le c+1$.

The gradient descent rule (2.19) with constant α is followed, with $\underline{\vec{u}}^i = -\nabla E_{\underline{\vec{w}}}(\underline{\vec{w}}^i)$. Together with the definition of the gradient (2.20), we find that the increment Δw_{ij} in a single weight w_{ij}^l of $\underline{\vec{w}}$ is:

$$\Delta w_{ij}^l = -\alpha \frac{\delta E(\underline{\vec{w}})}{\delta w_{ij}^l} = -\alpha \sum_\mu \frac{\delta E^\mu(\underline{\vec{w}})}{\delta w_{ij}^l} = -\alpha \sum_\mu \Delta^\mu w_{ij}^l \tag{2.33}$$

We have:

$$\Delta^\mu w_{ij}^l = \frac{\delta E^\mu(\underline{\vec{w}})}{\delta w_{ij}^l} = \frac{\delta E^\mu(\underline{\vec{w}})}{\delta \zeta_i^{\mu,l}} \frac{\delta \zeta_i^{\mu,l}}{\delta \hat{\zeta}_i^{\mu,l}} \frac{\delta \hat{\zeta}_i^{\mu,l}}{\delta w_{ij}^l} \tag{2.34}$$

Proceeding from right to left in (2.34):

$$\frac{\delta \hat{\zeta}_i^{\mu,l}}{\delta w_{ij}^l} = \zeta_j^{\mu,l-1} \tag{2.35}$$

$$\frac{\delta \zeta_i^{\mu,l}}{\delta \hat{\zeta}_i^{\mu,l}} = \frac{dg(\hat{\zeta}_i^{\mu,l})}{d\hat{\zeta}_i^{\mu,l}} = g'(\hat{\zeta}_i^{\mu,l}) \tag{2.36}$$

Assuming, for example, g to be the logistic function g_β (2.7), with zero offset θ, using (2.10):

$$g'(\hat{\zeta}_i^{\mu,l}) = \beta g_\beta(\hat{\zeta}_i^{\mu,l})[1 - g_\beta(\hat{\zeta}_i^{\mu,l})] = \beta \zeta_i^{\mu,l}(1 - \zeta_i^{\mu,l}) \tag{2.37}$$

The remaining expression $\frac{\delta E^\mu(\underline{\vec{w}})}{\delta \zeta_i^{\mu,l}}$ is the more delicate and constitutes the core of the algorithm. We develop it in two separate cases: $l = c+1$ and $l < c+1$. The first case corresponds to output neurons, for which an expression of the derivative is immediate from (2.32):

$$\frac{\delta E^\mu(\underline{\vec{w}})}{\delta \zeta_i^{\mu,c+1}} = -(y_{\mu,i} - \zeta_i^{\mu,c+1}) \tag{2.38}$$

Incidentally, if we collect all the results so far, assuming $c = 0$ (no hidden layers), the mentioned *delta rule* for non-linear single-layer networks is obtained:

$$\Delta^\mu w_{ij} = \alpha \sum_\mu (y_{\mu,i} - \zeta_i^\mu) g'(\hat{\zeta}_i^\mu) \zeta_j^{\mu,0} = \alpha \sum_\mu \delta_i^\mu x_{\mu,j} \tag{2.39}$$

where we have removed the superindex $l = 1$, (since $c = 0$) for clarity, and defined $\delta_i^\mu = (y_{\mu,i} - \zeta_i^\mu)g'(\hat{\zeta}_i^\mu)$. An equivalent name for the back-propagation algorithm is *generalized delta rule.* Consequently with (2.39), for $l \geq 1$, the deltas (that is, the errors local to a unit) are defined as:

$$\delta_i^{\mu,l} = \frac{\delta E^\mu(\underline{\vec{w}})}{\delta \hat{\zeta}_i^{\mu,l}} \tag{2.40}$$

and we have just seen that, for the last (output) layer $l = c+1$, $\delta_i^{\mu,l} = (y_{\mu,i} - \zeta_i^{\mu,l})g'_\beta(\hat{\zeta}_i^{\mu,l})$. In case g is (2.7), we shall have:

$$\delta_i^{\mu,l} = (y_{\mu,i} - \zeta_i^{\mu,l})\beta\zeta_i^{\mu,l}(1 - \zeta_i^{\mu,l}) \tag{2.41}$$

For the general case $l < c + 1$, we proceed as follows:

$$\frac{\delta E^\mu(\underline{\vec{w}})}{\delta \hat{\zeta}_i^{\mu,l}} = \sum_k \frac{\delta E^\mu(\underline{\vec{w}})}{\delta \hat{\zeta}_k^{\mu,l+1}} \frac{\delta \hat{\zeta}_k^{\mu,l+1}}{\delta \zeta_i^{\mu,l}} = \tag{2.42}$$

Now, $l < c + 1$ means that there is at least one more layer $l + 1$ to the *right* of layer l –according to Fig. (2.2)–, that is, a layer which is *posterior* in the *feed-forward* computation, but that *precedes* l in the opposite (back) direction. In other words, $\delta E^\mu(\underline{\vec{w}})$ functionally depends on layer $l + 1$ *before* than on layer l, and thus the derivative computation (through the chain rule) can be broken down in two pieces. The summation over k is due to the fact that $\delta E^\mu(\underline{\vec{w}})$ depends on every neuron in layer $l + 1$. Rewriting (2.42):

$$= \sum_k \frac{\delta E^\mu(\underline{\vec{w}})}{\delta \hat{\zeta}_k^{\mu,l+1}} \frac{\delta \hat{\zeta}_k^{\mu,l+1}}{\delta \zeta_i^{\mu,l}} = \sum_k \delta_k^{\mu,l+1} w_{ki}^{l+1} \tag{2.43}$$

since $\hat{\zeta}_k^{\mu,l+1} = \sum_j w_{kj}^{l+1}\zeta_j^{\mu,l}$ and thus $\frac{\delta \hat{\zeta}_k^{\mu,l+1}}{\delta \zeta_i^{\mu,l}} = w_{ki}^{l+1}$. Putting together (2.35), (2.37) and (2.43) in (2.34):

$$\Delta^\mu w_{ij}^l = \underbrace{\frac{\delta E^\mu(\underline{\vec{w}})}{\delta \zeta_i^{\mu,l}} \frac{\delta \zeta_i^{\mu,l}}{\delta \hat{\zeta}_i^{\mu,l}}}_{\text{this is } \delta_i^{\mu,l}} \frac{\delta \hat{\zeta}_i^{\mu,l}}{\delta w_{ij}^l} = \delta_i^{\mu,l}\zeta_j^{\mu,l-1} \tag{2.44}$$

where $\delta_i^{\mu,l} = g'(\hat{\zeta}_i^{\mu,l}) \sum_k \delta_k^{\mu,l+1} w_{ki}^{l+1}$. As it can be seen, this is again a valid version of the delta rule, but now we have a means to compute the deltas for the hidden units. To end the presentation, we show (Fig. 2.4) an algorithmic form for the derived back-propagation procedure, for a single training step (one presentation of the training set D):

```
forall $\mu$ in $1 \leq \mu \leq p$
    1. Forward pass. Present $\vec{x}^{\mu}$ and compute
       the outputs $\zeta_i^{\mu,l}$ of all the units.
    2. Backward pass. Compute the deltas $\delta_i^{\mu,l}$
       of all the units (the local gradients), as follows:
            a. $\underline{l = c+1}$ : $\delta_i^{\mu,l} = g'(\hat{\zeta}_i^{\mu,l})(y_{\mu,i} - \zeta_i^{\mu,l})$
            b. $\underline{l < c+1}$ : $\delta_i^{\mu,l} = g'(\hat{\zeta}_i^{\mu,l}) \sum_k \delta_k^{\mu,l+1} w_{ki}^{l+1}$
    3. $\Delta^{\mu} w_{ij}^{l} = \delta_i^{\mu,l} \zeta_j^{\mu,l-1}$
end
Update weights as $\Delta w_{ij}^{l} = \alpha \sum_{\mu} \Delta^{\mu} w_{ij}^{l}$
```

Figure 2.4: Back-propagation Algorithm pseudocode.

Annealing schedules

The term simulated annealing comes from termodynamics, where a physical solid is heated to an initially very high temperature, and slowly let to loose heat in a controlled way, until it reaches thermal equilibrium (a state of minimum energy). The integration of simulated annealing techniques to the workings of an ANN learning rule can give rise to enhanced algorithms able to escape from local minima to a certain extent [van Laarhoven and Aarts, 87]. The idea is to perform random changes in the weights and evaluate their effect on the global error. A decrease in error leads to the acceptance of the change. An increase leads to acceptance agreeing to a certain probability distribution, according to which the chances of acceptance are higher in earlier stages of the learning process. This probability can be either a **Boltzmann** or **Cauchy**:

Boltzmann: $$P(\delta w) = exp\{-\frac{\Delta E(\delta w)}{T(t)}\}$$

Cauchy: $$P(\delta w) = \frac{T(t)}{\{\Delta E(\delta w)\}^2 + T(t)}$$

where δw is the proposed change in a given weight w, $\Delta E(\delta w)$ is the change in error elicited by a change δw in w, and $T(t)$ is the annealing *temperature* at time t. This variable is updated after all weights have suffered this process, as:

$$T(t) = \frac{T(0)}{1+ln(t)} \quad \text{for the Boltzmann distribution} \tag{2.45}$$

$$T(t) = \frac{T(0)}{1+t} \quad \text{for the Cauchy distribution} \tag{2.46}$$

being $T(0)$ the initial temperature. The change δw is accepted if $P(\delta w) > \xi$, where ξ is a random value uniformly drawn from $[0, 1]$.

2.1.7 Learning algorithms for RBF networks

Learning in this kind of networks is characterized by the separation of the process in two consecutive stages [Haykin, 94], [Bishop, 95]:

1. Optimize the free parameters of the hidden layer using only the $\{\vec{x}\}_i$ in D. This is an *unsupervised* method that depends on the *input distribution.*

2. With these parameters found and freezed, optimize the $\{c_i\}_i$, the hidden-to-output weights, using the full information in D. This is a *supervised* method that depends on the given *task.*

There are many ways of optimizing the hidden-layer parameters. The aim is to form a representation of the probability density function of the data, by placing the centers in only those regions of the input space where significant data are present. One commonly used method is the *k-means algorithm* [McQueen, 67], which in turn is an approximate version of the maximum-likelihood (ML) solution for determining the location of the means of a mixture density of component densities (that is, maximizing the likelihood of the parameters w.r.t. the data). The Expectation-Maximization (EM) algorithm [Duda and Hart, 73] can be used to find the exact ML solution for the means and covariances of the density. It seems that EM is superior to k-means [Nowlan, 90].

Once these parameters are chosen and kept constant, assuming the output units are linear, the (square) error function is quadratic in them, and thus the hidden-to-output weights can be fast and reliably found by the mentioned delta rule (§2.1.5), based on a simple gradient descent over the quadratic surface of the error function.

The parameters of a RBF can also be optimized with a global gradient descent procedure on all the free parameters at once [Bishop, 95](p. 190), [Haykin, 94](p. 267). This brings back the problems of local minima, slow training, etc, discussed in §2.1.4. However, better solutions can be found, essentially because the unsupervised solution focuses on better estimating the probability density function of the input data, but the resulting disposition may not be the one minimizing the square error when one takes into account the kind of RBF unit chosen and *especially*, the desired target outputs.

2.1.8 Evolutionary learning algorithms for ANN

As we have seen in (§2.1.5), derivative-based methods (DBM) make a number of assumptions about the local error surface and its differentiability. In addition, the existence of local minima is often neglected or overlooked entirely. In fact, the possibility of getting caught in these minima is more than often circumvented by multiple runs of the algorithm (that is, multiple restarts from different initial points in weight space). This "sampling" procedure is actually an implementation of a very naïve stochastic process.

The alternative to these methods are Evolutionary Algorithms (EA) [Bäck, 96]. Although the number of successful specific applications of EA is counted by hundreds (see, for example, [Bäck, Fogel and Michalewicz, 97] for a review), only Genetic Algorithms (GA) [Holland, 75], [Goldberg, 89] and, to a lesser extent, Evolutionary Programming (EP) [Fogel, 92], have been broadly used for ANN optimization, since the earlier works of [Montana and Davis, 89]. There are comprehensive review papers and guides to the extensive literature on this subject: see [Shaffer, Whitley and Eshelman, 92], [Yao, 93], [Kuşçu and Thornton, 94] and [Balakrishnan and Honavar, 95].

One of their main advantages over methods based on derivatives is the global search mechanism. A global method does not imply that the solution is not a local optimum; rather, it eliminates the possibility of getting *caught* in local optima. Another appealing issue is the possibility of performing the traditionally separated steps of determining the best architecture and its weights *at the same time*, in a search over the joint space of structures and weights. An additional advantage is the use of potentially any cost measure to assess the goodness of fit or include structural information. Still another possibility is to embody a DBM into a GA, using the latter to search among the space of structures and the DBM to optimize the weights; this hybridization leads to extremely high computational costs. Finally, there is the use of EA solely for the numerical optimization problem.

In the neural context, this is arguably the task for which continuous EA are most naturally suited. However, it is difficult to find applications in which GA (or other EA, for that matter) have clearly outperformed DBM for supervised training of feed-forward neural networks [Whitley, 95]. It has been pointed out that this task is inherently hard for algorithms that rely heavily on the recombination of potential solutions [Radcliffe, 91]. In addition, the training times can become too costly, even worse than that for DBM.

In general, Evolutionary Algorithms -particularly, the continuous ones- are in need of specific research devoted to ascertain their general validity as alternatives to DBM in neural network optimization. Theoretical as well as practical work, oriented to tailor specific EA parameters for this task, together with a specialized operator design should pave the way to a fruitful assessment of validity.

The emergence of evolutionary techniques using different selection schemes and a direct representation of variables allows to bring new light on evolutionary approaches to supervised neural network training. Among the latest algorithms we find the Breeder Genetic Algorithm (BGA) [Mühlenbein and Schlierkamp-Voosen, 93], which is characterised by a truncation selection procedure and direct representation of continuous variables. Truncation selection is a simple use of rank-based selection, proven to be very useful in traditional GA [Whitley, 89]. The direct representation of variables eliminates the need for a coding scheme -that usually changes the search space- and permits to develop new continuous genetic operators.

The GA and the BGA are the two evolutionary algorithms chosen in this Thesis for many of the weight optimization processes carried out. To this end, an entire chapter (§6) is devoted to them. In it, the basic algorithms are described, tested extensively (especially the BGA, the least known of the two) and extended in a number of ways to cope with more general kinds of neural models in ANN optimization tasks.

2.1.9 Handling heterogeneity in ANN

Although a feed-forward neural network can in principle approximate an arbitrary function to any desired degree of accuracy, in practice a pre-processing scheme is often applied to the data samples to ease the task. Attention is specially put on continuous variables, usually (linearly) scaled to the unit interval or to zero-mean, unit standard deviation.

However, in many important domains from the real world, objects are described by a mixture of continuous and discrete variables, usually containing missing information and characterized by an underlying vagueness, uncertainty or imprecision. For example, in the well-known UCI repository [Murphy and Aha, 91] over half of the problems contain *explicitly declared* nominal attributes, let alone other discrete types or fuzzy information, usually unreported. This heterogeneous information has to be encoded (or better, cast) in the form of real-valued quantities. We review here traditional ways of encoding non-standard information in ANN [Prechelt, 94], [Bishop, 95], [Fiesler and Beale, 97], [Sarle, 99].

Ordinal variables

These variables correspond to discrete (finite) sets of values wherein an ordering has been defined (possibly only partial). They are more than often treated as real-valued, and mapped equidistantly on an arbitrary real interval. A second possibility is to encode them using a *thermometer*. To this end, let k be the number of ordered values; k new *binary* inputs are then created. To represent value i, for $1 \leq i \leq k$, the leftmost $1, \ldots, i$ units will be on, and the remaining $i+1, \ldots, k$ off.

The interest in these variables relies in that they appear frequently in real domains, either as symbolic information or from processes that are discrete in nature. Note that an ordinal variable need not be numerical.

Nominal variables

Nominal variables are unanimously encoded using a 1-out-of-k representation, being k the number of values, which are then encoded as the rows of the $I_{k \times k}$ identity matrix.

Missing values

Missing information is an old issue in statistical analysis [Little and Rubin, 87]. There are several causes for the absence of a value. For example, they are very common in Medicine and Engineering, where many variables come from on-line sensors or device measurements, or are simply too costly to be measured at the same rate as other variables (e.g., analytical tests). The causes are so variate that we just mention but a few:

- Technical limitations (e.g. sensors working only in a limited range of values, or for given periods of time). This includes sensor malfunctioning;

- Measures costly to perform in time or money or involving destructive methods (e.g., maximum weight supported by a lift or data from car crash tests);
- Measures not done by a number of reasons, or done but in invalid conditions, or simply the data item gets lost during transcription, transmission or storage.
- Senseless values, related to other variables (e.g., number of times pregnant in male adults). These values are not missing *per se*. The value is not there because it does not make sense. A value of zero for male pregnancy is also senseless and uninformative, because it does not express that no other value is possible. In other words, it is not the same *zero* than for females [Belanche, 91].
- Reluctance to supply the value (e.g., salaries, phone or credit card numbers, etc).
- Different time intervals in measurements. For example, a given variable v_1 can be measured every 1 hour, and another one v_2 every two. Then, in a data set including both, every value out of two for v_2 would be missing.

Missing information is difficult to handle, specially when the lost parts are of significant size. It can be either removed (the entire case) or "filled in" with the mean, median, nearest neighbour, or encoded by adding another input equal to one only if the value is absent and zero otherwise. Statistical approaches need to make assumptions about or model the input distribution itself. We review here the current approaches.

There are two basic ways of dealing with missing data:

1. Completing the object description in a hopefully optimal way, to be defined (that is, "fill in the holes");
2. Extend the methods to be able to work with incomplete object descriptions.

Of course, the possibility of simply *discard* the involved data can not be considered as a "method" and is also frustrating because of the lost effort in collecting the information. This can be done if the number of missing values is very small or else they are concentrated only in some objects. In this case, the entire example is discarded.

In practice, it is not uncommon that missing values are distributed randomly (that is, according to an unknown distribution but independently of the observed values) and hence, if their quantity is not negligible, it is likely to affect a significant number of examples (in the worst scenario, just one missing value per example) so that discarding them all can not be afforded. Discarding information can be also dangerous if the missing pieces are related with the non-missing ones (e.g. if a given variable exceeds a threshold, another variable is always missing), because it may result in a change of input distribution.

The vast majority of methods in the literature of ANN belong to the first kind. These include:

- Fill in by a *constant* (e.g. 0) or a *random* value. A fixed constant value can only harm the distribution and is rarely considered. In the latter case, an unconditional density estimation method can be used, based on a single Gaussian where the mean vector and covariance matrix are found by using the available values. Then, missing values may be filled in by sampling the distribution. Such approach can lead to poor results [Ghahramani and Jordan, 94].

- Fill in by the mean, median or nearest-neighbour. This is in general not a good idea, since the most probable is that it results in a significant alteration of the distribution, since it ignores the covariance in the observed data [Bishop, 95](p. 301). A problem often neglected is the multimodality of the distribution: in this case these values are entirely useless and only introduce noise.

- Fill in by the previous or next value (if the objects show temporal dependencies).

- Set a regression model for any variable having missing values over the present variables and use the obtained regression function to fill in the holes. This tends to cause problems because, since the regression function is noise-free, it tends to underestimate the covariance in the data, as the filled-in points will fall along the regression line.

- Create a second variable which is zero if the first variable is present, and is one otherwise. In this latter case, the involved variable is filled by a zero. This scheme results in an extra parameter (that will have to be estimated) and is not completely satisfactory, because it does not reflect the fact the new input is closely tied to the old one, and does not represent any specific value.

- Fill in by the values obtained from a maximum-likelihood estimation method, such as the Expectation-Maximization (EM) algorithm [Duda and Hart, 73]. This algorithm can estimate the parameters of a mixture model (e.g. of Gaussians), by maximizing their likelihood, even in presence of missing data [Ghahramani and Jordan, 94]. Although sound, this method requires the modeling of the input distribution itself, which can be a difficult and time-consuming task.

  Learning in this approach is an estimation problem which requires an explicit probabilistic model. In general, maximum-likelihood methods cannot handle the more complex situation of missing input distributions, namely, whenever the probability that an input is missing is a function of this same input, unless a model of the missing data mechanism is also learned [Ghahramani and Jordan, 94].

  Other associated problems are the determination of the number of component densities, and how to manage new data. In a typical ANN application with missing values, the trained network is likely to encounter incomplete *test* patterns, which can come in a real-time basis. These new patterns would require a periodic re-estimation of the mixture model parameters. Another claim is that missing values should be treated by averaging and not by maximizing, as the EM algorithm does [Ripley, 92].

Another statistical technique is based on *multiple imputation*, of which Bayesian back-propagation is a special case [Buntine and Weigend, 91]. Each missing value is replaced by m values sampled from a prior distribution of inputs. After this multiple imputation, m complete data sets exist, which can be trained with conventional methods and combined to form a unique response. The prior distribution requires the use of iterative simulation methods [Schafer, 94]. A related method estimates this distribution from the data [Tresp, Ahmad and Neuneier, 94]. In general, a principled statistical treatment of missing values prescribes a weighted integration over the missing variables, which requires the entire input distribution to be modeled, and is computationally intensive.

The main problem with missing data is that we never know if all the efforts devoted to their estimation will revert, in practice, in better-behaved data. This is also the reason why we develop on the treatment of missing values as part of the general discussion on data characteristics. The reviewed methods pre-process the data to make it acceptable by models that otherwise would not accept it. In the case of missing values, the data is *completed* because the available neural methods only admit complete data sets.

Uncertainty

Vagueness, imprecision and other sources of uncertainty are considerations usually put aside in the ANN paradigm. Nonetheless, many variables in learning processes are likely to bear some form of uncertainty. In Engineering, for example, on-line sensors are likely to get old with time and continuous use, and this may be reflected in the quality of their measurements. In many occasions, the data at hand are imprecise for a manifold of reasons: technical limitations, a veritable qualitative origin, or even we can be interested in introducing imprecision with the purpose of augmenting the capacity for abstraction or generalization [Esteva, Godo and García, 98], possibly because the underlying process is believed to be less precise than the available measures.

In general, therefore, it is likely that when a given continuous variable takes the value, say, 5.1, we *know* that this is just a reasonable approximation and that the actual (precise) value is unknown. Thus, the value actually supplied to the neural network (or other learning system, for that matter) is "approximately 5.1". The important point is that *we* know this is an approximate value, but have not told the system so. In Fuzzy Systems theory there are explicit formalisms for representing and manipulating uncertainty, that is precisely what the system best models and manages. It is perplexing that, when we supply to ANN this kind of input data, we require the network to approximate the desired output in a very precise way, trying to reduce the cost function to zero up to the smallest possible decimal digit.

Sometimes the known value takes an interval form: "between 5.1 and 5.5", so that any transformation to a real value will result in a loss of information. A more common situation is the absence of numerical knowledge. For example, consider the value "fairly tall" for the variable *height*. Again, Fuzzy Systems are comfortable, but for an ANN this is real trouble. The issue is in fact neglected because the variable is treated as ordinal (e.g., "short", "medium" and "tall") will correspond to 1, 2 and 3. We already have seen how these values are then treated. What is more, "fairly tall" would be encoded as just another value.

It is then interesting to review the possible situations for an input variable, following the example of people's height: *exactly* 1.80551 m (rather unreal), approximately 1.81, "between

1.80 and 1.81" and "fairly tall". The transition from precise to vague knowledge makes more apparent the nature of real data.

The integration of symbolic and continuous information is also important because numeric methods bring higher *concretion*, whereas symbolic methods bring higher *abstraction*. Their combined use is likely to increase the flexibility of hybrid systems. For numeric data, an added flexibility is obtained by considering imprecision in their values, leading to fuzzy numbers.

Integration of heterogeneity

There have been very few attempts to incorporate heterogeneous information into the workings of a neuron in a principled way. The main contribution is perhaps that encountered in the heterogeneous distance proposals by [Wilson and Martinez, 97], where separate distance calculations are used for nominal and continuous variables. Their interest is, however, devoted to an heterogeneous k-nearest neighbours algorithm. The idea is further developed in [Wilson and Martinez, 96], where the authors present an extension of the RBF model to nominal quantities and missing values. The RBF network is used in its original interpolative definition (i.e. there is one hidden node for each example in the training set). The extension consists in the use of the Value Difference Metric (VDM, [Stanfill and Waltz, 86]) to nominal distance computation. An Euclidean metric is used to account for the partial distances. Missing values are handled with a *pessimistic* semantics, defining the distance involving a missing value to be the maximum possible distance (actually a value of one). The results point to an increase in performance w.r.t. standard Euclidean distance when the amount of nominal information is significant.

This work can be seen as a particular instance of our approach, in which a specific kind of similarity (a global distance-based) is built by Euclidean aggregation of partial measures. The emphasis is not put on the development of a generic heterogeneous similarity measure but in extending one of the standard distances, as a basic step towards the construction of more valid metrics. Ordinal or vague information is not considered. Furthermore, it can be argued that the VDM metric has two drawbacks: first, it is useful only for classification problems, since it is a frequentist measure. Second, *stricto sensu* it is not a metric on the input data, because it makes use of extra information (the class labels) and thus it cannot be considered as a distance measure defined on the input space.

The interest in data heterogeneity not only relies in providing accurate treatment for the continuous, discrete, symbolic and other common data forms. Making a step forward to the future, once this heterogeneity is acknowledged, other, more complex data types could be considered, as logical, probabilistic, pictorial, hierarchical, taking the form of a lattice, and so on. These forms of data are commonly found in psychological tests, opinion polls, police inquest forms, etc, and their representation is extremely delicate.

2.2 Similarity in AI

The notion of **similarity** has been studied in its own right more thoroughly in psychology —and, in particular, in cognitive psychology— than in artificial intelligence (AI). In cognitive psychology, formulation and representation of similarity is a key issue. Understanding how (and whether) biological (perceptual) systems represent and use similarities has been notoriously difficult to formalize, in part because of the feature selection problem, a manifestation of the frame problem in AI systems. None the less, there is an agreement in that it is easier to respond intelligently to a stimulus if one can recall previous responses made under *similar* circumstances. The key point for a succeeding perceptual[4] system (biological or artificial) relies in its ability to represent internally the similarities between the different stimuli [Edelman, 93].

Similarity apparently has a bad reputation in the scientific world, because of its general elusive character, receiving less attention than related concepts such as fuzziness, distance or preference [Dubois and Prade, 97]. Still, the popularity of case-based reasoning (CBR) [Aamodt and Plaza, 94] in AI has boosted its use and signaled its key role in cognitive tasks. As a matter of fact, some form of similarity is inherent in the majority of AI approaches, including connectionist methods, and very specially in kernel-based methods, which exploit the idea of distance-based similarities. This background ubiquity and a general lack of operational foundations (with notable exceptions, as [Osborne and Bridge, 96]) have prevented an explicit use outside CBR.

On the other hand, similarity coefficients have had a long and successful history in the literature of cluster analysis and data clustering algorithms [Everitt, 77], [Jain and Dubes, 88]. There are *heterogeneous* similarity measures —the most notable of which, those of [Gower, 71] and [Diday, 74].

2.2.1 Similarity in CBR

A wide range of psychological theories of cognition in humans are based on the assumption that many cognitive processes are best explained through generalization from a large number of stored instances. This generalization capability is in turn presumed to depend on the similarity between new and old (known) instances. Analogously, CBR systems typically retrieve and compare information (the cases) by applying a similarity measure, usually constructed *ad hoc*. Their effectiveness is known to be critically dependent on the chosen similarity measure. A given measure may maximize the accuracy of a learner in a given experiment and perform poorly as compared to others in another one. This issue is also related to the number of examples needed to represent a particular function, and brings one to the conclusion that there is no "universal" similarity measure that efficiently represents any target function [Globig and Lange, 96]. Rather, performance is sensitive to the goals, context and other information on the specific task (i.e., on prior knowledge).

[4]Here the word perceptual can be taken as *embedded in an environment*, which includes neural learning.

2.2.2 Similarity in ANN

Almost all neural methods are concerned, in one way or another, with the computation of a certain type of similarity between input patterns and neurons in the network. This includes unsupervised learning approaches, which are distance-based [Kohonen, 88]. Nevertheless, there is little work on the explicit use of similarity measures in the context of ANN and, among these, by far the most prominent measures are based on scalar product or Euclidean distance.

A special focus has been given to the enhancement of the k-nearest neighbours algorithm (KNN) [Fukunaga, 90], which has a unique capability in the immediate addition or deletion of pieces of knowledge, without a degradation in performance on previously acquired knowledge. This characteristic is missing in the majority of ANN methods. The poor generalization ability sometimes exhibited by KNN, however, is mainly due to its higher dependence on the chosen (and fixed) similarity measure and the presence of irrelevant variables, caused by an incorrect weighting of features. An interesting solution tries to bring the best of both approaches by proposing a parameterized similarity measure in a variable interpolation kernel, optimized with the conjugate gradient (§2.1.6). The similarity measure used is a Gaussian composed to a weighted Euclidean distance and the approach lies in the side of an enhanced KNN system [Lowe, 93].

Among connectionist systems that make an explicit use of similarity measures, work can be divided in two independent subjects of study. The first is concerned with the development of neural architectures capable of emulating the ability of the brain to learn smooth approximations of the function that maps a perceptual stimulus to its desired response [Poggio, 90]. Specifically, given a number of sample stimuli, the value for novel ones is found by linearly combining a set of appropriately placed base functions. The second approach involves the use of connectionist methods to discover distributed representations of *psychological* similarity relations out of previously measured similarity data.

Perceptual similarity

It may be observed that a biological system can only base its inferences about the world on the firing of its neurons [Bialek *et al.*, 91]. At any stage in a flow of neural activation, differences between input stimuli only matter insofar as they concern activity patterns of a preceding stage. A collection of graded and highly overlapping receptive fields can collectively form a distributed representation, sufficient for discriminating among highly complex stimuli such as images of human faces.

In these systems, the base functions compute a similarity based on a *distance* between the neuron center and the previous patterns of activity elicited by the incoming stimulus (i.e., by the input representation). This is in general the case of artificial neural models that work out the idea of neurons as receptive fields, centered at particular points in input space, generally known as RBF networks [Poggio and Girosi, 89]. These functions are either centered on the sample stimuli, or on some of their representatives. In [Edelman, Reisfeld and Yeshurun, 92], a two-stage scheme is proposed, in which each of the units in the first stage (or layer, in ANN terminology) is trained to respond strongly only to a given face f_i, and less to the rest (ideally lesser and lesser the less similar they are to f_i). The second stage or layer takes the collection of graded responses and combines them to classify the input face.

The same idea is developed in [Poggio and Hurlbert, 93] by proposing modules of Hyper Basis Functions (HBF) -a generalization of RBF- for object recognition. It is postulated that cortical neurons are tuned to respond vigorously to specific multidimensional stimuli, while decaying for less similar inputs. This behaviour is suitable to be captured by HBF.

Corresponding arguments have been applied to the optimization of topographic cortical mappings. In [Goodhill, Finch and Sejnowski, 96], it is postulated that "... the cortex tries to represent similar inputs close together, and that similarity is given by the degree of correlation between the activities of points (cells)."

The ideal would be to transform the data (via the similarity measure) so that entities similar in input space should elicit close values of the chosen similarity measure in the representation space. This idea can be found behind some proposals of artificial systems for basic visual processing (see, for example, [Weiss and Edelman, 94]), where images (visual stimuli) are compared via their representations (as similarities) at successive levels of a processing hierarchy, mimicking the processing known to be done by the population activities, in the way from the retina to the non-linear cortical units.

In the same vain, [Würtz, 97] introduces a system for object recognition robust under translations, distortions and changes in background. The stored images (called *models*) are compared with a given input image via a global similarity function, obtained by combination of similarities of local features at corresponding points. The correspondence maps between the image and a model are found by coarse-fine matching in a Gabor pyramid. The local measures are given by modified angular similarities, which are then arithmetically averaged.

Psychological similarity

Similarity relations are in the basis of the construction of psychological spaces (PS). These are geometric coordinate spaces in which stimuli are represented by points. The construction of PS involves the application of one of the family of multi-dimensional scaling algorithms (MDS, [Shepard, 80]) to empirically obtained data corresponding to pairwise similarities. The objective is to map the observed similarities into a metric space in such a way that the original relations are represented as faithfully as possible.

The use of a neural network is explored in [Lee, 97] to obtain a connectionist version of MDS. An adapted RBF is used. In this case, the usual emphasis in function approximation is put aside in favour of cognitive interpretability. The modeling of similarity computations —traditionally tackled in cognitive psychology by geometric and contrast models [Tversky, 77]— has been approached by [O'Sullivan and Keane, 92] and [Rumelhart and Todd, 93]. The ANN is used to emulate the cognitive process of producing a judgement of psychological similarity, following the presentation of two stimuli, and using as target data the desired similarity responses.

In all these approaches, there is a specific cognitive or perceptual task for which a neural network is used as a model or simply to obtain a connectionist implementation. Furthermore, the networks utilized are conventional, and non-standard information, if present, is encoded in the usual way to obtain an overall Euclidean representation of input stimuli.

2.3 Fuzzy Neural Networks

Fuzzy Sets [Zadeh, 65] (see [Klir and Yuan, 95] or [Zimmermann, 92] for a modern presentation) are generalizations of classical sets where the indicator function for membership is continuous instead of binary, expressing that elements can belong to a set to a certain *degree.* We review here some of the more basic concepts.

Definition 2.8 (Fuzzy set) *Let X be a classical set (called the* universe *or* reference *set) and $\mu_A : X \rightarrow [0,1]$ a function. The set of ordered pairs $A = \{(x, \mu_A(x)) \mid x \in X\}$ is a* fuzzy set *on X; μ_A is called the* membership *function of A.*

Definition 2.9 (Support set) *The support of a fuzzy set A is the (classical) set $supp(A) = \{x \in X \mid \mu_A(x) > 0\}$*

Definition 2.10 (α-set) *The (classical) set of all elements in X that belong to a fuzzy set A with at least a membership of α, $A_\alpha = \{x \in X \mid \mu_A(x) \geq \alpha\}$, is called an α-set of A, $\alpha \in [0,1]$.*

Classical subsethood is extended in the following way to fuzzy sets:

Definition 2.11 (Subsethood) *Let A, B be two fuzzy sets on X. The set A is said to be a fuzzy subset of B, denoted $A \subset_f B$, if $\forall x \in X \, \mu_A(x) \leq \mu_B(x)$.*

Definition 2.12 (Power set) *Let A be a fuzzy set on X. The (classical) set of all fuzzy subsets of A is denoted $\tilde{P}(A) = \{B \mid B$ is a fuzzy set on X, $B \subset_f A\}$, and denoted* fuzzy power set.

According to this definition, identifying X with the (fuzzy) set defined as $X = \{(x,1) \mid x \in X\}$, then $\tilde{P}(X)$ is the set of all fuzzy sets on a universe X.

Definition 2.13 (Normalization) *A fuzzy set A on X is said to be* normalized *if $\exists x \in supp(X) : \mu_A(x) = 1$.*

Definition 2.14 (Compact support) *A fuzzy set A on X is said to have* compact support *if $\forall x,y,z \in supp(X) : x < y < z \wedge \mu_A(x)\mu_A(z) \neq 0 \Rightarrow \mu_A(y) \neq 0$.*

Definition 2.15 (Convexity) *A fuzzy set A on X is said to be* convex *if $\forall x,y,z \in supp(X) : x \leq y \leq z \Rightarrow \mu_A(y) \geq min(\mu_A(x), \mu_A(z))$*

Fuzzy Control systems [Pedrycz, 93] were originally created to accept uncertain data, to process them, and then to yield output values (crisp or fuzzy) to have control on some system, which in turn delivers new input data. This input/process/output operation is also of the kind performed in a feed-forward neural network. Thus, it is reasonable to think in hybrid ways of combining the two approaches.

In recent years, the hybridization of methods has become a very popular avenue of research. There are many benefits in joining different approaches, designing systems that make full use of every method in those tasks for which it is best. In the case of neural networks and fuzzy systems, it has lead to so-called *Neuro-Fuzzy Systems* [Nauck, Klawonn and Kruse, 97]. These are basically fuzzy systems (a fuzzy rule-base plus a fuzzy inference engine) enhanced in some way by means of ANN techniques. A typical situation is the use of neural learning algorithms to determine the parameters of the fuzzy sets in the rules. The overall system is always to be interpreted in terms of fuzzy IF-THEN rules. These constitute the vast majority of the approaches.

By considering fuzzy information in some of the input lines, the approach developed in this Thesis is more related to *fuzzy neural networks* [Gupta and Rao, 94]. These are ANN that use fuzzy methods to enhance the learning capabilities or to perform better in a more general sense. The primary intention is to *improve* on the neural network and the above interpretation is neither important nor possible.

These networks are fully fuzzified, in the sense that they are conventional networks where all the information is treated as fuzzy numbers and the network is working in fuzzy arithmetic. The corresponding learning algorithms can also be fuzzified [Lippe, Feuring and Mischke, 95]. In this latter work, which subsumes previous attempts [Hayashi, Buckley and Czogola, 92], the authors propose a classical MLP that accepts triangular fuzzy numbers (see §4.3) as well as crisp (real-valued) information (as degenerate fuzzy numbers). The back-propagation algorithm is adapted as a "fuzzy back-propagation" by applying the Extension Principle [Zadeh, 73]. These networks offer a number of advantages, of which the most obvious are the possibility to process imprecise data and to approximate fuzzy functions. The authors also claim that the fraction of input space covered by the (now fuzzy) training data is bigger than in the real case. Finally, the fuzziness of the weights can be interpreted as a means of softening the error surface, which can alleviate the possibility of getting trapped in local minima.

This approach is appealing, though it has several inherent drawbacks. First, triangular fuzzy numbers (TFN) lack a group structure under multiplication; specifically, the product of two TFN is no longer a TFN. Thus, linear approximations are needed to bring back the result of operations to the set of TFN. Second, the training times are superior because each atomic processing (e.g., a product of input by weight) requires three times the old computation plus the linearization operations. Third, fuzzy neural networks are not universal approximators in the classical sense -see Chapter (5). Rather, only the fuzzy *monotonic* functions can be approximated arbitrarily well by such a network [Feuring and Lippe, 95]. Monotonicity is in the fuzzy-function sense: being $\tilde{x}$ a fuzzy number and $\tilde{f}$ a fuzzy function, if the fuzziness of $\tilde{x}$ increases, then the fuzziness of $\tilde{f}(\tilde{x})$ increases.

In the present work, the fuzziness in the ANN is taken in the mentioned generic sense, to directly account for vague or imprecise knowledge, and increase the overall flexibility of the model. This may or may not imply to work in fuzzy arithmetic. An heterogeneous neuron computes a similarity function between its input and weight vectors. The proposed measures for computing similarity between two fuzzy numbers are functions of the form: $s : \mathbb{F}_n(X), \mathbb{F}_n(X) \rightarrow [0,1]$, where $\mathbb{F}_n(X)$ is the set of all fuzzy numbers on a universe X. Thus, if the p-dimensional input space is taken to be $(\mathbb{F}_n(X))^p$, we have a network that accepts all fuzzy inputs but is still working on real arithmetic.

However, as pointed out in (§4.4), other neuron models are possible, based on other similarity measures. In particular, if we devise an $s_n : \mathbb{F}_n(X), \mathbb{F}_n(X) \rightarrow \mathbb{F}_n([0,1])$, then we obtain a network that outputs a fuzzy similarity value in response of fuzzy inputs and its fuzzy weights (a natural situation). The chosen measure s_n should account for the nature of the problem (a direct choice is the fuzzification of a measure s). In any case, we obtain a fuzzy neural network. Thus, this kind of networks are a special case of the Heterogeneous Neural Networks (HNN) introduced in Chapter (4).

The advantage is that in HNNs we are not tied to the MLP neuron model, or to differentiable, non-linear, transfer functions and thus other, possibly simpler models can be devised that output a TFN (expressing a similarity in $[0,1]$) in response of the input and weight TFN, and that avoid the use of any product operation. The derivation of these fully fuzzified networks, however, falls out of the scope of this Thesis.

Chapter 3

Similarity as a basis for Neural Computation

"From causes which appear similar, we expect similar effects.
This is the sum total of all our experimental conclusions."

David Hume

The general view of an ANN as a similarity computing device is readily acknowledged in the ANN community. However, the majority of neural networks used in practice are based on a specific neuron model, namely the dot product one which, as seen in the Introduction, severely restricts the similarity computing capability of the network. On the other hand, RBF-based models narrow the guiding conception to distance-based similarities on metric spaces. Broader, more general and richer families of models can be obtained and studied if this general view of a neuron as a similarity computing unit is explicitly taken into account. To begin with this line of thought, a groundwork has to be set forth in order to develop a suitable base for a systematic investigation of the approach.

3.1 Preliminaries

The origins of learning by similarity in artificial neural systems can be traced back to the pioneering works of Hebb and his now classic book *The organization of behaviour* [Hebb, 49]. He postulated that the functionality of ANN had to be determined by the strengths of the connections between neurons and that this functionality should be adjusted to *increase* the likeliness of getting a *similar* response to *similar* inputs in the future, provided the elicited response is the desired one. In the opposite situation, the weights should be adjusted to decrease this likeliness.

In current ANN research, there are basically three "degrees of freedom":

1. The neuron model: what is the precise function performed by a single unit. This

includes the choice of the neuron's input and output spaces;

2. The architecture: how these neurons are to be arranged;
3. The learning algorithm: how the weights are to be determined.

There is a fourth element, the cost function: how is the goodness of a solution measured. This topic is well understood and two functions (the square error and the cross-entropy) have become fairly standard.

Undoubtedly, the most active of these three subjects is the third. Once the neuron model and architecture are decided, the problem of choosing the most suitable way of proposing a solution in weight space (fast and reliably) becomes the most important area of study. It is in fact remarkable how the majority of research attention in ANN has been devoted to the proposal of new learning algorithms or the improvement or somehow enhancement of existing ones (much more usual). Some of the techniques have been rediscovered several times or have been recast from methods known in statistics or other disciplines. Studying new variations in architecture, on the other hand, is of interest mostly for recurrent systems.

Still, it can be argued that there are at least *three* components strongly influence the task of finding a solution to a supervised ANN learning problem:

1. The *repertoire* of functions the network has, in order to approximate the target function;
2. The *function* that is *actually* being presented to the ANN via input/output examples;
3. The *algorithm* the network uses to select one of the functions of its repertoire.

This is true at least for the two most widespread neuron models, reviewed in (§2.1), the one used in MultiLayer Perceptrons (MLP) —basically a scalar product between the input and weight vectors plus an offset followed by a squashing function— and the one used in Radial Basis Function Neural Networks (RBF) —some distance metric between the input and weight vectors followed by a monotonic basis function (for instance, a Gaussian). In this latter case, only a hidden layer suffices.

By focusing only in these two neuron models or variants thereof, we are actually limiting the families of functions in point 1. In theory, very many functions can in principle work well (because they have a universal approximation capability, for example) but *in practice* the results can be very different for different selections, and the cornerstone for this selection is precisely the neuron model. Given a certain fixed network arrangement (such as the feed-forward), the practical computing capacity is completely determined by the repertoire of neuron models available. Final performance is also strongly influenced by point 2.; how exactly is the target function presented to the network will greatly condition the results. A more elaborate discussion follows.

Regarding the first point, one marked shortcoming of the neuron models existent in the literature is the difficulty of adding knowledge (either of the problem to be solved or of the data themselves) to the model. The neuron is used as a function (that is, a mapping) from

$\mathbb{R}^n$ to $\mathbb{R}$ (possibly, though rarely, with an internal state) as a black-box, for which the internal workings are often obscure. For such a blind processing element, part of the task is to find a structure in the data, to transform it from a *raw form* (possibly in a space of high dimension, only here and there covered by example cases) to a new space (the ***hidden space***, or space spanned by the hidden units) in such a way that the problem is simpler when projected to this new space. The same processing is repeated for all the subsequent hidden layers. In any case, the hidden layer's task is to find a new, more convenient representation for the problem *given* the data representation chosen, and eventually performing a reduction in dimensionality.

This leads directly to the second point, the way data are represented, which is extremely important. It not only includes which information is explicitly fed into the network —usually in the form of vectors of selected variables— and which is not. It also includes:

1. Input (and sometimes, output) preprocessing, mostly the normalization of variables or cases;
2. The way non-standard information is encoded, e.g. the representation of (discrete) ordinal and nominal variables;
3. The way missing values are handled.

Data representation can be a crucial factor for a succeeding learning process and can also have a great impact on generalization ability [Bishop, 95]. As Hinton points out, the correct generalization from a training set must depend on how the input and output vectors are encoded and, once a network has been correctly trained, the same network can be used to generalize in any other way we choose, simply by devising appropriate input and output transformations [Hinton, 89]. The task of the hidden layer transformation is therefore twofold: first, detection of data structure, redundancy and relative relevance of variables under the chosen representation and, second, approximation of the actual underlying function.

The difficulty of the general learning task is then exacerbated because, in the ANN paradigm, the data (and with them, the problem itself) are adapted to the neuron model (and, possibly, to the learning algorithm), and not otherwise. Instead, by making use of domain or *prior* knowledge, we shall be able to define a model more fit to the data. Prior knowledge is any information we possess about the problem itself *and* about the nature of its data or about the desired form of the solution, apart from the functional samples. If all this information (which is assumed relevant) is supplied to a neural network or included in its design, it shall not have to be discovered.

3.2 Scalar product as a pattern recognizer

Since the early days of pattern recognition, artificial neuron models were designed as computers of a "recognizing function" (giving rise to the name of the field). As such, one of the first activation function proposed was the Heaviside:

$$H_\theta(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ 0 & \text{if } z < \theta \end{cases} \tag{3.1}$$

This function was motivated by the logical analysis of computer circuits and the metaphor of computers as brains. It was used in the McCulloch & Pitts model [McCulloch and Pitts, 43], in many of the first non-linear associative memories (such as the *lernmatrix* [Steinbuch, 61]) and, fundamentally, in the Perceptron [Rosenblatt, 62]. The binary output reflected the original closeness to the biological counterpart. When an input vector (or pattern) is supplied to the neuron, its response is either 1 or 0, standing for "I *do* recognize it" or "I *do not* recognize it". The recognition part is dealt with by the aggregation function, the ubiquitous scalar (or inner) product. It is worth digressing a little about the reasons why this way of aggregating vectors has been so widely used.

In the first place, there is the once appealing analogy to the biological counterpart (the original inspiration). The slow, analog integration of incoming connections of a neuron, each one acting as (strong or weak) excitation or inhibition channels was modeled (see previous remarks on the abstraction process) by an instantaneous and discrete "integration" operator: the weighted summation. This leads to one of the definitions of scalar product.

$$\vec{x} \cdot \vec{y} = \sum_i x_i y_i \tag{3.2}$$

In the second place, the definition of the angle between two vectors can be related to their scalar product:

Definition 3.1 *The angle θ between two non-null vectors, $\vec{x}, \vec{y}$ is defined as*

$$\cos\theta = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\|\|\vec{y}\|} \in [0, \pi] \tag{3.3}$$

where $\|\cdot\|$ is the Euclidean norm.

The geometric interpretation of this definition is depicted in Fig. (3.1). Let us suppose that $\vec{x} = (x_1, x_2)$ and $\vec{y} = (y_1, y_2)$ and, without loss of generality, that $\|\vec{x}\| < \|\vec{y}\|$.

From trigonometry we have $\frac{r}{\|\vec{x}\|} = \cos\theta$ and therefore

$$r = \|\vec{x}\|\cos\theta = \frac{\vec{x} \cdot \vec{y}}{\|\vec{y}\|} \tag{3.4}$$

is the fraction of $\|\vec{x}\|$ that is pointing in the direction (with sign) of $\vec{y}$. Notice that if $\vec{y}$ is parallel to the x-axis, then $r = x_1$. In other words, $\vec{x} \cdot \vec{y}$ can be seen as $\|\vec{y}\|$ times the projection of $\vec{x}$ on $\vec{y}$, or vice versa ($\|\vec{x}\|$ times the projection of $\vec{y}$ on $\vec{x}$).

This *bounded* fraction of $\|\vec{x}\|$ gives an idea of how close or separate (in θ) are $\vec{x}$ and $\vec{y}$. It can also be regarded as how close are *the points* represented by $\vec{x}, \vec{y}$, as measured by $\|\vec{y} - \vec{x}\|$;

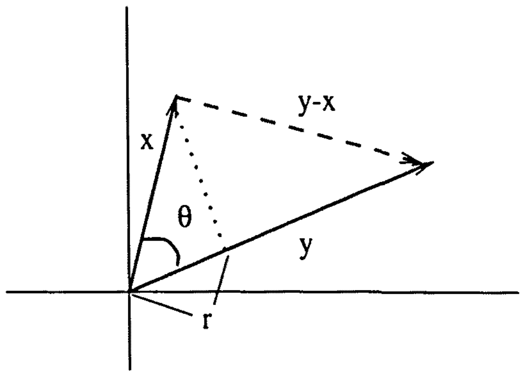


Figure 3.1: Two-dimensional geometric interpretation of scalar product.

the smaller the norm, the greater is r, and with it $\vec{x} \cdot \vec{y}$ (for constant $\vec{y}$), showing the relation between a model working on vector scalar product and one working on a norm of the vectorial difference. The value r is bounded by

$$-||\vec{x}|| \leq r \leq ||\vec{x}|| \tag{3.5}$$

The lower bound corresponds to the two vectors being colinear and going in opposite directions. The upper bound signals colinearity and the same direction. The magnitude is hence only a scalar factor that amplifies the value given by the angle. Therefore $\vec{x} \cdot \vec{y}$ is greater:

1. The closer (in θ) are $\vec{x}$, $\vec{y}$;
2. The longer (in $|| \cdot ||$) are $\vec{x}$, $\vec{y}$.

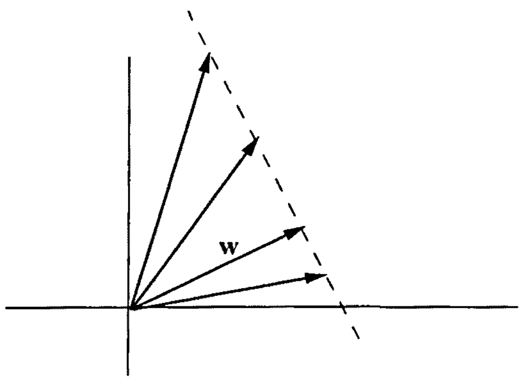


Figure 3.2: The scalar product between $\vec{w}$ and the other three vectors is constant.

leading to points of constant value lying in parallel hyperplanes -Fig. (3.2). Note also that, being the scalar product commutative, the initial assumption about their relative magnitude

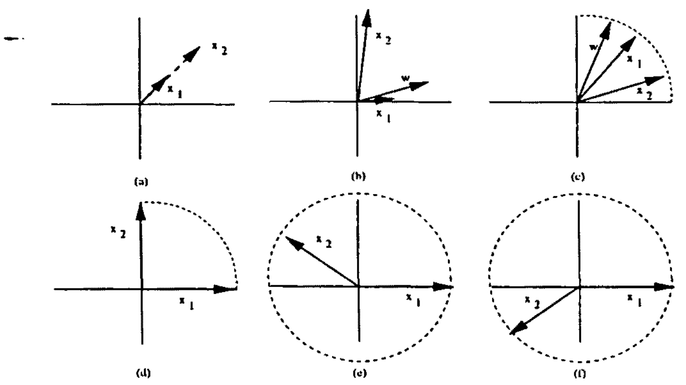


Figure 3.3: (a) Co-linear vectors ($\theta = 0$), $|\vec{x}_1 \cdot \vec{x}_2|$ is maximum. (b) $\vec{w} \cdot \vec{x}_1 < \vec{w} \cdot \vec{x}_2$ although $\vec{x}_1$ is more similar to $\vec{w}$ than $\vec{x}_2$ is. (c) Normalized vectors all lie on a circumference of radius one; $\vec{w} \cdot \vec{x}_1 > \vec{w} \cdot \vec{x}_2 > 0$ ($\theta \in [0, \frac{\pi}{2}]$). (d) Scalar product for orthogonal vectors is 0 ($\theta = \frac{\pi}{2}$). (e),(f) Scalar product is negative for $\theta \in [\frac{\pi}{2}, \frac{3\pi}{2}]$.

is irrelevant. Some of the possible situations are depicted in Fig. (3.3). In general, we have $|\vec{x} \cdot \vec{y}| \leq ||\vec{x}||\ ||\vec{y}||$ (the Cauchy-Buniakowski inequality). If magnitude effects are to be eliminated, normalization is necessary:

$$\vec{x}\,' = \frac{\vec{x}}{||\vec{x}||}; \qquad \vec{y}\,' = \frac{\vec{y}}{||\vec{y}||} \tag{3.6}$$

so that $\vec{x}\,' \cdot \vec{y}\,' = cos\,\theta \in [-1, +1]$ gives a measure of how *similar* $\vec{x}\,'$, $\vec{y}\,'$ are, traditionally used in statistics (Q-mode factor analysis) as a measure that preserves proportions. This makes scalar product a simple but effective *recognition* function, linear, continuous and differentiable. A neuron model $F_i(\vec{x})$ with weight vector $\vec{w}^i$ endowed with this measure will thus compute *the projection of* $\vec{x}$ *on* $\vec{w}^i$, multiplied by $||\vec{w}^i||$:

$$F(\vec{x}) = \vec{w}^i \cdot \vec{x} = ||\vec{w}^i||\ \underbrace{||\vec{x}||cos\,\theta}_{\text{projection}} \tag{3.7}$$

The problem of this measure is that, although it is *locally* bounded for a specific pair of vectors (the above inequality), the vectors themselves are not, and hence its usage as an explicit recognition device is severely limited. Variable standardization can be of help. A common input preprocessing scheme is to bring variables to zero mean and unit standard deviation. Chebyshev's inequality -see Appendix (D)- tells us that, in these conditions, by choosing a $k \in \mathrm{N}^+$,

$$\Pr(|x_i| > k) \leq \frac{1}{k^2}$$

and therefore we could propose a "bound" for $\vec{x}$ as $\vec{x}_{max} \approx (k, k, \ldots, k)$ so that $||\vec{x}_{max}|| = k\sqrt{n}$. A value of $k = 7$ should suffice (probability is upper-bounded by ≈ 0.02). However, the lack of an explicit control of the neuron (Is this really a good bound? How are the weights bounded?)

and the dependence on the k parameter[1] still suggest that a modification of the model (not of the data!) can lead to improved general (similarity) computing devices.

In conclusion, although not a proper similarity measure (we will see in (§4.2.5) that it does not fulfill one of the basic requirements to be called such) the conceptually nice dual view of scalar product as a weighted summation or as an "absolute" similarity (along with its clean analytical behaviour) is the rationale behind its use from the outset until today. Besides, both interpretations are easily linked by noting that (3.2) is in fact the Cartesian equation of the hyperplane written in a compact form:

$$\pi : x_1 w_1 + x_2 w_2 + \ldots + x_n w_n = 0 \tag{3.8}$$

with direction vector $\vec{w}$ (orthogonal to the plane) and constrained, in absence of an independent term, to pass through the origin. The situations in Fig. (3.3) can now be looked at again by taking one of the vectors as a reference and imagining a line perpendicular to it passing through the origin: it will be the hyperplane of which this vector is a director vector. Any other vector lying in the same side of the hyperplane as the reference will yield a positive scalar product, greater with increasing distance *to the hyperplane.* The situation is analogous for vectors lying in the opposite side and (increasingly) negative scalar products. Of course, any vector lying *on* the hyperplane will be orthogonal to the reference one, and will yield a null scalar product. This is the basic principle of the way Perceptrons work.

The basic Heaviside function was later generalized by softening it in a controlled way, leading to the family of squashing activation functions, of which the sigmoidal functions are well-known members. The main property of these functions (the "squashing property") is their asymptotic behaviour in the x-axis, which implies they are bounded functions. As an example, consider the logistic parameterized function:

$$g_{\beta,\theta}(z) = \frac{1}{1 + e^{-\beta(z-\theta)}}, \qquad \beta > 0 \in R \tag{3.9}$$

It can be shown that $\lim_{\beta \to \infty} g_{\beta,\theta}(z) = H_\theta(z)$. The net effect of a squashing activation function (not considering its effects on a derivative-based learning algorithm) can be regarded as producing a softened up recognition, a normalized (because of the boundedness characteristic) *degree of recognition.* Notice that this function, being a strictly increasing monotonic one, does not alter the discussion so far.

3.3 Similarity as a basis for Neural Computation

In theory, the design of artificial neural networks should follow the principle:

Principle 3.1 *Similar patterns should yield similar outputs.*

[1]Note that there is a compromise between high values of k (to avoid overflowing the bound) and low values (to prevent setting an unrealistically high bound, which would make the neuron hardly sensitive).

Although the smoothness criterion in (2.17) can be taken in some sense as a functional way to favour this principle, what "similar patterns" means is problem-dependent, and only in counted occasions will coincide with the fixed interpretation of similarity that the network is going to perform. There is even a more basic principle [Haykin, 94] to be satisfied:

Principle 3.2 *Similar patterns should have similar representations.*

The fulfillment of this principle easies the fulfillment of the former, although it does not directly imply it, because neuron models will make use of their own fixed "similarity measure" on the representations -see Fig. (3.4).

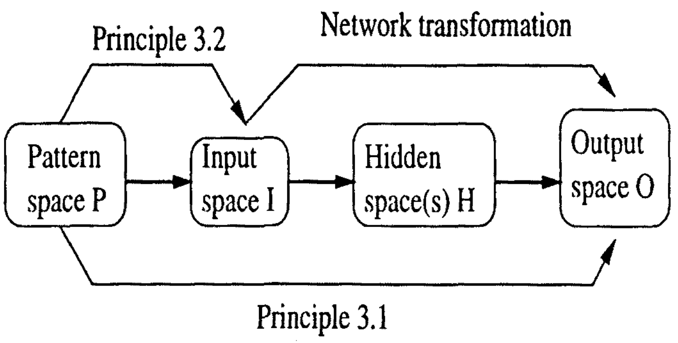


Figure 3.4: A Neural Network as a similarity transformer: The similarities in each space should be such that: *i*) $s_I(i,i')$ captures the original likeness s_P in pattern space, and *ii*) $s_I(i,i') \Rightarrow s_H(h,h') \Rightarrow s_O(o,o')$ (I :input, H :hidden, O :output).

Consider the example depicted in Table (3.1), where patterns include natural numbers (a situation much more common than not). Principle (3.2) says that their representation (the actual *input space* for the network) has to be respectful with their nature. If these pattern components are represented as, say, binary-coded bitstrings (which only respect the characteristic of being discrete) then Principle (3.2) is being violated and, with it, Principle (3.1). This is because the similarity behind natural numbers (possibly based on some metric in $\mathbb{N}$) does not correspond at all to that used in the same binary-coded numbers actually fed to the neurons. For example, two correlative numbers in $\mathbb{N}$ as 31 and 32 (thus very similar) are far in Hamming distance. This change of representation is not part of the original task and has to be additionally discovered by the network.

pattern space	input space
$\ldots, 30, 31, 32, \ldots$	$\ldots, 011110, 011111, 100000 \ldots$
(Metric in $\mathbb{N}$)	(Hamming metric)

Table 3.1: Representation of natural numbers as a binary code.

This is exacerbated in the case of scalar-product driven neurons, for which magnitude will also be important, and with magnitude into account the computation is further distorted. As an example, consider two vectors in $\vec{x}_1, \vec{x}_2 \in [0,1]^6$, depicted in Table (3.2).

	Case 1	Case 2
$\vec{x}_1$	111001	001001
$\vec{x}_2$	111111	010001
	$d_H(\vec{x}_1, \vec{x}_2) = 2$	$d_H(\vec{x}_1, \vec{x}_2) = 2$
	$d_E(\vec{x}_1, \vec{x}_2) = \sqrt{2}$	$d_E(\vec{x}_1, \vec{x}_2) = \sqrt{2}$
	$\vec{x}_1 \cdot \vec{x}_2 = 4$	$\vec{x}_1 \cdot \vec{x}_2 = 1$

Table 3.2: Influence of magnitude *in the representation* (d_E, d_H stand for the Euclidean and Hamming distances, respectively).

In both cases (1 and 2), scalar product yields quite different values, despite the fact that the vectors $\vec{x}_1, \vec{x}_2$ are equally near in Hamming and Euclidean distance. As for RBF units note that, denoting by b_2 the "base 10 to base 2 operator", it can be checked that

$$d_E(b_2(i), b_2(i')) = \sqrt{d_H(b_2(i), b_2(i'))} \; i, i' \in \mathrm{N} \tag{3.10}$$

This means that a neuron model based on Euclidean distance (instead of Hamming) on the bit representations will neither capture the original metric in N (assumed also Euclidean), because the change of representation relates all computations to the Hamming distance, thus violating Principle (3.2). Additionally, artificially augmenting the input dimension can only harm local approximation schemes like that performed by a RBF network.

All this means that, even if we devise a representation that is reasonably respectful to Principle (3.2), there is no guarantee that Principle (3.1) will be respected, because of the peculiarities induced by the neuron model. Additionally, the possibility of using a Gray code, for instance, would only fulfill Principle (3.2) locally, rendering it almost useless[2]. Even if we could make Euclidean distance behave proportionally in the representation, this would still be based on the assumption that the actual measure that should be used in general in N or one of its subsets is the Euclidean, certainly far from reality in some situations[3].

Therefore, the two requirements to ensure that both Principles are to be satisfied can be resumed in the following:

1. To identify and understand what does *similar* mean for the task at hand and propose a neuron model that best captures this functional similarity, at least for the first layer of neurons (the layer in contact with the input space). This deals with the *pattern similarity*;

2. To choose a representation —if still needed— that fulfills Principle (3.2). This has to do with the *pattern heterogeneity*;

[2]In a Gray code, two consecutive naturals would differ in only one value. This is a useful coding wherever a small change in representation involves a small change in the represented entity (as, for example, in Genetic Algorithms). However, medium or big differences do not elicit correspondingly high Hamming distances.

[3]Think of a quantity representing a date, ranging in 1..31; the circular behaviour should be realized somehow by altering the basic distance. For a variable (taking natural values) standing for the number of children, the distance between 7 and 9 is not the same "psychological" distance than that between 1 and 3 (which is triple). In both situations, domain knowledge is being used.

In current approaches these two requirements are more than often neglected, or let to the learning process. The following two sections elucidate these two topics (similarity and heterogeneity) in more detail.

3.4 Similarity in Artificial Neural Networks

The term *physical similarity* is coined to reflect the fact that the practical totality of approaches interpret the collection of input variables, whatever they are meant to represent, as the coordinates of a point in the n-dimensional Euclidean space $\mathbb{R}^n$. As a consequence, the appealingly simple and general physical similarity computation performed by a neuron working on scalar product or Euclidean distance has already been put in doubt as being truly general or conceptually right in all situations [Rumelhart *et al.*, 93], because it may not be the most suitable to capture the required *functional* or *psychological* similarity[4]. There are two basic *conceptual* flaws with this "raw" similarity computation.

The first one is the dependence on magnitude. Since the input variables are treated as vectors, their length is important. As we saw in previous sections, scalar product gives a value that is greater when *either* the angle is smaller or the product of magnitudes is greater. This implies that the measure is potentially unbounded, an undesirable feature, because if a function is to express a recognition degree, this degree has to be relative to some maximum. Of course, it can be argued that, at least in practical applications, the target function will have a compact support, and thus the "maximum scalar product" will be bounded. This remark is not useful since, on the one hand, we may not know the actual input distribution (it might, as is often the case, be covering a submanifold of the input space) and thus the theoretical maximum is an unreachable one and the actual maximum is unknown. On the other hand, what is the maximum length of a weight vector? Although in practice small weights are beneficial, there is no clear way of setting an a priori limit. Moreover, some recent experiments have shown that the range of possible weights strongly conditions the results of a learning process [Sopena, Romero and Alquézar, 99]. In conclusion, raw scalar product is an uninformative measure of similarity. In the classical neuron model (that of the MLP), the squashing activation function does part of the job: its boundedness permits to see the whole neuron computation as a bounded similarity function. However, it does not solve the problem to satisfaction: although the activation function is bounded, *its argument* is not, so that the function may be too easily brought close to its extreme values, loosing the sensibility the neuron should have. What is more, the unit is still computing the "physical similarity" which, quoting [Rumelhart *et al.*, 93]:

> "... may not be the best measure of the *functional* or *psychological* similarity we would like to employ at the output." (my italics)

This is the second basic flaw, easily illustrated in the following examples. Consider the three triangles of Fig. (3.5). To what other figure is more similar triangle number 1? To

[4]The reader may be interested in a more formal explanation of why Euclidean distance and scalar product are so closely tied to the Euclidean space $\mathbb{R}^n$, given in Appendix (A).

number 2 (because, although is bigger, it keeps the same proportions), or to number 3 (because is of the same size, though with different proportions)? One could argue that number 2 is *the same* triangle as number 1, though slightly scaled up and, thus, they should be treated as similar or, better, *equal.* This first example tells us that a similarity judgement depends on what input patterns really mean to us and how should they, as a consequence, be treated. This is an old problem in cognitive psychology and other disciplines.

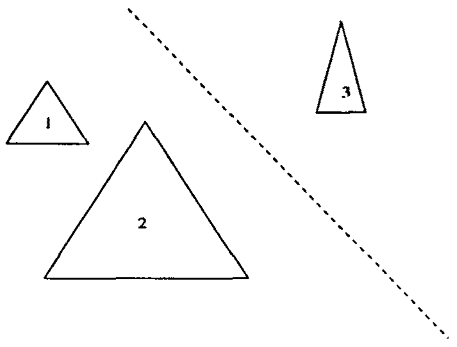


Figure 3.5: What triangle is more similar to triangle number 1?

As another example, consider a typical character recognition task, in which characters are encoded as binary (or continuous, for what matters) pixels in a grid, which is presented to a network as a vector in a space of very high dimension (the number of pixels in the grid) or, better, as a vector of features in the form of pixel combinations. The classification of the letters in the set $\{a, \ldots, z, A, \ldots, Z\}$ is clearly problem dependent. The precise classification the user has in mind (e.g., lower case vs. upper case or a/A vs. b/B vs. ... z/Z, or even vowels vs. consonants, etc) will surely not be the one the network "sees". In other words, patterns $\vec{x}$ that are similar to a neuron (because the inner product of $\vec{x}$ with its weight vector is high) need not (surely will not) be similar in the eyes of the user (read problem). In fact, the learning task consist precisely on this: to show the network what is the actual similarity relation. In a practical data clustering scenario reported in [Chandon and Pinson, 81], four skulls, one of an adult *Homo Sapiens*, one of an infant *Homo Sapiens*, one of an adult gorilla, and one of an infant gorilla had to be told apart. One reasonable outcome is to have both *Homo Sapiens* skulls grouped together (that is, to ignore the differences in skull size of adults and infants). The other is to have the two adult skulls separated from the two infant ones. These are different tasks; in both cases, though, the network is performing the *same* basic transformation. In these examples, the main problem is the dichotomy "what input patterns really mean to us"/"what they mean to a neuron". Much of the learning process is devoted to narrow this gap.

What is actually needed is a transformation to another representation form in which patterns requiring similar (for the problem) responses are indeed similar to one another (for the network). This is classically performed, in theory, by the hidden layer(s) in multilayer networks. Quoting again [Rumelhart *et al.*, 93]:

"... if we use a sequence of such transformations [the hidden layers], each involving certain non-linearities ... we can entirely rearrange the similarity relations among the original input vectors."

This view is conceptually right and task independent. In practice, it may be that for complex transformations several layers are needed, or very many neurons per layer if we restrict the maximum number of hidden layers to, say, one or two. In all, this is giving support to a deeper and more precise formulation of neuron models as similarity computing devices. If more information is implanted into the network design in the form of supplying a neuron model —at least for the input layer— which captures better the required similarity in the input space (which is also the space where the input-layer weights belong), the task of finding the required transformations may quite possibly be simpler to learn.

3.5 Heterogeneity in Artificial Neural Networks

The requirements involved with the fulfillment of Principle (3.2) bring us straight back to input representation. Real-world data come from many different sources (continuous or discrete numerical processes, symbolic information, etc.) and have their own peculiarities (vagueness, uncertainty, incompleteness), and thus may require different treatments. This heterogeneity is traditionally coped with by *preparing* the data using a number of coding schemes. However, this preprocessing is not part of the original task and may involve an abrupt change in input distribution and an increase in dimension.

These new high-dimensional patterns entail a lower data density, which in turn requires stronger, more accurate constraints on the problem solution [Cherkassky and Mulier, 98]. There is also a corresponding growth in the number of weights the network is forced to learn, their interpretation, the increase in training time, in the complexity of the error surface, and so on. We saw in (§2.1.9) the most commonly encountered kinds of non-standard data and how they were handled, if at all. In the following, the flaws of these encodings are discussed in detail.

3.5.1 Nominal information

For nominal variables with k values, the coding using a 1-out-of-k representation introduces the rows of the $I_{k \times k}$ identity matrix in the training set. This leads to an *structured* increase of model input variables, which translates in a notable increment in the number of parameters.

	x_i	.	.	.	x_{i+k-1}	
⋯	1	0	0	0	0	⋯
⋯	0	1	0	0	0	⋯
⋯	0	0	1	0	0	⋯
⋯	0	0	0	1	0	⋯
⋯	0	0	0	0	1	⋯

Table 3.3: A slice of the data set.

Let $[x_i, \ldots, x_{i+k-1}]$ be a "slice" of length k of $\vec{x} = (x_1, \ldots, x_n)$ (the entire input vector), corresponding to a given nominal encoding, as depicted in Table (3.3). The following statement holds:

$$\exists j : i \leq j \leq i+k-1 : (x_j = 1 \wedge \forall l : i \leq l \leq i+k-1 \wedge l \neq j : x_l = 0) \tag{3.11}$$

However, this information has not been told to the network and thus part of the task is to discover that all these inputs are strongly related and that, as a matter of fact, they represent a single one. Besides, thanks to the commutativity property, it would have been the same if the slice were broken in pieces and supplied in separate positions.

On the other hand, the increment in the number of network free parameters can become acute for moderate to big numbers of hidden units. It is important to emphasize that the problem with these extra parameters is that they are not collectively treated as an entity with a specific meaning, but as separate, independent variables.

3.5.2 Ordinal information

For ordinal variables, one possibility was to treat them as continuous and have them equidistantly mapped on an arbitrary real interval. First, this imposes a continuum where there is not (e.g., stating that there are potentially infinite possibilities between having two or three children) and also extends a finite to a potentially infinite range. Second, since these variables need not represent numerical entities (they could be, for example, the letters of the alphabet, or the days in the week), this encoding is assuring that Wednesday is three times Monday, or that "D" plus "E" is "J".

The representation as a thermometer is probably a worse choice. If the number of possibilities to be represented is not small (say, more than a dozen) the increase in variables may be simply not affordable. For a small such number, all the problems signaled for nominal variables apply. In this case, the structure to be discovered is of the form: $\exists j : i \leq j \leq i+k-1 : (\forall l : 1 \leq l \leq j : x_l = 1)$.

However, as we have seen, even this vision is misleading, since the "order" apparently imposed on this encoding is broken by the commutativity property. In other words, a permutation of the training data columns will result in an identical training set for the network.

3.5.3 Missing information

The possibilities and drawbacks of current approaches to handle missing information were already reviewed in (§2.1.9). The main trend was formed by those methods that complete the object description in a hopefully optimal way.

An alternative solution is simply to *ignore* what is not known, in such a way that it does not affect the treatment of the known pieces of information. This can lead to much simpler and equally intuitive schemes, based on not trying to estimate what is not known, because we could well not find the best estimation, in terms of the supervised learning process that we would like to apply to the obtained data. This is reminiscent of the discussion on the first training stage for RBF networks (§2.1.7).

This line of thought corresponds to the second type of methods explained in (§2.1.9). The idea is not new in data analysis. The ignorance of the absent components, followed by a simple normalization by the number of present ones, is in the base of the more classical similarity measures [Gower, 71]; empirically, it has been found superior to other treatments in standard data analysis experiments [Dixon, 79]. In our case, it translates in making the neuron model *handle* missing data. This fits nicely in the framework for neuron models proposed in this Thesis, because follows the principle of adapting the models to the data and not otherwise. Besides, since an heterogeneous weight is, by definition (see §(4.4)), of the same type than its corresponding input, a network weight can also be missing, with the consequent increase in flexibility and possible uses for network pruning. Note that a key assumption about this kind of treatment for missing values is that they are approximately equally distributed in training and in future test sets. In a sense, the network gets used to missing values from the outset, in the learning process itself.

It should be emphasized that we are not claiming that these are *better* ways to ultimately cope with missing information. This would need a full comparison between all the methods over a sufficiently large collection of data sets. Even then, the results might be inconclusive. Rather, we propose an alternative view which follows the general ideas of *missingness as ignorance*, and adaptation of the models and methods to the data. Of course, and regardless of the neuron model, it seems clear that complete data are to increase performance of whatever neuron model we use. Hence, methods for data completion could also be used together with the new neuron models proposed.

Of all the reviewed non-statistical methods, one of the most extended and arguably the less harmful is the addition of another variable. This has the appeal of being very intuitive because it can be regarded as *signaling* the abnormal situation to the network in the form of a *flag* that is turned on where appropriate. However, from the point of view of inductive learning systems (of which ANN are a class), for a data set wherein one attribute is often unknown, an algorithm may end basing its workings on that attribute taking the value "unknown", certainly an undesirable feature [Quinlan, 86]. For nominal variables, for example, with k different actual values, this technique is equivalent to an encoding with the rows of the $I_{(k+1)\times(k+1)}$ matrix. The missing value is treated as just another value. Consider the following Table (3.4):

	v_1	v_2	...
$\vec{x}_1$	3.8	0	...
$\vec{x}_2$	0	1	...
...	...	...	...

Table 3.4: Two patterns with their values in a data set. Pattern $\vec{x}_1$ is present for variable v_1 (with value 3.8), while pattern $\vec{x}_2$ is absent. The variable v_2 signals the situation.

Apart from the increase in parameters to be estimated, there is an *structure* known by the user but not by the network. For example, assuming zero is not a possible value for v_1, these two rules are fulfilled: $v_1 \neq 0 \Leftrightarrow v_2 = 0$ and $v_2 = 1 \Leftrightarrow v_1 = 0$. Besides, the value imputed to the variable whenever it is missing (zero in this case) is going to entirely change its distribution. Nevertheless, in mathematical terms, it is giving the network a chance to overcome the problem. Let $y(\vec{x}_i) = \sum_j w_j x_{ij}$, as usual. Then,

$$\begin{aligned} y(\vec{x}_1) &= w_1 x_{11} + w_2 x_{12} = w_1 x_{11} \\ y(\vec{x}_2) &= w_1 x_{21} + w_2 x_{22} = w_2 \end{aligned}$$

That is, when the value is present, its contribution to the overall sum can be made variable (it depends on x_{11}). When it is absent, there is a fixed contribution w_2. This seems a good idea in practice; hence, this is the preferred encoding method for comparisons.

3.5.4 Conclusions

Intuitive as they are, it is by no means clear what is the precise effect of these encodings on network performance, because of the notable change in input distribution, the increase (sometimes acute) in input dimension and other subtler mathematical effects derived from imposing an artificial order or extending one to an infinite continuum. A further problem is given by the significant amount of zero input values that all these methods introduce in the treated data sets. This has two important consequences: first, it means that the affected input is not counting at all in the aggregation function (whatever it is) and thus the corresponding weight is irrelevant. Second, and most important, for learning methods relying on derivatives, a null input also nullifies the updating of its weight[5] and makes the affected weight not subject to error correction.

As a consequence, in order to make data fit into the neuron model, these encodings are prone to cause a distorsion or result in a loss of information. In the ANN paradigm, hence, the inherent difficulty of the learning task can be aggravated by having the network to *discover* the new correlations derived from an structured increase in the number of inputs. Although the existence of prior knowledge allows for more sophisticated forms of preprocessing (e.g., non-linear), in most cases there is enough domain knowledge to design a more specific neuron model, so that these and other transformations can be embodied within the corresponding similarity measure.

3.6 A framework for similarity

Once we depart from the biological model and intend to use the artificial neuron as a computing device, there is no need to treat all the inputs in the same way, as is done in the practical totality of existing models. Scalar product and Euclidean distance are examples of

[5] Which is proportional to it, see the discussion on the back-propagation algorithm in (2.1.6). Roughly, being $x_j = 0$, $\Delta w_{ij} \propto error_i\, x_j = 0$.

a global aggregation measure, because they can be defined as a function of the weight and input vectors *as a whole*. The former can also be conveniently defined as a linear combination of components. In this case, all of the partial recognitions are performed using the same function (the correlation) and combined with a simple summation. In all, they are particular cases of performing a *combination* of *partial* recognitions. There is no local information or knowledge used to devise a particular recognition measure, *specifically tuned* to its input, nor are these partial measures combined in a special way.

To illustrate this point, let us consider an arbitrary neuron i computing a function $F_i(\vec{x})$, according to (2.1). For simplicity, we choose the vectors $\vec{x}, \vec{w}^i \in [0,1]^n \subset \mathbb{R}^n$, representing the input and weight vectors to neuron i, respectively. Let us define $s_1(x_j, w_{ij}) = x_j w_{ij}$ and $s_2(x_j, w_{ij}) = 1 - |x_j - w_{ij}|$.

The first function s_1 is the *correlation*. The second function s_2 is easily constructed from the standard metrics in $\mathbb{R}$, that is $d(x,y) = |x-y|$ for $x, y \in \mathbb{R}$, which is the base of all Minkowskian measures (2.5) (regardless of q). This can be checked by reducing (2.5) to the case $n = 1$.

These two measures can be said to compute the *similarity* (restricted to the domain $[0,1]$) between two scalars. Albeit capturing different aspects of their arguments, it is clear that the two functions are strongly related. On the one hand, both are continuous and exhaustive functions from $[0,1]^2$ to $[0,1]$. On the other hand, both are two ways of expressing an intuitive idea of similarity between two real numbers.

To be precise, s_1 is not a proper similarity measure, because it does not fulfill completely all the required properties (this will become clear in the next chapter). The point is that at this level —the computation of $F_i(\vec{x})$ for single vector components—, the two measures, though different, are actually computing the same kind of function.

Now let us define the extension to n components as a simple summation of the partial measures:

$$s_1^{(n)}(\vec{x}, \vec{w}^i) = \sum_{j=1}^{n} s_1(x_j, w_{ij}) = \vec{x} \cdot \vec{w}^i \tag{3.12}$$

is the classical scalar product, whereas

$$s_2^{(n)}(\vec{x}, \vec{w}^i) = \sum_{j=1}^{n} s_2(x_j, w_{ij}) = \sum_{j=1}^{n} 1 - |x_j - w_{ij}| = n - \sum_{j=1}^{n} |x_j - w_{ij}| = n - \|\vec{x} - \vec{w}^i\|_1 \tag{3.13}$$

is an inverse function of the norm of the difference (which is a distance in $[0,1]$). Let us plot them one against the other, in Fig. (3.6), for $n = 1$:

Their relative behaviour can be described as follows. First, the two measures yield a minimum value of 0 and a maximum of n. However, they do not have the same extrema: while the minimum is attained in a single point (at $[0,0]$) for s_1, this is attained at $[0,1]$ and $[1,0]$ for s_2. Regarding the maximum, it is reached only at $[1,1]$ for s_1, and along the line

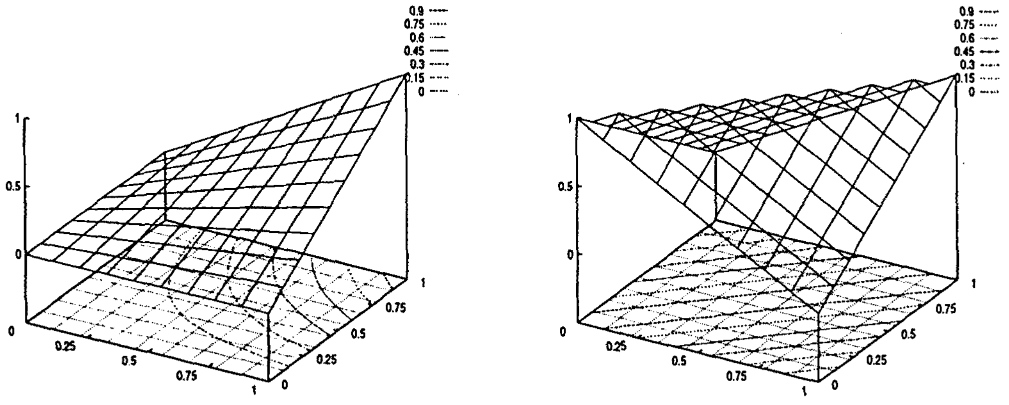


Figure 3.6: Left: $s_1(x,y) = xy, (x,y) \in [0,1]^2$. Right: $s_2(x,y) = 1 - |x-y|, (x,y) \in [0,1]^2$. In both cases, the contours are isoactivation lines.

segment $x = y$ for s_2. Notice that s_2 yields its maximum value whenever there is actually a "perfect match", that is, when $x_j = w_{ij}$. In contrast, s_2 gives it only when $x_j = w_{ij} = 1$.

In light of the maximum value attained, we can divide both functions by n, to obtain:

$$s_1^{(n)}(\vec{x}, \vec{w}^i) = \frac{\vec{x} \cdot \vec{w}^i}{n} \in [0,1] \tag{3.14}$$

$$s_2^{(n)}(\vec{x}, \vec{w}^i) = 1 - \frac{\|\vec{x} - \vec{w}^i\|_1}{n} \in [0,1] \tag{3.15}$$

The resulting normalized measures can be said to express a *degree of recognition* (ranging between zero and one) and can be shown to be kinds of similarity measures in $[0,1]$, according to the precise definition that will be given in (§4.2.2). Note that, since n is also the number of components, the "normalization" is actually the average of the partial quantities.

To sum up, we emphasize the more relevant points:

- The two measures above can be said to express the same concept: a similarity degree. The intuition behind (3.15) is that of the inverse of a distance, whereas the rationale of (3.14) was amply sketched in (§3.2).
- They have been built following the same line of reasoning. The design process has been given by two steps:
  1. Propose a *partial* measure for vector components.
  2. Integrate them into a global measure (in this case, with an arithmetic average).

Therefore, these two measures can be regarded as particular instances of an abstract or *generic model* of similarity computation. This model can then be used as an artificial neuron model for the degree of recognition expressed above, amply motivated in this Chapter.

Making a step beyond, and noticing that (3.14) and (3.15) are linear measures, we could think of composing them to a further, non-linear function $\check{s}$, in such a way that the computation of similarity is conceptually respected (for example, a monotonic distorsion). This leads to a generic model of the form:

$$F_i(\vec{x}) = s(\vec{x}, \vec{w}^{\,i}) = \check{s}(s_i(\vec{x}, \vec{w}^{\,i})) \tag{3.16}$$

where s_i is either (3.14) or (3.15). Notice that this conforms to the classical two-stage mapping (2.1) in current neural approaches. Additionally, we see that there is actually nothing that restrains the s_i to be linear: a non-linear choice for the partial measures could avoid the use of $\check{s}$. This is a clear departure from the classical model.

The above intuitions lead naturally to the development of more general measures, in which a similarity-based neuron model could be conceived by considering the following issues:

- If all the components are real-valued, we could think of devising partial (that is, local) measures of similarity other than the ones initially proposed.
- The way these partial measures are combined need not be the arithmetic average. More general *aggregation* functions can be defined, to obtain a similarity in $\mathbb{R}^n$.
- For input components that are *not* to be treated as real-valued, specific *heterogeneous* partial measures should be devised.
- There is no need to use such a simple transformation $\hat{s}(x) = 1 - x$, from a distance to a similarity, as is used in s_2. Other, parameterized and more controllable functions are conceivable. For instance, the choice $d(x, y) = |x - y|^2$ for $x, y \in \mathbb{R}$, ends in the expression $s_3^{(n)}(\vec{x}, \vec{w}^{\,i}) = n - ||\vec{x} - \vec{w}^{\,i}||_2^2$. Dividing by n as before, and making the choice $\hat{s}(z) = 1 - \sqrt{z}$ lead to $s_3^{(n)}(\vec{x}, \vec{w}^{\,i}) = 1 - \frac{||\vec{x} - \vec{w}^{\,i}||_2}{\sqrt{n}} \in [0, 1]$, a normalized similarity in $[0, 1]^n$, based on Euclidean distance.

  Moreover, it may possibly be of interest for the local distances to be bounded or, better, normalized, because this way the possible combination functions could be more general (by making them independent of data particularities), and eventually easier to formulate, so that a handful of them could serve in practically all situations.

The possibilities are many and rich to design new neuron models, provided the final measure (to be used as a neuron model) is a similarity in its input and weight space (which coincide). In all cases, domain knowledge can be used, to a greater or lesser degree, in the design of the neuron model and, by extension, into the function the ANN is computing. The objective is twofold: to gain insight into the workings of an artificial neuron model, and to use these new models to hopefully solve problems more satisfactorily.

3.7 Concluding remarks

In this chapter we have tried to provide a kind of overview and rationale for the design of neuron models cast under the common framework of similarity measures. This framework is meant to include the standard models used in MLP and RBF networks. However, unification of the plethora of models existent in the literature is not pretended (e.g. non-localized RBFs, or scalar product-based neurons using non-monotonic activation functions, as the sinusoidals, are not included).

A groundwork has also been set forth, by analyzing the basic postulates in ANN design, the extent to which current approaches deal with similarity and heterogeneity and how neuron models can be explained in light of these two related concepts.

Among the reviewed problems we find that the network is left to face an enormous task, including a discovery of structure, detection of relative relevance of variables under the chosen representation and approximation of the underlying function.

A reasonable solution to ease this task —and the one followed in this work— is to have, on the one hand, a normalized neuron computation, *instead* of having to normalize the inputs and, on the other hand, a neuron computation that is explicitly defined to take the form of an (heterogeneous) similarity measure.

There are other, additional advantages in defining a similarity function in a explicit form, by taking into account the data peculiarities:

- Practical: *understandability* of the function computed by the network (because of the domain knowledge used in its definition) and *readability* of the results (what do the weights mean).
- Theoretical: taking profit of the properties of similarity measures (e.g. boundedness, monotonicity, continuity or absence of input preprocessing).

The view of a neuron model as an explicitly defined similarity computing device allows to work directly in *pattern space*, thus making the identity *pattern space* = *input space* hold. Further, the direct treatment of heterogeneity not only fits conceptually in this view —if non-standard data are encoded, there is no heterogeneity— but is to lighten the learning task, as the problems or subtleties derived from the pre-processing mechanisms chosen are avoided.

Chapter 4

Similarity-Based Heterogeneous Neuron Models

Rien n'est plus semblable à l'identique
que ce qui est pareil à la même chose
et réciproquement.

Pierre Dac

In this chapter we construct a framework for the development of heterogeneous neuron models based on similarity relations or S-neurons, derived from a somewhat larger and abstract class of models (called H-neurons). Our focus of study will be centered on feed-forward architectures, though nothing would prevent the design of more general structures (as recurrent ones) by using S-neurons as building bricks. These networks, as stated in the Introduction, are out of the scope of the Thesis. The obtained class of feed-forward models includes some of the most widespread network models, as MLP and RBF networks.

4.1 Introduction

4.1.1 The definition of similarity

In classical AI systems, the notion of similarity appears in learning by analogy and case-based reasoning (CBR), where learning by analogy and instance-based learning (IBL) fuse. In particular, similarity coefficients have had a long history in the literature of cluster analysis and data clustering algorithms [Everitt, 77], [Jain and Dubes, 88]. However, similarity has been studied in its own right more thoroughly in psychology -and, in particular, in cognitive psychology- than in AI. Firstly, there is a well established school rooted in *psychophysics* which takes the form of geometric models. These models represent the objects as points in a high-dimensional coordinate space where a distance between points is to be defined based on the given data. This distance measure is to reflect the observed similarities and should correspond as closely as possible to the *objective* similarity between the objects that are represented by the points, which could be any kind of data with information about objective or empirical similarities (proximities) between pairs of objects.

Secondly, we find the general notion of similarity defined in set-theoretic terms -generally known as the *contrast* model- discussed in the work of Tversky [Tversky, 77], which summarizes a large portion of relevant literature and introduces a formal theory of similarity. According to it, a similarity judgement needs the existence of a essentially unique continuous real-valued function s that expresses a degree of similarity in a clearly defined sense. Such a function is presumably bounded and, taking two objects a and b represented by the feature sets A and B, respectively, expresses the (bounded) degree to which a and b are regarded as similar, written $s(a,b)$. This way, one can state comparisons of the type $s(a,b) > s(c,d)$ to say that object a is more similar to object b than object c is to object d. This endows s with a *semantics* expressing how much two objects resemble each other[1].

4.1.2 Features of similarity

The key point for a succeeding learning system, embedded in an environment, heavily relies in its ability to internally *represent the similarities* between the different stimuli and to arrange them so that similar incoming stimuli elicit resembling outcomes. We would like to bring to attention the analogy of this reasoning to the Principles (3.1), (3.2) formulated in the previous Chapter.

In cognitive psychology and other disciplines, formulation and representation of similarity has been a key issue. Understanding how (and whether) biological perceptual systems represent and use similarities has been notoriously difficult to formalize, mainly because of the feature selection problem, relevance determination, contextual effects, transitivity, semantic considerations and others.

In the ANN paradigm, however, these complex aspects are somewhat mitigated because, either they are not addressed (like transitivity) or assumed discovered by the network (like the assignment of relevance to input features). As for context, a basic assumption in ANN is that all the patterns are measured along the same fixed set of features. This means that some form of prior knowledge is implicit in the feature selection process and the neuron models are designed taking this into account. Nonetheless, in some recurrent architectures there is an explicit provision for context.

4.2 Similarity measures

4.2.1 Distance measures

Definition 4.1 (Distance) *Given a set X, any function d:*

$$d : X \times X \to \mathbb{R}^+ \cup \{0\}$$

such that $\forall x, y, z \in X$:

[1]Resemblance is the term used by Hume [Hume, 47].

i) $d(x,y) = 0 \Leftrightarrow x = y$

ii) $d(x,y) = d(y,x)$

iii) $d(x,y) \leq d(x,z) + d(z,y)$

is a distance in X.

A set X endowed with a distance d will be denoted (X,d) and referred to as a *metric space.* We denote the set of distance measures of a set X as $D(X)$. The following is a collection of useful propositions and properties about metric spaces and distance functions. The proofs are given in Appendix B.

Proposition 4.1 *Let* (X,d) *be a metric space and* $S \subset X$*; then* (S,d) *is a metric space.*

Proposition 4.2 $(\mathbb{R},d)$ *is a metric space for* $d(x,y) = |x-y|$. *This distance function will be referred to as the standard metric in* $\mathbb{R}$.

Proposition 4.3 $(\mathbb{Q},d)$, $(\mathbb{Z},d)$ *and* (N,d) *are metric spaces for the standard metric in* $\mathbb{R}$.

Definition 4.2 (Normalized distance) *Given a set* X, *a distance* $d \in D(X)$ *is normalized if:*

$$\exists x^*, y^* \in X, \qquad \forall x,y \in X,\ d(x^*,y^*) = d_{max} \geq d(x,y) \tag{4.1}$$

We denote such distances as functions $d : X \times X \to [0, d_{max}]$

The standard metric in $\mathbb{R}$ can be extended in various ways to $\mathbb{R}^n, n \in \mathrm{N}^+$. We study such ways as particular cases of *aggregation* operators.

Definition 4.3 (Aggregation operator) *Consider a collection of* n *non-negative real quantities. For convenience, we group these quantities as a vector* $\vec{z} = \{z_1, z_2, \ldots, z_n\}$. *An aggregation operator* Θ *is a function:*

$$\Theta : (\mathbb{R}^+ \cup \{0\})^n \to \mathbb{R}^+ \cup \{0\}$$

fulfilling:

i) Minimality. $\Theta(\vec{z}) = 0 \Leftrightarrow \forall i\ z_i = 0$.

ii) Symmetry. $\Theta(\vec{z}) = \Theta(\sigma(\vec{z}))$ *for any* $\sigma(\vec{z})$ *permutation of* $\vec{z}$.

iii) Strict monotonicity. $\Theta(\vec{z}) > \Theta(\vec{z}\,')$ *whenever* $\exists i\ z_i > z'_i \wedge \forall j : 1 \leq j \leq n \wedge j \neq i : z_j = z'_j$.

Remark 4.1 *These conditions imply that* $\Theta(\vec{z}) > 0$ *whenever there exists a* $z_i > 0$.

The use of the norm defined in (2.5) as an aggregation operator leads to general families of distances.

Definition 4.4 (Minkowskian aggregation) *Define the "Minkowskian aggregation" operator $\Theta^{q,n}(\cdot), \forall q \geq 1 \in \mathbb{R}$, for an n-dimensional vector $\vec{z}$, as*

$$\Theta^{q,n}(\vec{z};\vec{v}) = \left\{ \frac{1}{n'} \sum_{i=1}^{n} \left(\frac{|z_i|}{v_i} \right)^q \right\}^{\frac{1}{q}} \tag{4.2}$$

where $\vec{v}$ is a weighting vector and n' an averaging factor, either $n' = n$ or $n' = 1$.

The choice $n' = n$ eliminates the dependence on the number of components. It is required that all the v_i be positive and tied to the i-th component, in order to fulfill conditions (iii) and (ii), respectively, in Definition (4.3). Note that the weighting does not necessarily imply boundedness. However, if the original z_i are *normalized* distances in $[0, z_{i,max}]$, and the v_i equals the terms $\frac{|z_i|}{v_i}$ so to lie in the same $[0, z_{max}]$ (by choosing $v_i = \frac{z_{max}}{z_{i,max}}$), the choice $n' = n$ is mandatory so that this characteristic is kept.

Proposition 4.4 *The aggregation function $\Theta^{q,n}$ in Definition (4.4) is a valid distance aggregation operator in $\mathbb{R}^n$.*

Proposition 4.5 *The aggregation function $\Theta^{q,n}$ in Definition (4.4) is a norm in $\mathbb{R}^n$.*

On top of this general kind of aggregation operator, whole classes of distance measures can be defined.

Proposition 4.6 *The functions $d_q^{(n)}$ defined as:*

$$d_q^{(n)}(\vec{x},\vec{y}) = \{\Theta^{q,n}(\vec{x}-\vec{y};\vec{v}),\ \vec{v} \in (\mathbb{R}^+)^n\} \tag{4.3}$$

make $(\mathbb{R}^n, d_q^{(n)})$ a metric space.

This generic class of functions is specifically known as (weighted) Minkowskian distances. The measure obtained for $q = 1$ is generally known as the *City-block* or Manhattan distance. For $q = 2$ we have the Euclidean measure. This class of functions is characterized by an increasing attention to the bigger partial components d_i with increasing values of q, to the point that, in the limit:

$$\lim_{q\to\infty} d_q^{(n)}(\vec{x},\vec{y}) = \sup_{i=1,\ldots,n} d(x_i,y_i)$$

only the biggest value is taken into account. This last measure is known (for obvious reasons) as the *supremum* or *Chebyshev*. In general, though, we are not restricted to such a family of distances, and can make use of many possibilities of combining the individual distances that stem from the combinations of $\vec{v}, n'$ and q. Additionally, several other classes of distances can be derived as, for example:

$$d_q^{(n)}(\vec{x}, \vec{y}) = \Theta^{q,n}(\frac{\vec{x} - \vec{y}}{\vec{x} + \vec{y}}; \vec{v}) \tag{4.4}$$

$$d_q^{(n)}(\vec{x}, \vec{y}) = \Theta^{q,n}(\sqrt[q]{\vec{x}} - \sqrt[q]{\vec{y}}; \vec{v}) \tag{4.5}$$

where all the arithmetic operations on vectors are taken to be component-wise. These two classes are "self-normalizing" and require the vectors not having negative components or being null vectors. The following are classical examples of specific distance measures that can be derived from (4.3), (4.4) and (4.5). Denoting by $\vec{\mu}$, $\vec{\sigma}^2$ and $\vec{\sigma}_{dev}$, the vector of means, variances and maximum deviations of the variables,

1. (4.3) with $n' = 1, q = 2, \vec{v} = \vec{1}$ (*basic unweighted Euclidean distance*)
2. (4.3) with $n' = n, q = 1, \vec{v} = \vec{\mu}$ (*mean City-block distance weighted by the inverse of the mean*)
3. (4.3) with $n' = 1, q = 2, \vec{v} = \vec{\sigma}^2$ (*Pearson distance*)
4. (4.4) with $n' = 1, q = 2, \vec{v} = \vec{1}$ (*Coefficient of divergence or Clark distance*)
5. (4.4) with $n' = 1, q = 1, \vec{v} = \vec{1}$ (*Canberra distance*)
6. (4.3) with $n' = n, q = 1, \vec{v} = \vec{1}$ (*unweighted mean City-block distance*)
7. (4.3) with $n' = n, q = 2, \vec{v} = \vec{\sigma}_{dev}$ (*mean Euclidean distance weighted by the inverse of the maximum deviation*)
8. (4.5) with $n' = n, q = 2, \vec{v} = \vec{1}$ (*mean Duran-Odell distance*)

It can be noted that (4.) and (7.) above are normalized distances, with $d_{max} = 1$. It is not our intention to devise new distance measures, albeit by combining these factors many variations can be obtained. For convenience, the following simple ways for distance aggregation are derived from Definition (4.4), having the additional advantage of not being restricted to $\mathbb{R}^n$.

Definition 4.5 (Linear operator) *An unary operator π defined on a set X is linear if and only if, for every $x, y \in X$:*

1. $\pi(cx) = c\pi(x), \qquad c \in \mathbb{R}$

2. $\pi(x + y) = \pi(x) + \pi(y)$

Definition 4.6 ($\bar{n}$-Linear operator) *A n-ary operator is n-linear if it is linear on each of its arguments.*

Proposition 4.7 *Given $n \in \mathbb{N}^+$ and a collection of sets $X_1, X_2, \ldots, X_n$ for which corresponding distance functions $\vec{d} = \{d_1, d_2, \ldots, d_n\}$ have been defined, $d_i \in D(X_i)$, we denote $X = X_1 \times X_2 \times \ldots \times X_n$ (the usual Cartesian product). Then, for any n-linear Θ and positive $\vec{v}$, the pair $(X, \Theta(\vec{d}; \vec{v}))$ is a metric space.*

Proposition 4.8 *Given a positive weighting $\vec{v}$, the following is a valid n-linear Θ operator:*

$$\Theta(\vec{d}; \vec{v}) = \sum_{i=1}^{n} v_i d_i \qquad (\textit{weighted summation}) \tag{4.6}$$

In the common situation where the partial distances are all normalized to the same range of values, it is possible to devise specific aggregation operators. Let us suppose that, under the same assumptions of Proposition (4.7), it holds that $\forall i : 1 \leq i \leq n$: $d_i : X_i \times X_i \to [0, d_{max}]$. In these conditions, any $\Theta(\vec{d}) \in D(X)$ is a function of the form: $[0, d_{max}]^n \to [0, d_{max}]$.

Proposition 4.9 *For normalized distances $\vec{d} = \{d_1, d_2, \ldots, d_n\}, d_i \in [0, d_{max}]$, the following is a valid Θ operator:*

$$\Theta^{q,n}(\vec{d}) = \frac{1}{\sqrt[q]{n}} \|\vec{d}\|_q, \;\; q \geq 1 \in \mathbb{R} \tag{4.7}$$

(unweighted, normalized modulus)

This expression is obtained from Definition (4.4) by setting $n' = n$ and $\vec{v} = \vec{1}$. In general, in the conditions of Proposition (4.9), any operator of the form $\Theta^{q,n}(\vec{d}; \vec{v})$ leads to normalized distances provided the additional condition $\sum_{i=1}^{n} v_i = 1$ holds.

Proposition 4.10 *For normalized distances $\vec{d} = \{d_1, d_2, \ldots, d_n\}, d_i \in [0, d_{max}]$, the following is a valid Θ operator, being also n-linear:*

$$\Theta(\vec{d}) = \frac{1}{n} \sum_{i=1}^{n} d_i \qquad (\textit{arithmetic mean})$$

These are very simple and effective possibilities for aggregation families and shall be used through-out. Notice that an additional weight can be assigned to individual components of distance measures —not for normalization or standardization purposes, as in the previous list of measures— but to express relative *relevance* of components, in the form $\Theta(\vec{d}; \vec{v}, \vec{w})$ where each w_i multiplies d_i *before* the application of the aggregation operator. The condition is again that all the w_i be positive to preserve monotonicity. Normalized distances require also the condition:

$$\sum_{i=1}^{n} w_i = 1$$

Although it can be an interesting feature, these relevance weightings are not considered here, because relative relevance assessment is assumed to be part of the task of the ANN. Normalization should suffice to equal all variables to a common scale before learning takes place; in any case, the previous Propositions are still valid by associating each w_i to i, so that symmetry is kept. We now turn our attention to the problem of defining similarity measures.

4.2.2 Similarity measures

Let us represent objects, patterns, input stimuli, entities, or however we originally call them, belonging to a space X (of which nothing is assumed, apart that $X \neq \emptyset$) as a vector $\vec{x}_i$ of n components, where each component x_{ij} represents the value of a particular feature (descriptive variable) a_j for object i, from a predefined set of features $A = \{a_1, a_2, \ldots, a_n\}$, judged by the investigator as relevant to the problem. These "vectors" can be thought of as points in a n-dimensional vectorial space; this is possible provided that all the components have a dimension (i.e., there is a total order $\leq_{a_j}$ for all $a_j \in A$) and is irrelevant for the following definitions. It is assumed that the precise order of the $a_j \in A$ is of no importance, provided, of course, that is kept for all the $\vec{x}_i$. This is the usual representation system in which the components x_{ij} of a given object are the values that this object takes for the variables a_i in A. Now, the main question that arises is: how *like* are two $\vec{x}_i, \vec{x}_j$ in X?

We start by calling this vague notion of "likeness" a *proximity* index, one expressing the overall degree of either likeness or resemblance -the *similarity*- or the opposite -the *dissimilarity*. That is, a proximity index (or proximity measure) is a unique number expressing how *like* two given objects are, by only taking into account the selected set of features that characterize them.

For simplicity, let us denote by p_{ij} the proximity between $\vec{x}_i$ and $\vec{x}_j$, that is, $p : X \times X \to \mathbb{R}^+ \cup \{0\}$ and $p_{ij} = p(\vec{x}_i, \vec{x}_j)$. To begin with, we differentiate among two kinds of proximity indexes: similarity indexes s_{ij} and dissimilarity indexes δ_{ij}, defined as p above.

Definition 4.7 (Similarity, dissimilarity) *A proximity index*[2] *has to fulfill the following properties [Chandon and Pinson, 81]:*

1. Non-negativity. *The p_{ij} cannot be negative.*

$$s_{ij} \geq 0 \tag{4.8}$$
$$\delta_{ij} \geq 0 \qquad \forall \vec{x}_i, \vec{x}_j \in X \tag{4.9}$$

2. Symmetry. *The p_{ij} do not depend on the order of i, j.*

$$s_{ij} = s_{ji} \tag{4.10}$$
$$\delta_{ij} = \delta_{ji} \qquad \forall \vec{x}_i, \vec{x}_j \in X \tag{4.11}$$

[2]In this context, the terms *index* and *function* will be used interchangeably.

3. Boundedness. *This is where s_{ij} and δ_{ij} diverge. There is a maximum similarity (namely, that of an object with itself) and a minimum dissimilarity (idem).*

$$s_{ij} \leq s_{max} \tag{4.12}$$
$$\delta_{ij} \geq \delta_{min} \qquad \forall \vec{x}_i, \vec{x}_j \in X \tag{4.13}$$

4. Minimality *(Reflexivity in the strong sense). The extreme values are attained for equal objects, and only for them.*

$$s_{ij} = s_{max} \quad \Leftrightarrow \vec{x}_i = \vec{x}_j \tag{4.14}$$
$$\delta_{ij} = \delta_{min} \quad \Leftrightarrow \vec{x}_i = \vec{x}_j \quad \forall \vec{x}_i, \vec{x}_j \in X \tag{4.15}$$

5. Semantics. *The semantics of $s_{ij} > s_{ik}$ is that object i is more similar to object j than is to object k. Conversely, the semantics of $\delta_{ij} > \delta_{ik}$ is that object i is more dissimilar to object j than is to object k.*

Two possible (and, otherwise, arbitrary) values are $s_{max} = 1$ and $\delta_{min} = 0$. An index s_{ij} satisfying properties (1.) through (5.) is a *similarity index*. An index δ_{ij} satisfying properties (1.) through (5.) is a *dissimilarity* index. If, in property (4.), we replace the $\Leftrightarrow$ by just $\Leftarrow$, so that two different objects can be assigned s_{max} or δ_{min} (as if they were the same object) then we obtain, respectively, a *pseudo-similarity* and a *pseudo-dissimilarity* index. Note that by *different* objects $\vec{x}_i, \vec{x}_j$ we mean that $\exists k : 1 \leq k \leq n : x_{ik} \neq x_{jk}$.

Another interesting kind of similarity index can be defined by requiring the property:

$$s_{ij} = s_{max} \quad \Leftrightarrow \vec{x}_i = \vec{x}_j = \vec{x}_k \qquad \forall \vec{x}_i, \vec{x}_j \in X \tag{4.16}$$

where $\vec{x}_k$ is fixed in X. That is, two objects are regarded as more similar, the more similar they are with one another and with reference to a third "ideal" or prototypical object. The maximum similarity is attained only whenever the two objects are equal, and equal to the ideal: in other words, it is the similarity of the ideal with itself. An index with this property will be called *point similarity*.

A space X where a proximity index p_{ij} has been defined forms a semi-metric space, denoted (X, p), being p either s or δ. Other, more restrictive properties can be further demanded to such a space:

6. *Triangular inequality.* This property asks that any three different objects i, j, k are either aligned or forming a triangle, and can only be in general fulfilled by a dissimilarity.

$$\delta_{ij} \leq \delta_{ik} + \delta_{kj} \quad \forall \vec{x}_i, \vec{x}_j, \vec{x}_k \in X \tag{4.17}$$

This is equivalent to say that, given two different objects $\vec{x}_i, \vec{x}_j$, their dissimilarity must always be less or equal than the sum of their respective dissimilarities to a third object

$\vec{x}_k$. In other words, it is impossible for the way from objects $\vec{x}_i$ to $\vec{x}_j$ via a third object $\vec{x}_k$, to be less than the direct way from $\vec{x}_i$ to $\vec{x}_j$. A dissimilarity index fulfilling this property is called a *metric distance*, which will be denoted as d. Notice that this ultimately corresponds to the definition of distance already given in Definition (4.1). In what regards to similarities, it is perfectly possible for a similarity to either fulfill it or violate it -see Fig. (4.1).

7. *Ultrametric inequality.*

$$\delta_{ij} \leq max(\delta_{ik}, \delta_{kj}) \quad \forall \vec{x}_i, \vec{x}_j, \vec{x}_k \in X \tag{4.18}$$

This is equivalent to say that, given two different objects $\vec{x}_i, \vec{x}_j$, their dissimilarity (now metric distance) must always be less or equal than the maximum of their respective dissimilarities to a third object $\vec{x}_k$, asking that any three different objects $\vec{x}_i, \vec{x}_j, \vec{x}_k$ must be always forming an isosceles or equilateral triangle.

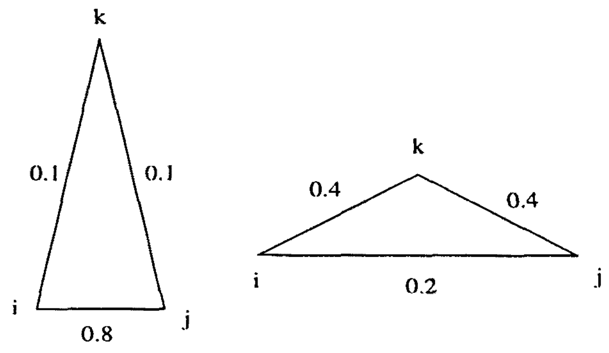


Figure 4.1: Two scenarios for the same similarity measure s taking values in $[0, 1]$. Left: i and j are very similar between each other and both are very dissimilar to k ($s_{ij} \geq s_{ik}+s_{kj}$). Right: i is moderately similar to k and k to j, and thus i is very dissimilar to j ($s_{ij} \leq s_{ik} + s_{kj}$).

The last two properties are particular cases of the more general statement:

$$\delta_{ij} \leq \{(\delta_{ik})^r + (\delta_{kj})^r\}^{\frac{1}{r}} \qquad \forall \vec{x}_i, \vec{x}_j, \vec{x}_k \in X, r \geq 1 \in \mathbb{R} \tag{4.19}$$

where (4.17) and (4.18) are obtained for $r = 1$ and $r = \infty$, respectively.

4.2.3 Aggregation of similarities

The way different partial similarities should be combined to give an overall score is by no means clear. The aggregation operator fulfills also a *semantic* role. Semantics are very important and is a means to express psychological aspects of similarity; for example, the presence of a particular attribute is neither a necessary nor a sufficient condition for the overall similarity judgement [Wallach, 58]. In these situations, a clearly defined semantics has to be stated. In our case, the adopted posture follows:

1. Even small contributions can only *add* something in favour for the overall measure;
2. The eventually missing pieces are regarded as *ignorance* and do not contribute in favour nor against the overall measure.

The following is a proposal of similarity aggregation formalization —an elusive topic [Hahn and Chater, 98]— through a compilation of desirable properties. Also, the definition is made easier by the fact that the partial values are, by definition, normalized. Incidentally, in comparison to Definition (4.3), the minimality condition can be relaxed and the other two conditions are still useful. Notice that monotonicity reflects point (1.) above.

For $z \in \mathbb{R}, n \in \mathbb{N}^+$ and an operator T, define:

$$z^n[T] = \begin{cases} z & \text{if } n = 1 \\ T(\underbrace{z, z, \ldots, z}_{n\ times}) & \text{if } n > 1 \end{cases} \tag{4.20}$$

Definition 4.8 (Θ_s aggregation operator) *Consider a collection of n_0 quantities, grouped by convenience as a vector $\vec{s}_0 = \{s_1, s_2, \ldots, s_{n_0}\}$, where $s_i \in [0, s_{max}] \cup \{\mathcal{X}\}$, with $[0, s_{max}] \subset \mathbb{R}, s_{max} > 0$. A similarity aggregation operator Θ_s is a function*

$$\Theta_s : ([0, s_{max}] \cup \{\mathcal{X}\})^{n_0} \to [0, s_{max}] \cup \{\mathcal{X}\}$$

where the symbol $\mathcal{X}$ denotes a missing component, for which only equality is defined, behaving as an incomparable *element w.r.t. any ordering relation. Let*

$$F_{\mathcal{X}} : ([0, s_{max}] \cup \{\mathcal{X}\})^{n_0} \to ([0, s_{max}])^n, \ (n \leq n_0)$$

be a filter *operator, which constructs a vector without the missing components, in the original order. Let $\vec{s} = F_{\mathcal{X}}(\vec{s}_0)$. For the present components, an aggregation operator for similarities is a function fulfilling:*

i) Minimality. $\Theta_s(\vec{s}) = 0 \Rightarrow \exists i : 1 \leq i \leq n : s_i = 0$ and $(\forall i : 1 \leq i \leq n : s_i = 0) \Rightarrow \Theta_s(\vec{s}) = 0$. This formulation allows (though not ensures) the 0 as an absorbing element.

ii) Symmetry. $\Theta_s(\sigma(\vec{s})) = \Theta_s(\vec{s})$ for any $\sigma(\vec{s})$ permutation of $\vec{s}$.

iii) Monotonicity. $\Theta_s(\vec{s}) > \Theta_s(\vec{s}\,')$ whenever there exists a s_i, $1 \leq i \leq n$ such that

$$s_i > s'_i \wedge \forall j : 1 \leq j \leq n \wedge j \neq i : s_j = s'_j$$

iv) Idempotency. For an (arbitrary) s_i, $s_i^n[\Theta_s] = s_i, \forall n \in \mathbb{N}^+$. Note that this specifically includes the boundary values $s_{max}^n[\Theta_s] = s_{max}$ and $0^n[\Theta_s] = 0$. Idempotency implies $s_i^{n+1}[\Theta_s] = s_i^n[\Theta_s], \forall n \in \mathbb{N}^+$. A less restrictive condition is to require:

(a) Idempotency for boundary values;

(b) The terms $\{s_i^n[\Theta_s]\}_{n\geq 1}$ forming a monotonic succession.

This property will be denoted relaxed *idempotency.*

v) Cancellation law. $\Theta_s(\{s_1, s_2\}) = \Theta_s(\{s_1, s_3\})$ *and* $s_1 > 0$ *implies* $s_2 = s_3$.

vi) Continuity. Θ_s *should be continuous in all its arguments so the reactions to small changes in an* s_i *are not "jumpy". This property also ensures that all possible values of* $\Theta_s(\vec{s})$ *can in principle be generated:*

$$\forall s : s \in [0, s_{max}] : (\exists \vec{s}^* : \ s = \Theta_s(\vec{s}^*))$$

vii) Compensativeness. Θ_s *should be a Cauchy mean, i.e.,*

$$\min_i s_i \leq \Theta_s(\vec{s}) \leq \max_i s_i$$

A Cauchy mean[3] *is such that a good (bad) score* s_i *can be compensated by a bad (good) score on another* s_j.

viii) Treatment of missing components. $\Theta_s(\vec{s}_0) = \Theta_s(F_{\mathcal{X}}(\vec{s}_0))$ *and, being* $\vec{s}_{\mathcal{X}} = (\mathcal{X}, \ldots, \mathcal{X})$ *of length* n_0, $\Theta_s(\vec{s}_{\mathcal{X}}) = \mathcal{X}$.

These last two conditions, together with monotonicity, are a way of introducing a *specific* semantics in the aggregation, and others should be possible. Note that some measures, like the product with arguments in $[0, 1]$, would violate *vii*). It will thus be considered as an optional requirement, although the majority of measures used in practice are to comply with it. Notice also that the *max* and *min* functions would not fulfill strict monotonicity.

Remark 4.2 *Weighted operators* $\Theta_s(\vec{s}; \vec{v})$ *are also valid, provided they make the operator fulfill the above conditions.*

Proposition 4.11 *A measure* s *obtained from any such aggregation operator* $s = \Theta_s(\vec{s})$, *provided* $\vec{s} = \{s_1, \ldots, s_n\}$ *are also similarities* s_i *in* X_i, *fulfills the properties of Definition (4.7), making* s *a similarity measure in* $X = X_1 \times X_2 \times \ldots \times X_n$.

Proposition 4.12 *The following are valid families of similarity aggregation functions:*

1. *Additive measures of the form:*

$$\Theta_s(\vec{s}; \vec{v}) = f^{-1}\left(\sum_{i=1}^{n} v_i f(s_i)\right) \tag{4.21}$$

where f *is a strictly increasing continuous function* $f : [0, s_{max}] \to [0, s_{max}]$ *such that* $f(0) = 0$ *and* $f(s_{max}) = s_{max}$, *and* $\sum_{i=1}^{n} v_i = 1$, *with* $v_i > 0$. *As an example, consider* $f(z) = z^m, m \in \mathbb{Z}, m \neq 0$. *For* $m = 1$ *we get the weighted arithmetic mean, which further reduces to a pure arithmetic average setting* $v_i = 1/n$. *For* $m = -1$ *a weighted harmonic mean is obtained.*

[3]This definition has been borrowed from [Roubens, 90].

2. *Multiplicative measures of the form:*

$$\Theta_s(\vec{s};\vec{v}) = f^{-1}\left(\prod_{i=1}^{n} f(s_i)^{v_i}\right) \tag{4.22}$$

with f and $\vec{v}$ in the same conditions. Consider again $f(z) = z^m, m \in \mathbb{Z}, m \neq 0$. The choice $m = n$ and $v_i = 1/n$ leads to the geometric mean, whereas $m = v_i = 1$ yields a pure correlation[4] *for $s_{max} = 1$. Note that pure correlation fulfills the relaxed idempotency.*

3. *The normalized modulus:*

$$\Theta_s(\vec{s}) = \frac{1}{\sqrt[q]{n}}\|\vec{s}\|_q,\ q \geq 1 \in \mathbb{R} \tag{4.23}$$

This is an interesting special case of an additive measure by taking $v_i = \frac{1}{n}$ and $f(z) = z^q, q \geq 1 \in \mathbb{R}$.

This last case is a simple and useful measure, unique in that it fulfills all the properties in Definition (4.8), including the optional ones, and is general enough to be applicable in many situations. Although other choices of $f(z)$ are possible, the family $f(z) = z^q$ is a simple one, and does not depend on any parameter, once q is fixed. In this sense, note that derived choices of the form $f(z) = \alpha z^q + \beta$, with an $\alpha \neq 0$ and β to be found, are of no interest since, by cancelling out, they play no significant role for additive measures, by application of a theorem due to Jensen -see Appendix (D).

Notice also that nothing is said about the degree of linearity of the resulting measure. It is useful to introduce a function that -regardless of its linear or non-linear character- maintains the similarity properties when applied to one such measure.

Definition 4.9 (Similarity keeping) *A similarity keeping function (denoted $\check{s}$) is a strictly increasing, bounded and continuous function $\check{s} : [0, s_{max}] \to [0, \check{s}_{max}]$ fulfilling:*

i) $\check{s}(0) = 0$

ii) $\check{s}(s_{max}) = \check{s}_{max} > 0$

iii) $\exists \epsilon > 0,\ \forall x \leq \epsilon\ \check{s}(x) \leq x$

Example functions are shown in Fig. (4.2). Note that $\check{s}(x) = x$ is valid, and fulfills $s_{max} = \check{s}_{max}$. In case this last condition is met, the functions are isomorphisms in $[0, s_{max}]$.

These functions keep (strict) monotonicity when composed to another (strictly) monotonic function and, since every strictly monotonic function is injective, when applied to a measure obtained by aggregation, the original properties that the aggregation operator as a whole could have —of particular interest, *(iii)*, *(iv)* and *(vi)*— are kept.

[4]For $s_{max} = 1$ there are no restrictions on $\vec{v}$.

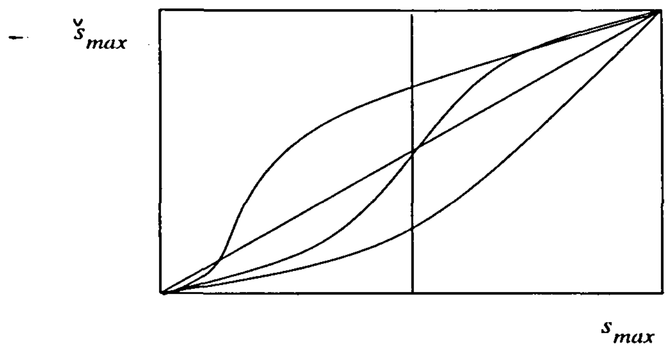


Figure 4.2: Different shapes for an $\check{s}$ function. One of them has a sigmoidal shape.

Proposition 4.13 *Let s be a similarity function in X defined on $[0, s_{max}]$ and $\check{s}$ a similarity keeping function (either linear or non-linear). Then $\check{s} \circ s$ is a similarity function in X.*

The final piece for constructing general similarity measures is a transformation operator to obtain a similarity out of a metric distance.

Definition 4.10 (Similarity transforming) *A similarity transforming function (denoted $\hat{s}$) is a strictly decreasing monotonic and continuous function $\hat{s} : [0, +\infty) \to [0, s_{max}]$ such that $\hat{s}(0) = s_{max}$ and $\lim_{z \to +\infty} \hat{s}(z) = 0$.*

Proposition 4.14 *Let d be a distance function defined on X and $\hat{s}$ a similarity transforming function (either linear or non-linear). Then $s(x, y) = \hat{s}(d(x, y))$ is a similarity function in X, for any two $x, y \in X$.*

Proposition 4.15 *Let $\check{s}, \hat{s}$ be two similarity keeping and transforming functions, respectively. Then $\check{s} \circ \hat{s}$ is a similarity transforming function in $[0, \check{s}_{max}]$.*

In case the original distance is normalized, the preceding statements can be further refined.

Proposition 4.16 *Let $d : X \to [0, d_{max}]$ be a normalized distance function defined on X and $\hat{s}$ a similarity transforming function (either linear or non-linear). Then, for an $\hat{s}$ such that $\hat{s}(d_{max}) = 0$, the function $s(x, y) = \hat{s}(d(x, y))$ is a similarity in X.*

Proposition 4.17 *For every $\hat{s}$ function defined for a non-normalized distance, there exists another $\hat{s}'$ function defined for a normalized one in $[0, d_{max}]$, of the same shape, and fulfiling the previous Proposition (4.16).*

The following are examples of $\hat{s}$ functions:

- Linear: $\hat{s}_0(z) = d_{max} - z$, for $z \in [0, d_{max}]$ $(s_{max} = d_{max})$.

- Inverse function: $\hat{s}_1(z) = \frac{a}{1+z}$ for $z \in [0, \infty), a > 0$ ($s_{max} = a$).
- Cauchy function: $\hat{s}_2(z) = \frac{a}{1+z^2}$ for $z \in [0, \infty), a > 0$ ($s_{max} = a$).
- Exponential function: $\hat{s}_3(z) = e^{-z}$, for $z \in [0, \infty)$ ($s_{max} = 1$).
- Inverted logistic: $\hat{s}_4(z) = a(1 - \sigma(z))$, for $z \in [0, \infty), a > 0$ ($s_{max} = a/2$).
  where $\sigma(z)$ is the logistic function.

Some of these functions (like $\hat{s}_0, \hat{s}_1$ and $\hat{s}_2$) are commonly found in the literature of in data analysis [Chandon and Pinson, 81] and will later on be generalized conveniently (§4.4.2).

4.2.4 The design of similarity measures

In general, we can devise a similarity measure either directly or by conversion from another proximity measure such as a dissimilarity, a distance or a pre-existent similarity. Measures in more than one dimension are constructed by making use of the aggregation operators set forth in Definitions (4.3) and (4.8). The following are basic possibilities for designing similarity measures:

$$\text{Similarity} \begin{cases} \text{from distance} \begin{cases} \text{globally} & (A) \\ \text{locally} & (B) \end{cases} \\ \text{directly } \ (C) \\ \text{from dissimilarity } \ (D) \end{cases}$$

The first two types (A) and (B) are distance-induced similarity functions. The difference stands in their local/global character. In (A), similarity is globally defined from a distance which is in turn decomposed in partial distances. In (B), similarity is built out of the aggregation of local (partial) similarities, each in turn coming from a (partial) distance.

In general, using a distance as a measure of the similarity between two objects expresses the idea that if two points are close together then they are relatively similar in terms of the features represented by their coordinates. Let X^n be a space of dimension n built out of the Cartesian product of one-dimensional spaces $X = \prod_{i=1}^{n} X^i$, in which a similarity relation has to be defined and let $\vec{x}, \vec{y} \in X^n$. The case (A) stands for a similarity s defined by means of a transformation from a metric distance in X^n, as follows:

$$s(\vec{x}, \vec{y}) = \hat{s}(\Theta(\vec{d})) \tag{4.24}$$

where $\vec{d} = \{d_1, \ldots, d_n\}$ is the vector of partial distance measures $d_i \in D(X^i)$ so that $\Theta(\vec{d}) \in D(X^n)$ and thus s is a similarity in X^n.

The case (B) stands for a similarity s defined by aggregation of partial similarities on one-dimensional spaces, each one of them coming from a transformation from a metric distance in those spaces, as follows:

$$s(\vec{x}, \vec{y}) = \check{s}(\Theta_s(\vec{s})) \tag{4.25}$$

where $\vec{s} = \{s_1, \ldots, s_n\}$ is the vector of partial similarity measures s_i defined on X^i as $s_i = \hat{s}(d(x_i, y_i))$ with $d \in D(X^i)$ and Θ_s is a similarity aggregation operator, so that s is a similarity in X^n. In the most general situation, the partial measures need not be equal, with $s_i = \hat{s}(d_i(x_i, y_i))$. Eventually, a non-linear similarity keeping function $\check{s}$ can be applied to produce the overall similarity.

The case (C) stands for a *direct* measure s, either by aggregation of partial direct measures,

$$s(\vec{x}, \vec{y}) = \check{s}(\Theta_s(\vec{s})) \tag{4.26}$$

where $\vec{s} = \{s_1, \ldots, s_n\}$ is the vector of direct partial similarity measures $s_i(x_i, y_i)$ defined on X^i or else as a globally defined similarity that takes non-local relations into account, as the one in example 2 below. An example of the former and perhaps the most widely used direct similarity, the scalar product is the more basic measure of linear dependency. It fits in this scheme by taking equal $s_i(x_i, y_i) = x_i y_i$ and Θ_s to be a simple summation. Other, more flexible functions, are the angular similarity, the covariance, and the correlation coefficient, which have seen a widespread and polemic use as similarity measures in classical data-analysis techniques [Chandon and Pinson, 81]. The $\check{s}$-function is not strictly necessary: if the measure obtained by aggregation is already non-linear, the choice $\check{s}(z) = z$ is perfectly possible.

The last situation (D) depicts the case in which a similarity measure is defined by transformation from a dissimilarity, in the form:

$$s(\vec{x}, \vec{y}) = \check{s}\left(\hat{s}(\delta(\vec{x}, \vec{y}))\right) \tag{4.27}$$

where δ is a dissimilarity measure as in Definition (4.7), conveniently transformed onto a similarity. In general, there are multiple possibilities for devising direct similarities or other weaker indexes –as a pseudo-similarity– even in cases for which an Euclidean (or Minkowskian) metric could in principle have been used. Consider the following two scenarios.

1. Given two vectors $\vec{x}, \vec{y} \in \mathbb{Z}^n$ the similarity we wish to apply should reflect the degree to which the two vectors have all their respective components different (i.e., to what degree the number of different $x_1, \ldots, x_n$ matches the number of different $y_1, \ldots, y_n$). Euclidean distance (or any member of its family, weighted or not) will not capture this simple relation. Even by normalizing the vectors to unit norm, the problem is still unsolved. Denoting

$$N(\vec{z}) = n - \#i : 1 \leq i \leq n : (\exists j : i < j \leq n : z_i = z_j) \tag{4.28}$$

   we have

$$d(\vec{x}, \vec{y}) = \frac{|N(\vec{x}) - N(\vec{y})|}{n - 1} \in [0, 1]$$

   This is not a distance in $\mathbb{Z}^n$, because it violates the first condition. However, the transformation $s(\vec{x}, \vec{y}) = 1 - d(\vec{x}, \vec{y})$ is a pseudo-similarity measure in $\mathbb{Z}^n$ (with $s_{max} = 1$), since $\vec{x} = \vec{y} \Rightarrow s(\vec{x}, \vec{y}) = 1$, although the opposite need not be true.

2. Given two vectors $\vec{x}, \vec{y} \in \mathbb{R}^n$, possibly coding some form of a temporal data series, we regard them as more similar the higher is their number of common elements. The number of such points can be defined as:

$$N(\vec{x},\vec{y}) = \#i : 1 \leq i \leq n : \\ (\#j : 1 \leq j \leq n : x_j = x_i) = (\#j : 1 \leq j \leq n : y_j = x_i) \tag{4.29}$$

and a direct measure can be defined:

$$s(\vec{x},\vec{y}) = \frac{N(\vec{x},\vec{y})}{n} \in [0,1]$$

that is a pseudo-similarity in $\mathbb{R}^n$, with $s_{max} = 1$.

In the first example, although the measure $d(\vec{x},\vec{y})$ proposed *fulfills* the triangular inequality -the hardest of the three conditions in (4.1)- the violation of the first condition prevents us to define a proper metric; therefore, on the one hand, scalar product or Euclidean measures will have a hard time discovering the similarity relation, which is in fact a simple one, *once it is known*. On the other, we are not able to devise an alternative distance measure that correctly captures the similarity in pattern space. In the second example, the underlying relation is not decomposable component-wise. In both situations, the relation can be defined in terms of a (in this case, pseudo) similarity and still get a working model.

An immediate derivation of the four defined types of similarity is that they can be combined, as long as an overall similarity measure is devised via aggregation, so as to keep within the introduced framework, leading to potentially more general and useful measures.

4.2.5 Proper scalar product-based similarity

In previous sections -(§3.2) and (§3.6)- we have seen how distance and similarity are opposite concepts (the more the distance, the less the similarity, and vice versa). In order to cast the classical neuron models defined in (2.2) and (2.3) as similarity computing devices, it is necessary to design a proper scalar product-based measure -by proper we mean fulfilling the required properties in Definition (4.7). To this end, it is useful to examine their precise relationship. Recall from (§3.4) that in $\mathbb{R}^n$ scalar product is tied to Euclidean distance. Let us assume, for simplicity, an unweighted Euclidean distance d in n-dimensions. We have:

$$\begin{aligned} d^2(\vec{x}-\vec{y}) = \|\vec{x}-\vec{y}\|^2 &= \sum_i^n (x_i - y_i)^2 \\ &= \sum_i^n x_i^2 + \sum_i^n y_i^2 - 2\sum_i^n x_i y_i \\ &= \|\vec{x}\|^2 + \|\vec{y}\|^2 - 2\vec{x}\cdot\vec{y} = \end{aligned}$$

At this point we see that they are effectively inverses. Now, assuming normalized vectors, as in (3.6):

$$= 2 - 2\vec{x}'\cdot\vec{y}' = 2(1 - \vec{x}'\cdot\vec{y}')$$

Thus, *if* both vectors are normalized, then scalar product is a direct function of the distance, with no other factor involved:

$$\vec{x}' \cdot \vec{y}' = 1 - \frac{d^2}{2} \in [-1, 1] \tag{4.30}$$

where $d \in [0, 2]$. Therefore,

$$s(\vec{x}', \vec{y}') = \frac{\vec{x}' \cdot \vec{y}' + 1}{2} \tag{4.31}$$

is a neat similarity function of type (A) in $[0, 1]$, easily obtained by substituting (4.30) in (4.31) and applying Proposition (4.14) to Euclidean distance using $\hat{s}(z) = 1 - \left(\frac{z}{2}\right)^2$, for $z = d(\vec{x}', \vec{y}')$, as:

$$s(\vec{x}', \vec{y}') = \hat{s}(d(\vec{x}', \vec{y}'))$$

This is a translated and scaled form of a measure usually known as ***angular similarity*** [Chandon and Pinson, 81]:

$$s(\vec{x}', \vec{y}') = s(\frac{\vec{x}}{||\vec{x}||}, \frac{\vec{y}}{||\vec{y}||}) = \left(\frac{\vec{x} \cdot \vec{y}}{||\vec{x}||\,||\vec{y}||} + 1\right)\frac{1}{2} = \frac{\cos(\theta) + 1}{2} \tag{4.32}$$

where $\theta \in [0, \pi]$ is the angle between $\vec{x}'$ and $\vec{y}'$. However, the normalization condition for both vectors is rarely found in ANN, and renders the measure expressed in (4.31) useless in practice. Unfortunately, if this condition is not met, the expression in (4.32), although it seems to be a "normalized" one, is *not* a proper similarity measure, since it does not fulfill condition (4) in Definition (4.7). It is, instead, a pseudo-similarity. This is caused by the fact that *different* vectors can be considered as maximally similar: it suffices that they are co-linear and point in the same direction.

It has to be remarked that other approaches to unification [Dorffner, 95] have focused on linking $\vec{x} \cdot \vec{y} + \theta$ directly with Euclidean distance, which only requires the condition $||\vec{x}|| = 1$, by setting $2\theta = -||\vec{y}||^2$. Still, this approximation is a link between *neuron models*, expressing how to make a P-neuron mimic Euclidean distance. To have a fair comparison, the smoothing factor in a R-neuron should be considered too. Besides, notice that $\vec{x} \cdot \vec{y} + \theta$ is *not* a scalar product on $\mathbb{R}^n$.

As a side note observe that, for general vectors, no measure directly derived as a function of $\vec{x} \cdot \vec{y}$–as (4.32)– can act as a RBF unit, because it cannot be centered in a particular point. Still, although there is a clear-cut conceptual difference between these two kinds of models, this does not mean that scalar product cannot be used to define a similarity measure.

To get a full-compliant scalar product-based similarity, let:

$$s(\vec{x}, \vec{y}) = \frac{1}{2}\left(\frac{\vec{x} \cdot \vec{y}}{||\vec{x}||\,||\vec{y}||} + 1\right)\frac{min(||\vec{x}||, ||\vec{y}||)}{max(||\vec{x}||, ||\vec{y}||)} \in [0, 1] \tag{4.33}$$

This is a *proper* similarity measure only based on the scalar product operator. To see this, making use of $||\vec{x}|| = \sqrt{\vec{x} \cdot \vec{x}}$, (4.33) can be written as:

$$s(\vec{x},\vec{y}) = \begin{cases} \frac{1}{2}\left(\frac{\vec{x}\cdot\vec{y}}{\vec{y}\cdot\vec{y}} + \sqrt{\frac{\vec{x}\cdot\vec{x}}{\vec{y}\cdot\vec{y}}}\right) & \text{if } \sqrt{\vec{x}\cdot\vec{x}} \le \sqrt{\vec{y}\cdot\vec{y}} \\ \frac{1}{2}\left(\frac{\vec{x}\cdot\vec{y}}{\vec{x}\cdot\vec{x}} + \sqrt{\frac{\vec{y}\cdot\vec{y}}{\vec{x}\cdot\vec{x}}}\right) & \text{if } \sqrt{\vec{x}\cdot\vec{x}} \ge \sqrt{\vec{y}\cdot\vec{y}} \end{cases} \tag{4.34}$$

Proposition 4.18 *The expression in (4.33) is a similarity measure in $\mathbb{R}^n$.*

Proof. Since $s \in [0,1]$, the non-negativity and boundedness conditions (with $s_{max} = 1$) are easily met. Symmetry is clearly fulfilled. Minimality deserves a special attention. For convenience, we denote by A the first term in brackets (including the $\frac{1}{2}$), and by B the second (the min/max fraction).

$\Rightarrow$ Let us suppose that $s(\vec{x},\vec{y}) = 1$. In these conditions, it is mandatory that both $A = 1$ and $B = 1$. The first is so only for co-linear vectors, and pointing in the same direction. The second is so only for vectors of equal magnitude. Thus, $\vec{x} = \vec{y}$.

$\Leftarrow$ Let us suppose that $\vec{x} = \vec{y}$. Then, $A = 1$ (properties of scalar product) and $B = 1$ (min=max). Thus, $s(\vec{x},\vec{y}) = 1$.

This measure cannot be assumed by any existing neuron model. Therefore, a new scalar product-based model could be obtained as:

$$F_i(\vec{x}) = \check{s}\left(s(\vec{x},\vec{w}^i)\right)$$

where s is as in (4.33), and $\check{s}$ is a non-linear similarity keeping function (e.g., a sigmoidal).

From the basic scalar product, it is possible to have a bounded measure without the need to perform vector normalizations, as follows. Let:

$$\begin{cases} \pi^+_{(K)} = \sup\limits_{\vec{x},\vec{y}\in K\subset\mathbb{R}^n} \vec{x}\cdot\vec{y} \\ \pi^-_{(K)} = \inf\limits_{\vec{x},\vec{y}\in K\subset\mathbb{R}^n} \vec{x}\cdot\vec{y} \end{cases} \tag{4.35}$$

to denote $\forall\vec{x},\vec{y} \in K : \vec{x}\cdot\vec{y} \in [\pi^-_{(K)}, \pi^+_{(K)}]$. It is then assumed that the target function (the function to be approximated by the neural network) has a compact support K, being $\mathbb{R}^n$ the space where input patterns are drawn from, so that there exists a $\vec{z}^*$ such that

$$\vec{z}^* = \operatorname*{argmax}_{\vec{z}\in K\subset\mathbb{R}^n} ||\vec{z}||^2 \tag{4.36}$$

Since in (4.35) we have a continuous function defined in a closed interval, its high and low bounds exist and are reached within the interval, so that $\pi^+_{(K)}, \pi^-_{(K)} \in \mathbb{R}$, and the measure:

$$s(\vec{x},\vec{y}) = \frac{\vec{x}\cdot\vec{y} - \pi^-_{(K)}}{\pi^+_{(K)} - \pi^-_{(K)}} \in [0,1],\ \vec{x},\vec{y} \in K \tag{4.37}$$

is a similarity in K. To be precise, what is obtained is a *point* similarity, which can serve as a basis to construct a P-neuron, where the ideal is $\vec{z}^*$ in (4.36). For simplicity, since in $\mathbb{R}^n$ a compact set always has the form of a closed and bounded set, let $K = [m, M]^n \subset \mathbb{R}^n, m \leq 0, M \geq 0$. In these conditions, define $\vec{\beta} = (\beta, \cdots, \beta)$ of dimension n for any scalar β and denote $\mu = max(|m|, M)$. The maximum and minimum scalar products in K are given by:

$$\begin{cases} \pi^+_{(K)} = & \|\vec{\mu}\|^2 \\ \pi^-_{(K)} = & -\|\vec{m}\|\,\|\vec{M}\| \end{cases} \tag{4.38}$$

An offset θ can now be incorporated into the definition, as required by a P-neuron. The need to obtain a normalized measure compels to set definite bounds on θ. Arbitrarily, we set $\theta \in [0,1]$. Hence, $\vec{x} \cdot \vec{y} + \theta \in [\pi^-_{(K)}, \pi^+_{(K)} + 1]$ and consequently the measure:

$$s(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y} + \theta - \pi^-_{(K)}}{\pi^+_{(K)} + 1 - \pi^-_{(K)}} \in [0,1], \ \vec{x}, \vec{y} \in K, \theta \in [0,1] \tag{4.39}$$

is a point similarity in K. A further simplification can be obtained by considering the particular case $m = -M$ (a symmetric zero-centered interval). In this situation, for $\vec{x}, \vec{y} \in K$, $\vec{x} \cdot \vec{y} \in [-\|\vec{M}\|^2, \|\vec{M}\|^2]$, and

$$s(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y} + \theta + \|\vec{M}\|^2}{2\|\vec{M}\|^2 + 1} \in [0,1] \tag{4.40}$$

is a point similarity measure in K.

4.2.6 Neural models of similarity

We are now in a position to summarize the casting of neuron models as similarity computers, in the following Propositions:

Proposition 4.19 *A R-neuron model of the kind declared in Definition (2.3) computes a similarity measure s in* $\mathbb{R}^n$ *of the form* $F_i(\vec{x}) = s(\vec{x}, \vec{w}^i)$.

Proof: these models, as described in (§4.2.4), fall within type (A) similarities in $\mathbb{R}^n$, in this case of the form $s(\vec{x}, \vec{w}^i) = \hat{s}(d_q^{(n)}(\vec{x}, \vec{w}^i))$, with $d_q^{(n)}(\vec{x}, \vec{w}^i) = \Theta^{q,n}(\vec{x} - \vec{w}^i; \vec{v})$, as in Proposition (4.3), with $v_i = \theta, \forall i : 1 \leq i \leq n$, followed by a similarity transforming function $\hat{s} = g$. By Proposition (4.14), this is a similarity in $\mathbb{R}^n$.

Proposition 4.20 *For normalized vectors, a P-neuron model of the kind declared in Definition (2.2) computes a similarity measure s in* $\mathbb{R}^n$ *of the form* $F_i(\vec{x}) = s(\vec{x}, \vec{w}^i)$.

Proof: in these conditions, this model behaves as a R-neuron, by taking $q = 2$ and $\hat{s}(z) = g(1 - (\frac{z}{2})^2)$, as seen in (§4.2.5). By using Propositions (4.15) and (4.19), this is a similarity in $\mathbb{R}^n$.

Proposition 4.21 *For general, non-normalized vectors, a neuron model based on angular similarity (4.32) can be constructed, of the form $F_i(\vec{x}) = \check{s}(s(\vec{x}, \vec{w}^i))$, computing a pseudo-similarity measure in $\mathbb{R}^n$.*

Proof. Since the only failing condition of those in Definition (4.7) is minimality for the left-to-right implication, s is a pseudo-similarity measure. That is, $\vec{x} = \vec{w}^i \rightarrow s(\vec{x}, \vec{w}^i) = s_{max}$ but the opposite is in general not true. The composition of a $\check{s}$-function to such an s, gives a pseudo-similarity measure.

Proposition 4.22 *For general, non-normalized vectors, a new scalar product-based neuron model using the measure defined in (4.33) can be obtained as: $F_i(\vec{x}) = \check{s}\left(s(\vec{x}, \vec{w}^i)\right)$, where s is a similarity measure in $\mathbb{R}^n$, and $\check{s}$ is a non-linear similarity keeping function (e.g., a sigmoidal).*

Proof. Being (4.33) a proper similarity, the composition of a similarity keeping function $\check{s}$ to $s(\vec{x}, \vec{y})$ makes the result a similarity, by Proposition (4.13).

For general, non-normalized vectors, a P-neuron model of the kind declared in Definition (2.2) can be shown *not* to compute a similarity measure, since minimality (the only failing condition) is not fulfilled in either direction of the implication. The scalar-product based model generically falls within type (C) or correlation-based similarities, plus an additional similarity keeping function $\check{s}(z) = g(z)$, as described in (§4.2.4). Being this last function bounded, a P-neuron model can be obtained computing the weaker minimality requirement of a point similarity. Such a measure is clearly symmetric and continuous. The semantics of this similarity was amply described in (§3.2).

Proposition 4.23 *For general, non-normalized vectors, a P-neuron model of the kind declared in Definition (2.2) can be constructed with the measure in (4.39) to compute a point similarity measure in $K \subset \mathbb{R}^n$, of the form $F_i(\vec{x}) = \check{s}(s(\vec{x}, \vec{w}^i))$.*

Proof: First, we prove that s in (4.39) is a similarity. Since $s \in [0, 1]$, the non-negativity and boundedness conditions (with $s_{max} = 1$) are easily met, as well as symmetry and a clear semantics.

Let $\underline{\vec{x}} = (1, x_1, \ldots, x_n)$ and $\underline{\vec{w}}^i = (\theta, w_1, \ldots, w_n)$, that is, for convenience offset term is included in the summation, as is common practice. Then, (4.39) can be written:

$$s(\vec{x}, \vec{w}^i) = \frac{\underline{\vec{x}} \cdot \underline{\vec{w}}^i - \pi^-_{(K)}}{\pi^+_{(K)} + 1 - \pi^-_{(K)}} \in [0, 1], \; \vec{x}, \vec{y} \in K, \theta \in [0, 1] \tag{4.41}$$

Clearly, this is a point similarity, where the ideal is $\underline{z}^* = (1, z_1^*, \ldots, z_n^*)$, being $\vec{z}^*$ in (4.36). The required minimality is met because:

$$s(\vec{x}, \vec{w}^i) = 1 \Longleftrightarrow \vec{x} = \vec{w}^i = \underline{z}^*$$

Second, again the composition of a similarity keeping function $\breve{s}$ to $s(\vec{x}, \vec{y})$ in (4.39) makes the result a similarity, by Proposition (4.13). Third, every function g –of a given parameterized family– of the required form in (2.2) can be transformed to a g' by choosing a different set of parameters (acting as constants) so that $g'(\vec{x} \cdot \vec{y}) = g(s(\vec{x}, \vec{y}))$.

For example, letting $g = g_{\beta,\underline{\theta}}$ the classical logistic function as defined in (3.9):

$$g_{\beta,\underline{\theta}}(z) = \frac{1}{1 + e^{-\beta(z-\underline{\theta})}}, \qquad \beta > 0 \tag{4.42}$$

and the particular case $m = -M$ (4.40), a model of the form:

$$g(s(\vec{x}, \vec{y})) = g(\frac{\vec{x} \cdot \vec{y} + \theta + ||\vec{M}||^2}{2||\vec{M}||^2 + 1})$$

is a P-neuron because the above expression can be equivalently written:

$$g(s(\vec{x}, \vec{y})) = g'(\vec{x} \cdot \vec{y} + \theta)$$

where $g' = g_{\beta',\underline{\theta}'}$ is just another logistic, of the same shape, though with different parameters:

$$\beta' = \frac{\beta}{2||\vec{M}||^2 + 1}; \; \underline{\theta}' = ||\vec{M}||^2(2\underline{\theta} - 1) + \underline{\theta} - \theta \tag{4.43}$$

This measure offers the additional advantage of being much cheaper to compute than (4.32), because the involved factors are constant. As an additional example, consider a collection of data properly normalized to lie in $D^n \subset \mathbb{R}^n = [0,1]^n$ (a common situation in practice, see [Prechelt, 94], [Sarle, 99]). Then $\pi^+_{(D^n)} = n$, $\pi^-_{(D^n)} = 0$ in (4.37), and thus:

$$s(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y} + \theta}{n+1} \in [0,1] \tag{4.44}$$

is a point similarity in D^n. In later parts of the work (§4.4), the assumption in (§4.2.5) expressed in (4.35) about a compact support K for both vectors will be additionally justified.

In summary, scalar product-based neuron models are in general used as similarity computing devices in a rather loose sense. A variety of derived measures can be obtained corresponding to the different types of similarities. In particular, in practice the assumption of a compactly supported target function is tenable, and thus a P-neuron can be effectively seen as computing a point similarity in $\mathbb{R}^n$.

4.3 Heterogeneous spaces

A process can be generally described by means of a set of variables that are thought to exert an influence on it. The values of these variables are either collected by the user or generated by a model. In the case of ANN, they are grouped in the form of input vectors, although the term input *tuple* would seem to be more accurate. Among the data collected there is information of very different nature.

4.3.1 Definition of heterogeneous measures

We consider in this work the following types of variables, for which corresponding similarity measures are to be defined.

Nominal (categorical) : non-numerical variable on which no order relation has been defined. It thus can be seen as having a *set* of values (finite or not).

Ordinal : variable (numerical or not) for which a linear order relation has been defined on a finite number of values, where each value has a crisp or precise sense.

Continuous : numerical and crisp variable for which a linear order relation has been defined on a continuum of values.

Set : variable whose values are sets.

Fuzzy Number : continuous variable whose values are expressing uncertainty in the form of *imprecision*.

Linguistic : ordinal variable whose values are expressing uncertainty in the form of *vagueness*.

The values of the last two variables can be obtained —where appropriate— by converting each of an existing set of crisp values (ordinal or continuous) into a fuzzy quantity.

Nominal variables

The basic similarity measure for these variables is the overlap. Let $\mathcal{N}$ be a *nominal* space and $x, y \in \mathcal{N}$.

$$s(x,y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases} \tag{4.45}$$

Proposition 4.24 *The function defined in (4.45) is a similarity measure in $\mathcal{N}$.*

Ordinal variables

It is assumed that a *linear order* exists such that the values of the variable form a total linearly ordered space. Let $\mathcal{O}$ be an *ordinal* space and $x, y \in \mathcal{O}$, with the notation $\#\{\cdot\}$ to denote cardinality. Let $\preceq$ be the linear order in $\mathcal{O}$. An equality relation $=$ is also assumed so that strict inequality

$$x' \prec x \equiv x' \preceq x \wedge \neg(x' = x)$$

is defined in the usual way. Let us define now:

$$\eta : \mathcal{O} \rightarrow [1, m] \subset \mathrm{N}^+$$

as

$$\eta(x) = \#\{x' \in \mathcal{O} : x' \prec x\} + 1, \qquad x \in \mathcal{O}$$

Since the order $\preceq$ is linear, this is a bijection. Since every finite set can be made well-ordered, $\mathcal{O}$ has a first (x_f) and a last element (x_l), i.e., elements such that $\not\exists x' \in \mathcal{O} : x' \prec x_f$ and $\not\exists x' \in \mathcal{O} : x_l \prec x'$, respectively. Therefore, $\eta(x_f) = 1$, $\eta(x_l) = \#\{\mathcal{O}\} = m$ and:

$$\eta(x') = \eta(x) + 1 \Leftrightarrow x' = succ(x)$$

where the $\Leftarrow$ holds by definition of η and $\Rightarrow$ by the linear character of $\preceq$. By doing this we can devise a distance in $\mathcal{O}$ resorting to a metric in $\mathbb{R}$, using Propositions (4.1) and (4.3) for $S = \mathcal{O}$, and making use of the fact that η is a bijection. Hence, for $x, y \in \mathcal{O}$, define d as:

$$d(x, y) = |\eta(x) - \eta(y)| \in D(\mathcal{O})$$

A normalized distance can be obtained as:

$$d(x, y) = \frac{|\eta(x) - \eta(y)|}{\#\{\mathcal{O}\} - 1} \in [0, 1], d \in D(\mathcal{O})$$

and, by selecting $\hat{s}(z) = 1 - z$, $s(x, y) = \hat{s}(d(x, y))$ is a similarity measure in $\mathcal{O}$, by Proposition (4.16) (though others choices for $\hat{s}(z)$ are possible).

Proposition 4.25 *Provided $\mathcal{O}$ is finite and linearly ordered:*

$$s(x, y) = 1 - \frac{|\eta(x) - \eta(y)|}{\#\{\mathcal{O}\} - 1} \qquad x, y \in \mathcal{O} \tag{4.46}$$

is a similarity measure in $\mathcal{O}$, with $s_{max} = 1$.

A measure different in nature can be defined as follows:

$$s(x, y) = \frac{min(\eta(x), \eta(y))}{max(\eta(x), \eta(y))}, \ x, y \in \mathcal{O} \tag{4.47}$$

which does not need any normalization in order to upper-bound it to unity. An equivalent form for (4.47) is:

$$s(x,y) = \begin{cases} \frac{\eta(x)}{\eta(y)} & \text{if } \eta(x) \leq \eta(y) \\ \frac{\eta(y)}{\eta(x)} & \text{otherwise} \end{cases} \tag{4.48}$$

The conceptual difference between this measure and (4.46) is that, while the latter is a function of $\eta(x) - \eta(y)$, this one is a function of $\frac{\eta(x)}{\eta(y)}$, which can be readily seen by rewriting it as:

$$g(z) = \begin{cases} z & \text{if } z \leq 1 \\ \frac{1}{z} & \text{if } z \geq 1 \end{cases} \tag{4.49}$$

so that $s(x,y) = g(\frac{\eta(x)}{\eta(y)})$. In general, $s(x,y)$ in (4.47) belongs to $[\frac{1}{m}, 1]$. Thus,

Proposition 4.26 *Provided $\mathcal{O}$ is finite and linearly ordered:*

$$s(x,y) = \frac{1}{m-1}\left(m\frac{min(\eta(x),\eta(y))}{max(\eta(x),\eta(y))} - 1\right) \in [0,1], \qquad x,y \in \mathcal{O} \tag{4.50}$$

is a similarity measure in $\mathcal{O}$, with $s_{max} = 1$.

The behaviour of this measure can be best seen with an example . Consider, for simplicity, $\mathcal{O} = \{a,b,c,d\}$ with the usual alphabetic linear order $a \preceq b \preceq c \preceq d$. Were it not for this ordering (that is, for *nominal* variables), according to (4.45) the similarities between *different* values would be:

$$s(a,b) = s(a,c) = s(b,c) = s(a,d) = s(b,d) = s(c,d) = 0$$

Taking the variables as described, and according to (4.46), these similarities are:

$$s(a,d) = 0;\ s(a,b) = s(b,c) = s(c,d) = \frac{2}{3};\ s(a,c) = s(b,d) = \frac{1}{3}$$

reflecting the effect of having defined an order. However, according to (4.50), they are shown to fulfill:

$$s(c,d) > s(b,c) > s(a,b) = s(b,d) > s(a,c) > s(a,d)$$

That is, the ordinal scale is compressed towards superior values in the scale. This could be a desired effect in some situations. For example, in a variable standing for the number of children, the distance between 7 and 9 is not the same psychological distance than that between 1 and 3 (it should be less), as pointed out in (§3.3). Whereas, assuming $m = 10$, $s(7,9) = s(1,3) = \frac{2}{9}$ by (4.46), $s(7,9) = \frac{7}{9} > s(1,3) = \frac{1}{3}$ by (4.50).

As another example, consider a numerical set $\mathcal{O} = [1, 1000] \subset \mathrm{N}^+$. Now, $s(101, 103) = s(10, 12)$ by (4.46), but $s(101, 103) > s(10, 12)$ by (4.50), because the difference (two) is smaller *relative* to the values being compared. The decision on which measure to use should be based on domain knowledge. Notice that (4.47) is also valid for infinite linearly-ordered and discrete sets, such as N, still upper-bounded by unity and asymptotically reaching zero as the low bound.

Finally note that, in a working implementation, the set $\mathcal{O}$ can be safely replaced by $[1, m] \subset \mathrm{N}^+$ as long as the elements are not arithmetically operated beyond (4.46) or (4.50).

Continuous variables

Let $x, y \in \Gamma = [r^-, r^+] \subset \mathbb{R}, r^+ > r^-$. By Propositions (4.1) and (4.2), the standard metric d in $\mathbb{R}$ is a metric in Γ. Therefore, any $s(x, y) = \hat{s}(d(x, y))$ is a similarity measure in Γ. A normalized distance can be obtained by setting:

$$d(x, y) = \frac{|x - y|}{\sup_{x,y \in \Gamma} |x - y|} \in [0, 1], d \in D(\Gamma) \tag{4.51}$$

which is the standard distance weighted by the maximum deviation.

Proposition 4.27 *Given $\Gamma \subset \mathbb{R}$ as above, the function:*

$$s(x, y) = 1 - \frac{|x - y|}{\sup_{x,y \in \Gamma} |x - y|}, \qquad x, y \in \Gamma \tag{4.52}$$

is a similarity measure in Γ, with $s_{max} = 1$.

Notice that this is equivalent to the ordinal case (4.50), where normalization is performed dividing by the number of elements in the space. Again, by selecting $\hat{s}(z) = 1 - z$, $s(x, y) = \hat{s}(d(x, y))$ is a basic similarity measure in Γ. Other choices for $\hat{s}(z)$ are possible and eligible by parameterizing it in appropriate ways. This is done in (§4.4.2), leading to the family $\hat{s}_0$.

Set variables

These can be regarded as generalized *nominal* variables whose values are not single elements of the space but subsets thereof. Let $\mathcal{S}$ be a *set* space and $x, y \in \mathcal{S}$.

Proposition 4.28 *Provided $\mathcal{S}$ is finite (so that all its subsets are),*

$$s(x, y) = \frac{\#\{x \cap y\}}{\#\{x \cup y\}}, \qquad x, y \neq \emptyset \tag{4.53}$$

is a similarity measure in S, with $s_{max} = 1$.

Note that this measure reduces to (4.45) for singleton sets.

Fuzzy variables

The word *fuzzy* is viewed in this work taking the *epistemic* interpretation of a fuzzy set, i.e., as describing the vague observation of an existing theoretically crisp object. The object is crisp, but its observation or realization is vague [Nauck, Klawonn and Kruse, 92]. The imprecision stems from the indetermination about the specific value of a variable, in a situation where the set of all its possible values is known. The fuzzy extension is understood as a relaxation of real-valued inputs, by considering more flexible situations, now tolerating imprecision.

The basic notions of fuzzy sets were reviewed in (§2.3). Recall $supp(F)$ denotes the *support* of a fuzzy set F and let $[\mathbb{R}]$ denote the set of all finite and closed real intervals:

$$[\mathbb{R}] = \{[a,b] \subset \mathbb{R} \mid a,b \in \mathbb{R}, a \leq b\}$$

Let us begin by defining a basic concept [Klir, 88]:

Definition 4.11 (Fuzzy number) *A fuzzy number in X is a convex and normalized fuzzy set F, with piecewise continuous μ_F, where X is the reference set, and such that $\exists! x \in X : \mu_F(x) = 1$. Symmetry of μ_F is a useful simplifying assumption although is not required.*

Let $\mathbb{F}_n(X)$ be the (crisp) set of all the fuzzy numbers in X:

$$\mathbb{F}_n(X) = \{F \in \tilde{P}(X) \mid F \text{ is a fuzzy number}\}$$

where X is assumed a continuum. Normally, we will consider X to be a real interval, that is, $X \in [\mathbb{R}]$. Notice that convexity of F ensures the compacity of $supp(F)$.

In general, a fuzzy set F with support in $[a,b] \in [\mathbb{R}]$ is represented by a function of the form:

$$\mu_F(x) = \begin{cases} \alpha & \text{if } x \in [a,b] \\ 0 & \text{if } x \notin [a,b] \end{cases} \tag{4.54}$$

where $\alpha \in [0,1]$. The above definition for fuzzy numbers (4.11) restricts the form $\mu_F(x)$ can take on $[a,b]$ so that it represents single values attached with uncertainty or inaccuracy. Among the several possible forms such a fuzzy set can take, two of the most popular are:

Triangular

Correspond to non-symmetric fuzzy sets of the form:

$$\mu_{[m]}^{[\alpha_1,\alpha_2]}(z) = \begin{cases} \frac{z-(m-\alpha_1)}{\alpha_1} & \text{if } z \in [m-\alpha_1, m] \\ \frac{(m+\alpha_2)-z}{\alpha_2} & \text{if } z \in [m, m+\alpha_2] \\ 0 & \text{otherwise} \end{cases} \tag{4.55}$$

In both situations, m is the mode or center and α_1, α_2 the left and right fuzziness, respectively. In case the number is symmetric, it may be written in a more compact form:

$$\mu_{[m,\alpha]}(z) = \begin{cases} 1 - \frac{|z-m|}{\alpha} & \text{if } |z-m| \leq \alpha \\ 0 & \text{otherwise} \end{cases} = 1 - min\left(\frac{|z-m|}{\alpha}, 1\right) \tag{4.56}$$

Gaussian

Correspond to symmetric fuzzy sets of the form:

$$\mu_{[m,\alpha]}(z) = exp\{-\left(\frac{z-m}{\alpha}\right)^2\} \tag{4.57}$$

Since it is unlikely that collected data come already in this fuzzified form, there has to be a *fuzzification* process, much as is done in Fuzzy Controllers [Jang and Sun, 95]. In these systems, crisp values u_0, in absence of other information, are transformed onto a fuzzy number of the form:

$$\mu_{u_0}(u) = \begin{cases} 1 & \text{if } u = u_0 \\ 0 & \text{otherwise} \end{cases} \tag{4.58}$$

In this work, however, a numerical variable will be considered a fuzzy number (and treated in a different way than a crisp one) *only* if additional information is available; otherwise it is considered as continuous (and crisp). One of the forms this information can take is about the reliability or error of measured values or expert estimates, which can be translated into the fuzzy number's left/right fuzziness. The inaccuracy is then transformed into an interval of possible values. The fact that not all these values equally belong to the interval makes it a fuzzy one.

For every $x \in \mathbb{R}$, let $\tilde{x}$ be a fuzzy number of a given form centered at x and with fuzziness generally defined as a function of x. The transformation:

$$\tilde{\cdot} : \quad x \in \mathbb{R} \xrightarrow{\sim} \tilde{x} \in \mathbb{F}_n(\mathbb{R}) \tag{4.59}$$

associates a fuzzy number $\tilde{x}$ to x where $\mu_{\tilde{x}} = \mu_{[x,\alpha(x)]}$ is a membership function expressing a fuzzy number –as (4.55) or (4.57)– and $\alpha(x)$ stands for as many fuzziness parameters as required. The resulting fuzzy number is to express the *degree of uncertainty*. This interpretation was proposed by Zadeh when introducing possibility theory [Zadeh, 78], in which a membership expression like $\mu_{\tilde{x}}(u)$ represents the degree of possibility that x has value u. In other words, the values $u \in supp(\tilde{x})$ have different degrees of uncertainty (given by $\mu_{\tilde{x}}(u)$), characterizing the extent to which u can be the possible value of the quantity.

Domain knowledge can be added to refine the transformation. For instance, let us suppose that the value x comes from a measuring device known to be biased towards underestimating the actual value at most a 2%, and overestimating it at most a 1%. A (non-symmetric) fuzzy number –depicted in Fig. (4.3)– could be derived, where $\mu_{\tilde{x}} = \mu_{[x,\alpha(x)]}$, $\alpha(x) = (\alpha_1(x), \alpha_2(x)) = (k_1 x, k_2 x)$ and $k_1 = 0.02, k_2 = 0.01$.

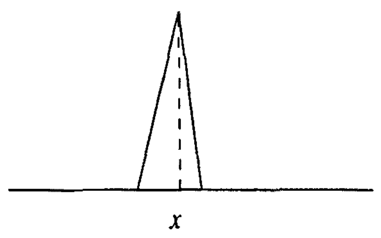


Figure 4.3: Example form for $\mu_{\tilde{x}}$ (not to scale).

Given two fuzzy numbers, the question is: how similar are they? For variables representing fuzzy sets, similarity relations from the point of view of fuzzy theory have been defined elsewhere [Dubois *et al.*, 96], [Dubois *et al.*, 97], and different choices are possible. In the present case, the situation is not that of a fuzzy similarity or proximity relation defined on real values, but a crisp relation between fuzzy entities.

In possibility theory two functions, called *possibility* (Π) and *necessity* (N), measure rank events by the degree of *unsurprisingness* and *acceptance*, respectively [Dubois and Prade, 97]. In particular, the first of these functions expresses the possibility of co-occurrence or simultaneity of two vague propositions, with a value of 1 standing for absolute certainty. For two fuzzy sets $\tilde{A}, \tilde{B}$ possibility is defined as:

$$\Pi_{\tilde{A}}(\tilde{B}) = \sup_{u \in X} (\mu_{\tilde{A} \cap \tilde{B}}(u))$$

where $\mu_{\tilde{A} \cap \tilde{B}}(u) = \min\ (\mu_{\tilde{A}}(u),\ \mu_{\tilde{B}}(u))$. Notice that this measure is reflexive in the strong sense and also symmetric, in that $\Pi_{\tilde{A}}(\tilde{B}) = \Pi_{\tilde{B}}(\tilde{A})$, both being required properties of general similarity measures.

Proposition 4.29 *Given* $X, Y \in \mathbb{F}_n(\Gamma), \Gamma \in [\mathbb{R}]$, *and making use of the transformation in (4.59), the function*

$$s : \mathbb{F}_n(\Gamma), \mathbb{F}_n(\Gamma) \to [0, 1]$$

defined as:

$$s(x, y) = \Pi^*(\tilde{x}, \tilde{y}) \tag{4.60}$$

where $\Pi^*(\tilde{x}, \tilde{y}) = \Pi_{\tilde{x}}(\tilde{y})$, *is a similarity measure in* $\mathbb{F}_n(\Gamma)$, *with* $s_{max} = 1$.

Proof: Since $s \in [0, 1]$, non-negativity and boundedness conditions (with $s_{max} = 1$) are easily met. There is also symmetry and a clear semantics. Minimality is met because, given $\alpha(x)$ is the same function for all $x \in \mathbb{R}$, it holds that $x = y \Leftrightarrow \tilde{x} = \tilde{y} \Leftrightarrow \Pi^*(\tilde{x}, \tilde{y}) = 1$.

Notice that, if one of the arguments, say $\tilde{x}$, is crisp, the measure yields its membership w.r.t. the other: $\Pi^*(x, \tilde{y}) = \mu_{\tilde{y}}(x)$. This is a simple and effective similarity measure under the context of *fuzzy numbers*. For more general fuzzy sets, other measures should be defined.

Linguistic variables

The integration of numeric and qualitative information —the latter in the form of fuzzy sets— has been pointed out to have several advantages in the framework of pattern recognition [Pedrycz, 97], either enriching the basic ideas or giving rise to completely new concepts. The addition of linguistic features for expressing vagueness both in the input patterns *and* in the inner workings of the system itself can lead to new architectures with enhanced expressiveness and flexibility. The linguistic approach [Zadeh, 76] considers the variables by means of linguistic terms (words in the most basic situation). Since words are less precise than numbers, this approach enables the use of vague information, in cases where a precise quantity is unknown or there is a need or convenience to abstract it out.

We set the following definitions:

Definition 4.12 (Fuzzy quantity) *A fuzzy quantity in $\mathbb{R}$ is a normalized fuzzy set F with continuous μ_F such that $supp(F) \in [\mathbb{R}]$.*

Definition 4.13 (Fuzzy interval) *A fuzzy interval in X is a convex fuzzy quantity in X. Symmetry of μ_F is a useful simplifying assumption although is not required.*

Definition 4.14 (Fuzzy p-quantity) *A fuzzy p-quantity in X is a fuzzy interval in X, such that μ_F can be decomposed in three parts (from left to right): a function with at most one inflection point and a positive first derivative, a flat zone (null derivative) and a function with at most one inflection point and a negative first derivative*[5].

Note that a fuzzy number is a unimodal fuzzy p-quantity. These quantities (depicted in Fig. 4.4) are useful for modeling the linguistic terms of a linguistic variable, and should be accurate enough in most situations. In particular, a p-quantity with support in $[a, b]$ models the vague proposition "roughly between a and b". Among the several possible forms such a fuzzy set can take, the most popular are probably the trapezoidal and bell-shaped.

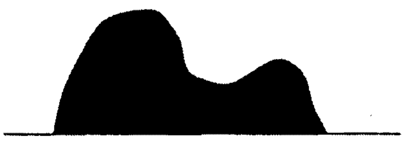
Trapezoidal

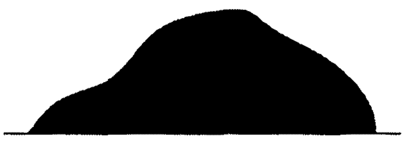
Correspond to fuzzy sets of the form:

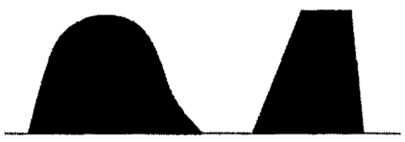
$$\mu_{[m_1,m_2]}^{[\alpha_1,\alpha_2]}(z) = \begin{cases} \frac{z-(m_1-\alpha_1)}{\alpha_1} & \text{if } z \in [m_1-\alpha_1, m_1] \\ \frac{(m_2+\alpha_2)-z}{\alpha_2} & \text{if } z \in [m_2, m_2+\alpha_2] \\ 1 & \text{if } z \in [m_1, m_2] \\ 0 & \text{otherwise} \end{cases} \tag{4.61}$$

where $m_2 \geq m_1$ and $\alpha_1, \alpha_2 > 0$ are offsets.

[5]This is a special case of a fuzzy interval of the LR-type [Zimmermann, 92], defined for convenience.


(a) A fuzzy (real) quantity.


(b) A fuzzy interval.


(c) Two fuzzy p-quantities.

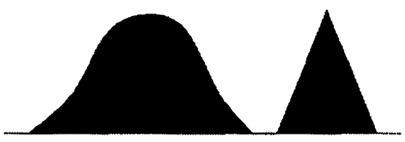

(d) Two fuzzy numbers.

Figure 4.4: Several specific forms for a continuous fuzzy set in $\mathbb{R}$.

Bell

Correspond to symmetric fuzzy sets of the form:

$$\mu_{[m,\alpha,\beta]}(z) = \frac{1}{1+(\frac{|z-m|}{\alpha})^{2\beta}} \tag{4.62}$$

Let $\mathbb{F}_q(X)$ be the (crisp) set of all the fuzzy p-quantities in X:

$$\mathbb{F}_q(X) = \{F \in \tilde{P}(X) \mid \ F \text{ is a fuzzy p-quantity}\}$$

Among the possible similarity measures between these elements, given two fuzzy p-quantities $\tilde{A}, \tilde{B} \in \mathbb{F}_q(\Gamma), \Gamma \in \mathbb{R}$, with support sets $\Gamma_{\tilde{A}}, \Gamma_{\tilde{B}} \in \Gamma$, respectively, we define their ratio as:

$$r(\tilde{A}, \tilde{B}) = \frac{\int_{\Gamma_{\tilde{A}} \cup \Gamma_{\tilde{B}}} \mu_{\tilde{A}\cap\tilde{B}}(u)du}{\int_{\Gamma_{\tilde{A}} \cup \Gamma_{\tilde{B}}} \mu_{\tilde{A}\cup\tilde{B}}(u)du} \tag{4.63}$$

which is the continuous version of one of the classical measures between fuzzy sets [Pappis and Karacapilidis, 93]. Being these measures general ones, they are not discriminative enough to capture all possible forms two fuzzy sets can take. The context in which they are applied here (fuzzy p-quantities) allows for an accurate assessment of similarity. Note that (4.63) can be regarded as a generalization of a measure for crisp sets representing subintervals of $\mathbb{R}$, in which a value either belongs or not to the interval. The step to fuzzy sets makes the integral operator necessary. Notice also that, although (4.60) could in principle be applied, its behaviour in some cases, namely, whenever $\#\{x \in X \mid \mu_{\tilde{A}\cap\tilde{B}}(x) = 1\} > 1$, makes it unsuitable as a similarity index (it does not fulfill minimality).

Proposition 4.30 *Given $\tilde{A}, \tilde{B} \in \mathbb{F}_q(\Gamma), \Gamma \subset \mathbb{R}$, the function $s(\tilde{A}, \tilde{B})$*

$$s : \mathbb{F}_q(\Gamma), \mathbb{F}_q(\Gamma) \to [0, 1]$$

as defined in (4.63), for $(\cap = min, \cup = max)$, is a similarity measure in $\mathbb{F}_q(\Gamma)$, with $s_{max} = 1$.

Proof: Symmetry, non-negativity and boundedness conditions (with $s_{max} = 1$) hold. There is also a clear semantics. Minimality is met because, as in crisp sets, the property $\tilde{A} \cap \tilde{B} = \tilde{A} \cup \tilde{B} \Leftrightarrow \tilde{A} = \tilde{B}$ holds (provided $\{\cap = min, \cup = max\}$ are used).

These variables are to be used whenever there is a knowledge that moves us to define linguistic terms instead of simple ordinals as, for example, for common situations of values as: "small", "increasing", "young", etc. If precise membership functions are known, these can be directly used.

Note that the so-constructed fuzzy quantities $\tilde{x}$ and $\tilde{X}$ are respectful with the respective orderings in $\Gamma \in \mathbb{R}$ and $\mathcal{O}$. It is ensured that the ordering $\preceq$ in $\mathcal{O}$ is respected, since this is a basic knowledge. Denoting $\tilde{X}$ the linguistic term associated with a given ordinal $x \in \mathcal{O}$, and $[\tilde{X}]$ its center of gravity —for example, $\frac{m_1+m_2}{2}$ in (4.61) and m in (4.62)— it holds that:

$$\forall x, y \in \mathcal{O} : \quad [\tilde{X}] \leq [\tilde{Y}] \Leftrightarrow x \preceq y$$

Analogous arguments can be done for $x \in \Gamma$, $\Gamma \in [\mathbb{R}]$ and the corresponding $\tilde{x} \in \mathbb{F}_n(\Gamma)$. Additionally, a complete coverage and a correct degree of overlap should be made sure, and the left and rightmost values shouldered. It is thus a design decision —involving the existence of a continuous substrate and enough domain knowledge— which variables are amenable to be treated as linguistic and which as ordinal. All these similarity measures are conveniently defined such that $s_{max} = 1$, so to allow a simpler and more general design of aggregation functions.

4.3.2 Definition of heterogeneous spaces

An heterogeneous space, denoted $\hat{\mathcal{H}}^n$, is defined as the Cartesian product of a number n of *source* sets, as follows. Consider a collection of extended sets $\hat{\mathcal{R}}_1, \ldots, \hat{\mathcal{R}}_{n_r}$, where $\hat{\mathcal{R}}_i = \mathbb{R}_i \cup \{\mathcal{X}\}, \mathbb{R}_i \in [\mathbb{R}]$, and $1 \leq i \leq n_r$. Consider also collections of extended sets $\hat{\mathcal{O}}_1, \ldots, \hat{\mathcal{O}}_{n_o}$, with $\hat{\mathcal{O}}_i = \mathcal{O}_i \cup \{\mathcal{X}\}, 1 \leq i \leq n_o$, where each $\mathcal{O}_i$ is a finite and linearly ordered set, and of extended sets $\hat{\mathcal{N}}_1, \ldots, \hat{\mathcal{N}}_{n_n}$, with $\hat{\mathcal{N}}_i = \mathcal{N}_i \cup \{\mathcal{X}\}, 1 \leq i \leq n_n$, where each $\mathcal{N}_i$ is a finite and unordered set [Valdés and García, 97].

Consider now the collection of n_f extended fuzzy sets of the form $\hat{\mathcal{F}}_1, \ldots, \hat{\mathcal{F}}_{n_f}$, where $\hat{\mathcal{F}}_i = \mathcal{F}_i \cup \{\mathcal{X}\}, \mathcal{F}_i \subset \mathbb{F}_q(\Gamma_i), \Gamma_i \in [\mathbb{R}]$, and $1 \leq i \leq n_f$. Notice that $\mathbb{F}_n(\Gamma_i) \subset \mathbb{F}_q(\Gamma_i)$.

The respective Cartesian products can now be constructed:

$$\begin{array}{rcl} \hat{\mathcal{R}}^{n_r} & = & \hat{\mathcal{R}}_1 \times \ldots \times \hat{\mathcal{R}}_{n_r} \\ \hat{\mathcal{O}}^{n_o} & = & \hat{\mathcal{O}}_1 \times \ldots \times \hat{\mathcal{O}}_{n_o} \\ \hat{\mathcal{N}}^{n_n} & = & \hat{\mathcal{N}}_1 \times \ldots \times \hat{\mathcal{N}}_{n_n} \\ \hat{\mathcal{F}}^{n_f} & = & \hat{\mathcal{F}}_1 \times \ldots \times \hat{\mathcal{F}}_{n_f} \end{array} \tag{4.64}$$

In all cases, the extension is given by the special symbol $\mathcal{X}$, which denotes the *unknown* element (missing information) for which only equality is defined and behaving as an *incomparable* element w.r.t. any ordering relation. In these conditions, the space formed by the Cartesian aggregation of the previous sets:

$$\hat{\mathcal{H}}^n \equiv \hat{\mathcal{R}}^{n_r} \times \hat{\mathcal{O}}^{n_o} \times \hat{\mathcal{N}}^{n_n} \times \hat{\mathcal{F}}^{n_f}$$

is such that $n = n_r + n_f + n_o + n_n$, with $\hat{\mathcal{R}}^0 = \hat{\mathcal{O}}^0 = \hat{\mathcal{N}}^0 = \hat{\mathcal{F}}^0 = \emptyset$ and $n > 0$. According to this definition, elements of $\hat{\mathcal{H}}^n$ are general tuples of n components, among which there may be real numbers, fuzzy sets (representing fuzzy numbers and p-quantities), ordinals, nominals and missing data. Other data types, e.g. sets, should be accommodated in an analogous way. We will call such a structure an *heterogeneous space.*

Proposition 4.31 *The Definitions and Propositions in (§4.2), referring to distance and similarity measures, apply also to an heterogeneous space $\hat{\mathcal{H}}^n$.*

Proof: in all these statements, there is no assumption about the structure of the space itself, provided basic distance and similarity functions are definable. The existence of aggregation operators is only conditioned to the fulfillment of their respective properties.

To build an heterogeneous distance $d \in D(\hat{\mathcal{H}}^n)$ two basic steps are needed:

1. Define the partial measures $d_i \in D(\hat{\mathcal{H}}^i), 1 \leq i \leq n$;
2. Define a distance d by an aggregation operator $\Theta(\vec{d};\vec{v})$ where $\vec{d} = \{d_1, \dots, d_n\}$.

However, with such kinds of heterogeneity, uncertainty or missing information, only in some cases it will be possible to define a distance measure in the pattern space. The difficulties in constructing metrics in heterogeneous spaces can in general be associated with:

- Presence of categorical variables (absence of an order);
- Presence of missing values (regardless of the data type);
- Uncertainty and imprecision in some variables.

Moreover, we have seen that in some occasions, although we *could* in principle define a metric (such as an Euclidean), it is preferable to devise a direct similarity measure incorporating domain theory. From Propositions (4.11) and (4.31), we then derive the following Corollary:

Corollary 4.1 *Let Θ_s be an aggregation operator as defined in Proposition (4.8). Given a group of partial similarities $\vec{s} = \{s_1, \dots, s_n\}$, where:*

$$s_i : \hat{\mathcal{H}}_i, \hat{\mathcal{H}}_i \to [0, s_{max}] \cup \{\mathcal{X}\}$$

Then, the measure:

$$s: \hat{\mathcal{H}}^n, \hat{\mathcal{H}}^n \rightarrow [0, s_{max}] \cup \{\mathcal{X}\}$$

defined as:

$$s(\vec{x}, \vec{y}) = \Theta_s(s_1(x_1, y_1), \ldots, s_n(x_n, y_n))$$

for $\vec{x}, \vec{y} \in \hat{\mathcal{H}}^n$, *is a similarity measure in* $\hat{\mathcal{H}}^n$, *where* $\hat{\mathcal{H}}^n = \hat{\mathcal{H}}_1 \times \cdots \times \hat{\mathcal{H}}_n$.

4.4 A framework for general neuron models

Definition 4.15 (H-neuron) *Given* $\vec{x} \in \hat{\mathcal{H}}^n$, *an heterogeneous neuron or H-neuron is a function of the form:*

$$F_i(\vec{x}) = \{h(\vec{x}, \vec{w}^i), \vec{w}^i \in \hat{\mathcal{H}}^n\} \tag{4.65}$$

with $h: \hat{\mathcal{H}}^n, \hat{\mathcal{H}}^n \rightarrow \hat{\mathcal{H}}$, *where the domains* $\hat{\mathcal{H}}^n$ *are equal heterogeneous spaces of n components, and the codomain* $\hat{\mathcal{H}}$ *is a single heterogeneous space.*

This definition accounts for neuron models -depicted in Fig. (4.5)- performing general mappings from an (heterogeneous) input space to an (heterogeneous) output space. Therefore, H-neurons are classified according to the nature of their codomain (which must not be necessarily restricted to a subset of the reals). In the present study, a model with a codomain given by $\hat{\mathcal{H}} = \mathbb{R}$ is called of the *real kind.*

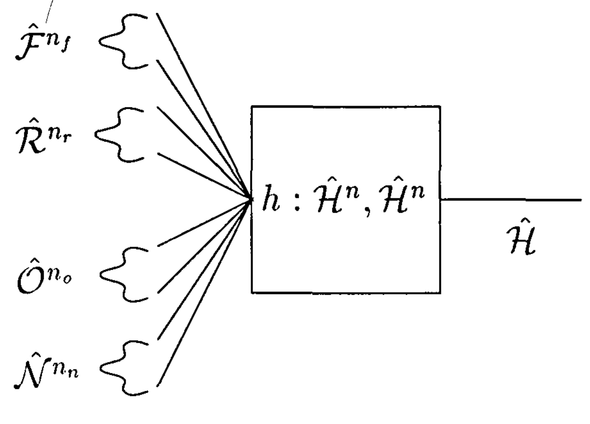

$\hat{\mathcal{F}}^{n_f}$
$\hat{\mathcal{R}}^{n_r}$
$\hat{\mathcal{O}}^{n_o}$
$\hat{\mathcal{N}}^{n_n}$
$h: \hat{\mathcal{H}}^n, \hat{\mathcal{H}}^n$
$\hat{\mathcal{H}}$

Figure 4.5: The H-neuron model.

The reason to consider in a first place neuron models of the real kind is twofold. On the one hand, there is their natural coupling with other classical neuron models (i.e. accepting only real-valued and complete inputs), thus leading to hybrid networks in a straightforward way. On the other hand, in this work the emphasis has been put on the definition of scalar similarity functions acting on real subintervals. None the less, the definition makes provision for neurons, either not working on similarity or computing similarity functions in other image sets $s: \hat{\mathcal{H}}^n, \hat{\mathcal{H}}^n \rightarrow \hat{\mathcal{H}}$ where in general the range of possible values of s is a subset $H[s] \subseteq \hat{\mathcal{H}}$.

Definition 4.16 (S-neuron) *A similarity-based neuron or S-neuron (of the real kind) is an H-neuron computing device for which $h(\cdot,\cdot)$ is a similarity, pseudo-similarity or point similarity in $\hat{\mathcal{H}}^n$ and $\hat{\mathcal{H}} = \Gamma_s \cup \{\mathcal{X}\}, \Gamma_s = [0, s_{max}] \in [\mathbb{R}]$.*

Accordingly, the S-neuron is sensitive to the degree of similarity between its inputs and its weights. The extent of this sensitivity is determined by the precise shape of the similarity function h, whose choice should account for the heterogeneous nature of neuron inputs and the presence of missing data. This leads to the notion of neuron and network *sensitivity*, which will be dealt in (§4.4.3).

When the model is constructed as the composition of two mappings, it is immediate to realize that the classical neuron models defined in (2.3) and (2.2) are included in the above definition, by setting $\hat{\mathcal{H}}^n = \mathbb{R}^n$, $n = n_r$ (and thus $n_f = n_o = n_n = 0$) and no missing data at all, and making use of Propositions (4.19) and (4.23). The four types of similarities described in (§4.2.4) are also included, since nothing is required about the precise form of h.

Within this framework, as described in (§4.2), all possible variations of *localized* RBF units of the kind described in (2.3) are included in similarity measures of type (A). It suffices to take real-valued and complete inputs, and a metric followed by a $\hat{s}$ function. In theory, there is no need for a RBF unit to have a localized response. Choices as $g(z) = z^2 lnz$ (thin-plate spline), $g(z) = \sqrt{z^2 + c^2}, c \in \mathbb{R}$ (multiquadric), or even simpler functions as $g(z) = z^3$ (cubic) are in the list of valid radial basis functions for data interpolation [Poggio and Girosi, 89], although they require a polynomial to be added to the general expression of the network (a linear term for the above functions). However, non-localized functions are scarcely used in RBF networks applications because –albeit they are still radially centered– their unbounded response departs from the clear conceptual view of setting local models that respond strongly to a small (input) region of attraction and gently (or rapidly) fade away outside this region. This view is in the basis of RBF networks and any conceivable distance-based localized model fits nicely in the similarity framework.

In any case, a network formed of type (A) S-neuron models, with heterogeneous inputs, and an output layer of P-neurons, where each partial distance d_i is tailored to its data type, $d_i \in D(\hat{\mathcal{H}}^i)$, can be seen as an *heterogeneous* RBF network, thus amenable to be trained like a standard RBF with its characteristic two-stage process (§2.1.7).

Factorizable RBF networks (F-RBFNN), an idea introduced in [Poggio and Girosi, 89], are specifically included as particular cases of type (B) similarity measures. These networks are conceptually seen as three layered ones. With respect to a standard RBF network, an extra layer is placed before the hidden layer. In this extra layer, there is a neuron per each input dimension for every neuron in the hidden layer. These neurons compute a one-dimensional distance between their (scalar) input and weight, followed by a localized function. Their outputs are then seen as "factors" and grouped by a hidden neuron. This grouping is a product operator. This scheme is intuitively appealing for Gaussian units working on Euclidean distance, since in this case the Gaussian can be factorized in individual terms as follows:

$$s(\vec{x},\vec{w}^{\,i}) = e^{-\|\vec{x}-\vec{w}^{\,i}\|^2} = e^{-\sum_{i=1}^{n}|x_j-w_ij|^2} = \prod_{j=1}^{n} e^{-|x_j-w_{ij}|^2} \tag{4.66}$$

As in our framework the partial similarity measures s_j can be seen as representing a single unidimensional unit specifically detecting similarity in one input dimension, equation (4.66) is cast as:

$$s(\vec{x},\vec{w}^{\,i}) = \Theta_s(\vec{s}) = \prod_{j=1}^{n} e^{-|x_j-w_{ij}|^2} \tag{4.67}$$

with $\vec{s} = \{s_1,\ldots,s_n\}$, $s_j = \hat{s}(d(x_j,w_{ij}))$, $\hat{s}(z) = e^{-z^2}$ and $d(x_j,w_{ij}) = |x_j - w_{ij}|$ (thus $s_{max} = 1$ for all i) and Θ_s as in Proposition (4.12, 2.) with $m = v_i = 1$.

In the line of RBF units, other expressions have been proposed in search for new models. For example, choosing the same $\Theta_s(\vec{s}) = \prod_{j=1}^{n} s_j$, and $s_j = \hat{s}(d(x_j,w_{ij}))$, with $d(x_j,w_{ij}) = \frac{|x_j-w_{ij}|}{\sigma_j}$ and $\hat{s}(z) = \frac{1}{1+z^2}$, the resulting expression is the neuron model:

$$s(\vec{x},\vec{w}^{\,i}) = \prod_{j=1}^{n} \frac{1}{1+\left(\frac{|x_j-w_{ij}|}{\sigma_j}\right)^2} \tag{4.68}$$

This is a type (B) measure, proposed in [Duch and Jankowski, 95]. By replacing the product by a summation in (4.66), and adding a weighting scheme v_j, that is, by using $\Theta_s(\vec{s};\vec{v}) = \sum_{i=1}^{n} v_j s_j$, the resulting neuron model:

$$s(\vec{x},\vec{w}^{\,i}) = \sum_{j=1}^{n} v_j e^{-\frac{(x_j-w_{ij})^2}{\sigma_j^2}} \tag{4.69}$$

is known as the *Gaussian bar* [Park and Sandberg, 91]. In general, single units need not be Gaussian nor their combination needs to be done in this way.

Measures of type (C) include scalar product as a special case and, by simple extension, the sigmoidal unit defined in (2.2). As for measures of type (D), similarities from dissimilarities, they do not correspond or include any neuron model as used in ANN. The situation makes sense whenever it is sensible to define a distance, but the proposed index does not fulfill triangular inequality. If the rest of conditions hold, the transformation functions $\hat{s}$ and $\check{s}$ are valid and could be applied.

The flexibility of this framework carries with it several advantages:

1. Any valid set of unidimensional similarity measures –not necessarily distance-based– possibly designed in accordance with the nature of their respective inputs, can be grouped by means of a similarity aggregation operator.

2. There is no strict order between linear/non-linear functions, as long as the overall (similarity) measure is non-linear.

3. Domain knowledge can be put into work in the design of the model.

4. The functions $\check{s}$ and $\hat{s}$ and the aggregation operators play analogous roles regardless of the specific similarity type.

In addition, Definition (4.15) explicitly states that the weight w_i associated to a given input line x_i is an element of the same space than x_i. This in turn implies that, whenever $\hat{\mathcal{H}}^n = \mathbb{R}^n$ and, more precisely, $\hat{\mathcal{H}}^i = \Gamma_i \in [\mathbb{R}]$, then *both* $x_i, w_i \in \Gamma_i$, thereby bounding the weight to be within the same limits than the input it is weighting, giving an additional basement to the assumption (4.35). Notice that, as a further consequence of the definition, the presence of missing values is allowed in the weights. This can be beneficial to a learning algorithm, since it adds for more flexibility by enabling the units to ignore some of their inputs.

Among the many different architectures, we consider in the first place hybrid feed-forward ones, consisting of a hidden layer of S-neurons and an output layer of P-neurons -illustrated in Fig. (4.6)- since the outputs of the former (which are real-valued) can be directly used as inputs for the latter. Although, in essence, these architectures are not fully compliant with the ideas exposed so far[6], a simple parsimony principle leads to begin by choosing the simplest architectures, among those with enough theoretical computing capacity. The output neurons can be regarded as performing a linear combination of the similarities reported by the hidden layer. Applying a further non-linear function $\sigma : \mathbb{R} \to \mathbb{R}$ to these neurons leads to the P-neuron model for the output units.

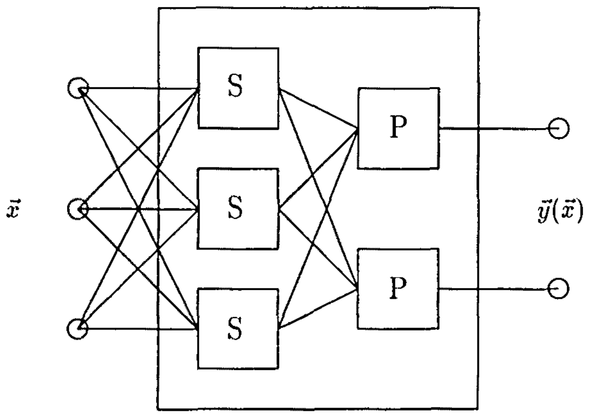


Figure 4.6: Intuitive composition of S-neurons with P-neurons.

More elaborated architectures, where outputs are also computed by S-neurons, would constitute the immediate step. From a conceptual point of view, these units would be computing a similarity measure between vectors of similarities, and yielding a scalar quantity in account of the input vector fed to the network. A simple choice for the aggregation of such

[6]Note that, though the inputs of the P-neurons are bounded, their weights are not.

similarity vectors would be given by a weighted average, a type of additive measure (4.21), of the form:

$$\Theta_s(\vec{s},\vec{c}) = \frac{\sum_{i=1}^{h_1} c_i s_i}{\sum_{i=1}^{h_1} c_i} \tag{4.70}$$

where $\vec{s}$ is the vector of similarities as computed by the hidden units, and c_i are the hidden-to-output weights. The expression (4.70) is obtained taking $f(s_i) = s_i$ in (4.21) and $\vec{v} = \frac{\vec{c}}{\sum_{i=1}^{h_1} c_i}$ to ensure normalization. Networks with more than one hidden layer of S-neurons are also a clear choice and subject of study, albeit it is conjectured that one hidden layer should suffice for most applications, especially if the network is acting as an (heterogeneous) RBF network.

Definition 4.17 (HNN) *A feed-forward Heterogeneous Neural Network or HNN is a m-FFNN[c] for which $c \geq 1$, where the F^l are S-neurons, $1 \leq l \leq c$, and the output units are either S-neurons or P-neurons.*

For networks with a single output unit ($m = 1$), the general aspect of a feed-forward HNN is depicted in Fig. (4.7) for $c = 1$. If the output unit is a P-neuron, the overall computation $y(\vec{x})$ is given by Proposition (2.1), where the computation performed by each hidden S-neuron is simply $F_i(\vec{x}) = s(\vec{x}, \vec{w}^i)$.

In case it is a S-neuron computing a similarity measure s_0 –like, for example, that in (4.70)– then:

$$y(\vec{x}) = s_0(\vec{c}, \vec{F}(\vec{x}))$$

where $\vec{c} = (c_1, \ldots, c_{h_1})$ and $\vec{F}(\vec{x}) = (F_1(\vec{x}),, \ldots, F_{h_1}(\vec{x}))$.

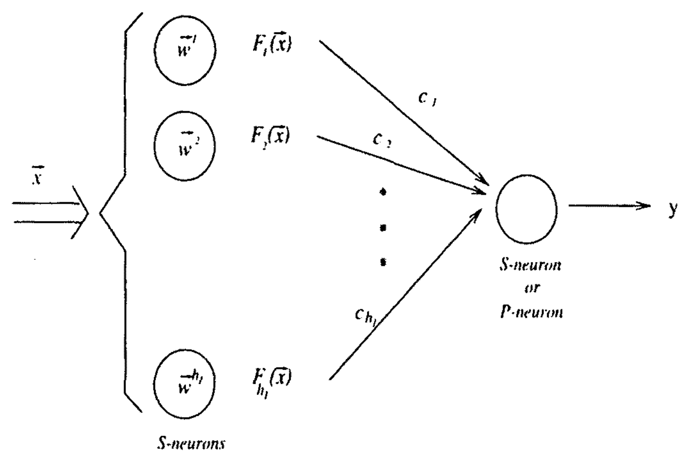


Figure 4.7: A one-hidden-layer HNN with h_1 hidden S-neurons and a single output unit, either a S-neuron or a P-neuron, where $F_j(\vec{x}) = s(\vec{x}, \vec{w}^j), 1 \leq j \leq h_1$.

4.4.1 An example of a S-neuron

A basic but very useful heterogeneous measure can be devised by making use of a Gower-like similarity index, well-known in the literature on multivariate data analysis [Gower, 71]. This coefficient has its values in the real interval $[0,1]$ and for any two objects $\vec{x}_i$, $\vec{x}_j$ given by tuples of cardinality n, is given by the expression:

$$s_G(\vec{x}_i,\vec{x}_j) = \frac{\sum_{k=1}^{n} g_{ijk}\,\delta_{ijk}}{\sum_{k=1}^{n} \delta_{ijk}} \tag{4.71}$$

where:

- g_{ijk} is a similarity *score* for objects $\vec{x}_i$, $\vec{x}_j$ according to their value for variable k. These scores are in the interval $[0,1]$ and are computed according to different schemes for numeric and qualitative variables.

- δ_{ijk} is a binary function expressing whether the two objects are comparable or not according to their values w.r.t. variable k, as follows:

$$\delta_{ijk} = \begin{cases} 1 & \text{if } x_{ik} \neq \mathcal{X} \wedge x_{jk} \neq \mathcal{X} \\ 0 & \text{otherwise} \end{cases} \tag{4.72}$$

In particular, in Gower's original work, the partial similarity measures used are those defined in (4.45) for nominal and (4.52) for continuous variables. For ordinal, set, fuzzy and linguistic variables, their scores can be computed by using the measures in (4.46), (4.53), (4.60) and (4.63), respectively. Setting $g_{ijk} = \mathcal{X}$ if $\delta_{ijk} = 0$, the expression in (4.71) can be cast as an additive similarity aggregation operator, obtained taking $f(s_i) = s_i$ in (4.21), $n' = \#k : 1 \leq i \leq n : \delta_{ijk} = 1$ (the number of actually performed comparisons), and a weighting vector $\vec{v} = (\frac{1}{n'}, \ldots, \frac{1}{n'})$.

This treatment of missing values adopted respects the philosophy described in (§4.2.3). To see this, let $\vec{x}|i$ denote the slice of a vector $\vec{x}$ minus the i-th component. Then, for two vectors $\vec{x}_i, \vec{x}_j$ for which $\delta_{ijk} = 0$, it holds $s(\vec{x}_i, \vec{x}_j) = s(\vec{x}_i|k, \vec{x}_j|k)$. For example, for these two numeric vectors:

$$\vec{x}_1 = \{0.3, \mathcal{X}, \ldots, 8.1\}$$

$$\vec{x}_2 = \{0.5, 2.3, \ldots, 3.1, \}$$

This aggregation policy yields $s(\vec{x}_1, \vec{x}_2) = s(\vec{x}_1|2, \vec{x}_2|2)$. The possibility that none of the partial similarities can be performed has to be handled with special care. Formally, given $\vec{x}_i, \vec{x}_j$, if it holds that:

$$\forall k : 1 \leq k \leq n : \delta_{ijk} = 0$$

the result of (4.71) is $\frac{0}{0}$. As in Gower's original work, the measure would remain undefined. This is unfeasible from the point of view of a neural network computation. There are at least two immediate solutions:

1. To set $s_G(\vec{x}_i, \vec{x}_j) = 0$, denoting that they are incomparable (corresponding to a *pessimistic* interpretation of similarity).
2. To set $s_G(\vec{x}_i, \vec{x}_j) = 1$, denoting that they are indistinguishable (corresponding to an *optimistic* interpretation of similarity). This option carries with it the redefinition of when two objects are equal.

In our framework, however, there is a third possibility, namely, to set $s_G(\vec{x}_i, \vec{x}_j) = \mathcal{X}$, expressing that a specific unit's computation of similarity is lacking. This missing value can be processed by other heterogeneous neurons in the next layer.

The fact that a network weight can also be missing is interpreted by the neuron in the same way than for the case of a missing input. In consequence, the input is ignored (as if the connection were not there), which is consistent with the fact that a similarity measure must be symmetric. Notice that, if at the end of a training process, a given weight is missing, then the connection can be removed, since it is to be ignored permanently. This can be seen as a form of network pruning.

An added advantage stems from the fact that (4.71) is not affected by any linear normalization -see Appendix (D)- thus rendering this usual preprocessing step unnecessary. Furthermore, being a linear aggregation operator, a non-linear component can be added to in the form of a similarity keeping function $\check{s}$ to form an S-neuron of the type described in Definition (4.16), as follows:

$$\check{s}_G(\vec{x}, \vec{w}^i) = \check{s}(s_G(\vec{x}, \vec{w}^i)) \qquad \vec{x}, \vec{w}^i \in \hat{\mathcal{H}}^n \tag{4.73}$$

The widespread logistic functions can be used to work as a non-linear $\check{s}$ by adapting it so as to map the real interval $[0,1]$ on $(0,1)$, by taking $\check{s} = g_{11,0.5}$ in (3.9). However, this function is not cheap to compute. For this reason, other parameterized families of sigmoidal activation functions can be especially designed to operate within the $[0,1]$ interval [Valdés and García, 97], such as $\check{s} = g(\cdot, k)$, where:

$$g(x,k) = \begin{cases} \frac{-k}{(x-0.5)-a(k)} - a(k) & \text{if } x \leq 0.5 \\ \frac{-k}{(x-0.5)+a(k)} + a(k) + 1 & \text{if } x \geq 0.5 \end{cases}$$

$$a(k) = \frac{-0.5 + \sqrt{0.5^2 + 4k}}{2} \tag{4.74}$$

being $a(k)$ an auxiliary function with $k > 0$ a real-valued parameter controlling the global curvature, usually set in the experiments to $k = 0.1$. This family of functions meets the conditions in Definition (4.9) and is displayed in Fig. (4.8). They correspond to isomorphisms (monotonic bijections) in $[0,1]$, fulfilling:

$$\forall k \in \mathbb{R}^+,\ g(0,k) = 0; \qquad g(1,k) = 1; \qquad \lim_{k \to \infty} g(x,k) = x; \qquad \text{and } g(x,0) = H_{0.5}(x)$$

being H the Heaviside function (3.1). The expression for $a(k)$ is the solution of the equation: $a(k)^2 + \frac{a(k)}{2} - k = 0$, which results of imposing the above first two equalities.

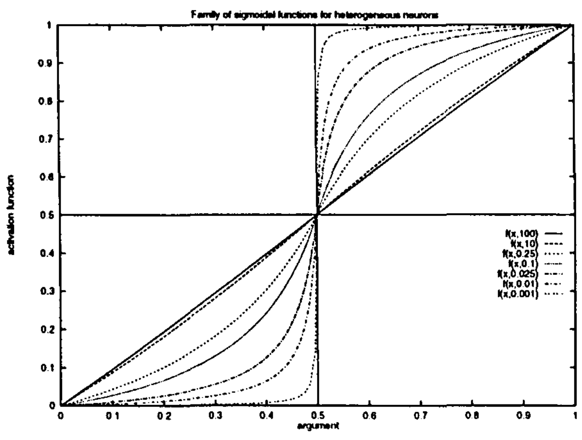


Figure 4.8: The family of sigmoidal functions $g(x, k)$, for different values of k.

Especially noteworthy is the following situation. If the heterogeneous space $\hat{\mathcal{H}}^n$ includes only elements of $\mathbb{R}, \mathcal{O}$ or $\mathcal{N}$, for which similarities -as shown in (4.52), (4.46) and (4.45)- can be distance-based, *and* there is no missing information, then the overall measure in (4.71) can be viewed as coming from a distance $d_G \in D(\mathcal{H}^n)$, as follows:

$$\check{s}_G(\vec{x}, \vec{w}^i) = \check{s}(1 - d_G(\vec{x}, \vec{w}^i)) \qquad \vec{x}, \vec{w}^i \in \hat{\mathcal{H}}^n \tag{4.75}$$

The above expression fits the RBF unit scheme since it is a type (A) similarity measure of the form (4.24) with $\hat{s}(z) = 1 - z$ applied to an aggregated distance d_G, built as in (4.3) with $n' = n$, $q = 1$ and $v_i = 1$. Additionally, the expression can be further simplified by making use of Proposition (4.15) and collapsing the activation function into a single similarity transforming one:

$$\hat{s}'(z) = \check{s}(1 - z)$$

where either $\check{s} = g_{11,0.5}(\cdot)$, $\check{s} = g(\cdot, k)$ or other suitable function. In any case, the neuron computation corresponds to a localized distance-based —the response of which is shown in Fig. (4.9)— thereby making (4.73) an instance of a model acting as an extended RBF unit (because it accepts ordinal data). In general, in presence of fuzziness, incompleteness and other kinds of data peculiarities, in the components of the heterogeneous input space, the function d_G in (4.75) cannot be guaranteed to be a distance in $D(\mathcal{H}^n)$ and the RBF philosophy is departed. In any case, it is behaving as a similarity-driven model.

Additionally, it is noteworthy to observe how the sigmoidal-like functions are by no means tied to the MLPNN (or, in general, to be used as $\check{s}$-functions). They can most conveniently be used as an $\hat{s}$-function by inverting its behaviour. For instance, by setting $a = 2$ in the Example $\hat{s}_4$ in page (84), the function $\sigma_0(z) = 2(1 - \sigma(z))$ is a similarity whenever z is a distance, with a response much like that in Fig. (4.9). This transformation is well-known although scarcely used in the ANN context. In case $z \in [0, 1]$, the relation is more compactly

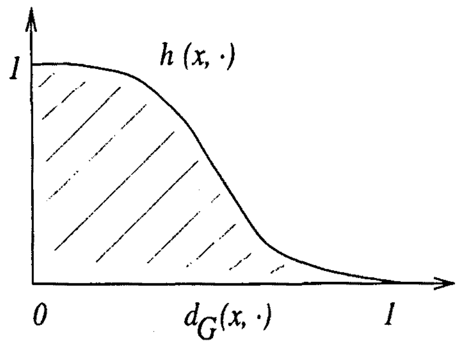


Figure 4.9: General response of the example S-neurons (for fixed weights).

written as $\sigma_0(z) = 2\sigma(-z)$. Notice that in all these situations only the upper half of the functions is being used.

The expression in (4.71) has many advantages: is a simple one, fulfills the desired properties, including the policy for missing values, is computationally cheap and is also a generic measure, in the sense that it does not take assumptions about the partial similarities.

4.4.2 Classes of $\hat{s}$ functions

In this section, several classes of functions that can act as similarity transformers are presented, obtained by generalizing and extending the functions listed in page (84). Some of their instances are well known and have been proposed in the context of RBF networks as theoretically valid radial basis functions [Poggio and Girosi, 89]. In the proposed framework, the requirement for inclusion in the list is the fulfillment of Definition (4.10). Without loss of generality, all the functions are set to yield $s_{max} = 1$. The following represent flexible classes of $\hat{s}$ functions:

$$\hat{s}_0(z) = \left(1 - (az)^d\right)^\alpha, 0 < d \leq 1, \alpha \geq 1, a > 0, z \in [0, d_{max}] \tag{4.76}$$

$$\hat{s}_1(z) = \frac{1}{1 + (az)^\alpha}, \alpha > 0, a > 0, z \in [0, \infty) \tag{4.77}$$

$$\hat{s}_2(z) = \frac{1}{1 + e^{(az)^\alpha}}, \alpha > 0, a > 0, z \in [0, \infty) \tag{4.78}$$

$$\hat{s}_3(z) = e^{-(az)^\alpha}, \alpha > 0, a > 0, z \in [0, \infty) \tag{4.79}$$

$$\hat{s}_4(z) = 1 - g_{11,0.5}(cz), c \geq 1, z \in [0, 1] \tag{4.80}$$

where g in (4.80) is the logistic (3.9). The variety of such classes of functions is enhanced by the introduction of parameters to control their precise shape within a specific analytic expression. In practical terms, this variety translates in to how *sensitive* a neuron endowed with a given function will be w.r.t. its argument. Since this argument is a distance, the sensitivity can be thought of as a means to control how *far* apart (in heterogeneous distance) two entities (in this case, an input and its weight) should be to be considered as, for example, "half" similar (recall the similarity is always expressing a degree).

The first family of functions is designed to explicitly take this controlled sensitivity into account. The basic distance to similarity transformation $\hat{s}_0(z) = 1 - z$ being linear, this generalization achieves also the purpose of making it non-linear in a controlled way. As desired, $\hat{s}_0(0) = 1$, and s_{max}, d_{max} are related. For $s_{max} = 1$, $d_{max} = \frac{1}{a}$, and thus $\hat{s}_0(d_{max}) = 0$. Normally, a simple choice $a = 1$ can be taken, leading to $s_{max} = d_{max} = 1$. Note that this does not imply any specific value for d or α.

Let us show how this is a valid generalization of the standard transformation $t(z) = 1 - z$, assuming z is a normalized distance in $[0,1]$.

Proposition 4.32 *Given z a normalized distance, $z \in [0,1]$, the family of functions*

$$\hat{s}_0(z) = (1 - z^d)^\alpha, 0 < d \leq 1, \alpha \geq 1$$

is a similarity in $[0,1]$.

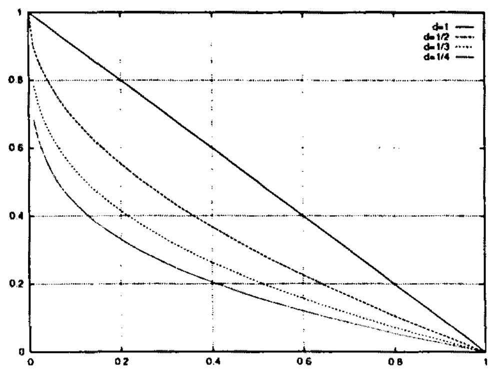
Proof. First, since z is a distance, z^d, for $0 < d \leq 1$, so is, by Lemma (A.4) in the Appendix. Second, $\hat{s}_0(z) = 1 - z$ is a valid similarity transforming function, one of those in the list at page (84), for $d_{max} = 1$ -Proposition (4.16). And third, $\check{s}(z) = z^\alpha, \alpha \geq 1$ is clearly a valid similarity keeping function.

For low values of d, α, a logistic non-linearity $g_0 = g_{11,0.5}$ can be further applied, as a $\check{s}$-function, like in (§4.4.1). In (4.80), the same function is used in a parameterized way, this time as a $\hat{s}$ function. Note that, by making use of the equality $1 - g_{\beta,\theta}(cz) = g_{\beta,\theta}(-cz)$, both (4.78) and (4.80) are obtained out of the general logistic $g_{\beta,\theta}(z)$ (3.9), the former as $g_{\beta,0}(-z)$ and the latter as $g_{11,0.5}(-cz)$. Though the general appearance of (4.78) and (4.80) is obviously much like that of (4.79), their conception as a generalized and inverted logistic can be conveniently instantiated into more specific functions, for which the centered and symmetric behaviour over a diagonal axis is no longer needed. Instead, its utility stems from its smooth and monotonically decreasing response, controlled by parameter c.

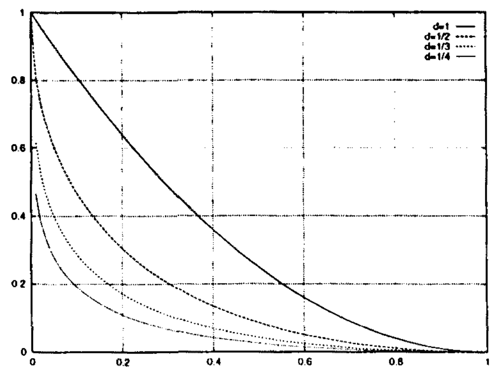
The response of functions (4.76) to (4.80) is displayed in Figs. (4.10 to 4.14), where their diverse behaviour is plainly observed, reacting quite differently to distance values within the same input range. Therefore, it is of interest to establish a means to compare their relative response and a mechanism to determine which of them are more suited to be used as similarity transformers in practice.

4.4.3 Sensitivity relations

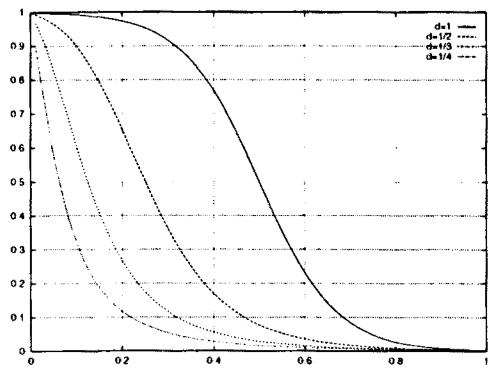
The idea behind the sensitivity of a $\hat{s}$ function can be informally stated as follows: given a similarity value $s_0 \in [0, s_{max}]$, what is the required distance value d_0 to yield s_0? Since $s_0 = \hat{s}(d_0)$, we have in general $d_0 = \hat{s}^{-1}(s_0) \in [0, \infty)$.



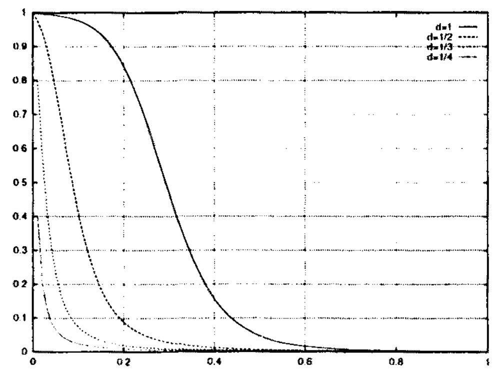
(a) Varying d for $\alpha = 1$



(b) Varying d for $\alpha = 2$

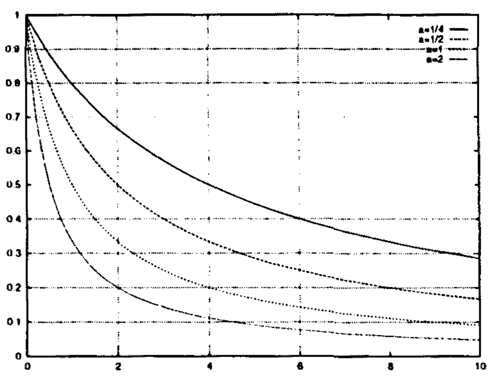


(c) Same as (a) after applying $g_0 = g_{11,0.5}$

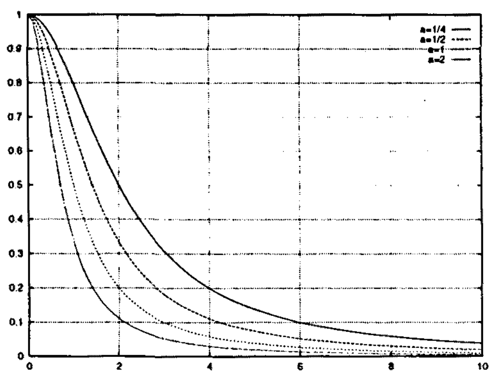


(d) Same as (b) after applying $g_0 = g_{11,0.5}$

Figure 4.10: Response of the class $\hat{s}_0(z)$. Notice the effect of g_0.

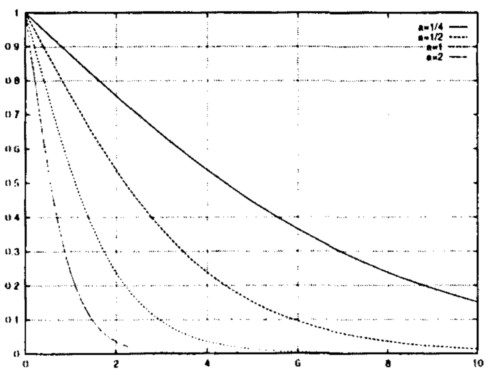


(a) Varying a for $\alpha = 1$

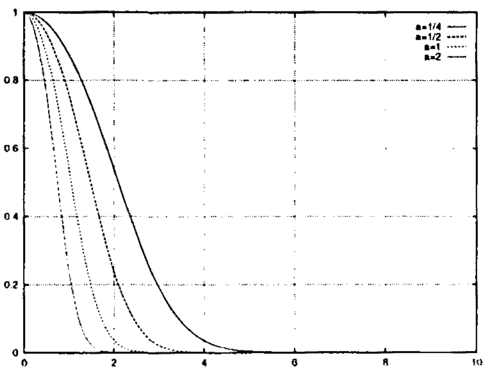


(b) Varying a for $\alpha = 2$

Figure 4.11: Response of the class $\hat{s}_1(z)$.

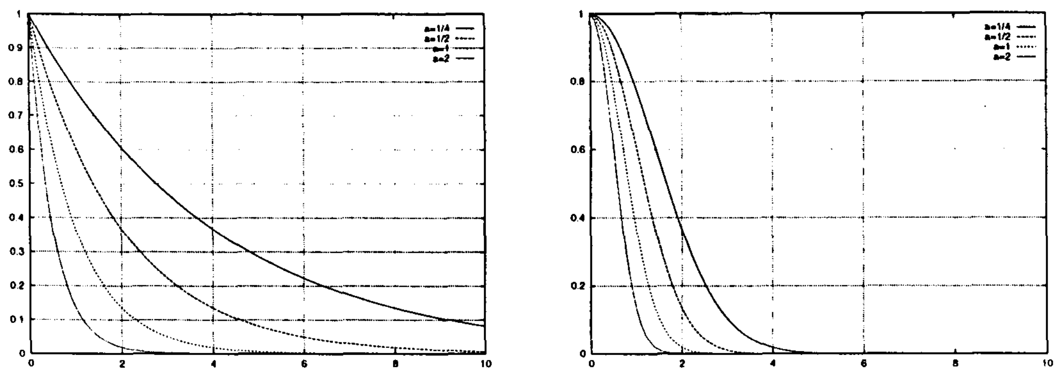


(a) Varying a for $\alpha = 1$



(b) Varying a for $\alpha = 2$

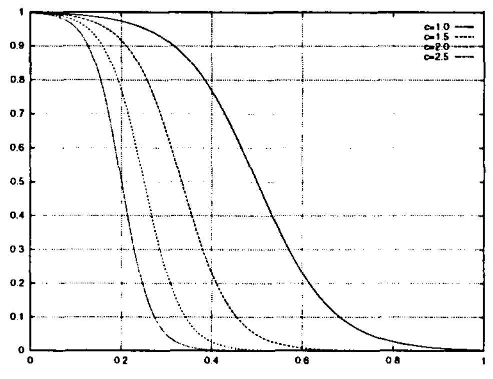
Figure 4.12: Response of the class $\hat{s}_2(z)$.



(a) Varying a for $\alpha = 1$

(b) Varying a for $\alpha = 2$

Figure 4.13: Response of the class $\hat{s}_3(z)$.



(a) Plots of $\hat{s}_4(z)$ for varying c.

Figure 4.14: Response of the class $\hat{s}_4(z)$.

Assuming $\hat{s} : [0, d_{max}] \to [0, s_{max}]$, its inverse $\hat{s}^{-1}$ is guaranteed to exist, and to be strictly increasing and continuous in $[0, s_{max}]$, by the properties of $\hat{s}$. The smaller d_0, the less sensible $\hat{s}$ is, since it takes a small distance value to reach a given similarity. Accumulating all this values over all $s_0 \in [0, s_{max}]$, and applying a proper normalization, we have a measure $\odot$ of sensitivity for a given $\hat{s}$ function.

The accumulation is nothing more than the integral:

$$\odot\hat{s} = \int_{[0,s_{max}]} \hat{s}^{-1}(z)dz$$

As long as $\hat{s}^{-1}(z) \in [0, d_{max}]$ –which implies that $\hat{s}(d_{max}) = 0$, by Proposition (4.16)– the above integral is guaranteed to exist (that is, to be finite) and $\odot\hat{s} \in [0, s_{max}d_{max}]$; therefore, the sensitivity can be normalized. In addition, by making use of the following equality:

$$\int_{[0,s_{max}]} \hat{s}^{-1}(z)dz = \int_{[0,d_{max}]} \hat{s}(z)dz \tag{4.81}$$

the inverse can be avoided. Denoting $\hat{S}$ the set of all $\hat{s}$ functions satisfying Definition (4.10), we end by defining the *sensitivity* of a $\hat{s}$ function as a function: $\odot : \hat{S} \to [0, 1]$ as:

$$\odot\hat{s} = \frac{\int_{[0,d_{max}]} \hat{s}(z)dz}{s_{max}d_{max}} \tag{4.82}$$

This measure gives a value closer to 1 the more sensible a function is; in other words, the easier is for a distance to elicit a given similarity value, accumulated over all the possible distance values. It is non-negative since the integrated functions are; being all the $\hat{s} \in \hat{S}$ monotonic and continuous, they are Riemann-integrable in $[0, d_{max}]$ [Jarauta, 93]. Additionally, (4.82) induces an ordering relation $\sqsubseteq$ on $\hat{S}$, defined as:

$$f_1 \sqsubseteq f_2 \Longleftrightarrow \odot f_1 \leq \odot f_2, \qquad f_1, f_2 \in \hat{S} \tag{4.83}$$

expressing that f_1 is *less* sensible than f_2. It is straightforward to see that $\sqsubseteq$ is a preorder (a reflexive and transitive relation) and hence $(\hat{S}, \sqsubseteq)$ is a *preordered* set. The relation is neither symmetric (obviously) nor antisymmetric. This is illustrated in Fig. (4.15), in which it is assumed that $\odot f_1 = \odot f_2$.

Mathematically, the comparison between different $\hat{s}$ makes sense because, in general:

$$\forall \hat{s} \in \hat{S}[a, b], \qquad \exists \xi(\hat{s}) \mid \int_{[a,b]} \hat{s}(z)dz = \xi(\hat{s})(b - a) \tag{4.84}$$

In our case, $[a, b] = [0, d_{max}]$ and is constant for all the functions so that, being (4.81) in $[0, s_{max}d_{max}]$, the number:

$$\xi(\hat{s}) = \frac{1}{d_{max}} \int_{[0,d_{max}]} \hat{s}(z)dz$$

represents the *fraction* (w.r.t. s_{max}) of area filled by $\hat{s}$ in $[0, d_{max}]$, depicted in Fig. (4.16). Dividing by s_{max}, we get the normalized measure (4.82).

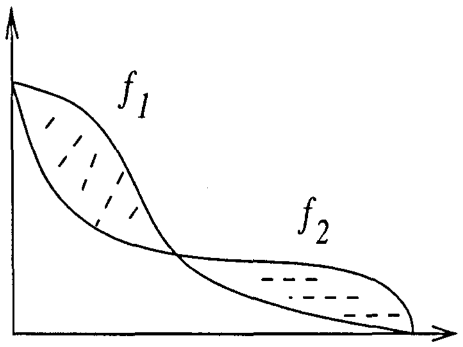


Figure 4.15: An example of non-antisymmetry. Assuming both marked areas are equal, it follows that $f_1 \sqsubseteq f_2 \wedge f_2 \sqsubseteq f_1$ but $f_1 \neq f_2$.

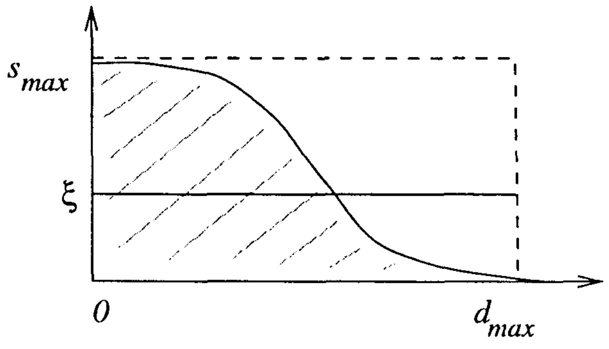


Figure 4.16: Sensitivity as a similarity ratio.

4.4.4 Sensitivity in practice

To clarify the relationship of different instances of the proposed families of $\hat{s}$ functions and to assess their usefulness as parts of recognition devices, a numerical experiment is performed with a twofold purpose. On the one hand, to check whether sensitivity is an important quantity and to what degree it exerts an influence on neuron performance. On the other, to ascertain what is the best range of values for the sensitivity. Too sensitive a neuron would make it useless as a recognizer because of the loss in discrimination ability, while too unsensitive would translate in a very specialized neuron model, one yielding high outputs only for a specific pattern (its weight vector) and vigorously decaying moving away from it. This has to do in general with the *degree of overlap* between neurons, a topic already found in RBF networks [Orr, 96]. Furthermore, Notice that the application of a $\hat{s}$ function is not limited to the case in which the neuron as a whole is computing a distance –as in type (A) similarities–, but can most conveniently be applied to individual neuron input lines –specifically to those that are distance-based– as in type (B) similarities. The general form of the function computed by the neuron allows the aggregation of partial similarities with no assumption to their distance-based character. Therefore, in those situations where the similarity measure is not completely of either type (A) or (B) the overall sensitivity of a neuron has to be assessed in a different way.

To this end, we define the β-sensitivity of a S-neuron i with weight vector $\vec{w}^i$, computing a similarity measure $s : K, K \to [0,1]$ over a support set $K \subset \hat{\mathcal{H}}^n$ as:

$$\beta_i(K,s) = E\{s(\vec{x}, \vec{w}^i)\}_{\vec{x} \in K}, \qquad \vec{w}^i \in K \tag{4.85}$$

where $E\{\cdot\}$ denotes the statistical expectation operator. One way to evaluate the sensitivity of s is by assessing its initial response in K. This can be achieved by computing the β-sensitivity as:

$$E\{s(\vec{x}, E\{\vec{w}_0^i\})\} \tag{4.86}$$

where $\vec{w}_0^i$ stands for the initial weight vector of neuron i in a training process. Expanding (4.85) we get, for $(\vec{x}, \vec{w}_0^i) \in K \times K \subset \hat{\mathcal{H}}^{2n}$:

$$\beta_i(K,s) = \int_{K \times K} s(\vec{x}, E\{\vec{w}_0^i\}) p(\vec{x}, \vec{w}_0^i)\, d\vec{x}\, d\vec{w}_0^i \tag{4.87}$$

The joint distribution can be decomposed in two, for the two variables are independent:

$$\beta_i(K,s) = \int_{K \times K} s(\vec{x}, E\{\vec{w}_0^i\}) p_x(\vec{x}) p_w(\vec{w}_0^i)\, d\vec{x}\, d\vec{w}_0^i \tag{4.88}$$

Let us denote by $D[X, p, \vartheta](\vec{x})$ a set of ϑ samples of $\vec{x}$ drawn from a set X with probability density function (pdf) $p(\vec{x})$. The pdf p_w for the initial weights is known albeit to be determined; we can generally assume it to be uniform though it is not necessary. In any case, we have a (finite) set $W_0^i = D[K, p_w, \vartheta_w](\vec{w}_0^i)$ of available samples at the outset of the training process. The pdf p_x from which the input patterns are drawn is not known. Instead, in general all we have is a set of samples $K_0 = D[K, p_x, \vartheta_x](\vec{x})$. Therefore, the integral (4.88) has to be estimated by evaluating it on the (finite) set $K_0 \times W_0^i$ so that:

$$\tilde{\beta}_i(K,s) = \frac{1}{\vartheta_x \vartheta_w} \sum_{\vec{x} \in K_0} \sum_{\vec{w}_0^i \in W_0^i} s(\vec{x}, \vec{w}_0^i) \tag{4.89}$$

yields an estimate of the true $\beta_i(K,s)$ in (4.85), where p_w is taken to be uniform. In practice, the measure (4.89) is likely to be more accurate the bigger the two sets of samples. As for W_0^i, we have in principle an unlimited (in the sense that it can be as big as needed) set of initial weights. With regard to K_0, having a training set as large and, most important, representative of the underlying distribution as possible is a key condition for acceptable performance on novel data. In addition, considering a network with a layer of h_1 hidden neurons allows for a supplementary source of independent estimates (at least initially), which can in turn be averaged over. The final quantity, denoted *mean initial similarity* is computed as:

$$\tilde{\beta}(K,s) = < \tilde{\beta}_i(K,s) >_i, \;\; 1 \le i \le h_1 \tag{4.90}$$

The value yielded by (4.90) gives an idea of how initially sensitive the network is to a precise data set. The question arises: what is the most convenient value for it? There surely is no value generally best, although, in practice, lower and upper bounds are likely to exist. It also depends on the way partial measures (be distances or similarities) are aggregated, and on the way these are computed. Therefore, the answer traces back to the sensitivity of the $\hat{s}$ functions as defined in (4.82). Since s_{max} has been set to 1, we could expect $\tilde{\beta}(K, s)$ to be about 0.5, signaling a middle initial average similarity. We hypothesize that this will certainly not be the case in general, for reasons that will soon be introduced. It will indeed be the case for the standard P-neurons as defined in Definition (2.2) and working on point similarities -see Proposition (4.23). As a matter of fact, it is general advice to set the initial weights of these neurons so to elicit an activation about one half[7] so that the learning process moves the weights towards saturation, thus committing the neuron to the extreme bounds of the non-linearity [Haykin, 94].

However, for general neuron models working on similarity, especially those using a combination of *different* partial measures, either distance-based or direct, for each input variable, the answer is not so obvious. Looking at it from the point of view of a similarity leads to think that 0.5 over 1.0 corresponds to a high degree of overall similarity. It means that, by guessing initially blind, untrained weight vectors, the expected similarity (that is, the degree of recognition) is yet half the maximum possible. It it thus reasonable to presume that a lower (possibly much lower, depending on the precise model) initial value would be more tenable.

4.5 Preliminary Experimental Comparison between Similarity Measures

4.5.1 Experiment description

To study the validity of the general approach and, at the same time, to preliminary assess how well each of the proposed neuron models (acting as basis functions) is able to approximate a sampled function, they are experimentally compared on a two-dimensional benchmark function [Su and Sheen, 92], the non-linear system identification problem:

$$y(k) = y_1(k-1) + y_2(k-1)$$

where

$$\begin{aligned} y_1(k) &= 2.5y(k)sin[\pi e^{-u^2(k)-y^2(k)}] \\ y_2(k) &= u(k)[1+u^2(k)] \end{aligned} \tag{4.91}$$

The output $y(k)$ depends on the previous input $u(k-1)$ and on the previous output $y(k-1)$. A learning set of $\vartheta_x = 100$ input/output pairs is constructed by setting $y(0) = 0$ and randomly exciting the system using a signal $u(k)$ uniformly drawn from $[-2.0, 2.0]$.

In this task there is no data heterogeneity -the two variables are continuous- and there is no missing information. It has been chosen as a preliminary case study precisely because

[7]That is, in the mid part of the sigmoid linear range: see the last part of (§2.1.2).

of that: the primary interest being to compare the relative usefulness of several instances of the parameterized functions (4.76) to (4.80) as bricks for constructing basic distance-based similarity models on continuous variables, as well as to have them compared to the more classical neural models. A fuzzy model is also investigated. The following are the explored models:

1. The standard scalar-product neuron (`SclProd`) plus a trainable bias and a logistic $g_{1,0}$, and the three derived measures, (4.44) (`SclProd2`), (4.32) (`SclProd3`) and (4.33) (`SclProd4`), all of them using a similarity keeping sigmoidal (4.74), with $k = 0.1$.
2. Two RBF models, using Euclidean distance as aggregation, and a Gaussian ($\hat{s}_3$ for $\alpha = 2$) as activation function. The first model, `StdRBF` has a different variance (to be learned) for every hidden unit. The second, `NormRBF`, computes a normalized distance —that in Proposition (4.9), for $q = 2$— followed by a Gaussian adapted to an argument in $[0,1]$, by setting $\alpha = 2, a = 6$ in (4.79).
3. A representative set of $\hat{s}$ functions -displayed in Table (4.1)- chosen from (4.76) to (4.80), applied to partial distance computations, as explained below.
4. A fuzzy model, obtained by considering the original crisp data as triangular fuzzy numbers.

The partial similarity measures between the two variables in point 3. above are computed using a distance-based similarity function, obtained by application of the different $\hat{s}$ to the basic distance (4.51) on continuous variables. Notice that S0_0 -the first obtained function- in Table (4.1) directly corresponds to (4.52). They are all aggregated using a simple arithmetic mean:

$$s(\vec{x}, \vec{w}^i) = g_{11,0.5}\left(\frac{\hat{s}(d(x_1, w_1^i)) + \hat{s}(d(x_2, w_2^i))}{2}\right) \tag{4.92}$$

complemented with a similarity keeping function, thus being type (B) similarities, whilst the two RBF models are examples of type (A). In other words, no measure in point 3. above, although being distance-based and accepting only real-valued and complete data, can be seen as a classical RBF computation, because the former express a similarity formed by aggregation of partial distance-based measures, while the RBF expresses a similarity obtained from a global distance (in this case, in $\mathbb{R}^2$), which results in a different computation since the $\hat{s}$, except S0_0, are non-linear.

Besides, the partial non-linear measures are linearly aggregated to form a non-linear similarity so that, in principle, there is no need to apply a further non-linearity (in the form of a similarity keeping function). The behaviour of (4.92) is illustrated in Fig. (4.17) for several choices of $\hat{s}_0$. These variations are to be tested experimentally.

The fuzzy model uses (4.60) for the partial similarities, the same aggregation and $\check{s} = g_{11,0.5}$ as above. The imprecision is roughly estimated as follows. Data is generated from (4.91) by rounding to four significant digits (three decimal); the application of equation (4.91)

Function Index	Parameters	Expression
S0_0	$\hat{s}_0$ for d=1, α=1	$1-z$
S0_1	$\hat{s}_0$ for d=1, α=2	$(1-z)^2$
S0_2	$\hat{s}_0$ for d=0.5, α=1	$1-\sqrt{z}$
S0_3	$\hat{s}_0$ for d=1, α=4	$(1-z)^4$
S0_4	$\hat{s}_0$ for d=0.25, α=1	$1-\sqrt[4]{z}$
S0_5	$\hat{s}_0$ for d=0.5, α=2	$(1-\sqrt{z})^2$
S0_6	$\hat{s}_0$ for d=0.5, α=4	$(1-\sqrt{z})^4$
S0_7	$\hat{s}_0$ for d=0.25, α=2	$(1-\sqrt[4]{z})^2$
S0_8	$\hat{s}_0$ for d=0.25, α=4	$(1-\sqrt[4]{z})^4$
S1_0	$\hat{s}_1^*$ for a=1, α=0.25	$\frac{2}{1+\sqrt[4]{z}}-1$
S1_1	$\hat{s}_1^*$ for a=1, α=0.5	$\frac{2}{1+\sqrt{z}}-1$
S1_2	$\hat{s}_1^*$ for a=1, α=1.0	$\frac{2}{1+z}-1$
S1_3	$\hat{s}_1^*$ for a=1, α=2.0	$\frac{2}{1+z^2}-1$
S3_0	$\hat{s}_3$ for a=6, α=0.5	$exp\{-6\sqrt{z}\}$
S3_1	$\hat{s}_3$ for a=6, α=1.0	$exp\{-6z\}$
S3_2	$\hat{s}_3$ for a=6, α=2.0	$exp\{-6z^2\}$
S4_0	$\hat{s}_4$ for c=1.0	$1-g_{11,0.5}(1.0z)$
S4_1	$\hat{s}_4$ for c=1.5	$1-g_{11,0.5}(1.5z)$
S4_2	$\hat{s}_4$ for c=2.0	$1-g_{11,0.5}(2.0z)$
S4_3	$\hat{s}_4$ for c=2.5	$1-g_{11,0.5}(2.5z)$

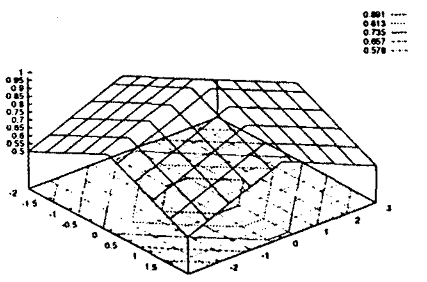
Table 4.1: The different $\hat{s}(z)$ used in the experiment for partial similarity computations on continuous variables. The symbol $(*)$ denotes the adaptation to an argument $z \in [0,1]$, so that $\hat{s}_1^*(z) = 2\hat{s}_1(z) - 1$. In all cases, $s_{max} = 1$.

magnifies the errors up to a factor of 10. Therefore, symmetric triangular fuzzy numbers $\tilde{x}$ centered at the original value x are associated to x as $\mu_{\tilde{x}} = \mu_{[x,\alpha(x)]}$, $\alpha(x) = (\alpha_1, \alpha_2) = (kx, kx)$, $k = 5 \cdot 10^{-3}$.

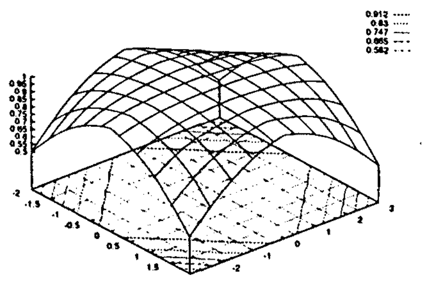
4.5.2 Experimental setup

All of the tested models use exactly the same experimental environment, in which everything is kept constant except the neuron model itself. The two input variables are normalized to lie in $[0,1]$. This is not needed by the heterogeneous neurons because they compute a normalized measure, but is beneficial for the standard models. The output is not normalized. A fixed architecture composed of $h_1 = 8$ neurons arranged in a hidden layer plus an output linear neuron is used. In general, the number of free parameters for the standard models in 1. and 2. is given by:

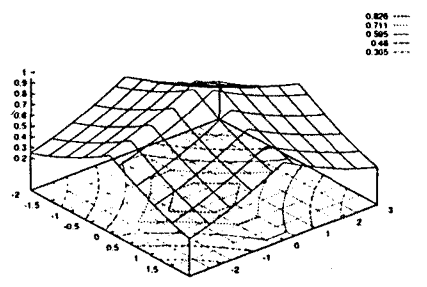
$$(n+1)h_1 + h_1 + 1 = h_1(n+2) + 1 = 33 \tag{4.93}$$



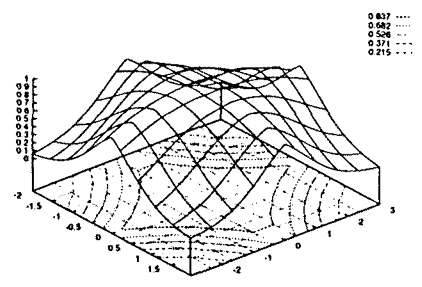
(a) $d = 1, \alpha = 1$, no $\check{s}$



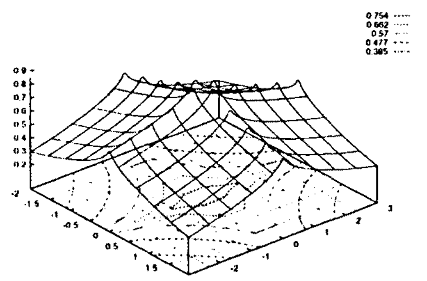
(b) $d = 1, \alpha = 1$, with $\check{s}$



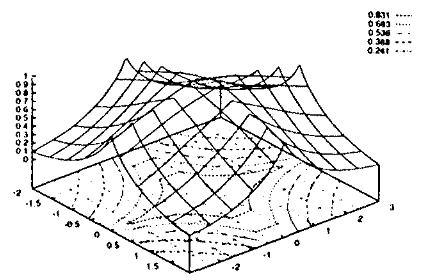
(c) $d = 1, \alpha = 2$, no $\check{s}$



(d) $d = 1, \alpha = 2$, with $\check{s}$



(e) $d = 0.5, \alpha = 1$, no $\check{s}$



(f) $d = 0.5, \alpha = 1$, with $\check{s}$

Figure 4.17: Example response of (4.92) for a reference weight vector $\vec{w}^i = (0, 0)$ and $\vec{x} \in [-2, 2] \times [-3, 3]$, with and without a further smoothing $\check{s} = g_{11,0.5}$ non-linearity. Note how the response in (c) and (e) is less sensitive than in (a). The same happens for (d) and (f) against (b).

where $n = 2$ is the number of inputs. For the heterogeneous neurons in 3. it is given by:

$$nh_1 + h_1 + 1 = h_1(n+1) + 1 = 25 \tag{4.94}$$

while for the fuzzy model is:

$$2nh_1 + h_1 + 1 = h_1(2n+1) + 1 = 41 \tag{4.95}$$

The weights (including biases and standard deviations) for the models in 1. and 2. are let to vary in $[-10, 10]$, a sufficiently wide range given the normalization chosen. The only exception is the measure (4.44) (SclProd2) for which, according to its definition, the weights and bias belong to $[0, 1]$. The interval $[-10, 10]$ is also used for the hidden-to-output weights in all the networks.

The training procedure employed is the *Breeder Genetic Algorithm* (BGA), characterized by *truncation* selection, a deterministic procedure driven by a *breeding* mechanism, an artificial selection method in which only the best individuals —usually a fixed percentage τ of total population size μ— are selected and enter the gene pool to be recombined and mutated, as the basis to form a new generation. A detailed description can be found in Chapter (6). Notice that the training method used is not an studied aspect here. It is implicitly assumed that different procedures are not to alter the *relative* behaviour of the models.

The BGA is used with the following parameters: $\mu = 60, \tau = 25$, EIR recombination with $\delta = 0.45$, and continuous mutation with $\rho = 0.5, k = 8$. A number of NRuns=10 independent runs are performed, each of which is allowed 10,000 error function evaluations, a moderate amount of computation for an ANN optimization task (a total of $\lfloor 10,000/60 \rfloor = 166$ generations are to be carried out).

Function	Mean MSE $\pm\ \sigma/\sqrt{n}$	Best MSE	$\beta(K,s)$
S0_0	1.228 ± 0.045	1.010	0.795
S0_1	0.752 ± 0.053	0.521	0.541
S0_2	0.448 ± 0.041	0.328	0.475
S0_3	0.371 ± 0.037	0.223	0.289
S0_4	0.273 ± 0.015	0.211	0.172
S0_5	0.356 ± 0.033	0.266	0.176
S0_6	0.840 ± 0.074	0.456	0.053
S0_7	1.312 ± 0.060	0.959	0.040
S0_8	5.315 ± 0.176	4.333	0.008

Table 4.2: Performance of the heterogeneous models using $\hat{s}_0$.

This means that $\vartheta_w = h_1\mu$NRuns $= 4,800$ is the sample size of initial weights as they are generated to form the various initial populations of individuals. For each tested model, the *mean square error* is reported (as MSE) plus/minus the normalized standard deviation in the usual form. The best MSE found is also reported. The last column is the mean initial similarity, a measure of network sensitivity, computed as in (4.90). The results of the training process are displayed in Tables (4.2) to (4.5) for the different models explored. The following remarks are in order:

Function	Mean MSE $\pm \sigma/\sqrt{n}$	Best MSE	$\breve{\beta}(K,s)$
S0_0	0.745 ± 0.050	0.633	0.692
S0_1	0.500 ± 0.029	0.350	0.524
S0_2	0.362 ± 0.044	0.176	0.487
S0_3	0.235 ± 0.017	0.133	0.352
S0_4	0.220 ± 0.015	0.145	0.302
S0_5	0.258 ± 0.045	0.151	0.280
S0_6	0.309 ± 0.023	0.173	0.124
S0_7	0.324 ± 0.053	0.149	0.117
S0_8	3.050 ± 0.090	2.704	0.028

Table 4.3: Same as Table (4.2), without using the $\breve{s} = g_{11,0.5}$ non-linearity.

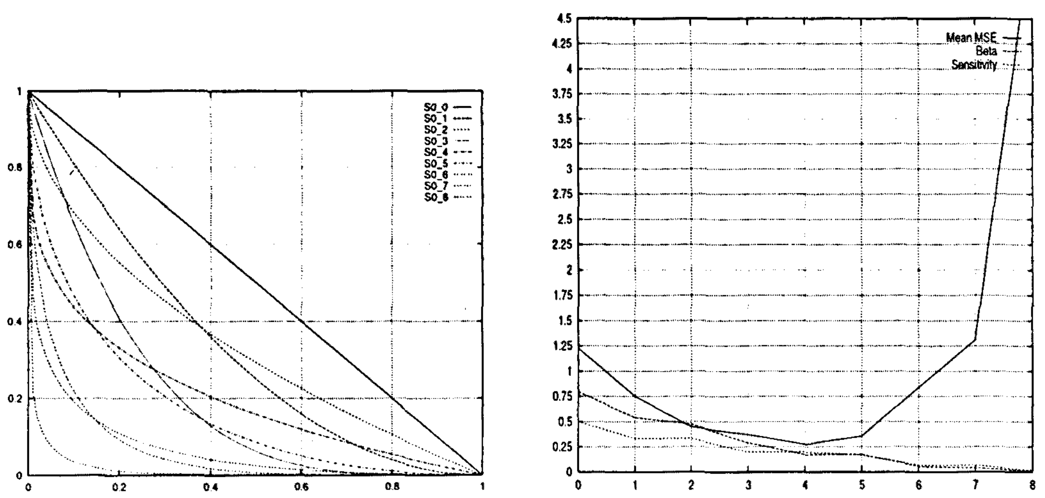


Figure 4.18: Left: The functions named S0_0 to S0_8. Right: Plot of Mean MSE, $\odot$ and $\breve{\beta}(K,s)$, for the functions S0_0 to S0_8 (x-axis), according to Table (4.2).

1. The most interesting family is the S0, because of its richer set of functions, displayed in Fig. (4.18, left). As it can be derived, the parameters d, α are in control of the overall sensitivity of the model. Obviously, for increasing α or decreasing d, high similarities are obtained only for really similar vectors, so that the neuron becomes less sensitive. More precisely, the $\odot$ sensitivities for the functions S0_0 to S0_8 are, respectively,

$$\{\frac{1}{2}, \frac{1}{3}, \frac{1}{3}, \frac{1}{5}, \frac{1}{5}, \frac{1}{6}, \frac{1}{15}, \frac{1}{15}, \frac{1}{70}\}$$

This is reflexed in Table (4.2) where, on the one hand, there is a correlation between the factor $\breve{\beta}(K,s)$ and the above $\odot$ sensitivities. On the other hand, there is a deep relation to the obtained performance. Note how $\odot$S0_0=1/2 is half the maximum (perhaps too high) and $\odot$S0_8=1/70 $\approx$ 0.014 is a small value, leading to an almost unsensitive neuron. The three factors are plot together in Fig. (4.18, right). The networks using these two

Function	Mean MSE $\pm\ \sigma/\sqrt{n}$	Best MSE	$\tilde{\beta}(K,s)$
S1_0	0.491 ± 0.050	0.311	0.072
S1_1	0.313 ± 0.042	0.143	0.245
S1_2	0.768 ± 0.057	0.534	0.613
S1_3	1.110 ± 0.022	0.954	0.882
S3_0	1.676 ± 0.095	1.276	0.036
S3_1	0.470 ± 0.088	0.173	0.212
S3_2	1.397 ± 0.134	0.751	0.630
S4_0	0.597 ± 0.025	0.444	0.838
S4_1	1.514 ± 0.083	1.207	0.607
S4_2	0.920 ± 0.076	0.596	0.450
S4_3	0.895 ± 0.109	0.504	0.355

Table 4.4: Performance of the heterogeneous models using $\hat{s}_1$, $\hat{s}_3$ and $\hat{s}_4$.

extreme models S0_0, S0_8 achieve the worst results (much worse for S0_8) for different reasons, displaying the highest (0.795) and lowest (0.008) $\tilde{\beta}$, respectively. For this family, the relations are the following:

$$\mathrm{S0\_8} \sqsubseteq \begin{matrix}\mathrm{S0\_7}\\ \mathrm{S0\_6}\end{matrix} \sqsubseteq \mathrm{S0\_5} \sqsubseteq \begin{matrix}\mathrm{S0\_3}\\ \mathrm{S0\_4}\end{matrix} \sqsubseteq \begin{matrix}\mathrm{S0\_1}\\ \mathrm{S0\_2}\end{matrix} \sqsubseteq \mathrm{S0\_0} \tag{4.96}$$

This translates in general in the mean initial similarity $\tilde{\beta}$, higher for higher $\odot$ sensitivities, as expected. In performance, which is markedly better in the middle of the relation scale (4.96), for functions S0_3, S0_4 and S0_5, corresponding to a $\tilde{\beta}$ in the range $[0.176, 0.289]$. Notice also how the sensitivity of these three functions is very similar (1/5, 1/5, 1/6) and different from the rest. The standard deviation of the errors reported for these functions are also the lowest, signaling less variability, as is the best result found (Best MSE).

2. In Table (4.3), the same experiment is repeated, this time not using the non-linear logistic as activation function. It can be seen that the results have generally improved those in the previous experiment, the possible reason being that the non-linearities in the proposed measures for partial similarities are enough to capture the relation (4.91). Notoriously, S0_0 has also improved, although it is now a linear model. The $\tilde{\beta}$ estimations have suffered a slight increase, due to the absence of the logistic, which tended to lower similarity values under 0.5 and raise those above 0.5. The results are in general accordance with Table (4.2), being again S0_3, S0_4 and S0_5 the best performing models.

3. Results for the other $\hat{s}$ functions tested are shown in Table (4.4), related as follows (inter-family relations not shown):

$$\mathrm{S1\_0} \sqsubseteq \mathrm{S1\_1} \sqsubseteq \mathrm{S1\_2} \sqsubseteq \mathrm{S1\_3} \tag{4.97}$$

Function	Mean MSE $\pm \sigma/\sqrt{n}$	Best MSE	$\tilde{\beta}(K,s)$
SclProd	0.549 $\pm$ 0.055	0.314	0.507
SclProd2	1.551 $\pm$ 0.028	1.380	0.218
SclProd3	0.917 $\pm$ 0.026	0.794	0.506
SclProd4	0.968 $\pm$ 0.034	0.814	0.004
StdRBF	1.573 $\pm$ 0.090	1.224	0.354
NormRBF	1.974 $\pm$ 0.052	1.628	0.522
Fuzzy	0.482 $\pm$ 0.041	0.252	0.064

Table 4.5: Performance of the models based on scalar product, the two RBF models and the fuzzy model.

$$\mathrm{S3\_0} \sqsubseteq \mathrm{S3\_1} \sqsubseteq \mathrm{S3\_2} \tag{4.98}$$

$$\mathrm{S4\_3} \sqsubseteq \mathrm{S4\_2} \sqsubseteq \mathrm{S4\_1} \sqsubseteq \mathrm{S4\_0} \tag{4.99}$$

The only acceptable performance (compared to the already found results) is that of S1_1 (the Gaussian function) yielding a network with estimated $\tilde{\beta} = 0.245$, inside the previous range. As for the S3 family, only S3_1 shows a fair performance ($\tilde{\beta} = 0.212$), while the logistic itself (S4 family) does not seem to be appropriate, possibly due to its characteristic S-shape.

4. The classical models are displayed in Table (4.5). The first remarkable point is the poor results achieved by the two RBF models, possibly caused by an insufficient number of basis functions, although factors as input normalization, different standard deviation per unit, a representative training set and the presence of only two variables are generally beneficial. Incidentally, notice how performance is comparable to that of S3_2 (which is also using a Gaussian as activation).

5. The mean result for the scalar-product (0.549) can be taken as a reference for the similarity-driven models. For those in Table (4.2), all the models from S0_2 to S0_5 yield a better performance. For those in Table (4.3), all the models from S0_1 to S0_7 —that is, all but the two extreme ones— outperform it, in various degrees. The standard deviation of the errors and the Best MSE are also lower. Notice how the estimated $\tilde{\beta} = 0.507$ for SclProd is close to the expected value 0.500. Interestingly, the nearest model in performance (S0_1) yields a similar $\tilde{\beta} = 0.524$.

6. The measures SclProd2 reports also relatively poor results, though in the line of those for the RBF. We conjecture that its estimated $\tilde{\beta} = 0.218$ is too low for a scalar-product, possibly caused by the input distribution. The other two measures base on scalar product show to be valid as neuron models, performing in between the true scalar product SclProd0 and the two RBF measures. Note that while SclProd3 still behaves as a scalar product (indicated by its $\tilde{\beta} = 0.506$), SclProd4 can be seen as what it is: a

true similarity measure for reflexivity is taken in the strong sense, as indicated by its low $\tilde{\beta} = 0.004$. However, both yield an analogous performance. More work should be devoted to study these measures, possibly by controlling their sensitivity.

7. Regarding the fuzzy model, the first notable point is its very low sensitivity ($\tilde{\beta} = 0.064$), due to the way the partial similarities are computed. The fairly good obtained results (Mean MSE=0.482) indicate that explicitly considering uncertainty in a network leads to valid models enjoying an added flexibility -because they work on a superset of the reals. This compensates for the increase in the number of free parameters, less likely to be properly constrained by a limited size data set [Bishop, 95].

This experiment has been intended as an illustrative example of how to develop neuron models under the introduced framework. It has been chosen deliberately simple (two variables, no data heterogeneity, no missing values) and yet the variety of models permits to elucidate many of the introduced ideas and hypotheses.

The estimated value $\tilde{\beta}(K, s)$ is to give a further insight into the workings of each model. For example, for S0_0 in Table (4.2), its value of nearly 0.8 is expressing many things when compared to that for scalar product (0.5) in Table (4.5). This value is explained when thinking that, for scalar product, not being a proper similarity, the partial measures (the correlations) can be negative, thus contributing *negatively* to the total. In contrast, the other measures are pure similarities and the partials are always non-negative. Besides, the adopted semantics for partials ensure that even small contributions can only *add* something to the overall measure, thus leading easily to high values.

This is not inherently wrong —and agrees with the idea that a negative similarity is senseless— as long as we do not allow the overall measure to easily reach these high values, for stimuli only fairly similar. This is precisely what is achieved in a simple way by controlling the partial measures via the parameters d, α. That is, setting less sensitive partial measures, thereby reducing their contribution to the total, though keeping it still positive. An indication of how sensitive on average (at least initially) is a network is given by $\tilde{\beta}(K, s)$.

Perhaps the most noteworthy conclusion is the relation between model sensitivity $\odot$, estimated network $\tilde{\beta}(K, s)$ and final network performance. In particular, sensitivity has been shown to have a deep impact on performance. In a working model, its value is to depend on two factors: the training set K_0 and the actual measure s being used. Holding the first factor constant, varying the shape of s reflects in the resulting performance of the system, and this is in turn related to the $\tilde{\beta}(K, s)$ estimation. In the presented experiment, the overall results are only indicative because of the simplicity of this preliminary study -only two variables (both of them significant), no missing values and no complex interactions present- but they are in general accordance with the discussion presented.

4.6 Concluding Remarks

One of the main contributions of the approach presented in this chapter consists in providing a conceptual framework for the study of neuron models as similarity computing devices.

The intention is not to embody all the existing neuron models –a difficult topic about which some excellent material exists [Reyneri, 99]. Rather, it is to provide a precise though comprehensive framework that fastens the most common models under the general cover of a similarity measure, thereby exploiting the various advantages stemming from this. We use the term **Heterogeneous Neural Network** for the broad class of architectures making use of neuron models derived from the framework and show the relation between neuron models of the kind used in MLP and RBF networks. The rationale behind the approach is that, by replacing, whenever convenient, the traditional scalar-product by an *explicit*, *normalized* and *heterogeneous* similarity measure, it is expected that the resulting model is to capture more satisfactorily the underlying process (in terms of quality of the approximation, generalization ability or readability of the solution found).

Perhaps a more important prospect is that it offers the possibility to develop *new* models, eventually taking into account data heterogeneity, a further extension that fits nicely in the framework. For example, standard (localized) RBF models are seen as particular cases whenever all the partial similarity computations are distance-based and continuous, using a specific aggregation measure and in the absence of missing values. The scheme stresses also the fact that certain activation functions are by no means tied to specific neuron models. Instead, what counts is the properties they fulfill and its capacity to adapt to different roles; in this sense, the sigmoidal family of functions is a flexible means to do so.

In general, the similarity relation is to be data dependent, and possibly problem dependent, since the same data could be treated differently for different tasks. In any case, it is ensured that any derived instance is to behave as a (soft) pattern recognizer and that similar patterns for the input units are to elicit similar responses in the hidden units. This may add an extra insight in the processing carried out in an artificial neuron (especially in the hidden ones) because of the known and controlled response characteristic and the possibility to add prior knowledge in its design. The $\hat{s}$ functions proposed, being parameterized families, allow to have a control on the exact shape of the neural response, leading to the idea of *sensitivity*. In this vein, the specific analysis of the subspace within $[0, s_{max}]^{h_1}$ spanned by the h_1 hidden neurons and the way it is covered should give a deeper insight in the overall process and opens possibilities for further refinement.

All this suggests that there are useful alternatives to classical neuron models. The task of selecting the most appropriate is however an open question. A given measure may maximize the accuracy of a learner in an experiment but perform poorly as compared to others in another one. This in turn relates to the number of examples needed to represent a specific function, bringing to the conclusion that there is no "universal" similarity measure that efficiently represents any target function [Globig and Lange, 96].

Clearly, applications on real-world size problems as well as thorough studies in ANN benchmark data (as [Prechelt, 94, Murphy and Aha, 91]) are needed to assess the validity experimentally. These problems and benchmarks should specially include rich forms of data, as discrete, nominal or linguistic variables, and missing information, traditionally resistant to classical models.

Chapter 5

The Universal Approximation Property for HNN

All exact science is dominated by the idea of approximation.

Bertrand Russell

This Chapter is devoted to the analysis of the representational ability of Heterogeneous Neural Networks (HNN). In subsequent chapters, it is illustrated how satisfactory solutions can be obtained in practice, for several real-world problems, by various types of HNN.

5.1 Introduction

The universal approximation property for ANN relates to the ability of approximating an unknown target function to any desired degree of accuracy. From a theoretical point of view, the property is important because it ensures that a satisfactory solution is always to exist. However, the practical concern is limited, in most occasions, to an existence proof, and it does not address the question of the precise form of the network (as number of hidden layers and units per layer) nor how to find adequate weights for it. For this reason, experimental work is the best means to assess the validity of the property because it illustrates the *quality* of the actually obtained approximations. Nevertheless, the property can act as a complement to the more practical results, and at the same time give a further insight into the involved neuron models, because it requires an additional effort of formalization.

Since HNN constitute a very generic framework of families of similarity-based models, a common and general proof is not amenable to be obtained, unless making hypotheses about the specific similarity functions being computed by the network. In consequence, we split the proof into classes of functions –into the types of similarity measures worked out in previous chapters (cf. §4)–, assuming precise decompositions of the overall computation. We show in this chapter that, under certain conditions, several types of feedforward HNN are classes of universal approximators. In all cases, we make use of the properties fulfilled by these neuron models. This scheme is also followed to handle the different kinds of data heterogeneity.

The rest of the chapter is organized as follows. The next Section (§5.2), deals with preliminary material, presenting the neuron models in a useful uniform notation. In (§5.3), the formal notion of universal approximation is discussed, along with the required basic concepts, and the basic results in the literature relevant to neural networks are reviewed. Section (§5.4) contains the results concerning the feed-forward networks worked out in this Thesis. Finally, some conclusions to the chapter are presented in (§5.5).

5.2 Preliminaries

The problem of learning an input/output relation from a set of examples can be regarded as the task of approximating an unknown function $f(\vec{x})$ from a set of data points, which are possibly sparse. The question of approximation by classical feed-forward ANN has been addressed by several authors: [Funahashi, 89], [Cybenko, 89], [Hornik, Stinchcombe and White, 89], [Hecht-Nielsen, 89], [Hornik, 90], [Leshno *et al.*, 93] and others for MLP networks, and [Hartmann, Keeler and Kowalski, 90], [Girosi and Poggio, 89], [Park and Sandberg, 91] and others for RBF networks. In essence, these networks are equivalent to a parametric approximating function $\mathcal{F}_{\underline{\vec{w}}}(\vec{x})$ and have been shown to be able of representing generic classes of functions —as the continuous or integrable functions— to an arbitrary degree of accuracy.

In general, there are three questions that arise when defining one such parameterized family of functions $\mathcal{F}_{\underline{\vec{w}}}(\vec{x})$, where $\underline{\vec{w}}$ represents the whole collection of network parameters (see §2.1):

i) What is the most adequate form of $\mathcal{F}_{\underline{\vec{w}}}(\vec{x})$, for a given problem?

ii) How to find the best $\underline{\vec{w}}$ for the chosen $\mathcal{F}_{\underline{\vec{w}}}(\vec{x})$?

iii) What classes of functions $f(\vec{x})$ can be represented (i.e., approximated) by $\mathcal{F}_{\underline{\vec{w}}}(\vec{x})$, regardless of $\underline{\vec{w}}$, and how well?

The universal approximation property concerns only the third question. The other two are usually tackled by model selection or network design techniques, and learning algorithms, respectively. The property is shared by algebraic and trigonometric polynomials, as is shown by the classical Stone-Weierstrass theorem [Stone, 48]. There is nothing special nor exclusive about ANN in this respect. In fact, many classical approximation schemes exist that can be represented as a "neural" network with a single hidden layer, and exhibiting the Stone-Weierstrass property [Girosi and Poggio, 89]. Yet, the existence of powerful learning algorithms and the variety of neuron models endow ANN with a special practical interest and raise the above question *iii)* as a natural one to be asked.

In an ANN approximation scheme, the approximating functions are members of a parameterized family. Existent work has been almost entirely devoted to the properties of one-hidden-layer networks. A network with no hidden layer is not capable of approximating generic non-linear continuous functions [Widrow, 90]. On the other hand, two or more hidden layers are scarcely_used in practice and the proof that they share the property is simple, once we have it for single-hidden-layer networks [Scarselli and Tsoi, 98]. Hence, the majority of work in the literature deals with single-hidden-layer networks.

Consequently, in the following we shall concentrate on networks composed of one hidden layer of h_1 units and one output layer consisting of a single unit, whose task is to produce a linear combination of the outcomes for the hidden units, plus an offset term. This is the most simple architecture which should be capable of approximating a target function, given some very general conditions are met. It is sometimes common to consider an additional non-linear function σ in the output neuron. Such a network represents the class of functions:

$$\begin{aligned} S_\Gamma = \{\mathcal{F}_{\underline{\vec{w}}} : \mathbb{R}^n \to \mathbb{R} \mid \mathcal{F}_{\underline{\vec{w}}}(\vec{x}) = \sigma\left(\sum_{i=1}^{h_1} c_i\gamma_i(\vec{x}) + c_0\right), \\ (\vec{c}, c_0) \in \mathbb{R}^{h_1} \times \mathbb{R}, \sigma : \mathbb{R} \to \mathbb{R}, h_1 \in \mathrm{N}^+, \gamma_i \in \Gamma, 1 \leq i \leq h_1\}, \qquad \vec{x} \in \mathbb{R}^n \end{aligned} \tag{5.1}$$

where γ_i is the *transfer* function of neuron i, $1 \leq i \leq h_1$, in the hidden layer. The usual definition of layer forces these γ_i to belong to the same (parameterized) family Γ. In the ANN context, the γ_i have a very definite form:

$$\Gamma = \{\gamma_i : \mathbb{R}^n \to \mathbb{R} \mid \gamma_i(\vec{x}) = g(h(\vec{x}, \underline{\alpha}^i)),\ \underline{\alpha}^i \in A\}, \qquad \vec{x} \in \mathbb{R}^n \tag{5.2}$$

where $g : \mathbb{R} \to \mathbb{R}$ is the activation function and A is the neuron *parameter space*. This parameterized set of functions correspond to the type of hidden computing units in the network. In classical models, $A \subseteq \mathbb{R}^s$, being $s \in \mathrm{N}^+$ a function of n, denoted $s(n)$. It is normally required that $\Gamma \subset C(\mathbb{R}^n)$, where $C(A)$ denotes the set of all continuous functions $f : A \to \mathbb{R}$. The general expression $\sum_{i=1}^{h_1} c_i\gamma_i(\vec{x})$ is a semiparametric characterization, also known as a *dictionary method*, since the choice of the basis function family Γ corresponds to a particular dictionary [Friedman, 94].

We are assuming that the $\gamma_i(\vec{x})$ do not represent higher-order functions of $\vec{x}$, because this would require a higher-than-linear number of parameters per neuron, w.r.t. the number of inputs. In other words, we shall consider $\gamma_i(\vec{x})$ such that $s(n) = \delta n + \gamma, \delta \in \mathbb{Z}^+, \gamma \in \mathrm{N}$. This is consistent with the common assumptions encountered in the literature [Scarselli and Tsoi, 98]. Dedicated proofs for higher-order units can be found, based on the proofs for linear units (see, for example, [Li and Chow, 96]).

The complexity $\hat{c}$ of the function $\mathcal{F}_{\underline{\vec{w}}}$ realized by the ANN can be measured by the length of its overall vector of parameters $\underline{\vec{w}}$, which in turn is a linear function of the number h_1 of hidden units, as:

$$\hat{c}(s, n, h_1) = h_1\{s(n) + 1\} + 1 \tag{5.3}$$

in the case of one-hidden-layer networks, as those in (5.1). In these conditions,

$$\underline{\vec{w}} = (\underline{\alpha}^1, \ldots, \underline{\alpha}^{h_1}, c_1, \ldots, c_{h_1}, c_0) \tag{5.4}$$

and $|\underline{\vec{w}}| = \hat{c}(s, n, h_1)$. The fraction of weight space $\mathbb{R}^{|\underline{\vec{w}}|}$ that contains a solution depends on two factors:

1. The capacity (information theoretic) of the network. This depends primarily on the representation capacity of the neuron model, which is the cornerstone for a good approximation.
2. The complexity of the problem: the function to be approximated and the availability of representative samples.

By far the most well-known and used in practice neuron models, the P-neuron and R-neuron give rise to the following networks[1].

- The multi-layer perceptron (MLP). $\underline{\alpha}^i = (\vec{w}^i, \theta_i) \in \mathbb{R}^n \times \mathbb{R}, s(n) = n + 1$. This leads to:

$$\gamma_i(\vec{x}) = g(h(\vec{x}, \underline{\alpha}^i)) = g(h(\vec{x}; \vec{w}^i, \theta_i)) = g(\vec{x} \cdot \vec{w}^i + \theta_i) \tag{5.5}$$

  where g is a bounded sigmoidal function. This is the *scalar product + bias* model.

- The radial basis function network (RBF). We consider three basic variants:
  1. Simple RBF. $\underline{\alpha}^i = (\vec{w}^i, \sigma_i) \in \mathbb{R}^n \times \mathbb{R}, s(n) = n + 1$. This leads to:

$$\gamma_i(\vec{x}) = g(h(\vec{x}, \underline{\alpha}^i)) = g(h(\vec{x}; \vec{w}^i, \sigma_i)) = g(\frac{1}{\sigma_i}||\vec{x} - \vec{w}^i||_q) \tag{5.6}$$

     where g is a localized-response function and the $||\cdot||_q$ are norms. This is the *center + variance* (or smoothing factor) model.
  2. Weighted RBF. $\underline{\alpha}^i = (\vec{w}^i, \vec{\sigma}^i) \in \mathbb{R}^n \times \mathbb{R}^n, s(n) = 2n$. This leads to:

$$\gamma_i(\vec{x}) = g(h(\vec{x}, \underline{\alpha}^i)) = g(h(\vec{x}; \vec{w}^i, \vec{\sigma}^i)) = g(||\vec{x} - \vec{w}^i; \vec{\sigma}^i||_q) \tag{5.7}$$

     where the $||\cdot;\cdot||_q$ are weighted norms and g is as before. This is the *center + variances* model.
  3. Generalized RBF. $\underline{\alpha}^i = (\vec{w}^i, M^i) \in \mathbb{R}^n \times \mathbb{R}^{n^2}, s(n) = \frac{n(n+3)}{2}$. This leads to:

$$\gamma_i(\vec{x}) = g(h(\vec{x}, \underline{\alpha}^i)) = g(h(\vec{x}; \vec{w}^i, M^i)) = g\left((\vec{x} - \vec{w}^i) \cdot M^i \cdot (\vec{x} - \vec{w}^i)\right) \tag{5.8}$$

     where M^i is a positive definite matrix and g is as before. This is the *center + co-variances* model, the most general form a RBF unit can take. This case is not considered here because is a higher-order model; notice that $s(n)$ is quadratic, though smaller than $n^2 + n$ due to the symmetric character of M^i.

[1]See also the introductory text in (§2.1).

It can be seen that the function h in all the RBF models always represents a distance in $\mathbb{R}^n$. Recall from (2.5) that the norm definitions are, for $\vec{z} \in \mathbb{R}^n$ and $q \geq 1 \in \mathbb{R}$:

$$\|\vec{z}\|_q = \left(\sum_{j=1}^{n} |z_j|^q\right)^{\frac{1}{q}} \tag{5.9}$$

and for $\vec{v} \in (\mathbb{R}^+)^n$:

$$\|\vec{z};\vec{v}\|_q = \left\{\sum_{j=1}^{n} \left(\frac{|z_j|}{v_j}\right)^q\right\}^{\frac{1}{q}} \tag{5.10}$$

Incidentally, notice that these two norms were shown to be valid ways for the aggregation of partial (one-dimensional) distances, with operators of the form $\Theta^{q,n}(\vec{z};\vec{v}) = \|\vec{z};\vec{v}\|_q$.

The expressions in (5.5) are called *ridge functions.* These functions are constant for input vectors $\vec{x}$ lying in planes orthogonal to $\vec{w}^i$. Almost all neuron transfer functions commonly developed in theory and used in practice can be classified as either radial basis or ridge functions.

The proofs existent in the literature concentrate on the approximation of scalar functions, of the form: $f : K \subset \mathbb{R}^n \to \mathbb{R}, n \in \mathbb{N}^+$, by means of networks with a single output neuron. The extension to codomains of the type $\mathbb{R}^m, m \in \mathbb{N}^+$, in what concerns approximation theory, is normally considered as straightforward via the use of m separate functions. We will also follow this policy in this work.

5.3 Universal Approximation

In mathematical terms, the capacity of a class of functions S_Γ of arbitrarily approximating any member of another class of functions -as, for example, $C(\mathbb{R}^n)$- is called the **universal approximation** (UA) property. The property is related to **denseness**, in the sense that the class of functions of interest (in our case, the set S_Γ of functions represented by neural networks of a given form) is **dense** in, say, $C(\mathbb{R}^n)$.

5.3.1 Basic definitions and notation

We first review some of the basic concepts related to sets, norms and denseness. The following definitions can be found in any standard textbook on Functional Analysis [Kolmogorov and Fomin, 75], [Yosida, 74]:

Definition 5.1 (Limit point) *A point $x \in K$ is a limit or accumulation point of a subset $M \subset K$ if every neighbourhood of x contains at least one point $m \in M$ different from x.*

Definition 5.2 (Closure) *Let K be a set of elements. The closure $[K]$ of K is defined as $[K] = K \cup K_{lim}$, being K_{lim} the set of all limit points of K.*

Definition 5.3 (Closed an open set) *A set K is closed if $[K] = K$. Otherwise it is open.*

Definition 5.4 (Compact set) *A set K is compact if every infinite $M \subset K$ contains at least one limit point.*

In spaces of finite dimension where a metric has been defined, every closed and bounded set is compact. In fact, in $\mathbb{R}^n$ the set of compact sets coincides with the set of closed and bounded sets (T. of Bolzano-Weierstrass).

Definition 5.5 (Dense set) *Let $M \subset K$. The set M is dense in K if $[M] = K$.*

This last definition means that each $k \in K$ can be approximated arbitrarily well by elements $m \in M$. The classical example is $M = \mathbb{Q}$, that is dense in $K = \mathbb{R}$ (every real number can be approximated as accurately as desired by a rational number). The set of polynomials is also dense in the set of continuous functions in compacta.

In general, $L^p(\Delta)$ is the set of all the measurable functions, that is, $f : \Delta \to \mathbb{C}$, with $\Delta \subseteq \mathbb{R}^n$ measurable, such that the (Lebesgue) integral:

$$\int_\Delta |f(\vec{x})|^p \, d\vec{x} \tag{5.11}$$

exists (i.e., is finite), where $\vec{x} = (x_1, \ldots, x_n)$. Restricting the discussion to real functions, defined almost everywhere in $\mathbb{R}^n$ w.r.t. the Lebesgue measure –that is, for which the set $\{\vec{x} \mid f(\vec{x}) \text{ is undefined}\}$ is of zero Lebesgue measure– three common functional norms are used. The first one is:

$$||f||_{L^p(\mathbb{R}^n)} = \left\{ \int_{\mathbb{R}^n} |f(\vec{x})|^p \, d\vec{x} \right\}^{\frac{1}{p}} \tag{5.12}$$

The set of functions bounded by this norm (i.e., those with a finite integral) is denoted $L^p(\mathbb{R}^n)$. For example, $\varphi \in L^1(\mathbb{R})$ means that:

$$\int_{-\infty}^{+\infty} |\varphi(x)| \, dx < \infty$$

The second norm is:

$$||f||_{L^\infty(\mathbb{R}^n)} = \sup_{\vec{x} \in \mathbb{R}^n} f(\vec{x}) \tag{5.13}$$

This one can be obtained as the limit $p \to \infty$ of (5.12), with the usual interpretation of the ∞-norm. The set of functions bounded by this norm is denoted $L^\infty(\mathbb{R}^n)$, and is known as the set of essentially bounded functions. The third norm is:

$$\|f\|_{L^p(\mu)} = \left\{ \int_{\mathbb{R}^n} |f(\vec{x})|^p \, d\mu(\vec{x}) \right\}^{\frac{1}{p}} \tag{5.14}$$

The set of functions bounded by this norm is denoted $L^p(\mu)$, where μ is a finite measure in $\mathbb{R}^n$. Similar norms $\|f\|_{L^p(K)}, \|f\|_{L^\infty(K)}$ and corresponding sets of functions $L^p(K), L^\infty(K)$ can be defined in a compact domain $K \in \mathbb{R}^n$.

When a class of functions is said to be *dense* in $L^p(\mathbb{R}^n), L^\infty(\mathbb{R}^n)$ or $L^p(\mu)$ (the sets defining the *target* functions), this is generally meant w.r.t. the corresponding norms, which are used to evaluate the difference between two functions. For example, denseness in $L^p(\mathbb{R}^n)$ is understood w.r.t. the $\|\cdot\|_{L^p(\mathbb{R}^n)}$ norm. Hence, without an explicit statement, we say that the class of functions S_Γ is **dense** in $L^* \in \{L^p(\mathbb{R}^n), L^\infty(\mathbb{R}^n), L^p(\mu), L^p(K), L^\infty(K)\}$ if:

$$\forall f \in L^*,\ \forall \epsilon > 0,\ \exists \varphi \in S_\Gamma \mid \|\varphi - f\|_{L^*} < \epsilon \tag{5.15}$$

Sometimes, the denseness result will be in other sets as $C(\mathbb{R}^n)$ or $C(K)$, for compact sets $K \in \mathbb{R}^n$; in these cases, and indication of the used norm will be given.

The **convergence** of a series of functions $\{f_n\}$ towards f in L^* is defined by:

$$\lim_{n \to \infty} \|f_n(\vec{x}) - f(\vec{x})\|_{L^*} = 0 \tag{5.16}$$

The convergence in $L^\infty(\mathbb{R})$ or $L^\infty(K)$ is called *uniform* convergence. A subset $S \subseteq C(\mathbb{R}^n)$ is said to be *uniformly dense* on compacta if, for every $K \subset \mathbb{R}^n$, K compact set, S is dense in $C(K)$ w.r.t. the $L^\infty(K)$ norm.

Together with (5.15), this means that:

$$\forall f \in C(K),\ \forall \epsilon > 0,\ \exists \varphi \in S_\Gamma \mid \sup_{\vec{x} \in K} |\varphi(\vec{x}) - f(\vec{x})| < \epsilon \tag{5.17}$$

Let $D = \{< \vec{x}_i, y_i > \mid f(\vec{x}_i) = y_i\}$ be a set of samples of a target function f, where $\vec{x}_i \in \mathbb{R}^n$ is the stimulus, $y_i \in \mathbb{R}$ is its response and $|D| = p$. The apparent loss $\tilde{\mathcal{L}}(D, \varphi)$ can be defined, like in (§2.1.4), as:

$$\tilde{\mathcal{L}}(D, \varphi) = \sum_{<\vec{x}_i, y_i> \in D} \lambda^q(y_i, \varphi(\vec{x}_i))$$

An immediate consequence of denseness is that, for the usual loss functions of the form $\lambda^q(a, b) = |a - b|^q, q \in [1, +\infty)$, there exists a $\varphi \in S_\Gamma$ such that $\tilde{\mathcal{L}}(D, \varphi)$ is within any arbitrary $\epsilon > 0$ of the theoretical minimum or **true loss** (equal to zero in the noiseless case), which is equal to $||f - \varphi||_{L^q(\mathbb{R}^n)}$ [Fiesler and Beale, 97]. This is the usual framework in which supervised training of ANN is carried out.

The classical example in function approximation is given by the Weierstrass theorem:

Any $f \in C(K), K \subset \mathbb{R}^n$ compact set, can be uniformly approximated by a polynomial. In other words, the class $\mathcal{P}(K) = \{P_n(K) \mid n \in \mathrm{N}\}$ given by the union, for all n, of all the polynomic functions of degree n in K, is dense in $C(K)$, w.r.t. the $L^\infty(K)$ norm.

This is also one of the examples of classes of functions having the UA or denseness property, which are closer to the ANN approach, because they can be represented as a linear combination of basis functions (that is, as a dictionary method). Among the polynomials, we find:

Algebraic polynomials (in $K \subset \mathbb{R}$)

Taking the base functions $\gamma_i(x) = x^i$, the functions:

$$\mathcal{F}_{\underline{\vec{w}}}(x) = c_0 + \sum_{i=1}^{n} c_i x^i \tag{5.18}$$

represent polynomials of degree n by making $h_1 = n$. Comparing this expression to (5.1), here σ is the identity function, c_0 is the polynomial of degree zero, and $\underline{\vec{w}} = (c_0, c_1, \ldots, c_n)$.

Trigonometric polynomials (in $K \subset \mathbb{R}$)

The functions:

$$\mathcal{F}_{\underline{\vec{w}}}(x) = c_0 + \sum_{i=1}^{n} c_i sin(ix) + \sum_{i=1}^{n} c_i' cos(ix) \tag{5.19}$$

represent (trigonometric) polynomials of degree n by making $h_1 = 2n$, with σ and c_0 in the same conditions, and $\underline{\vec{w}} = (c_0, c_1, \ldots, c_n, c_1', \ldots, c_n')$.

5.3.2 Results in the literature

Early studies concerning universal approximation departed from Kolmogorov's celebrated theorem[2], which can be stated as follows [Kolmogorov, 57]:

Theorem 5.1 *Let $I^n = [0,1]^n$ denote the closed unit interval of dimension n. Any continuous function $f : I^n \to \mathbb{R}$ can be represented in the form:*

[2]Celebrated because it answered (negatively) one of the hypotheses (No. 13) raised by Hilbert in 1900.

$$f(x_1,\ldots,x_n) = \sum_{j=1}^{2n+1} \phi_j \left(\sum_{i=1}^{n} \psi_{ij}(x_i) \right) \tag{5.20}$$

where ϕ_j, ψ_{ij} are continuous functions of one variable. In particular, the ψ_{ij} are monotone functions not depending on f.

This result was later enhanced and rewritten by [Sprecher, 72]. The concern for neural networks begins with R. Hecht-Nielsen, who uses the theorem to show that a network with three layers (two hidden) can in principle represent any continuous mapping $f : I^n \to \mathbb{R}$, with the units in the first (hidden) layer computing the ψ_{ij}, those in the second (hidden) layer computing the inner summation and the ϕ_j, and the one in the third (output) layer performing the outer summation [Hecht-Nielsen, 89]. The problem with this approach is that it was far from the actually used neural networks. For example, the "units" in the theorem are not of the smooth sigmoidal type. However, it respects the basic idea of (5.20) of representing a continuous function of many variables by superposition of continuous functions of only one variable.

The first formal results for ANN were obtained for single-hidden-layer sigmoidal MLP networks by [Funahashi, 89], nearly but independently followed by [Cybenko, 89] and [Hornik, Stinchcombe and White, 89]. Their results show that, for the scalar product-based neuron model (5.5), if g is a sigmoidal function, that is, a continuous, monotonically increasing and bounded real function $g : \mathbb{R} \to (g_{min}, g_{max})$ fulfilling:

$$\lim_{x \to -\infty} g(x) = g_{min} \tag{5.21}$$

$$\lim_{x \to +\infty} g(x) = g_{max} \tag{5.22}$$

normally called a "squashing" function, and σ is the identity function, then S_Γ in (5.1) is uniformly dense in $C(K)$, for $K \subset \mathbb{R}^n$ compact set. The result was extended to networks with more than one hidden layer in [Hornik, Stinchcombe and White, 89]. Cybenko, independently, showed that for any bounded sigmoidal, the class S_Γ in (5.1) is dense in $C(K)$, with $K = [0,1]^n \subset \mathbb{R}^n$.

Later, these results were greatly extended in [Hornik, 93]:

1. If g is *analytic* and *non-polynomial*, the previous results hold with a *single* bias $\theta \in \mathbb{R}$, common to all the hidden units, provided it is a point where g and all its derivatives are non-null. This last condition is not difficult to be met. The key assumption is the non-polynomial character of g.

2. If g is *Riemann-integrable* and *non-polynomial* (the bias $\theta_i \in \mathbb{R}$ being no longer a shared one, but different for all the hidden units), the previous result still holds.

   It is known that a function is **Riemann-integrable** if it is bounded on compact sets having a set of discontinuities with zero Lebesgue measure [Kolmogorov and Fomin, 75].

This is one of the most general results concerning uniform convergence by MLP in the literature. If g is a polynomial, then the input distribution must be concentrated in a finite set (i.e., have a finite support).

3. Denoting by $S_\Gamma(A, B)$ the same set S_Γ in (5.1) but restricting all the weight vectors $\vec{w}^i$ and the bias θ_i of the γ_i so that $\forall i : 1 \leq i \leq h_1 : (\vec{w}^i \in A \wedge \theta_i \in B)$ then, provided B is closed and g is defined on B, and A contains a neighbourhood of the origin, then again $S_\Gamma(A, B)$ is uniformly dense in $C(K)$, for $K \subset \mathbb{R}^n$ compact set.

4. Finally, if g is essentially bounded (recall this means that its norm $||g||_{L^\infty}$ is finite) instead of being Riemann-integrable, then $S_\Gamma(A, B)$ is dense in $L^p(\mu)$, for any compactly supported measure μ.

These results were complemented by [Leshno *et al.*, 93]. They showed that if g is *Riemann-integrable* and locally bounded (this means that $||g||_{L^p(K)}$ is finite for every compact set $K \subset \mathbb{R}$) then S_Γ is dense in $L^p(\mu)$ and in $C(\mathbb{R}^n)$ if and only if g is *not* a polynomial. Also significant is the work of [Chen, Chen and Liu, 95], wherein it is shown that, for every bounded sigmoidal, S_Γ is dense in $C(K)$, for $K \subset \mathbb{R}^n$ compact set.

The first results concerning approximations by RBF networks can also be traced back to the late 80s, where [Girosi and Poggio, 89] show, for units of the form (5.6), that the particular choices:

$$h(\vec{x}; \vec{w}^i, \sigma_i) = \frac{1}{\sigma_i}||\vec{x} - \vec{w}^i||_2 \tag{5.23}$$

and $g(z) = e^{-\frac{1}{2}z^2}$ lead to one-hidden-layer networks showing the UA property, by application of the Stone-Weierstrass theorem. This result was obtained in parallel though independently by [Hartmann, Keeler and Kowalski, 90]. There is a more powerful result in [Poggio and Girosi, 89], namely, that all networks derived from regularization theory are dense in $C(K)$, for $K \subset \mathbb{R}^n$ compact set. This outcome includes, in particular, RBF networks with the basis function being the Green's function of a self-adjoint differential operator, associated to the Tikhonov stabilizer [Tikhonov and Arsenin, 77]. Such Green functions include most of the known classical approximation schemes, such as the Gaussian and several types of splines. This scheme does *not* include the MLP, because its neuron model cannot be obtained from regularization techniques [Girosi, Jones and Poggio, 93].

A major breakthrough was achieved by Park & Sandberg. In their work, the authors state and prove the following theorem:

Theorem 5.2 ([Park and Sandberg, 91]) *For generic neuron models of the form:*

$$\gamma_i(\vec{x}) = K\left(\frac{\vec{x} - \vec{w}^i}{\sigma}\right), \qquad \sigma > 0 \in \mathbb{R},\ \vec{x}, \vec{w}^i \in \mathbb{R}^n \tag{5.24}$$

where $K : \mathbb{R}^n \to \mathbb{R}$ is called a **kernel** *function, then, provided the following conditions are met:*

1. *K is an integrable bounded function*

2. *K is continuous almost everywhere*

3. $\int_{\mathbb{R}^n} K(\vec{x})\, d\vec{x} \neq 0$

then S_Γ is dense in $L^p(\mathbb{R}^n)$, $\forall p \in [1, +\infty)$.

For future convenience, we define the integral operator $[\cdot]$ for functions $f : \mathbb{R}^n \to \mathbb{R}$, as $[f]_n = \int_{\mathbb{R}^n} f(\vec{x})\, d\vec{x}$. Note that the three conditions above basically require K to be integrable (i.e., $[K]_n < +\infty$) and such that $[K]_n \neq 0$. Note also that the smoothing factor σ is the same for all the hidden units.

Before examining what is the concern for RBF networks of general kinds, let us mention some related results. The first one is an extension of (5.24) where the smoothing factors σ_i are not necessarily equal for all i.

Theorem 5.3 ([Park and Sandberg, 91]) *For generic neuron models of the form:*

$$\gamma_i(\vec{x}) = K\left(\frac{\vec{x} - \vec{w}^i}{\sigma_i}\right), \qquad \sigma_i > 0 \in \mathbb{R},\ \vec{x}, \vec{w}^i \in \mathbb{R}^n \tag{5.25}$$

where $K : \mathbb{R}^n \to \mathbb{R}$ is a **kernel** *function, provided the following conditions are met:*

1. $K \in L^1(\mathbb{R}^n)$

2. $[K]_n \neq 0$

then the class S_Γ is dense in $L^1(\mathbb{R}^n)$.

The following four theorems were introduced in the authors's subsequent work [Park and Sandberg, 93], and serve as a complement to the main theorems (5.2) and (5.3).

Let $K : \mathbb{R}^n \to \mathbb{R}$ be an integrable function. Then, for neuron models of the form (5.24), the class S_Γ is dense in $L^1(\mathbb{R}^n)$ if and only if $[K]_n \neq 0$.

Let $K : \mathbb{R}^n \to \mathbb{R}$ be an integrable function. Then, for neuron models of the form (5.25), the class S_Γ is dense in $L^1(\mathbb{R}^n)$ if and only if $[K]_n \neq 0$.

Let $K : \mathbb{R}^n \to \mathbb{R}$ be an integrable function. Then, for neuron models of the form (5.25), for which $K \in L^p(\mathbb{R}^n), \forall p \in (1, +\infty)$ and $[K]_n \neq 0$, the class S_Γ is dense in $L^p(\mathbb{R}^n)$.

The last one is in fact a corollary of the preceding material:

Let $K : \mathbb{R}^n \to \mathbb{R}$ be an integrable, continuous function, satisfying $[K]_n \neq 0$. Then, for neuron models of the form (5.24), the class S_Γ is dense in $C(W)$, for $W \subset \mathbb{R}^n$ compact set.

5.3.3 Relation to RBF networks

The concern of theorems (5.2) and (5.3) for RBF networks —understanding by such, networks whose neuron model for the hidden units is the R-neuron— is immediate. As a preliminary work towards our main results, we first show their applicability to classical RBF networks, and then establish its generic applicability for more general models. For kernels K radially symmetric, that is, whenever:

$$\forall \vec{x}, \vec{y} \in \mathbb{R}^n \qquad \|\vec{x}\|_2 = \|\vec{y}\|_2 \Rightarrow K(\vec{x}) = K(\vec{y}), \tag{5.26}$$

we obtain the particular case $K(\vec{z}) = g(\|\vec{z}\|_2)$, where g is any continuous function $g : [0, +\infty) \to \mathbb{R}$. Thus, we can write kernels of the form:

$$\gamma_i(\vec{x}) = K\left(\frac{\vec{x} - \vec{w}^i}{\sigma}\right) = g\left(\frac{\|\vec{x} - \vec{w}^i\|_2}{\sigma}\right) \tag{5.27}$$

Most often, g is a localized-response function, according to the proposed definition (5.6).

In general, we are not limited to the Euclidean norm. We can have $K(\vec{z}) = g(\|\vec{z}\|_q), q \geq 1 \in \mathbb{R}$. The behaviour of these models is not radial anymore. Rather, it is "radial" w.r.t. the norm $\|\cdot\|_q$ being used. In any case, any Minkowskian norm (5.9) is valid and leads to neuron models of the general form:

$$\gamma_i(\vec{x}) = g\left(\left\|\frac{\vec{x} - \vec{w}^i}{\sigma}\right\|_q\right) = g\left(\frac{1}{\sigma}\|\vec{x} - \vec{w}^i\|_q\right) \tag{5.28}$$

Values of $q \neq 2$ are rarely used (in this case, the most usual is to have $q = 1$). Notice that the first condition for the applicability of Park & Sandberg's Theorem (5.2) is that the neuron model be written in the general form $\gamma_i(\vec{x}) = K\left(\frac{\vec{x} - \vec{w}^i}{\sigma}\right)$, where the σ can (though need not) be the same for all $i : 1 \leq i \leq h_1$. The second condition is that $[K]_n < +\infty$, and such that $[K]_n \neq 0$.

We have just seen how, for basic RBF models, $h(\vec{x}; \vec{w}^i, \sigma) = \frac{1}{\sigma}\|\vec{x} - \vec{w}^i\|_q$. Therefore, defining $K(\vec{z}) = g(\|\vec{z}\|_q)$, by (5.28) the models are of the required form $\gamma_i(\vec{x}) = K\left(\frac{\vec{x} - \vec{w}^i}{\sigma}\right)$ in (5.24). Similarly, for the simple models in (5.6):

$$h(\vec{x}; \vec{w}^i, \sigma_i) = \frac{1}{\sigma_i}\|\vec{x} - \vec{w}^i\|_q,$$

we have $\gamma_i(\vec{x}) = K\left(\frac{\vec{x} - \vec{w}^i}{\sigma_i}\right)$, for the same definition of K, that is a neuron model of the form in (5.25). Whereas, for the weighted models in (5.7):

$$h(\vec{x}; \vec{w}^i, \vec{\sigma}^i) = \|\vec{x} - \vec{w}^i; \vec{\sigma}^i\|_q,$$

we have $\gamma_i(\vec{x}) = K\left(\frac{\vec{x}-\vec{w}^i}{\vec{\sigma}^i}\right)$, for the same definition of K, where the vector division is performed component-wise.

Specially noteworthy is the fact that there is no requirement for radial symmetry of the kernel function K in the above theorem; as the authors readily point out:

"The theorem is stronger than necessary [in the sense of being more generally applicable] for RBF networks, and might be useful for other purposes." [Park and Sandberg, 91]

In our case, these other purposes involve the use of the theorem for more general ANN, like the HNN. To begin with, more general norms $\|\cdot\|_{L^*}$ can be used, fulfilling $\|\vec{x}\|_{L^*} = \|\vec{y}\|_{L^*} \Rightarrow K(\vec{x}) = K(\vec{y})$, with $\|\cdot\|_{L^*}$ any norm in $\mathbb{R}^n$. This means we can always see the obtained kernels as distance-based, since a distance can be defined simply by the existence of a norm: $d^*(\vec{x},\vec{y}) = \|\vec{x}-\vec{y}\|_{L^*}$ [Kolmogorov and Fomin, 75]. Notice also that, as stated in Appendix (A), all the norms in $\mathbb{R}^n$ are equivalent. In the present case, we have kernels $K^*(\vec{z}) = g(\|\vec{z}\|_{L^*})$. It then follows that:

$$\gamma_i(\vec{x}) = g\left(\|\frac{\vec{x}-\vec{w}^i}{\sigma}\|_{L^*}\right) = g\left(\frac{1}{\sigma}\|\vec{x}-\vec{w}^i\|_{L^*}\right) = g\left(\frac{d^*(\vec{x},\vec{w}^i)}{\sigma}\right) \tag{5.29}$$

and the neuron model is again a distance-based one of the general required form $\gamma_i(\vec{x}) = K^*\left(\frac{\vec{x}-\vec{w}^i}{\sigma}\right)$, where $K^*(\vec{z}) = g(\|\vec{z}\|_{L^*})$. This is the way generic RBF models are defined.

It is clear that we can make further steps in developing more general neuron models, since there is no need for a norm of any kind. The final applicability of the theorem is only subject to the use of generic kernels K fulfilling the conditions $[K]_n \in \mathbb{R}$ and $[K]_n \neq 0$, and defining neuron models of the form $\gamma_i(\vec{x}) = K\left(\frac{\vec{x}-\vec{w}^i}{\sigma}\right), \sigma > 0$.

Additionally, if K is a function satisfying $K(c\vec{z}) = cK(\vec{z})$, for a scalar $c \in \mathbb{R}^+$, then the smoothing factor can be integrated in the weights c_i since, according to (5.24):

$$c_i\gamma_i(\vec{x}) = c_iK\left(\frac{\vec{x}-\vec{w}^i}{\sigma}\right) = \frac{c_i}{\sigma}K(\vec{x}-\vec{w}^i) \tag{5.30}$$

5.4 Main results

5.4.1 Summary of main results

In the following Sections, proofs for the universal approximation ability of some of the introduced families of neuron models are developed. It has to be noted that the versatility and general extent of the potential neuron models included in the framework prevents a generic proof to be attempted. Instead, dedicated demonstrations are worked out on particular cases –corresponding to specific families of functions– or on specially interesting models, by making corresponding assūmptions. The proofs for the forthcoming propositions are mainly based on the general applicability of Theorem (5.2) and on another theorem due to Hornik [Hornik, 93].

1. In (§5.4.2), similarity models of type (A) (globally distance-based) are considered for real-valued variables, and a specific proof is developed for the standard metric in $\mathbb{R}$, the normalized modulus as aggregation operator, and several families of $\hat{s}$ functions, representative of many of the neuron models constructed in this Thesis.
2. In (§5.4.3), similarity models of type (B) (locally distance-based) are considered. The same choices are studied.
3. In (§5.4.4), the generic measure (4.73), developed in (§4.4.1), and based on Gower's similarity index, is examined as a particular case of the previous results, for real-valued and complete variables and for the logistic functions (4.74) and (3.9), this last one adapted to the interval $[0, 1]$.
4. In (§5.4.5), a P-neuron computing the point similarity measure (4.39), directly derived from scalar product, is considered.
5. In (§5.4.6), the result in (§5.4.3) is extended by considering partial similarity measures betwéen fuzzy numbers.
6. In (§5.4.7), the result in (§5.4.4) is extended to incorporate missing information.

The considered neuron models belong to the following class of functions:

$$G_s = \{\gamma_i : \mathbb{R}^n \rightarrow \mathbb{R} \mid \gamma_i(\vec{x}) = s(\vec{x}, \vec{w}^{\,i}),\ \vec{w}^{\,i} \in \mathbb{R}^n\}, \qquad \vec{x} \in \mathbb{R}^n \tag{5.31}$$

where $s : \mathbb{R}^n \times \mathbb{R}^n \rightarrow [0, s_{max}]$ is a similarity, pseudo-similarity or point similarity function in $\mathbb{R}^n$. This class corresponds to the generic S-neuron in Definition (4.16) where, in cases 1. to 4., $\hat{\mathcal{H}}^n = \mathbb{R}^n$.

5.4.2 Similarity-based models of type (A)

These measures are of the general form $s(\vec{x}, \vec{w}^{\,i}) = \hat{s}(d(\vec{x}, \vec{w}^{\,i}))$, $d \in D(\mathbb{R}^n)$, where $d(\vec{x}, \vec{w}^{\,i}) = \Theta(\vec{d}(\vec{x}, \vec{w}^{\,i}))$, with $\vec{d}(\vec{x}, \vec{w}^{\,i}) = \{d_1(x_1, w_{i1}), \ldots, d_n(x_n, w_{in})\}$, $d_j \in D(\mathbb{R})$, Θ is a distance aggregation operator, and $\hat{s}$ a similarity transforming function.

Proposition 5.1 *For the neuron models in (5.31), corresponding to measures of type (A) as above, where $d_j(x, y) = \alpha_j |x - y|$, for constant $\alpha_j \in \mathbb{R}^+, x, y \in \mathbb{R}, 1 \leq j \leq n$, that is, for distances proportional to the standard metric in $\mathbb{R}$, for the distance aggregation operator Θ in Definition (4.4), and for $\hat{s}$ integrable in $[0, \infty)$ and fulfilling the (mild) condition $\hat{s}(z, a) = \hat{s}(az, 1)$ (see below), the class S_Γ in (5.1) is dense in $L^p(\mathbb{R}^n)$, $\forall p \in [1, +\infty)$.*

The proof is divided in two parts. In the first part, we show how these similarities are valid neuron models, as kernel functions of the appropriate form in (5.24), regardless of $\hat{s}$. In the second part, we show the integrability conditions, which depend on $\hat{s}$.

Kernel functions

We have that the computation of a hidden node is a (similarity) function of the form:

$$\gamma_i(\vec{x}) = s(\vec{x}, \vec{w}^{\,i}) = \hat{s}(d(\vec{x}, \vec{w}^{\,i})) \tag{5.32}$$

We have to show how (5.32) is a kernel function K of the form $K\left(\frac{\vec{x}-\vec{w}^{\,i}}{\sigma}\right), \sigma > 0$. For $x_j, w_{ij} \in \mathbb{R}$, we have $d_j(x_j, w_{ij}) = \alpha_j |x_j - w_{ij}|$. Therefore, defining:

$$\vec{d}(\vec{x}, \vec{w}^{\,i}) = \{d_1(x_1, w_{i1}), \ldots, d_n(x_n, w_{in})\} = \{\alpha_1|x_1 - w_{i1}|, \ldots, \alpha_n|x_n - w_{in}|\}, \tag{5.33}$$

the overall distance computation $d(\vec{x}, \vec{w}^{\,i})$ is formed out of the components of $\vec{d}(\vec{x}, \vec{w}^{\,i})$ by means of a Minkowskian aggregation operator as in Definition (4.4), with $n' = n$. The choice $n' = 1$ would be equally valid, but by introducing an additional factor, the former is slightly more general:

$$\Theta^{q,n}(\vec{z}; \vec{v}) = \frac{1}{\sqrt[q]{n}} \left\{ \sum_{j=1}^{n} \left(\frac{|z_j|}{v_j} \right)^q \right\}^{\frac{1}{q}}, \quad q \geq 1 \in \mathbb{R} \tag{5.34}$$

We have then $d(\vec{x}, \vec{w}^{\,i}) = \Theta^{q,n}(\vec{x} - \vec{w}^{\,i}; \vec{\alpha}^{-1})$ where $\vec{\alpha}^{-1} = \left(\frac{1}{\alpha_1}, \ldots, \frac{1}{\alpha_n}\right)$. Notice that $\vec{\alpha}$ does not depend on i (that is, these weightings are equal for all the hidden units). This is reasonable, because these values typically depend on the input distribution of the variables -e.g., the vector of variances, etc, see (§4.2.1).

Now,

$$\hat{s}(d(\vec{x}, \vec{w}^{\,i})) = \hat{s}\left(\Theta^{q,n}(\vec{x} - \vec{w}^{\,i}; \vec{\alpha}^{-1})\right) \tag{5.35}$$

The smoothing factor $\frac{1}{\sigma}$ is usually provided by the $\hat{s}$ function itself (in this type of similarities, the $\hat{s}$ corresponds to a classical activation function). This free parameter, according to Theorem (5.2), can be the same for all the units (that is, it does not depend on i). For convenience, we write $\hat{s}(z, a)$ to express the dependence on the a parameter in $\hat{s}$. For example, considering $\hat{s}(z) = e^{-(az)^\alpha}$, $a > 0, \alpha > 0$ (5.40), this family is probably the most widely used for RBF networks, for $\alpha = 2$ (Gaussian function) and $\alpha = 1$ (exponential function). In particular, $\alpha = 2$ yields $\hat{s}_{\text{Gauss}}(z) = e^{-a^2 z^2}$. Making the substitution $a = \frac{1}{\sigma}$, we get the usual $\hat{s}_{\text{Gauss}}(z) = e^{-\frac{z^2}{\sigma^2}}$ form.

Returning to the main argument, from (5.35), and using the property $\hat{s}(z, a) = \hat{s}(az, 1)$:

$$\hat{s}(d(\vec{x}, \vec{w}^{\,i}), a) = \hat{s}\left(\Theta^{q,n}(\vec{x} - \vec{w}^{\,i}; \vec{\alpha}^{-1}), a\right) = \hat{s}\left(\Theta^{q,n}(a(\vec{x} - \vec{w}^{\,i}); \vec{\alpha}^{-1}), 1\right) \tag{5.36}$$

This last step can also be done because $a > 0$ and $\Theta^{q,n}$ fulfills the property $\Theta^{q,n}(a\vec{z}; \vec{v}) = a\Theta^{q,n}(\vec{z}; \vec{v}), \forall q \geq 1 \in \mathbb{R}, n \in \mathbb{N}^+$. Hence, by setting $a = \frac{1}{\sigma}$ in (5.36) and (5.32):

$$\gamma_i(\vec{x}) = \hat{s}\left(\Theta^{q,n}(\frac{\vec{x}-\vec{w}^i}{\sigma};\vec{\alpha}^{-1}),1\right) = K\left(\frac{\vec{x}-\vec{w}^i}{\sigma}\right), \tag{5.37}$$

where the kernel is more compactly written $K(\vec{z}) = \hat{s}\left(\Theta^{q,n}(\vec{z};\vec{\alpha}^{-1}),1\right)$ and the neuron model (5.37) is clearly a function of $\frac{\vec{x}-\vec{w}^i}{\sigma}$ and this completes the proof. Notice that any other constant factor coming from the Θ operator itself (as $\frac{1}{\sqrt[q]{n}}$ in this case), can be integrated in $\frac{1}{\sigma}$ (or, if desired, in the $\vec{\alpha}$).

Integrability conditions

The integrability condition of the kernel requires two things:

1. $[K]_n < +\infty$ (K is integrable)
2. $[K]_n \neq 0$ (its integral is non-null)

The first thing to point out is that the multiplicative constants α_j (being all positive) do not affect any of these conditions. The normalized modulus function is defined as $||\cdot||_q : \mathbb{R}^n \to \mathbb{R}^+ \cup \{0\}$, and is continuous everywhere. On the other hand, by definition, any $\hat{s} \in \hat{S}$

$$\hat{s} : \mathbb{R}^+ \cup \{0\} \to [0, s_{max}], \quad s_{max} > 0 \in \mathbb{R}$$

is a continuous, strictly monotonically decreasing function, fulfilling $\hat{s}(0) = s_{max}$, and $\hat{s}(z)$ fades away to zero as z goes to infinity. This means that the composition of the two is of the general form depicted in Fig. (5.1),

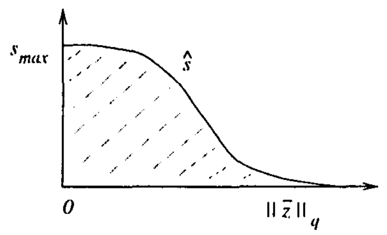


Figure 5.1: Plot of $\hat{s}(||\vec{z}||_q)$.

and thus $\lim_{||\vec{z}||_q \to \infty} \hat{s}(||\vec{z}||_q) = 0$. This guarantees (for the properties of $\hat{s}$) that $K(\vec{z})$ is always a strictly positive and bounded function of $\vec{z}$, implying that the integral -in case it exists- is strictly positive and thus different than zero. It does not, however, ensure that $[K]_n$ exists, provided we may not count on the integrability of $\hat{s}$. It depends on the exact form of $\hat{s}$. In fact, it depends on the exact form of the *composition* of $\hat{s}$ and the $||\cdot||_q$ norm. In consequence, we first focus on specific families for the functions $\hat{s}$, and carry out a separate proof for each family of $\hat{s}$ functions considered, though Proposition (5.1) is valid for any integrable $\hat{s}$.

We center on those $\hat{s}$ functions introduced in Chapter (§4): (4.77), (4.78) and (4.79) for the $\hat{s}$, reproduced here for convenience. Again, we write $\hat{s}(z,a)$ to express the dependence on the a parameter in $\hat{s}$. The following functions can be shown to fulfill the property $\hat{s}(z,a) = \hat{s}(az,1)$:

$$\hat{s}_1(z,a) = \{\frac{1}{1+(az)^{\alpha}}, a > 0, \alpha > 0\}, \qquad z \in [0,+\infty) \tag{5.38}$$

$$\hat{s}_2(z,a) = \{\frac{1}{1+e^{(az)^{\alpha}}}, a > 0, \alpha > 0\}, \qquad z \in [0,+\infty) \tag{5.39}$$

$$\hat{s}_3(z,a) = \{e^{-(az)^{\alpha}}, a > 0, \alpha > 0\}, \qquad z \in [0,+\infty) \tag{5.40}$$

and let $\hat{S} = \hat{s}_1 \cup \hat{s}_2 \cup \hat{s}_3$ generically denote the set of all members of these families.

Lemma 5.1 *The family of $\hat{s}$ functions (5.39) makes the kernel $K(\vec{z}) = \hat{s}\left(\Theta^{q,n}(\vec{z};\vec{\alpha}^{-1}),1\right)$ integrable in $\mathbb{R}^n$, where $\Theta^{q,n}$ is as in (5.34).*

Proof: The factors α_j of $\vec{\alpha}^{-1}$ (positive and constant) can be dropped, for they do not affect integrability. The same can be said about the factor $\frac{1}{\sqrt[q]{n}}$. By Proposition (C.5, 1.), this family of functions is integrable provided $a > 0, \alpha > 0$. In our case, since $a = 1$ and their argument is the $\|\cdot\|_q$ norm, defining $\underline{z} = \sum_j |z_j|^q \in [0,+\infty)$, the relevant part of the kernels can most conveniently be written as:

$$K(\vec{z}) = \hat{s}(\|\vec{z}\|_q, 1) = \frac{1}{1+e^{(\|\vec{z}\|_q)^{\alpha}}} = \frac{1}{1+e^{(\sum_j |z_j|^q)^{\frac{\alpha}{q}}}} = \frac{1}{1+e^{(\underline{z})^{\frac{\alpha}{q}}}}$$

Since $a > 0$ and $\frac{\alpha}{q} > 0$, this completes the proof.

Lemma 5.2 *The family of $\hat{s}$ functions (5.40) makes the kernel $K(\vec{z}) = \hat{s}\left(\Theta^{q,n}(\vec{z};\vec{\alpha}^{-1}),1\right)$ integrable in $\mathbb{R}^n$, where $\Theta^{q,n}$ is as in (5.34).*

Proof: Analogously, and by Proposition (C.5, 3.), we obtain the conditions $a > 0, \frac{\alpha}{q} > 0$.

Lemma 5.3 *The family of $\hat{s}$ functions (5.38) makes the kernel $K(\vec{z}) = \hat{s}\left(\Theta^{q,n}(\vec{z};\vec{\alpha}^{-1}),1\right)$ integrable in $\mathbb{R}^n$, where $\Theta^{q,n}$ is as in (5.34), provided $\frac{\alpha}{q} > 1$.*

Proof: Analogously, and by Proposition (C.5, 2.), we obtain the conditions $a > 0, \frac{\alpha}{q} > 1$.

Immediate examples for $\hat{s}$ can be derived, with $a > 0$, for the simple case $\frac{\alpha}{q} \in \mathbb{N}^+$:

- $\hat{s}(z) = \frac{1}{1+e^{az}}$ (inverse logistic) for the $\|\cdot\|_1$ or $\|\cdot\|_2$ norms.
- $\hat{s}(z) = \frac{1}{1+a^2z^2}$ (Cauchy function) for the $\|\cdot\|_1$ norm.

- $\hat{s}(z) = e^{-az}$ (exponential function) for the $\|\cdot\|_1$ norm.
- $\hat{s}(z) = e^{-a^2z^2}$ (Gaussian function) for the $\|\cdot\|_2$ norm.

This last example is one of the most classical RBF settings (a Gaussian applied to a, possibly weighted, Euclidean distance). It is derived here as a special case. If $\hat{s}$ is integrable in $\mathbb{R}$, a simpler and more direct proof can be derived.

Proposition 5.2 *For $\hat{s}$ integrable in $\mathbb{R}$, the kernel $K(\vec{z}) = \hat{s}\left(\Theta^{q,n}(\vec{z};\vec{\alpha}^{-1})\right)$ is integrable in $\mathbb{R}^n$, where $\Theta^{q,n}$ is as in (5.34).*

Proof. Again, the factors α_j of $\vec{\alpha}^{-1}$ (positive and constant) can be dropped, for they do not affect integrability. The same can be said about the factor $\frac{1}{\sqrt[q]{n}}$. Making use of Proposition (C.13), the relevant (for integrability) part of the kernel $K(\vec{z}) = \hat{s}(\|\vec{z}\|_q)$ —being $\hat{s}$ a positive and monotonically decreasing function in $[0,+\infty)$ by its definition (4.10)— is integrable in $\mathbb{R}^n$, for all $q \geq 1 \in \mathbb{R}$.

Since the considered neuron models $\gamma_i(\vec{x})$ in (5.32) are in the conditions of Theorem (5.2), the class S_Γ in (5.1) using these $\gamma_i(\vec{x})$ is dense in $L^p(\mathbb{R}^n)$, $\forall p \in [1,+\infty)$.

5.4.3 Similarity-based models of type (B)

These measures are of the general form $s(\vec{x},\vec{w}^i) = \check{s}(\Theta_s(\vec{s}(\vec{x},\vec{w}^i)))$, where $\vec{s}(\vec{x},\vec{w}^i) = \{s_1(x_1,w_{i1}),\ldots,s_n(x_n,w_{in})\}$, and $s_j(x_j,w_{ij}) = \hat{s}_j(d_j(x_j,w_{ij}))$, $d_j \in D(\mathbb{R})$, $1 \leq j \leq n$, with Θ_s a similarity aggregation operator, and $\check{s}$ a similarity keeping function.

Proposition 5.3 *For the neuron models in (5.31), corresponding to measures of type (B) as above, where $d_j(x,y) = \alpha_j|x-y|$, for constant $\alpha_j \in \mathbb{R}^+, x,y \in \mathbb{R}, 1 \leq j \leq n$, that is, for distances proportional to the standard metric in $\mathbb{R}$, for the similarity aggregation operator Θ_s in (5.34), for $\hat{s}_j$ integrable in $[0,\infty)$ and fulfilling the (mild) condition $\hat{s}_j(z,a) = \hat{s}_j(az,1), 1 \leq j \leq n$, and for $\check{s}$ a similarity keeping function, the class S_Γ in (5.1) is dense in $L^p(\mathbb{R}^n)$, $\forall p \in [1,+\infty)$.*

The proof is again divided in two parts. In the first part, we show how these similarities are valid neuron models, as kernel functions of the appropriate form in (5.24), in this case regardless of the precise form of Θ_s. In the second part, we show the integrability conditions, by assuming the mentioned specific form for Θ_s. Notice that we could suppose all the $\check{s}, \hat{s}_j$ to be equal (regardless of i) by definition of a neural layer, and thanks to the condition of a unique smoothing factor in (5.24).

Kernel functions

We have:

$$\gamma_i(\vec{x}) = s(\vec{x},\vec{w}^i) = \check{s}(\Theta_s(\vec{s}(\vec{x},\vec{w}^i))) \tag{5.41}$$

$$\vec{s}(\vec{x}, \vec{w}^i) = \{\hat{s}_1(\alpha_1|x_1 - w_{i1}|), \ldots, \hat{s}_n(\alpha_n|x_n - w_{in}|)\} \tag{5.42}$$

Again, the overall (similarity) computation $s(\vec{x}, \vec{w}^i)$ is formed out of the components of $\vec{s}$ by means of a similarity aggregation operator Θ_s of any of the forms in Proposition (4.11) or of any form compliant with Definition (4.8) and Remark (4.2) -as those in (4.12)- so that we write $s(\vec{x}, \vec{w}^i) = \check{s}(\Theta_s(\vec{s}(\vec{x}, \vec{w}^i); \vec{v}))$. In any case, for notational convenience, we define the special vector operator $< \cdot ; \cdot >_{\{.\}}: \mathbb{R}^n \times \mathbb{R}^n \times \hat{S} \rightarrow \mathbb{R}^n$ as follows:

$$< \vec{z}; \vec{r} >_{\{\hat{s}_j\}_{1 \leq j \leq n}} = \{\hat{s}_1(r_1 z_1), \ldots, \hat{s}_n(r_n z_n)\}, \tag{5.43}$$

so that, extending the absolute value operator to vectors in a straightforward way as $|\vec{z}| = (|z_1|, \ldots, |z_n|)$, we can write $\vec{s}(\vec{x}, \vec{w}^i) =< |\vec{x} - \vec{w}^i|; \vec{\alpha} >_{\{\hat{s}_j\}_{1 \leq j \leq n}}$ and then:

$$s(\vec{x}, \vec{w}^i) = \check{s}\left(\Theta_s(< |\vec{x} - \vec{w}^i|; \vec{\alpha} >_{\{\hat{s}_j\}_{1 \leq j \leq n}}; \vec{v})\right) \tag{5.44}$$

The smoothing factor can be supplied by the $\hat{s}_j$ functions, much as it was done in Proposition (5.1). We again write $\hat{s}_j(z, a)$ to express the dependence on the $a \in \mathbb{R}^+$ parameter in the $\hat{s}_j$. Then, from (5.44), and using $\hat{s}_j(z, a) = \hat{s}_j(az, 1), 1 \leq j \leq n$:

$$\check{s}\left(\Theta_s(< |\vec{x} - \vec{w}^i|; \vec{\alpha} >_{\{\hat{s}_j(\cdot,a)\}_{1 \leq j \leq n}}; \vec{v})\right) = \check{s}\left(\Theta_s(< |a(\vec{x} - \vec{w}^i)|; \vec{\alpha} >_{\{\hat{s}_j(\cdot,1)\}_{1 \leq j \leq n}}; \vec{v})\right) \tag{5.45}$$

For $a = \frac{1}{\sigma}$, we obtain the usual dependence on $\frac{\vec{x} - \vec{w}^i}{\sigma}$ for the neuron model:

$$\gamma_i(\vec{x}) = \check{s}\left(\Theta_s(< \left|\frac{\vec{x} - \vec{w}^i}{\sigma}\right|; \vec{\alpha} >_{\{\hat{s}_j(\cdot,1)\}_{1 \leq j \leq n}}; \vec{v})\right) = K\left(\frac{\vec{x} - \vec{w}^i}{\sigma}\right) \tag{5.46}$$

where the kernel is more compactly written $K(\vec{z}) = \check{s}\left(\Theta_s(< |\vec{z}|; \vec{\alpha} >_{\{\hat{s}_j(\cdot,1)\}_{1 \leq j \leq n}}; \vec{v})\right)$ and this completes this part of the proof. Again, any other constant factor coming from the Θ operator itself can be integrated in $\frac{1}{\sigma}$. The factors v_i cannot be integrated with the α_i (or vice versa) because the function $\hat{s}_j$ may be non-linear.

Integrability conditions

To study the integrability conditions, since the $\hat{s}_j$ are applied to each distance computation separately, we analyze first the effect of $\hat{s}_j$ on a single component of the similarity vector. This vector of partial similarities is simply denoted by $\vec{s} = (s_1, \ldots, s_n)$, where $s_j = \hat{s}_j(\alpha_j|x_j - w_{ij}|)$, $1 \leq j \leq n$. The overall analysis is simpler this time because each partial similarity s_j is a scalar function of its argument, so that we have for each s_j a similar picture as in (§5.4.2) where, by the properties of $\hat{s}_j$, it holds: $\lim_{z \to \infty} \hat{s}_j(z) = 0, 1 \leq j \leq n$.

We follow with the assumption that the aggregation operator Θ_s is specifically the family in (4.23) –reproduced in (5.34)– corresponding to the normalized modulus. This operator, as shown in (§4.2.3), should be general enough for most purposes and fulfills the totality of required good properties for such operators.

A basic condition for kernel integrability is then that the partial measures $\hat{s}_j$ be integrable in $[0, \infty)$. These measures can thus be seen as *partial kernels* $K_j(z) = \hat{s}_j(|z|), 1 \leq j \leq n$. Note again that the constant terms α_j, v_j (positive and multiplicative) can be ignored, for they do not alter the integrability conditions.

We depart from the hypothesis that the integrals $\int_{\mathbb{R}} K_j(z)dz$ exist and are non-null, for $1 \leq j \leq n$; this last condition is always ensured being the $\hat{s}_j$ positive. In practice, it is likely that all the $\hat{s}_j$ are the same function (i.e., all the similarities between real values are analogously computed) but this is of no importance here. Example $\hat{s}_j$ functions are those in (5.38) to (5.40), integrable in $[0, \infty)$ and thus $\hat{s}_j(|\cdot|)$ integrable in $\mathbb{R}$, by Proposition (C.5). Nonetheless, any other $\hat{s}_j$ integrable in $\mathbb{R}$ is possible.

Under the hypothesis of partial kernel integrability, the main task is to show that the relevant part of the whole kernel function:

$$K(\vec{z}) = \check{s}\left(\frac{1}{\sqrt[q]{n}}||\vec{K}(\vec{z})||_q\right), \tag{5.47}$$

where $\vec{K}(\vec{z}) = (K_1(z_1), \dots, K_n(z_n))$ is integrable in $\mathbb{R}^n$ and its integral is non-null. By using (5.9), we have:

$$||\vec{K}(\vec{z})||_q = \left\{\sum_{j=1}^{n} K_j(z_j)^q\right\}^{\frac{1}{q}} \tag{5.48}$$

The proof is developed by looking on the values of q. Let us begin with the simplest case, $\mathbf{q} = 1$, and let us also suppose, only for the moment, that $n = 2$ (i.e., we are in front of a two-dimensional problem). For the sake of clarity, we follow the discussion with two generic functions f and g, and later apply the results to the partial functions $K_j(z_j)$ in (5.48).

Now, given f, g two bounded functions defined in $\mathbb{R}$, the function $h(x, y) = f(x) + g(y)$ is *not* integrable in $\mathbb{R}^2$, even if f, g are, by Proposition (C.8). The reason is that the integrals w.r.t. x for constant values of y extend to infinity, and the value of these integrals is the infinite summation of a finite, constant value (the integral w.r.t. y), which is infinite.

To overcome this problem, we make use of a technique, to force integrability without altering the nature of the function. Define, for a given $\varepsilon > 0 \in \mathbb{R}$:

$$h(x, y) = \begin{cases} G(f(x), g(y)) & \text{if } f(x) \leq \varepsilon \vee g(y) \leq \varepsilon \\ f(x) + g(y) & \text{otherwise} \end{cases} \tag{5.49}$$

where $G : \mathbb{R}^2 \to \mathbb{R}$ is the function $G(u, v) = uv$, that makes $G(f(x), g(y)) = f(x)g(y)$

integrable in $\mathbb{R}^2$ -Proposition (C.10). A simpler and more "direct" choice would be $G(u,v) = 0$, but this choice will not meet future requirements (because it makes the function h nullify). In these conditions, we make the following claim:

Theorem 5.4 *Given f, g two bounded, positive and continuous functions integrable in $\mathbb{R}$, and the function $G(u,v) = uv$, the function $h(x,y)$ defined in (5.49) is integrable in $\mathbb{R}^2$, $\forall \varepsilon > 0 \in \mathbb{R}$.*

Proof. Let $A_\varepsilon = \{(x,y) \in \mathbb{R}^2 \mid f(x) > \varepsilon \wedge g(y) > \varepsilon\}$.

$$\int\int_{\mathbb{R}^2} h(x,y)\,dxdy = \int\int_{A_\varepsilon} [f(x) + g(y)]\,dxdy + \int\int_{\mathbb{R}^2 \setminus A_\varepsilon} G(f(x), g(y))\,dxdy \tag{5.50}$$

The first term in (5.50) deals with a continuous function, bounded in a closed and bounded set (a compact set) A_ε that, by Lebesgue's Theorem (C.1), is integrable in A_ε. The second term is an integrable function by hypothesis. Notice that the so-defined function (5.49) may not be continuous though this does not affect integrability.

In view of this result, that is extendable to any $n \in \mathbb{N}^+$, let us define the kernels K_ε:

$$K_\varepsilon(\vec{z}) = \begin{cases} \sum_{j=1}^n K_j(z_j) & \text{if } z \in A_\varepsilon \\ G(\vec{K}(\vec{z})) & \text{if } z \notin A_\varepsilon \end{cases} \tag{5.51}$$

where $K_j(z) = \hat{s}_j(|z|)$, $\hat{s}_j$ bounded and integrable in $[0, \infty)$, $1 \leq j \leq n$ by hypothesis, and $A_\varepsilon = \{(z_1, \ldots, z_n) \in \mathbb{R}^n \mid \forall j : 1 \leq j \leq n : K_j(z_j) > \varepsilon\}$, and $G(\vec{u}) = \prod_{j=1}^n u_j$.

Lemma 5.4 *The kernels (5.51) are integrable in $\mathbb{R}^n$, $\forall \varepsilon > 0 \in \mathbb{R}$.*

Proof.

$$\int_{\mathbb{R}^n} K_\varepsilon(\vec{z})d\vec{z} = \int_{A_\varepsilon} \sum_{j=1}^n K_j(z_j)d\vec{z} + \int_{\mathbb{R}^n \setminus A_\varepsilon} \prod_{j=1}^n K_j(z_j)d\vec{z} \tag{5.52}$$

The first term in (5.52) deals with a continuous function, bounded in a compact set A_ε that, by Lebesgue's Theorem (C.1), is integrable in A_ε. The second term is an integrable function by Proposition (C.10).

Lemma 5.5 $\forall \varepsilon > 0 \in \mathbb{R}$, *the kernels (5.51) are in the conditions of Theorem (5.2).*

Proof.

(*i*) $\forall \varepsilon > 0 \in \mathbb{R}$, the kernels K_ε fulfill $[K_\varepsilon]_n < \infty$, by application of Lemma (5.4).

(*ii*) $\forall \varepsilon > 0 \in \mathbb{R}$, the kernels K_ε fulfill $[K_\varepsilon]_n \neq 0$. Since the K_j are positive functions, provided the integral $[K_\varepsilon]_n$ exists (point (*i*)), it must be positive.

Proposition 5.4 *Let $f \in L^p(\mathbb{R}^n)$ be the target function to be approximated. Let $\delta > 0 \in \mathbb{R}$ be given. Then, $\forall \varepsilon > 0 \in \mathbb{R}$, there exist $h_1^* \in \mathbb{N}^+$, $\vec{w}^{i*}, 1 \leq i \leq h_1^*, \vec{w}^{i*} \in \mathbb{R}^n$, $\sigma^* > 0 \in \mathbb{R}$ and $c_i^* \in \mathbb{R}, 1 \leq i \leq h_1^*$, such that:*

$$\forall \vec{x} \in \mathbb{R}^n, \qquad \left\| f(\vec{x}) - \sum_{i=1}^{h_1^*} c_i^* K_\varepsilon\left(\frac{\vec{x} - \vec{w}^{i*}}{\sigma^*}\right) \right\|_{L^p} < \delta$$

Proof.

(*i*) $\forall \varepsilon > 0 \in \mathbb{R}$, the kernels K_ε are in the conditions of Theorem (5.2) by Lemma (5.5).

(*ii*) By application of Theorem (5.2), and using (5.15), the function f can be approximated to any degree of accuracy δ by an expression of the form $\sum_{i=1}^{h_1} c_i K_\varepsilon\left(\frac{\vec{x}-\vec{w}^i}{\sigma}\right)$, according to the L^p norm in $\mathbb{R}^n$.

(*iii*) For the given degree of accuracy $\delta > 0 \in \mathbb{R}$, let $h_1^*, \vec{w}^{i*}, \sigma^*, c_i^*, 1 \leq i \leq h_1^*$ denote the set of required parameters.

The entire process can be developed for any $q \geq 1 \in \mathbb{R}$, that is, for any power of the partial kernels, and a corresponding inverse power of the whole kernel, because they generate positive and bounded functions. To see this, let us define the kernels K_ε^q, for $q \geq 1 \in \mathbb{R}$:

$$K_\varepsilon^q(\vec{z}) = \begin{cases} \left\{\sum_{j=1}^n K_j^q(z_j)\right\}^{\frac{1}{q}} & \text{if } z \in A_\varepsilon \\ G(\vec{K}(\vec{z})) & \text{if } z \notin A_\varepsilon \end{cases} \tag{5.53}$$

where $K_j(z), A_\varepsilon$ and $G(\vec{u})$ are in the same conditions as for (5.51).

Lemma 5.6 *The kernels (5.53) are integrable in $\mathbb{R}^n$, $\forall \varepsilon > 0 \in \mathbb{R}$.*

Proof.

$$\int_{\mathbb{R}^n} K_\varepsilon^q(\vec{z})d\vec{z} = \int_{A_\varepsilon} \left\{\sum_{j=1}^n K_j^q(z_j)\right\}^{\frac{1}{q}} d\vec{z} + \int_{\mathbb{R}^n \setminus A_\varepsilon} \prod_{j=1}^n K_j(z_j)d\vec{z} \tag{5.54}$$

The first term in (5.54) deals with a continuous function, bounded in a compact set A_ε that, by Lebesgue's Theorem (C.1), is integrable in A_ε. The second term is an integrable function by Proposition (C.10).

Lemma 5.7 *$\forall \varepsilon > 0 \in \mathbb{R}$, the kernels (5.53) are in the conditions of Theorem (5.2).*

Proof. It can be developed analogously as for Lemma (5.5).

This means that Proposition (5.4) is still valid for the whole kernels $K_\varepsilon^q, q \geq 1 \in \mathbb{R}$. The final part of the proof concerns the application of a $\check{s}$ function to K_ε^q. In the following, we identify $h(x_1, \dots, x_n) = K_\varepsilon^q(x_1, \dots, x_n)$, and assume it integrable in $\mathbb{R}^n$.

To begin with, $\check{s}$ is a bounded function -see Definition (4.9). Second, by the same definition, it holds that $\exists \epsilon > 0 \mid \forall x \leq \epsilon,\ \check{s}(x) \leq x$. Hence, noting that h is a positive function $h : \mathbb{R}^n \to \mathbb{R}$, their combination is such that:

$$\exists h_0 > 0,\ \forall x_1, \dots, x_n \mid\ h(x_1, \dots, x_n) < h_0,\ \ \check{s}(h(x_1, \dots, x_n)) < h(x_1, \dots, x_n) \tag{5.55}$$

Let $H_0 = \{(x_1, \dots, x_n) \in \mathbb{R}^n \mid\ h(x_1, \dots, x_n) \geq h_0\}$, which has a finite measure $\mu(H_0)$. In these conditions, we have that:

$$\int_{\mathbb{R}^n} \check{s}(h(x_1, \dots, x_n)) dx_1, \dots, dx_n =$$
$$\int_{H_0} \check{s}(h(x_1, \dots, x_n)) dx_1, \dots, dx_n + \int_{\mathbb{R}^n \setminus H_0} \check{s}(h(x_1, \dots, x_n)) dx_1, \dots, dx_n \tag{5.56}$$

The first term in (5.56) deals again with a continuous function, bounded in a compact set H_0 that, by Lebesgue's theorem (C.1), is integrable in H_0, and its integral is bounded by $\check{s}_{max}\mu(H_0)$. The second term, by the previous remarks, is bounded by:

$$\int_{\mathbb{R}^n \setminus H_0} h(x_1, \dots, x_n) d\,x_1, \dots, d\,x_n$$

which, since $h(x_1, \dots, x_n)$ is integrable in $\mathbb{R}^n$, $\forall \varepsilon > 0 \in \mathbb{R}$ -Lemma (5.6)- is finite. The proof is now complete for the kernels $\check{s}(K_\varepsilon^q), q \geq 1 \in \mathbb{R}$.

Given any desired accuracy $\delta > 0 \in \mathbb{R}$, any function $f \in L^p(\mathbb{R}^n)$ can be approximated to this accuracy, and thus the class S_Γ in (5.1) is dense in $L^p(\mathbb{R}^n)$, $\forall p \in [1, +\infty)$, by application of Theorem (5.2).

In practice (in computational terms) any target function can be approximated as accurately as desired by taking a sufficiently large compact set containing A_ε. In other words, choosing a sufficiently small (as small as needed) $\varepsilon > 0$ in Proposition (5.4) -that is valid for *all* $\varepsilon > 0$- so that the K_ε^q can be made as close as needed to the actually used K in (5.47), by making K_ε^q always work inside A_ε. This technique ensures integrability in theory and does not affect the neuron models in practice.

5.4.4 Similarity models based on Gower's score

One of the most widely used similarity measure throughout this work has its roots on Gower's score [Gower, 71], introduced in (§4.4.1). For real-valued and complete variables, this measure can be seen as a special case of a distance-based type (B) similarity, as those defined in Proposition (5.3). In the absence of missing values, from (4.73) and (4.71), these measures correspond to functions of the form $\check{s}_G(\vec{x}, \vec{w}^i) = \check{s}(s_G(\vec{x}, \vec{w}^i))$, with:

$$s_G(\vec{x}, \vec{w}^i) = \frac{1}{n} \sum_{j=1}^{n} s_j(x_j, w_{ij}) \tag{5.57}$$

where the s_j are the partial similarities defined on the j-th components of the input and weight vectors. We then derive the following corollary of Proposition (5.3).

Corollary 5.1 *Given the neuron model $\gamma_i(\vec{x}) = \check{s}(s_G(\vec{x}, \vec{w}^i))$, with s_G in (5.57), and for sigmoidal $\check{s} : [0,1] \to [0,1]$ functions, the class S_Γ in (5.1) is dense in $L^p(\mathbb{R}^n)$, $\forall p \in [1, +\infty)$.*

Proof. It suffices to use Proposition (5.3) for $q = 1$.

We develop now a specific proof for the combination of s_G with *sigmoidal* $\check{s}$ functions. This topic has already been addressed in (§5.4.3) for general forms of these functions. We shall develop an alternative and hopefully more illustrative proof, by assuming a sigmoidal shape for $\check{s}$, and then apply it to the particular choices amply used in this work.

Let $\varphi : \mathbb{R}^n \to [0,1]$ be a function of interest. In our case, $\varphi(\vec{x})$ can be identified as $\varphi(\vec{x}) = s_G(\vec{x}, \cdot)$ in (5.57), integrable in $\mathbb{R}^n$ by Proposition (5.3). Since $\vec{w}^i$ is constant in s_G, this renaming will add to the clarity of the discussion. Let $\check{s}_\sigma$ be a sigmoidal-shaped and bounded $\check{s}$ function defined as $\check{s}_\sigma : [0,1] \to [0,1]$. Hence, for our purposes, $\check{s}_G(\vec{x}, \cdot) = \check{s}_\sigma(\varphi(\vec{x}))$.

Let us define $g : \mathbb{R} \to [0,1]$ as:

$$g(x) = \begin{cases} \check{s}_\sigma(x) & \text{if } x \in [0,1] \\ 0 & \text{if } x \notin [0,1] \end{cases} \tag{5.58}$$

This function is depicted in Fig. (5.2). Contrary to $\check{s}_\sigma(x)$, it is defined in $\mathbb{R}$. In $[0,1]$, both functions coincide and thus have the same integral. Furthermore,

$$\int_{\mathbb{R}} g(x)\, dx = \int_{[0,1]} \check{s}_\sigma(x)\, dx + \int_{\mathbb{R}\setminus[0,1]} 0\, dx = \int_{[0,1]} \check{s}_\sigma(x)\, dx$$

In these conditions, we have that:

Lemma 5.8

$$\int_{\mathbb{R}^n} |g(\varphi(\vec{x}))| d\vec{x} \text{ is finite.}$$

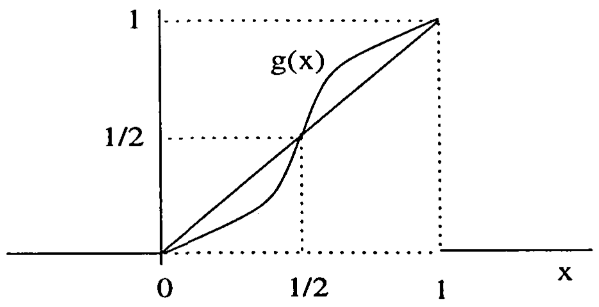


Figure 5.2: The sigmoidal function $g(x)$.

Proof. First, defining $A = \{\vec{x} \in \mathbb{R}^n \mid \varphi(\vec{x}) \geq \frac{1}{2}\}$, we have:

$$\int_{\mathbb{R}^n} |\varphi(\vec{x})| d\vec{x} = \int_{\mathbb{R}^n} \varphi(\vec{x}) d\vec{x} = \int_A \varphi(\vec{x}) d\vec{x} + \int_{\mathbb{R}^n \setminus A} \varphi(\vec{x}) d\vec{x} \tag{5.59}$$

since $\varphi(\vec{x})$ is positive. The integral $\int_A \varphi(\vec{x})\, d\vec{x}$ must be finite: given that $\varphi(\vec{x})$ is positive and $\int_{\mathbb{R}^n} \varphi(\vec{x})\, d\vec{x}$ is finite, both $\int_A \varphi(\vec{x})\, d\vec{x}$ and $\int_{\mathbb{R}^n \setminus A} \varphi(\vec{x})\, d\vec{x}$ are positive and finite.

In addition, A is a set of finite measure $\mu(A) < \infty$. To see this, let us suppose that it is infinite. Then, being $\varphi(\vec{x}) \geq \frac{1}{2}$, for $\vec{x} \in A$:

$$\mu(A) = \infty \Rightarrow \int_A \varphi(\vec{x})\, d\vec{x} \geq \int_A \frac{1}{2}\, d\vec{x} = \frac{1}{2}\mu(A) = \infty$$

but $\int_A \varphi(\vec{x})\, d\vec{x} < \infty$. Thus $\mu(A) < \infty$ (note that this implies $\mu(\mathbb{R}^n \setminus A) = \infty$).

Second, we have, being g also positive:

$$\int_{\mathbb{R}^n} |g(\varphi(\vec{x}))| d\vec{x} = \int_{\mathbb{R}^n} g(\varphi(\vec{x})) d\vec{x} = \int_A g(\varphi(\vec{x})) d\vec{x} + \int_{\mathbb{R}^n \setminus A} g(\varphi(\vec{x})) d\vec{x} < \infty \tag{5.60}$$

The first term, being $g(x) \in [0,1]$, is bounded by $1 \cdot \mu(A)$. The second term, being $g(x) \leq x$, for $x \leq \frac{1}{2}$, is bounded by $\int_{\mathbb{R}^n \setminus A} \varphi(\vec{x})$, which, as we have seen, is finite.

Intuitively speaking, we have shown that, for points $\varphi(\vec{x})$ above $\frac{1}{2}$, which form a closed set, the composition of the functions is bounded (by the unity function). For the rest of the points, the composition of φ with g keeps the integrability of φ because $g(x) \leq x$. Finally, note that the definition of g as zero outside $[0,1]$ –since our $\varphi(\cdot)$ has its image inside this interval– is just to have it defined in $\mathbb{R}$ in a safe way.

There are two distinguished sigmoidal-shaped $\check{s}$ functions, defined in (§4.4.1), that have been used extensively in the experimental part. The first one is the family of functions $g(\cdot, k) : [0,1] \to [0,1]$ (4.74), displayed in Fig. (4.8), where the parameter k controls the curvature of g.

It is not difficult to see that these functions fulfill the mild conditions of the previous corollary. In particular, the functions are continuous[3] and:

$$\begin{cases} g(x,k) \geq x & \text{for } x \geq \frac{1}{2} \\ g(x,k) \leq x & \text{for } x \leq \frac{1}{2} \end{cases} \tag{5.61}$$

as required, and where $g(\frac{1}{2},k) = \frac{1}{2}$. This value can be identified with h_0 in (5.55) and thus taking $h_0 = \frac{1}{2}$ suffices in this case. The second sigmoidal is the widespread logistic function which, as shown in (§4.4.1), can be used to work as a non-linear $\check{s}$ adapting it so that it maps the real interval $[0,1]$ on $(0,1)$, by taking $\check{s} = g_{11,0.5}$ in (3.9). Note that the change in codomain to the open $(0,1)$ does not affect the discussion.

5.4.5 Similarity models based on scalar product

We now show how the denseness property is shared by a scalar product-based similarity measure, fulfilling the required properties in Definitions (4.7) and (4.8), which is an example of a type (C) measure. In the conditions of (§4.2.5), let $f \in C_c(\mathbb{R}^n)$ (the set of compactly supported continuous functions in $\mathbb{R}^n$) the function to be approximated, and let $K \subset \mathbb{R}^n$ be the compact support of f. Given a compact set $W \subset \mathbb{R}^n$, define, as in (4.35):

$$\begin{cases} \pi^+_{(W)} = \sup\limits_{\vec{x},\vec{y} \in W \subset \mathbb{R}^n} \vec{x} \cdot \vec{y} \\ \pi^-_{(W)} = \inf\limits_{\vec{x},\vec{y} \in W \subset \mathbb{R}^n} \vec{x} \cdot \vec{y} \end{cases} \tag{5.62}$$

These values are guaranteed to exist, being the scalar product a continuous function and W a closed and bounded set. Let us recall the similarity (4.39):

$$s_p(\vec{x},\vec{y})_{(W)} = \frac{\vec{x}\cdot\vec{y} + \theta - \pi^-_{(W)}}{\pi^+_{(W)} + 1 - \pi^-_{(W)}} \in [0,1], \qquad \vec{x},\vec{y} \in W, \theta \in [0,1] \tag{5.63}$$

The measure s_p is a point similarity in W -Proposition (4.23). This measure does not come from any distance and we are not assuming any normalization on the vectors.

Proposition 5.5 *For neuron models in (5.31), corresponding to measures of type (C), where $s(\vec{x},\vec{w}^i) = \overline{g}(s_p(\vec{x},\vec{w}^i))$, with s_p the similarity measure in (5.63), and $\overline{g} : [0,1] \to (\overline{g}_{min}, \overline{g}_{max})$, a sigmoidal function, the class S_Γ in (5.1) is uniformly dense in $C(K)$.*

To prove it, we make use of the following theorem, previously mentioned in (§5.3.2):

[3]This is not strictly required: boundedness suffices.

Theorem 5.5 ([Hornik, 93]) *Let $g(z)$ be a Riemann-integrable and non-polynomial function, defined in a closed interval B. Let $K \subset \mathbb{R}^n$ a compact set. Denoting by $S_\Gamma(A,B)$ the set S_Γ in (5.1) with a neuron model as in (5.5), and restricting all the weight vectors $\vec{w}^i$ and the bias θ_i of the γ_i so that $\forall i : 1 \leq i \leq h_1 : (\vec{w}^i \in A \wedge \theta_i \in B)$ then, provided B is closed and g is defined on B, and A contains a neighbourhood of the origin, then $S_\Gamma(A,B)$ is uniformly dense in $C(K)$.*

and of the following Lemma:

Lemma 5.9 *Let $g : \mathbb{R} \to (g_{min}, g_{max})$ a sigmoidal function, and $\gamma_i(\vec{x}) = g(\vec{x}\cdot\vec{w}^i + \theta_i)$ the P-neuron model (5.5). The neuron model obtained by bringing g to an equivalent $\overline{g} : [0,1] \to (\overline{g}_{min}, \overline{g}_{max})$ of the same shape, is a P-neuron model.*

Proof. We develop a proof for the well-known family of logistic functions (3.9) (with $\theta = 0$), although any parameterized sigmoidal is conceivable, and the discussion would be analogous. We first deal with the adaptation to $[0,1]$. Let:

$$g_{\beta,0}(z) = \frac{1}{1+e^{-\beta(z)}} \in (0,1), \beta > 0 \tag{5.64}$$

An adapted function can be constructed as:

$$\overline{g}_{a,b}(z) = g_{\beta,0}(a(z-b)),\ a > 0 \tag{5.65}$$

with a, b such that $\overline{g}$ is defined in $[0,1]$ (e.g., $a = 11, b = 0.5$). We now show how the $\overline{g}_{a,b}$ are still logistic functions of scalar product with a bias, and of the same shape than $g_{\beta,0}$. First, we note that:

$$g_{\beta,0}(a(z-b)) = g_{\beta',\theta'}(z) \tag{5.66}$$

with $\beta' = \beta a$ and $\theta' = b$. Then, using (5.65), (5.66) and (5.63):

$$\overline{g}_{a,b}(s_p(\vec{x},\vec{w}^i)) = g_{\beta',\theta'}(s_p(\vec{x},\vec{w}^i)) = g_{\beta',\theta'}\left(\frac{\vec{x}\cdot\vec{w}^i + \theta - \pi^-_{(K)}}{\pi^+_{(K)} + 1 - \pi^-_{(K)}}\right) = g_{\beta'',\theta''}(\vec{x}\cdot\vec{w}^i + \theta) \tag{5.67}$$

with $\beta'' = \frac{\beta'}{\pi^+_{(K)}+1-\pi^-_{(K)}} = \frac{\beta a}{\pi^+_{(K)}+1-\pi^-_{(K)}}, \theta'' = \pi^-_{(K)} + (\pi^+_{(K)} + 1 - \pi^-_{(K)})\theta' = \pi^-_{(K)} + (\pi^+_{(K)} + 1 - \pi^-_{(K)})b$. Note that $\pi^+_{(K)} + 1 - \pi^-_{(K)} \neq 0$, and that $\beta'' > 0$ as required. Therefore, this function is just another member of the family of logistic functions, and (5.67) is a P-neuron.

Proof of Proposition (5.5). Let $f \in C_c(\mathbb{R}^n)$ an arbitrary function to be approximated, and let $K \in \mathbb{R}^n$ its compact support. The compact set K in which f is defined can always be transformed into a K_0 that contains a neighbourhood of the origin, by a simple transformation of the input variables, as follows:

Let $t : K \rightarrow K_0$, with t a continuous and injective function (in each coordinate) and K_0 containing a neighbourhood of the origin. Let $f_0 : K_0 \rightarrow \mathbb{R}$ defined as $f_0 = t^{-1} \circ f$. Clearly, $f_0 \in C(K_0)$. If we are able to approximate f_0, since $f = t \circ f_0$, and t is known, we will have an approximation to f in K as accurate as desired. The neuron model used to approximate f_0 in K_0 is of the form: $\gamma_i(\vec{x}) = \overline{g}(s_p(\vec{x}, \vec{w}^i)_{(K_0)})$, with $s_p(\vec{x}, \vec{w}^i)_{(W)}$ defined in (5.63), where the weight vectors are restricted to belong to K_0. The activation function $\overline{g} : [0,1] \rightarrow (\overline{g}_{min}, \overline{g}_{max})$ is a sigmoidal function, Riemann-integrable in compacta and non-polynomial, and in this case defined on the closed real interval $[0,1]$.

By Lemma (5.9), the neuron model $\gamma_i(\vec{x}) = \overline{g}(s_p(\vec{x}, \vec{w}^i)_{(K_0)})$ is a P-neuron. By application of Theorem (5.5), the class $S_\Gamma(K_0, [0,1])$ (corresponding to S_Γ with the weights restricted to K_0 and the bias restricted to $[0,1]$) is uniformly dense in $C(K_0)$. This means we can have an approximation to f_0 as accurate as desired w.r.t. the L^∞ norm, with the neuron models $\gamma_i(\vec{x})$. Since f can be constructed with the $f_0 \in C(K)$, we can have an approximation to $f \in C(K)$ for every $K \in \mathbb{R}^n$ compact set (that is, for $f \in C_c(\mathbb{R}^n)$) as accurate as desired, and hence these neuron models lead to approximation schemes that are dense in $C_c(\mathbb{R}^n)$. It is known that $C_c(\mathbb{R}^n)$ is dense in $L^p(\mathbb{R}^n)$, $\forall p \in [1, +\infty)$ [Rudin, 66]. The transformation t performed can be as simple as a linear one, of the kind usually carried out as a pre-processing in neural networks.

5.4.6 Extension to fuzzy numbers

To account for other data types, on which a similarity measure has been defined, profit can be made of Proposition (5.3), dealing with the obtention of generic measures by aggregation of partial ones, thereby extending the models in (5.31) to $\vec{x}, \vec{w}^i \in \hat{\mathcal{H}}^n$. In general, the extension can be achieved by showing that similarities defined on other data types lead also to integrable *partial* kernels of the required form. These kernels also depart from the real-valued ones in that they are not necessarily distance-based. An informal outline of the proof for fuzzy numbers is given, in the understanding that a more formal demonstration should be made in the future.

Let $\tilde{x}, \tilde{y} \in \mathbb{F}_n(\mathbb{R})$. From (4.60), we have that $s(\tilde{x}, \tilde{y}) = \Pi_{\tilde{x}}(\tilde{y})$ is a similarity in $\mathbb{F}_n(\mathbb{R})$, where Π is the possibility measure. When implanted in a particular input of a neuron model, a function of the form:

$$s(\tilde{x}, \tilde{w}^i) = \check{s}\left(\Pi_{\tilde{x}}(\tilde{w}^i)\right) \qquad (5.68)$$

can be built, where $\check{s}$ is a non-linear similarity keeping function, supplying the smoothing factor. As usual, we split the discussion in two parts:

i) To show that $s(\tilde{x}, \tilde{w}^i)$ is a function $K_{\mathbb{F}}$ of $\frac{x-w^i}{\sigma}$. For simplicity, we consider the factor $\frac{1}{\sigma}$ as provided by the $\check{s}$ function.

ii) This partial kernel $K_{\mathbb{F}}$ must be integrable in $\mathbb{R}$, that is, $\int_{\mathbb{R}} K_{\mathbb{F}}(z)\, dz < \infty$, and have a positive integral.

Part *i)*. We assume a specific transformation from x to $\tilde{x}$, given by a function $\tilde{\cdot} : \mathbb{R} \rightarrow \mathbb{F}_n(\mathbb{R})$, so that $\mu_{\tilde{x}}$ represents a symmetric fuzzy number in $\mathbb{R}$, where the mode is given by x, and the spread e_x by a number proportional to it, $e_x = kx$, normally $k \in [0, 1]$.

In case the chosen representation is *triangular* (4.55), a number x results in a fuzzy number $\tilde{x}$ of the form depicted in Fig. (5.3, left). Another possibility is the Gaussian form (4.57), depicted in Fig. (5.3, right).

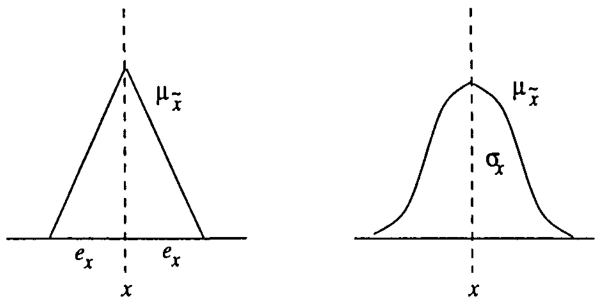


Figure 5.3: A triangular (left) and a Gaussian (right) fuzzy number.

Note that both functions are integrable in $\mathbb{R}$ and their integral is positive. The expressions for these functions are, respectively:

$$\mu_{\tilde{x}}(u) = 1 - min\left(\frac{|x-u|}{e_x}, 1\right), \qquad u \in \mathbb{R} \tag{5.69}$$

$$\mu_{\tilde{x}}(u) = exp\left\{-\frac{1}{2\sigma_x^2}(x-u)^2\right\}, \qquad u \in \mathbb{R} \tag{5.70}$$

The similarity between fuzzy numbers was defined as $s(\tilde{x}, \tilde{y}) = \Pi_{\tilde{x}}(\tilde{y})$ (4.60), with:

$$\Pi_{\tilde{x}}(\tilde{y}) = \sup_{u \in \mathbb{R}} \{min(\mu_{\tilde{x}}(u), \mu_{\tilde{y}}(u))\}$$

Since $\Pi_{\tilde{x}}(\tilde{y}) = \Pi_{\tilde{y}}(\tilde{x})$, for the neuron model in (5.68) we can write:

$$s(\tilde{x}, \tilde{w}^i) = \check{s}\left(\Pi_{\tilde{w}^i}(\tilde{x})\right), \qquad x, w^i \in \mathbb{R} \tag{5.71}$$

hence considering $\tilde{x}$ as the kernel free parameter and $\tilde{w}^i$ as fixed. Since $\tilde{x}$ is constructed exclusively out of a scalar x (and $\tilde{w}^i$ out of a scalar w^i) the above expression can be seen as a function of x and w^i. It needs to be shown that it is also a function of $x - w^i$. Let $\tilde{z} = \tilde{x} - \tilde{w}^i$, defined as:

$$\mu_{\tilde{z}}(u) = 1 - min\left(\frac{|x - w^i - u|}{|e_x - e_{w^i}|}, 1\right), \qquad u \in \mathbb{R} \tag{5.72}$$

obtained from $\mu_{\tilde{x}}, \mu_{\tilde{w}^i}$ thanks to the Extension Principle [Zadeh, 73]. This fuzzy number can be cast as a function of $x - w^i$, as follows. The inner expression is:

$$\frac{|x-w^i-u|}{|e_x-e_{w^i}|}=\frac{|x-w^i-u|}{|kx-kw^i|}=\frac{|x-w^i-u|}{k|x-w^i|}=\frac{1}{k}\left|\frac{x-w^i-u}{x-w^i}\right|=\frac{1}{k}\left|1-\frac{u}{x-w^i}\right|$$

and thus (5.72) can be written:

$$\mu_{\tilde{z}}(u)=1-min\left(\frac{1}{k}\left|1-\frac{u}{x-w^i}\right|,1\right),\qquad u\in\mathbb{R} \tag{5.73}$$

Therefore, the whole expression $\mu_{\tilde{z}}$ is an analytic function of $x-w^i$ in each point. The value yielded by (5.71) is a function of how close $\tilde{z}$ is to the fuzzy set $\tilde{0}$. In fact, it holds that $\Pi_{\tilde{w}^i}(\tilde{x})=\Pi_{\tilde{0}}(\tilde{x}-\tilde{w}^i)$. This situation is depicted in Fig. (5.4).

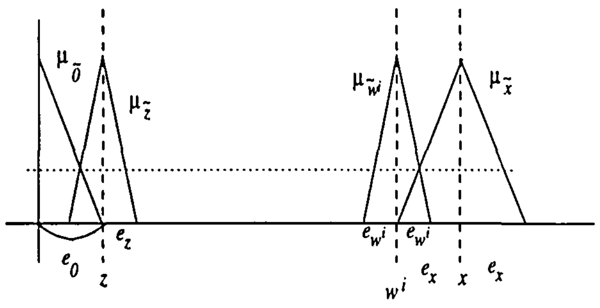


Figure 5.4: An example situation of $\tilde{x}$, $\tilde{w}^i$ and $\tilde{z}=\tilde{x}-\tilde{w}^i$, where $x\geq w^i$, $z=x-w^i, e_z=|e_x-e_{w^i}|$ (5.72). Also shown is the appropriate $\tilde{0}$ fuzzy set. The dotted line is the desired value of similarity $\Pi_{\tilde{0}}(\tilde{x}-\tilde{w}^i)$.

The expression for the $\tilde{0}$ fuzzy number (whose mode is 0) is given by:

$$\mu_{\tilde{0}}(u)=1-min\left(\frac{|u|}{e_0},1\right),\qquad u\in\mathbb{R} \tag{5.74}$$

Some tedious manipulations lead to $e_0=(e_x+e_{w^i})-|e_x-e_{w^i}|$. To be precise, e_0 is *not* properly a function of $x-w^i$, because $(e_x+e_{w^i})$ depends on $x+w_i$ (or, alternatively, on $x-w_i$ and w_i). A solution to overcome this is to set a constant spread, that is, the fuzziness of the constructed fuzzy numbers is taken to be a constant value $e_c>0\in\mathbb{R}$ for the corresponding variable in a given domain (it could be different for different variables). This constant fuzziness may be useful in some practical situations, but certainly is not the most general case. However, by setting a constant fuzziness, $\tilde{z}$ happens to be crisp, $e_0=2e_c$, and the rest of the discussion is not altered. Since both fuzzy numbers ($\tilde{z}$ and $\tilde{0}$) are built out of $x-w^i$, the result of $\Pi_{\tilde{0}}(\tilde{x}-\tilde{w}^i)$ (a scalar) also is. This leads to kernels of the form:

$$K_{\mathbb{F}}(z)=\breve{s}\left(\Pi_{\tilde{0}}(z)\right)=\breve{s}\left(\mu_{\tilde{0}}(z)\right) \tag{5.75}$$

In practice, this requirement can be relaxed by letting the spreads to be variables rather than constants. Specifically, the fuzziness of the weight e_{w^i} can be let to be another free parameter of the model. This is not harmful in theory (we know a theoretical solution still exists) and will certainly add to the practical flexibility.

Part *ii)*. We first show that the kernel $K_{\mathbb{F}}$ in (5.75) is integrable w.r.t. z in $\mathbb{R}$, in a graphical way, in Fig. (5.5). Since z is crisp, we have:

$$\int_{\mathbb{R}} \Pi_{\tilde{0}}(z)dz = \int_{\mathbb{R}} \mu_{\tilde{0}}(z)dz = e_0 = 2e_c > 0 \in \mathbb{R} \tag{5.76}$$

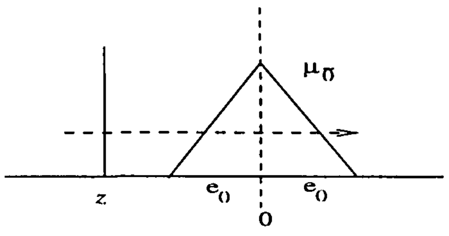


Figure 5.5: Integrability of $\Pi_{\tilde{0}}$.

The composition to an $\check{s} : [0,1] \to [0, \check{s}_{max}]$ keeps integrability since $K_{\mathbb{F}}$ is only defined in the closed interval $[-e_0, e_0]$ (being zero elsewhere) and the $\check{s}$ is, by definition (4.9), continuous and bounded in $[0,1]$.

In case the fuzziness of the weight e_{w^i} is set as a free parameter of the model, the difference $\tilde{z} = \tilde{x} - \tilde{w}^i$ is in general not crisp and we have a kernel of the form

$$K_{\mathbb{F}}(z) = \check{s}\left(\Pi_{\tilde{0}}(\tilde{z})\right) \tag{5.77}$$

We show now that this kernel is still integrable. The discussion is probably simpler by considering instead the function $\Pi_{\tilde{w}^i}(\tilde{x})$ and showing that it is integrable in $\mathbb{R}$ w.r.t. x. This is analogous to the required integral $K_{\mathbb{F}}(z)$ w.r.t. z, since $\Pi_{\tilde{w}^i}(\tilde{x}) = \Pi_{\tilde{0}}(\tilde{x} - \tilde{w}^i) = \Pi_{\tilde{0}}(\tilde{z})$, where $z = x - w^i$.

Again, we show the function $\Pi_{\tilde{w}^i}(\tilde{x})$ to be integrable w.r.t. x in $\mathbb{R}$ in a graphical way, in Fig. (5.6).

For given w_i and e_{w^i}, $\tilde{w}^i$ is build. Let us suppose that $\mu_{\tilde{x}}, \mu_{\tilde{w}^i}$ are of the triangular form (the proof for Gaussians is analogous). Integrating over all $x \in \mathbb{R}$ means building an $\tilde{x}$ (which only depends on x) and calculating $s(\tilde{x}, \tilde{w}^i) = \Pi_{\tilde{w}^i}(\tilde{x})$. From the picture, it is clear that the support of $\Pi_{\tilde{w}^i}$ in $\mathbb{R}$ is $z \in [w^i - (e_x + e_{w^i}), w^i + (e_x + e_{w^i})]$. The limits of this interval correspond to the cases where the bases of the triangles are touching in a single point at level 0, either to the left or to the right.

Hence,

$$\int_{\mathbb{R}} \Pi_{\tilde{w}^i}(\tilde{x})\,dx = \int_{-\infty}^{w^i-(e_x+e_{w^i})} \Pi_{\tilde{w}^i}(\tilde{x})\,dx + \int_{w^i-(e_x+e_{w^i})}^{w^i+(e_x+e_{w^i})} \Pi_{\tilde{w}^i}(\tilde{x})\,dx + \int_{w^i+(e_x+e_{w^i})}^{+\infty} \Pi_{\tilde{w}^i}(\tilde{x})\,dx \tag{5.78}$$

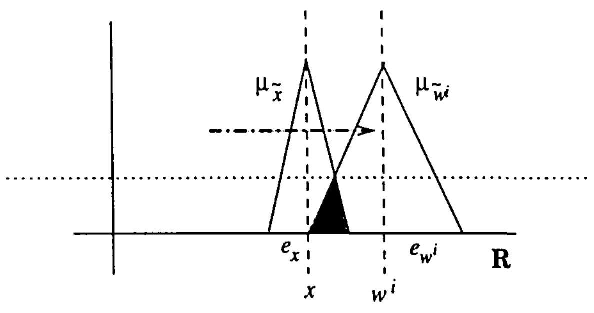


Figure 5.6: Given a fixed $\tilde{w}^i$, obtained from the current w_i and e_{w^i}, integrating means sliding for consecutive $\tilde{x}$ and calculating the similarity $\Pi_{\tilde{w}^i}(\tilde{x})$. The dotted line is the value $\Pi_{\tilde{w}^i}(\tilde{x})$ for an arbitrary $\tilde{x}$.

The first and third integrals on the right-hand side of (5.78) are zero. By thinking graphically, we can imagine the $\tilde{x}$ triangle sliding slowly from left to right across $\tilde{w}^i$ (that is fixed). For all the x points in the above interval each similarity between $\tilde{x}$ and $\tilde{w}^i$ yields a number in $[0,1]$. The net result across $\mathbb{R}$ is another triangular fuzzy number $\mu_{\tilde{s}}$, depicted in Fig. (5.7).

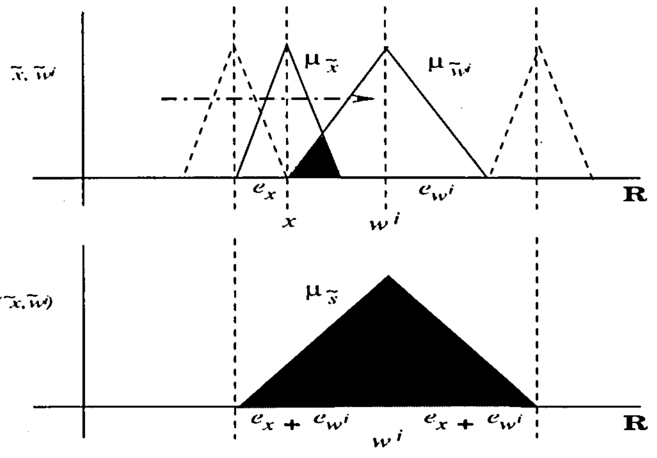


Figure 5.7: The integral (5.78) in graphical form.

Since $\Pi_{\tilde{w}^i}(\tilde{x}) = \mu_{\tilde{s}}(x)$, the above integral can be written:

$$\int_{w^i-(e_x+e_{w^i})}^{w^i+(e_x+e_{w^i})} \mu_{\tilde{s}}(x)\,dx \tag{5.79}$$

where $\mu_{\tilde{s}}$ is a symmetric fuzzy number centered at w_i (like $\tilde{w}^i$) with a spread $e_s = e_x + e_{w^i}$. In any case, this last integral exists, and is clearly positive (to be precise, its value is e_s). Again, the composition to an $\tilde{s}$ keeps integrability, for analogous reasons. This time, however, there is an interesting remark to be done. The addition of a new free parameter (the spread of the weight $e_{w^i} > 0 \in \mathbb{R}$) to the neuron model is equivalent to letting it constant to a value of zero (that is, considering only crisp weights) and using a neuron model which is directly of the required form $K_{\mathbb{F}}\left(\frac{x-w^i}{\sigma}\right)$, without the need of a $\check{s}$ function. In this model, the smoothing factor σ takes the role of the spread of the weight.

To see this, we depart from the kernel (5.77) and manipulate it by setting $e_{w^i} = 0$, considering no spread in the weights. This implies $e_0 = (e_x + e_{w^i}) - |e_x - e_{w^i}| = e_x - |e_x| = 0$ (since $e_x > 0$) and therefore $\tilde{0} = 0$. Thus, the similarity is performed between a general fuzzy number ($\tilde{z}$) and the crisp number zero. In these conditions,

$$K_{\mathbb{F}}(z) = \check{s}\left(\Pi_{\tilde{0}}(\tilde{z})\right) = \check{s}\left(\Pi_0(\tilde{z})\right) = \check{s}\left(\Pi_{\tilde{z}}(0)\right) = \check{s}\left(\mu_{\tilde{z}}(0)\right) \tag{5.80}$$

For clarity, we denote this kernel $K'_{\mathbb{F}}(z)$, where:

$$\mu_{\tilde{z}}(u) = 1 - min\left(\frac{|z-u|}{e_c}, 1\right), \qquad u \in \mathbb{R} \tag{5.81}$$

with $e_c > 0 \in \mathbb{R}$ as before, accounting for the constant fuzziness of x. Then, the construction of a kernel of the required form, given $\sigma > 0 \in \mathbb{R}$, leads to:

$$K'_{\mathbb{F}}\left(\frac{x-w^i}{\sigma}\right) = \check{s}\left(1 - min\left(\frac{\left|\frac{x-w^i}{\sigma}\right| - 0}{e_c}, 1\right)\right) = \check{s}\left(1 - min\left(\frac{|x-w^i|}{\sigma e_c}, 1\right)\right) \tag{5.82}$$

On the one hand, we see that the resulting expression is a function of $\frac{x-w^i}{\sigma}$, with $w^i \in \mathbb{R}, \sigma > 0 \in \mathbb{R}$ as required by Theorem (5.2), where the kernel can be simply written:

$$K'_{\mathbb{F}}(z) = \check{s}\left(1 - min(\frac{|z|}{e_c}, 1)\right)$$

On the other hand, it is clear that σ is taking the role of a variable fuzziness $e'_c = \sigma e_c$ in (5.82), letting it free in the positive part of $\mathbb{R}$, as required. Therefore, the model is self-contained in the number of required free parameters. This justifies the elimination of $\check{s}$, if desired. The obtained model is integrable, with non-null integral equal to e_c (without $\check{s}$) and equal to $2\int_0^1 \check{s}(z)dz$ with a $\check{s}$.

5.4.7 Extension to missing information in Gower's measure

In this last section, the result in (§5.4.4) is extended to incorporate missing information. The inclusion of missing elements in heterogeneous space, as prescribed by Gower's measure (4.71), consists in counting only the non-missing computations and normalizing by their number. We depart from kernels of the form:

$$K(\vec{z}) = \check{s}\left(\frac{1}{n}\sum_{j=1}^{n} K_j(z_j)\right), \qquad \vec{z} \in \mathbb{R}^n \tag{5.83}$$

The extension of K to account for missing information implies the extension of the partial kernels K_j to accept a missing value (herein denoted $\mathcal{X}$) as part of their domain. In other words, the domain of a K_j is now $\hat{\mathcal{R}} = \mathbb{R} \cup \{\mathcal{X}\}$, where $\mathcal{X}$ is an element defined as incomparable w.r.t. any ordering relation in $\mathbb{R}$, for which only equality is defined in $\hat{\mathcal{R}}$. Therefore, K is extended to be defined in $\hat{\mathcal{R}}^n = (\mathbb{R} \cup \{\mathcal{X}\})^n$. Specifically, Gower's prescription to aggregate the partial measures in presence of missing values (4.71) leads to new kernels:

$$\hat{K}(\vec{z}) = \check{s}\left(\frac{\sum_{j=1}^{n} \delta_j \hat{K}_j(z_j)}{\sum_{j=1}^{n} \delta_j}\right), \qquad \vec{z} \in \hat{\mathcal{R}}^n \tag{5.84}$$

where

$$\delta_j = \begin{cases} 1 & \text{if } z_j \neq \mathcal{X} \\ 0 & \text{if } z_j = \mathcal{X} \end{cases} \tag{5.85}$$

Before proceeding, we make a restriction to get through with the discussion: we only consider missing values in the *inputs*, that is, in the $\vec{x}$ and hence, $\vec{w}^i \in \mathbb{R}^n$. For the kernels, this ensures that at least one z_j is non-missing, and hence the denominator in (5.84) is non-null. It could only be null if all the components of $\vec{x}$ were missing, a situation discarded by hypothesis.

A simple way to achieve the extension is to place this "alien" element in a new dimension, that is, to work in $\mathbb{R}^2$ for each partial kernel. We depart from the hypothesis that the K_j in (5.83) were valid kernels, always positive, with $[K_j]_1 < \infty$ and $[K_j]_1 > 0, 1 \leq j \leq n$. In these conditions, let:

$$K'_j(z,m) = \begin{cases} K_j(z) & \text{if } |m| \leq \epsilon \\ 0 & \text{if } (z,m) = (0,1) \\ 0 & \text{otherwise} \end{cases} \tag{5.86}$$

with $0 < \epsilon < 1$. In particular, the point $(0,1)$ in $\mathbb{R}^2$ is taken to represent the missing element $\mathcal{X} \in \hat{\mathcal{R}}$. These functions are depicted in Fig. (5.8).

The K'_j are defined in $(\mathbb{R} \times [-\epsilon, +\epsilon]) \cup \{(0,1)\} \subset \mathbb{R}^2$. Once we have the missing element represented in each partial kernel, we define the whole kernel K' as:

$$K'(\vec{z}) = \check{s}\left(\frac{1}{n'} \sum_{z_j \in \vec{z}|\neg\mathcal{X}} K'_j(z_j, 0) + \frac{n'-1}{n'} \sum_{z_j \in \vec{z}|\mathcal{X}} K'_j(0,1)\right), \qquad \vec{z} \in \hat{\mathcal{R}}^n \tag{5.87}$$

where $\vec{z}|\neg\mathcal{X} = \{z_i : 1 \leq i \leq n : z_i \in \mathbb{R}\}$, $\vec{z}|\mathcal{X} = \{z_i : 1 \leq i \leq n : z_i = \mathcal{X}\}$ and $n' = \#(\vec{z}|\neg\mathcal{X})$. This kernel (5.87) is so-defined for clarity: note that the second summation is null. The task is then twofold:

1. To show that the obtained kernel K' behaves analogously to (5.84).
2. To show that the obtained kernel K' is integrable.

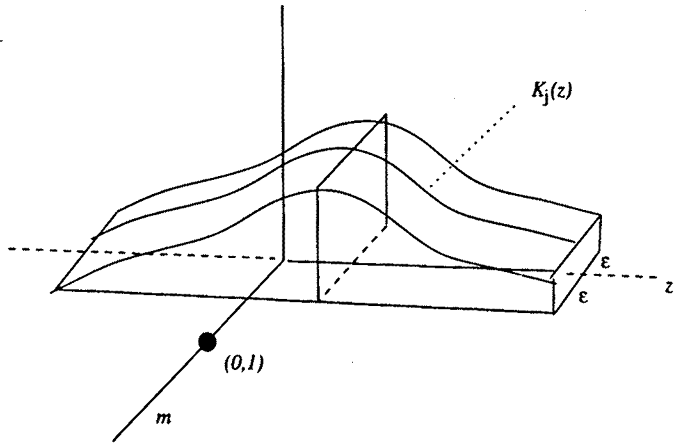


Figure 5.8: The functions (5.86).

Proposition 5.6 *The function K' is equivalent to the Gower-based aggregation of partial similarities, in presence of missing values, defined in (5.84).*

Proof. We prove that $K'(\vec{z}) = \hat{K}(\vec{z}),\ \forall \vec{z} \in \hat{\mathcal{R}}^n$. We first note that $\sum_{j=1}^{n} \delta_j = \#\,(\vec{z}|\neg\mathcal{X})$.

(i) If $\vec{z} \in \mathbb{R}^n$, that is, if there are no missing elements in $\vec{z}$, we have $\hat{K}(\vec{z}) = K(\vec{z})$, since (5.83) is equivalent to (5.84). Moreover,

$$\begin{aligned}
K'(\vec{z}) &= \breve{s}\left(\tfrac{1}{n'} \textstyle\sum_{z_j \in \vec{z}|\neg\mathcal{X}} K'_j(z_j, 0)\right) \\
&\qquad (n' = n, \vec{z} = \vec{z}|\neg\mathcal{X}) \\
&= \breve{s}\left(\tfrac{1}{n} \textstyle\sum_{j=1}^{n} K'_j(z_j, 0)\right) \\
&\qquad (\text{def. of } K'_j) \\
&= \breve{s}\left(\tfrac{1}{n} \textstyle\sum_{j=1}^{n} K_j(z_j)\right) = K(\vec{z})
\end{aligned}$$

(ii) In presence of missing elements in $\vec{z}$, we have $0 < n' < n$ and:

$$\begin{aligned}
\hat{K}(\vec{z}) &= \breve{s}\left(\frac{\sum_{j=1}^{n} \delta_j \hat{K}_j(z_j)}{\sum_{j=1}^{n} \delta_j}\right) \\
&\qquad (\text{def. of } \delta_j) \\
&= \breve{s}\left(\tfrac{1}{n'} \textstyle\sum_{z_j \in \vec{z}|\neg\mathcal{X}} K_j(z_j)\right) \\
&\qquad (\text{def. of } K'_j;\, K_j \text{ always positive}) \\
&= \breve{s}\left(\tfrac{1}{n'} \textstyle\sum_{z_j \in \vec{z}|\neg\mathcal{X}} K'_j(z_j, 0)\right) = K'(\vec{z})
\end{aligned}$$

Proposition 5.7 *The partial kernels K'_j are integrable in $\mathbb{R}^2$.*

Proof. Let $M_\epsilon = \{(z, m) \in \mathbb{R}^2 \mid |m| \leq \epsilon\}$.

$$\int\int_{\mathbb{R}^2} K'_j(z,m)\ dzdm = \int\int_{M_\epsilon} K'_j(z,m)\ dzdm + \int\int_{\mathbb{R}^2\setminus M_\epsilon} K'_j(z,m)\ dzdm =$$

$$\int_{\mathbb{R}} 2\epsilon K_j(z)\ dz + 0 = 2\epsilon \int_{\mathbb{R}} K_j(z)\ dz$$

Since $\int_{\mathbb{R}} K_j(z)\ dz < \infty$ and is non-null, so is the new obtained integral, since $\epsilon > 0$.

Intuitively speaking, we have shown that the required extended kernels $\hat{K}(\vec{z}), \vec{z} \in \hat{\mathcal{R}}^n$ (5.84) can be simulated by defining

$$\hat{K}_j(z_j) = \begin{cases} K'_j(0,1) & \text{if } z_j = \mathcal{X} \\ K'_j(z_j,0) & \text{if } z_j \neq \mathcal{X} \end{cases} \qquad z_j \in \hat{\mathcal{R}} \tag{5.88}$$

An alternative way of dealing with the extension would have implied extending the topology in $\mathbb{R}$ to a topology in $\hat{\mathcal{R}}$, thereby extending the concept of integrability to $\hat{\mathcal{R}}$. This remains as an avenue of future work.

5.5 Conclusions

We have shown how several classes of similarity-based neuron models share the universal approximation property. Some assumptions and technicalities have been worked out in order to get through with the proofs and, wherever possible, to further illustrate the studied models.

While the Propositions are general enough to cover most of the neuron models developed and used in this Thesis, including many previously existent in the literature, there clearly is a potential for more generic proofs, or for the enhancement of the proposed ones. In this vein, normalized distance-based measures, or more general measures based on scalar product are specific avenues of future study.

In addition, the incorporation of nominal, ordinal and linguistic information has not been considered. The first two types, being discrete, pose a problem which transcends the scope of this work. There is also the integration of discrete and continuous information into general approximation schemes. To this end, new theorems for neural networks, concerning approximation in discrete or hybrid spaces should be needed. For linguistic variables, a clear line of thinking is to consider them as generalizations of fuzzy numbers.

The approximation ability does not give clues about the effective *learnability*, that is, how to build the approximation. Hence, a failure to reach a desired performance can be due to a handful of combined causes, as an inadequate architecture, the presence of noise in the samples, an insufficient number or lack of representativeness thereof, or an uneffective learning algorithm. In addition, most of the universal approximation results do not take into account the limited precision with which computers work with real numbers. In this sense, it has been pointed out that current standards are not enough to meet the requirements addressed in theory [Wray and Green, 95].

Besides, there may exist models very useful or however effective in practical applications, but such that, for one or other reason, a proof for their UA property is difficult or cannot be obtained with the current definition. Generally speaking, the fulfillment of the property does not convey a clear-cut practical concern, and viceversa. All this should make us to revise current standpoints towards more realistic settings, in which the number of samples and hidden units, noise, computer precision, roundoff errors and related concerns are tackled. Otherwise, the proof of the property, will remain a an issue far from everyday practice.