

MS COMPREHENSIVE EXAMINATION
Spring 1994
23
Torsten Felzer
Artificial Neural Networks

University of Colorado at BOULDER
Department of Computer Science

March 4 – April 4, 1994

PDF version by TORSTEN FELZER (Dec. 10, 2002)

Abstract

“Artificial neural networks” is a frequently used expression in everyday language, particularly among computer scientists. When you ask some people who are working in this area what they are really doing, you sometimes get the short answer: “We are simulating the brain”.

The first time I got this response, I found it rather offending. What are they doing? Simulating the brain!? Do they *know* what the brain does? Who do they think they are? God?!?

After having learned more about the field, after having got more familiar with the topic and after having started getting a more detailed view of the whole thing, I realized that not only were these the wrong questions to ask, but also that this simple answer to the question what they are doing was much too imprecise.

In this paper, I intend to demonstrate that there is a lot more to artificial neural nets than the simple response stated above. I want to show what neural nets are, what can be done with them, *how* this is done, and what they are good for. I talk about the development of artificial neural networks over the past 50 years and say some words on practical applicability. I will also try to summarize the current status of the research in the field and give some examples of objectives for the future.

Contents

1	Introduction	2
2	Artificial Neural Networks – Description of the Development	3
2.1	Origins and Motivation	4
2.2	Basic Structure	4
2.3	Learning and Training	7
2.3.1	Supervised	7
2.3.2	Unsupervised	10
2.4	Backpropagation	11
2.5	Historical Overview	13
2.6	Current Status	14
2.7	Relations to different areas of Computer Science	15
2.7.1	Artificial Intelligence	15
2.7.2	Theory of Algorithms	15
2.7.3	Programming Languages	16
2.7.4	Numerical Computation	17
2.7.5	Computer Architecture	17
3	Applicability to Practical Problems	18
3.1	Pattern Recognition	18
3.2	“Predicting the Future”	19
3.3	Composing Music	20
3.4	Solving Systems of Linear Equations	21
4	Desirable Future Directions	22
5	References	23

1 Introduction

The human brain is organised as a huge network of numerous very simple computational units, called *neurons*. During the past half century, the study of *artificial neural networks*, modeled after those “natural prototypes” has become more and more popular. Particularly *after* the advent of VLSI technology, as computers became more powerful and running simulations of large artificial networks therefore became easier and faster, the field of *connectionism* (which is another name for the area of artificial neural networks) began to gain importance. The past decade has been the probably most significant one in the development of connectionist models.

In this paper, I want to talk about this development. I chose this topic because of mainly two reasons. First of all, it is at the heart of present-day research, it represents a relatively young and progressive field, which makes it extremely interesting. Second, the connectionist approach is directly applicable to problems of the real world, a very impressive property for such a recent development in computer science.

Although there are a few experts in neural networks who are trying to “play God”, the main goal of the research on artificial neural networks is *not* to reconstruct the human brain in order to “create” a new, perfect, intelligent, artificial being. This is a very common misunderstanding among those who are not familiar with the subject. Many of them think *Artificial Intelligence* (AI) is nothing else but the attempt to construct artifacts that are more intelligent than humans. But the true objectives for the research in the field of Artificial Intelligence in general and on artificial neural networks in particular are completely different.

On the one hand, one tries to gain new insights and to get new ideas concerning problems related to psychology, linguistics, and (last but not least) biology by modelling the human brain to some extent. Knowledge (or theories) about what methods the brain uses to implement certain tasks might help humans (not artifacts) to improve their way of thinking.

On the other hand, artificial neural networks provide for a new level of computation, entirely distinct to the conventional von Neumann approach. With this new paradigm, one tries to apply the (so far very limited) knowledge about what is going on in the brain to take advantage of the brain’s enormous computational power.

Artificial neural networks might make it possible to solve interesting problems that could not be solved without them (at least not with an acceptable overhead). This refers to problems that are extremely hard to solve in the usual way with

conventional computer programs, but very easy for the human brain, for instance tasks related to pattern or speech recognition. Furthermore, artificial neural networks allow the development of a totally new kind of algorithms, algorithms with a highly parallel structure.

I start the actual paper with a description of the historical and theoretical foundations in the following section, including a discussion of *training* an artificial neural network and an explanation of the basic concepts of the biological equivalent. Besides, I try to give a fair summary of the current status of the research on artificial neural networks, which is not very easy, since there are efforts of numerous groups of scientists to make progress in the field, and so the current status is subject to ongoing rapid changes. I then want to relate the development of artificial neural networks to different areas of computer science, such as Artificial Intelligence and Computer Architecture. By this, I want to provide for some “broader understanding” of the topic, and it should become clear that artificial neural networks represent *more* than a concept that only people in AI need to know about.

Section 3 mentions some of the applications of artificial neural networks to real world problems. At first sight, one might get the impression that artificial neural networks are nothing else but another “academic invention” that is merely of theoretical interest. In this third section, I intend to demonstrate that this is definitely not the case by presenting a bunch of interesting, practical problems that can be solved with the help of the connectionist approach.

In section 4, I point to some desirable future directions. I want to talk about breakthroughs that researchers wish for, which would lead to large steps toward the ultimate goals mentioned above.

Section 5 contains a list of all books and papers I used when writing this paper. The list shall provide the reader with information about where to find additional treatment of the topic. I cannot treat a subject like artificial neural networks in complete detail in such a paper, so I just *have to* refer the interested reader to the literature.

2 Artificial Neural Networks

This section is entitled “*Artificial Neural Networks*”. However, I begin with a description of the “biological prototype” in the first subsection, the gigantic neural net in the human brain.

2.1 Origins and Motivation

Computers have been developed chiefly to facilitate the lives of their users. Those artificial systems perform tasks that humans therefore do not have to do by themselves any more, e.g. complicated mathematical calculations. The problem is that some tasks which are extremely easy for a human, e.g. pattern recognition, are extremely difficult if not almost impossible to program on a conventional computer.

The logical question that arose from this fact was: “What does the brain do?” As far as current knowledge goes, the brain consists of about 5 to 10 billion very special cells, called *neurons*, that are interconnected in a gigantic net. Neurons receive electrochemical signals from other neurons, some of which exciting the overall activation of the cell, others inhibiting it. The cell “sums up” all those incoming signals and determines whether or not this sum exceeds a certain threshold. If yes, the neuron “fires”, which means it transmits the same kind of electrochemical signal to other neurons. In other words, the neurons are extremely simple computational units.

The overwhelming power of the human brain originates in the huge interconnection network. The excitatory or inhibitory character of a connection (in the context of biological neural networks, and sometimes even for artificial ones, the connections are called *synapses*) can change over time, the network *learns*. The brain acts like a very complicated associative memory, where the activation of certain neurons results in the firing and activation of certain other neurons. This works so well that some very impressive tasks, such as, for example, recognizing a familiar face, can be performed in a surprisingly short amount of time.

The next natural question which arose after that was: “If the brain works so beautifully well, why can’t we construct computers that do the same?” One wanted to mimic the brain’s behaviour in order to build computers that are capable of performing tasks that were not “covered” by conventional computers. This was one major motivation for the development of artificial neural networks.

2.2 Basic Structure

Like in the natural model, artificial neural networks also consist of a number of simple computational units (also called neurons or sometimes also *nodes*), connected with each other. Associated with each connection is a so-called *weight* which corresponds to a synapse in the biological model. Those weights might be positive (excitatory) or negative (inhibitory).

Except for the so-called *input units* (whose values are set to the input to the

network), each of these artificial neurons has several connections feeding into it. These connections carry output values of “earlier” neurons. The unit multiplies each of its input values with the corresponding weight and sums the resulting products. This sum is then mapped (by a function f) to an output value which serves as input to “later” neurons (except for the so-called *output units* whose output values represent the output of the entire network).

Theoretically spoken, if i_1, i_2, \dots, i_n are the inputs of unit j , and $w_{j1}, w_{j2}, w_{j3}, \dots, w_{jn}$ are the weights associated with these inputs, then the output o_j of this unit is determined by

$$o_j = f \left(\sum_{k=1}^n w_{jk} \cdot i_k \right),$$

where f is the so-called *activation function*.

The outputs of artificial neurons are usually restricted to the range between 0 and 1, and the activation function has to assure this property, so in this case, f can be the sigmoidal “squashing” function (depicted in figure 1) or a thresholded piecewise linear function. In some applications, f is just the identity function, in which case the units are called *linear* and its output is just the weighted sum of its inputs. Binary units have only two discrete possible output values, 0 or 1. The activation function is in this case a step function, like the signum function.

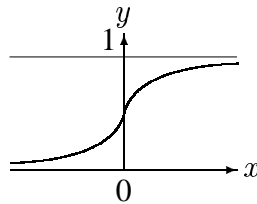


Figure 1: The sigmoidal “squashing” function

The nodes in an artificial neural network are usually grouped in *layers*, each consisting of one or more artificial neurons. There exist several types of topologies that determine how the interconnection networks look like. Common to all those network topologies is that there is one *input layer* and one *output layer*. The units of the input layer do not perform any computation, the values at their outputs are just set equal to their input values.

The simplest type of artificial neural network is a so-called *perceptron* which consists merely of an input layer, an output layer, and weighted connections in

between¹ (see for instance [25]). The perceptron is also the oldest type of artificial neural network. It can solve a bunch of interesting problems, but unfortunately, it is *not* applicable to many other problems. For many tasks, it is necessary that there are several (at least one) layers, called *hidden layers*, between the inputs and the outputs.

The probably most popular and most frequently used type of topology is represented by so-called *feed-forward networks*, where the units of each layer are connected only to units of “later” layers. In *recurrent networks*, there are also *feedback connections* to units in “earlier” layers or even *within* one layer. Figure 2 depicts a typical representative for each of the network types mentioned above.

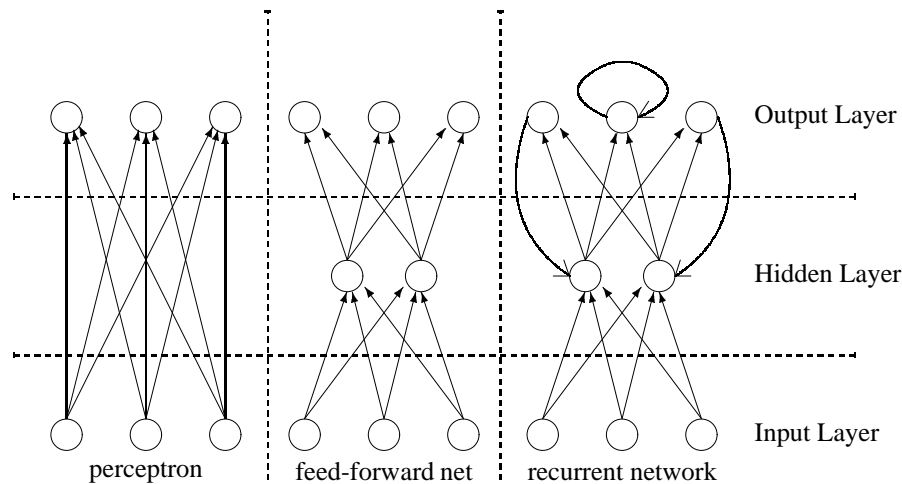


Figure 2: Different network topologies (all connections are weighted)

There are some artificial models that differ from the network types presented in this subsection, such as the Hopfield nets² or Boltzmann machines³. For consistency reasons, in order to avoid confusion, I decided not to talk about them in detail. For further information, please see, for example, [11], [12], or [20].

In the remainder of this paper, biological neural networks are of minor interest. Therefore, I can get rid of the adjective “artificial”. Whenever I use the notion “neural network” from this point on, I mean artificial ones. When I want to specifically mention the natural model, I will explicitly say so.

¹Sometimes such a network is called *single-layer* network. The input layer (which does *not* consist of “complete” neurons, because input units do not have an activation function) is in this case not counted.

²graph-like networks of binary neurons with symmetric connections

³networks where each artificial neuron updates its state according to a *stochastic* decision rule

2.3 Learning and Training

One way to look at a neural net is to see it as a special kind of computer memory. Unlike the structure of conventional memory, there are no such things as memory cells that contain a bunch of binary bits and that can be accessed via some fixed address. “Neural memories” (i.e. memories realized by means of a neural net) are different in that they associate input patterns with certain output patterns. Thus, neural nets are often referred to as *associative memories*. In this sense, the inputs stand for an “address” and the corresponding outputs stand for the information stored at this address.

To “store” information, one has to adjust the weights in a neural net. The weights determine, from layer to layer, which units of a “later” layer are to be activated depending on the activation of units in “earlier” layers. So in total, the weights are responsible for the mapping from input to output patterns. Therefore, adjusting the weights, *adapting* them to react to certain input patterns appropriately, needs to be done before one can try to use the net to perform anything useful.

Therefore, the usage of a neural net is in most cases divided into two parts. First, a so-called *training* or *learning phase*, and second, the actual application phase. The training phase serves as an initialization step. The information to be stored gets coded into the weights. The adaptation of the weights is done by confronting the net over and over again with the patterns to be stored. In this sense, the network *learns* in a trial-and-error fashion by adjusting its weights.

Many neural networks continue adapting their weights during the application phase based on recently “learned” concepts in order to improve their performance. But in general, during the application phase, neural nets make use of the information stored in the weights in some way or another.

I now want to talk about the two basic methods used in the training phase: *supervised* and *unsupervised learning*.

2.3.1 Supervised

There are two kinds of supervised learning. The most popular and most widely used technique for training a neural net is called *learning by error correction*. The other branch of supervised learning is entitled *reinforcement learning*.

Learning by Error Correction If a network is trained using the method of *error correction*, the net is considered to learn a certain mapping from inputs to

outputs. The net is presented with a set of input patterns, which are entered at the input layer and “shuffled” through the net all the way up to the output layer. A teacher (e.g. a human operator) compares the resulting output pattern with a desired or target output pattern. According to the current error (the deviation of the two patterns or *vectors* from each other), the weights in the neural network are adapted using one of several learning algorithms in order to reduce (or *correct*) the error.

One of the first learning algorithms is given by the *delta rule*⁴ (see [18]):

$$\begin{aligned}\Delta w_{jk} &= \eta(t_j - o_j)i_k, \\ w_{jk}^{new} &= w_{jk}^{old} + \Delta w_{jk}.\end{aligned}$$

Here, w_{jk} is the synaptic weight from input neuron k to output neuron j , t_j is the *target output* of neuron j , o_j its actual output, i_k the value of input k , and η a *positive* factor of proportionality called *learning rate*. The rule tells us that we get the new value of a weight by adding a certain, weight-specific “delta”⁵.

In other words, the change of a weight feeding into an output unit is the bigger the more the actual output deviates from the target output. Besides it is proportionate to the value coming from input unit k . This both makes sense. If the difference between actual and target output is zero, no changes need to be made to this weight. Secondly, the bigger the “flow” through this connection is (the bigger i_k), the bigger is the contribution of this weight to the error in o_j . That means for example, if i_k is zero, there is no justification for changing the weight w_{jk} , because this weight cannot be responsible for the error in o_j .

This rule works well for single-layer networks (i.e. perceptrons), but cannot be applied to general feed-forward nets, since there is no information concerning “targets” for hidden units, so the delta rule does not tell us how to change the weights feeding into these units. On the other hand, we cannot limit ourselves to networks *without* hidden units, since it has been proved that they are necessary for some tasks (see [4, 18, 23]).

To build general networks that are able to implement arbitrary types of mappings from input patterns to output patterns, there is no way to get around hidden units. However, the human operator does not know in advance how the net will implement the task at hand. Weights feeding into hidden layers correspond to internal representations which are important for the net, but not for the environment.

⁴sometimes also called *Widrow-Hoff rule*

⁵This, by the way, provided the name for this adaptation rule.

Rumelhart and his research group introduced in 1986 a generalization of the delta rule that works also for hidden units [18]. Their algorithm is called *backpropagation*, and it represents *the* state-of-the-art learning algorithm for multi-layer feed-forward nets. As far as I know, there is almost no recent paper on artificial neural networks that does *not* mention backpropagation in some way or another. That is why I decided to dedicate the following subsection entirely to this algorithm.

Backpropagation does not only work for feed-forward nets, it can also be applied to recurrent networks, as Rumelhart shows in his original paper [18].

Genetic Algorithms In addition to the method of adapting the weights according to a certain learning rule, a more evolutionary approach can be used. *Genetic Algorithms* involve *creating* a “generation” of “individuals” (e.g. a set of neural networks with different topologies and/or different weights) and “producing” new generations which are hopefully “better” or “fitter”. This is done by having the individuals *mutate* their features (e.g. change some of the weights in the network) or *reproduce* themselves and by allowing *crossovers* between two individuals, all this according to stochastic rules.

It is then determined which individuals of the new generation may survive and which of them have to die by means of evaluating a *fitness function*, which measures the quality of an individual (e.g. the conformity of the actual output of a neural net to the desired output)⁶. The whole process is then repeated generation by generation (and always the “fittest” are most likely to survive) until finally an “optimal” individual is found⁷. This “trial-and-error” approach imitates the “natural” evolution process to some extent.

Evaluating the fitness function is somewhat similar to calculating the deviation between actual and desired outputs, so *Genetic Algorithms* are related to *learning by error correction* in some sense. Genetic Algorithms represent a very interesting subject, but unfortunately, it would exceed the range of this paper, if I discussed it in full detail. Hence, I have to refer the reader to the literature (e.g. [12, 20]).

Reinforcement Learning In *reinforcement learning*, there is no teacher telling the net how good its output is and how it has to be corrected. The only exterior feedback to the net is a signal given by a critic, telling if the current output is good

⁶again, together with a stochastic rule

⁷which is, of course, not always possible

or bad, right or wrong, desired or not. That means the net is only provided with a scalar information, a single bit saying yes or no.

If the signal from the critic indicates that the current output is incorrect, the net does therefore *not* know *what* is wrong or *how far* its answer deviates from the desired one. The solution to this problem involves generating a teacher's signal from the critic's response and then using methods similar to those described above when talking about learning by error correction [25].

The major drawback of the reinforcement learning paradigm is that it is somewhat unrealistic in the sense that critics often provide *more* than just a scalar yes/no information (see [25]). The situation of a child trying to learn to ride a bike, for example, does definitely *not* belong to error correction learning, since the child is not told in detail which muscles to use or which movements to make. Rather, it is part of reinforcement learning, since the child notices what happens if certain movements are made in a certain order. This exterior information can be interpreted as the signal of a critic, and it is not at all scalar. The child not only learns whether or not it does the right things, it also learns *how* well it does.

2.3.2 Unsupervised

When there is no signal whatsoever providing the net with information about the quality of the current output, neither a teacher indicating a desired output nor a critic deciding whether or not the net works correctly, the learning is said to be *unsupervised*.

In this case, learning no longer means memorizing or storing input/output mappings, but discovering regularities in the input data. Thus, unsupervised learning cannot take place without those regularities. Here, the net can be really said to “program itself”, it “acts” completely independently.

Some researchers think that unsupervised learning is the *only* biologically sound learning paradigm (see [23]), but on the other hand, supervised techniques are in widespread use and yield excellent results concerning technological problems.

Finally, I want to add that one approach to apply unsupervised learning involves, like in reinforcement learning, generating a target output in a certain way and thus providing the net with a teacher signal [25]. Then the adaptation of weights can again be done using “supervised” procedures like backpropagation.

2.4 Backpropagation

The most important and most popular learning algorithm was introduced by Rumelhart and his colleagues in 1986 [18, 19]. The algorithm works with a generalization of the delta rule to multi-layer feed-forward networks, that means in particular to networks with hidden units. The algorithm is called *backpropagation*, and it is based on the mathematical method of *gradient descent*. Rumelhart begins his original paper by presenting a derivation of the delta rule that shows that the delta rule also implements gradient descent.

The idea is to define an error function

$$E = \frac{1}{2} \sum_j (t_j - o_j)^2,$$

where j varies over all output units *and* all input/output patterns. This function is assumed to be a function of the weights, and the ultimate goal is to minimize this function (i.e. minimize the error) by adjusting the weights. There is no formula for the error function, the only known pieces of information are the current point in weight space and the current output, or its deviation from the desired output, respectively.

The great breakthrough now was to show that it is somehow possible to calculate the gradient of the error function, i.e. the partial derivatives of the error function with respect to each of the weights. The derivation of this result involves the one-dimensional and higher-dimensional chain rules, and I do not want to go into details (see [18]). What I only want to say is that in a backpropagation net, one first performs a forward pass, shuffling the input data through the network, and then a backward pass, computing, with the help of a recursive formula⁸, certain *error signals*, which are then used to calculate the partial derivatives. The backward pass with the error propagation provided the name of the algorithm.

To implement gradient descent, one has to move the weight vector in the direction of the steepest descent which is the direction of the negative gradient. And this is exactly the adaptation rule for backpropagation. The size of the step in this direction is determined by the learning rate which serves (like in the delta rule) as a factor of proportionality.

Moving the weight vector in the direction of the negative gradient means moving it towards a (local) minimum of the error function. However, to be mathematically precise, the step size should be infinitely small, but this would take infinitely

⁸beginning at the output units and then layer by layer down to the first hidden layer above the input units

long. So, backpropagation uses a bigger step size, thus only approximating gradient descent and risking to “jump over” the minimum in one step. To illustrate this, let me assume the one-dimensional case, where E is a function of only *one* independent variable w . Suppose, for simplicity, that E is the parabola depicted in figure 3.

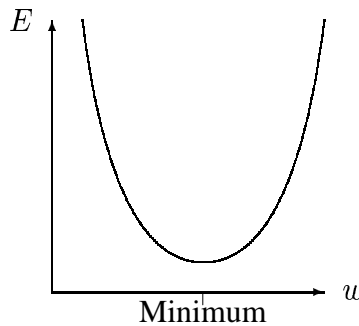


Figure 3: A one-dimensional error function

This special error function has a single local and global minimum⁹. If the current value for w is too far left, the derivative $\frac{dE}{dw}$ is negative, and moving in the direction of the negative derivative means really marching right, i.e. adding a positive value, which takes w closer to the minimum (if the step size is small enough, so that w does not jump over the minimum). If w is too big, it is the same: moving towards the negative derivative means marching towards the minimum.

This adaptation method is applied iteratively, again and again for the input/output patterns to be learned¹⁰, until the deviation of the actual output from the desired output is satisfyingly small, or until the weight vector converges to a local minimum. Although it may take very long until the process converges or until the “optimal” learning rate is found, and despite the problem with the local minima, the backpropagation algorithm often yields very good results, particularly since it can be applied very easily (and for all weights of one layer in parallel).

There are numerous learning algorithms that are based on backpropagation. Many of them just try to eliminate disadvantages, like for example *adaptive backpropagation* [21], where each weight has its own individual learning rate, and these learning rates are adapted during learning¹¹. Others seem to point in a to-

⁹That this is not always so, that error functions in fact often *do* have multiple local minima, is actually a big problem for backpropagation.

¹⁰starting with a randomly generated initial weight vector

¹¹This shall eliminate the sometimes frustrating search for the right learning rate.

tally different direction, like the *scaled conjugate gradient algorithm*¹² [13] or the *cascade-correlation algorithm*¹³ [3, 27]. But I claim that also these new developments would not have been possible without the invention of and inspiration by the backpropagation algorithm.

2.5 Historical Overview

One of the first points in history associated with the development of artificial neural networks is the year 1943, where McCulloch and Pitts introduced their *M-P neuron* (see [23]). These neurons receive a number of inputs and produce one single binary output depending on whether or not a certain threshold is reached or exceeded by the weighted sum of the inputs. The weights can only take on one of two values: +1 or -1.

An interesting application of this model is to use one single neuron as a perceptron to implement logic gates. For example, if there are m inputs, all of the weights are +1, and the threshold is set to m , then the output turns out to be the AND of the inputs. If the threshold is set to 1 with the same setting as far as weights are concerned, the output returns the logical OR of the inputs. Finally, if there is only one (binary) input whose weight is set to -1, and the threshold is set to 0, then the output represents the NOT of the input.

Hebb postulated in 1949 that an important property of the strengths of the connections in the biological neural network of the human brain are changing in time as the organism learns. Hebb's proposal triggered a lot of research effort in the field of adaptive neural networks, and a preliminary climax was reached when, in 1958, Rosenblatt showed how to train a network consisting of M-P neurons. In 1960, Widrow and Hoff proposed a variant of the perceptron: the *adaline* (an acronym for ADaptive LINear Element). They also introduced the now famous delta rule, which I already mentioned in the section about supervised learning.

Let me return to the idea of implementing logical functions with neural networks. Minsky and Papert proved in 1969 that it is not possible for elementary perceptrons to learn the XOR (exclusive or) function. They showed that one single M-P neuron is not enough and that hidden units are necessary to implement really *all* kinds of mappings. So the focus in the neural network community changed from elementary perceptrons to multi-layer feed-forward nets.

¹²which is based on a different mathematical optimization technique, involving second order information, i.e. the second derivative of the error function with respect to the weights

¹³which tries to find an optimal network by also adapting the *topology* of the net

The problem with multi-layer networks is that there are no “target values” for hidden units. Therefore, the delta rule, which adjusts the weights feeding into a neuron proportionately to the deviation of its output from the target output, is not applicable. Thus, it was extremely difficult to train these networks for a relatively long period of time, because one simply was not sure how. In 1986, as already mentioned several times, a group around Rumelhart introduced a learning algorithm that generalizes the delta rule to general feed-forward networks [18]. This algorithm, which is called *backpropagation* (for a description see the section on this learning method), has been since then the standard technique for training neural nets.

2.6 Current Status

It is kind of difficult to summarize the current status of the research in the area of artificial neural networks, because this topic is so new and so popular that every month hundreds of papers are published, each of which is dealing, so it seems, with yet another small subtopic. For this reason, such a summary always *has* to be incomplete, at least in a paper like this. However, I will try to point out some major features that characterize the current status in the field.

One important aspect in this context is that the artificial model is not *too* near to the biological one. For example, it is true that backpropagation yields nice results, but there is no indication at all that something like the propagation of errors is going on in the brain, and I claim that it is evident that the learning in the brain does not work with applying the higher-dimensional chain rule – that is just too complicated.

Moreover, the structure of artificial neural networks is only marginally similar to that of the cerebral net. The size and the complexity of the connections, for instance, does not even reach the gigantic dimensions of the brain. It is furthermore common belief that the neurons and the synapses in the brain are located in 3-dimensional space whereas the units and the weights in an artificial net both have no spatial location. Finally, all the signals in an artificial net are of the same unique type, whereas the brain is believed to use several different signal types (see [22]).

This gap between the two models mainly comes from the fact that the emphasis concerning artificial models was thus far on computational power and not on biological fidelity [8]. With all this in mind, it is not really astonishing that even experts in the field are not 100% convinced that the connectionist approach will ever be adequate for modeling neural computations (in the biological sense) [22].

Nevertheless, connectionist models are becoming more and more important, more and more popular. There is so much research going on in the field that it seems rather unlikely that the topic will *not* grow in the future, that there will *not* be a lot of progress, and that the connectionist approach will *not* provide valuable information concerning the study of the human mind.

Application of the approach is widespread, and there are already a great deal of tasks that seemingly could not be solved, if it were not for artificial neural networks. The perspective that, one day, general purpose computers could be built, implementing a huge neural net that would program itself, and help the human operator of the computer with a lot of tasks, e.g. reading aloud from a book or receiving (and understanding!) spoken commands, makes this field to a major research subject of immense interest.

2.7 Relations to different areas of Computer Science

One very nice property of connectionism is that it has relations to many very different areas of computer science. In the following subsections, I want to shed a little bit of light on these relationships.

2.7.1 Artificial Intelligence

AI is the “home” of connectionism. The area of artificial neural networks developed as a branch in Artificial Intelligence [26].

Meanwhile, it has gained so much interest, importance, and maturity that AI can even be divided into connectionist and non-connectionist Artificial Intelligence.

2.7.2 Theory of Algorithms

There are two very different aspects, actually two different *levels* characterizing the relationship between neural nets and the theory of algorithms.

On the one hand (the *higher* level), neural nets rely on efficient *training algorithms*, and since the introduction of backpropagation, dozens of learning techniques have been proposed [1, 2, 3, 10, 13, 15, 16, 17, 21]. The algorithms developed thus far have mostly been oriented mathematically, so designing a biologically sound and at the same time computationally powerful method seems to be a major objective for the future.

On the other hand (the *lower* level), weight adjustments done by the net during learning sort of imply algorithms for the task that was learned¹⁴. For example, the internal representations (in the weights) for the symmetry problem, the problem of classifying input vectors as to whether or not they are symmetric about their center, turn out to be very easily interpretable in terms of an algorithmic, non-connectionist approach (again see [18]).

In spite of this phenomenon, connectionism differs very much from the algorithmic way¹⁵ of solving problems in that one does not have to *tell* the net *how* to implement the solution of a problem, the net is only told *what* it is supposed to learn, a certain set of input/output patterns. The way this information gets encoded into the weights is entirely up to the net itself and is of almost no interest for the human operator (which is very convenient).

It is thus even possible to solve problems for which algorithms do not exist at all. For example, one does not know how to solve the problem of speech recognition algorithmically [9]. With the help of a connectionist model, one simply needs to specify a mapping from inputs to outputs by giving several training examples – the rest is done by the neural net.

2.7.3 Programming Languages

Like for the theory of algorithms, the ties between neural nets and programming languages are twofold.

Firstly, neural nets represent a means to implement a problem completely differently from the conventional way which involves formulating the algorithm for solving the problem in terms of a certain programming language. To “program” (i.e. train) a neural net, one simply has to provide training examples, the actual implementation is done by the net itself.

Secondly, there are some programming languages (actually, they are called neurosoftware languages; they admittedly constitute specification, rather than programming languages) developed for neural nets. These languages (among them are P3, Panspec, AnSpec, and Axon) allow describing neural networks in a high-level machine-independent way [9].

¹⁴In short, a low-level algorithm in the above sense might tell you how to swim and a high-level (or *meta-*) algorithm how to learn to swim.

¹⁵with respect to the lower level

2.7.4 Numerical Computation

The inputs feeding into a neural network always are numerical representations of the true input (e.g. a two-dimensional image, a handwritten numeral, or a piece of music), and the outputs are also numerical (before “decoding”). Internally, the computational units of the net only work with the numbers without relating them to any “concrete object”, and the basic building blocks, the artificial neurons, are nothing but a combination of adders, multipliers, and a non-linear activation function.

Therefore, it is obvious that neural nets are well suited to solve certain types of numerical problems. For example, as I intend to show in the next section on applicability, neural networks can be used to implement a parallel version of an algorithm that computes the solution of a system of linear equations [4]. But of course, things like rounding error analysis become extremely complicated in this context.

2.7.5 Computer Architecture

When artificial neural nets were introduced about 50 years ago, the natural implementation choice was to simulate them on a conventional von Neumann computer. Now, as technology advances, as it is possible¹⁶ to build larger and faster circuits using less room, one begins to successfully replace this rather inefficient strategy by implementing neural nets directly in hardware, as coprocessors supplementing a conventional machine [9].

Examples include the MARK III (1985) and MARK IV (1986), the PARAL-LON 2 (1987) and the ANZA PLUS (1988). The last one, for instance, contains 1 million processing elements (i.e. artificial neurons), interconnected in a network comprising 1.5 million connections. That sounds pretty large, but it still is by several orders of magnitude smaller than the huge network in the human brain.

Since all the computations of the units within one layer go on in parallel (at least in non-recurrent, feed-forward networks) these *neurocomputers* represent one kind of *parallel architecture*. But nobody has so far proposed a stand-alone neurocomputer (which is *not* configured as a coprocessor to a standard serial computer), since common neural networks do not handle I/O, so this task is left to the host machine [9].

Like the biological prototype, artificial neural nets have the very nice property that they are not as sensitive to damage as conventional parallel computers, which

¹⁶thanks to VLSI

makes them superior to those machines. If a processing element in a neural net breaks down (and hundreds of neurons die every day in the brain), others can take over its function.

3 Applicability to Practical Problems

An extremely interesting aspect of neural networks is its widespread applicability to all kinds of real world problems. Since the popularity of the topic is constantly increasing, and since progress is being made very fast, the seemingly unlimited list of potential and actual applications will supposedly grow even further. In this section, I talk about some of the problems neural nets are currently applied to.

3.1 Pattern Recognition

The big keyword in the context of neural networks is *generalization*. “Teaching” a neural net involves specifying training examples, i.e. input/output patterns that have to be memorized. But in almost any case (at least in those cases that are of interest concerning real world problems), only the network’s performance on new data and its ability to “forecast” the correct output answering a previously unknown input is important, not how well the net memorizes the training examples. In other words, although the net has to “learn” the training examples up to a reasonable degree of perfection, what matters later in the application phase is that the net is able to guess the right answer, even if it has never “seen” the corresponding input before.

This ability is referred to by the notion of generalization. A neural net maps similar input patterns to similar output patterns. It sort of detects regularities in the training data and encodes them into its synaptical weights. The consequence is that if, for example, the net has been trained to distinguish between handwritten Arabic numerals from 0 to 9 (in some appropriate representation), a digit is mostly still classified correctly even if it is written somehow different from the representation learned in the training phase, e.g. with a different slant or something like that, up to an often surprisingly large degree of deformation [6].

Often times, the set of known patterns is divided into two parts. The first part is used for training the net and the second part for evaluating the network’s ability to generalize. After the learning phase, the net is presented with patterns that are new for *it*, but well-known for the human operator, and it is checked if the net is able to “guess” the correct output. The network is not applied to unknown data,

before it performs reasonably well on the untrained data. The task of dividing the data set is not at all trivial, because the training examples have to contain a somewhat representative selection¹⁷.

The phenomenon of generalization makes neural nets ideal for all kinds of classification or pattern recognition tasks [5, 6, 9]. Examples include “reading” written text, recognizing familiar faces, distinguishing between several 3-dimensional objects, or understanding spoken words. All of these tasks require a network structure that is specific to the individual problem. For instance, for classifying handwritten numerals, a net with a 2-dimensional input layer could be used, i.e. the numeral to be classified is approximated with a (2-dimensional) grid of pixels (which are either on or off, black or white, visible or not), and this pattern is shuffled into the input units of the net. The outputs could be trimmed to return a noise-free representation of the numeral.

The connectionist approach does not only provide “ideal” or “natural” solutions for classification problems. Sometimes it is really *necessary* to solve a problem by just giving examples of which inputs shall be mapped to which outputs, since for those tasks, e.g. speech recognition, no algorithmic solution is known [9]. Therefore, conventional computers could not be programmed to perform the task, so we have the choice to either use neural nets or to do it on our own.

3.2 “Predicting the Future”

The problem of continuing a time series is closely related to the notion of pattern recognition. A time series is a sequence of measurements that are taken at different points in time. Assuming a non-random, non-accidental behaviour of the time series, the value at a point t (which shall be denoted x_t) can be expressed as a function of the values at times $t - 1, t - 2, \dots, t - d$, where d is a problem-dependant parameter (see [24]):

$$x_t = F(x_{t-1}, x_{t-2}, \dots, x_{t-d}).$$

The problem now is to find the function F , and this can be done with the help of a neural net. One possible architecture, for example (a sort of “logical” choice), involves d input units (for $x_{t-1}, x_{t-2}, \dots, x_{t-d}$), one output unit (for x_t) and some number of hidden units¹⁸. If the training set is large enough, the net will be able

¹⁷otherwise, generalization cannot work

¹⁸Determining this number is a rather critical task, as explained below.

to detect the general structure of the mapping that F represents¹⁹, so it is able to “predict” values for time points in the future.

This could help, for example, to forecast earthquakes and their magnitudes, at least seismic activity during an earthquake could be predicted. Another possible application is related to medicine (e.g. predicting the continuation of electrocardiograms).

As I said above, the values x_t are *measurements*, so, like always with measured data, they do not seem to follow an exact, fixed function F . There is always some noise and deterioration, it looks as if the function F varies over time²⁰. This brings up another issue, an issue not only related to time series prediction but to any application that involves any kind of noise: the problem of *overfitting* (see [7]).

Overfitting relates to the fact that a neural net will begin to learn also the *noise* if its size is too big. The net will then encode too much information into the weights, and generalization will thus not yield any reliable results.

In summary, it can be said that a neural net can be used to predict the future in some sense, but only if its size is appropriate for the problem.

3.3 Composing Music

The idea of prediction can be used to make a neural net appear to be creative. Interpreting a piece of music as a special kind of time series (where one note depends on several previous ones) leads to a way to apply neural networks to music “composition”.

The net used for this task could be a recurrent one, taking the first few notes (in some representation encoding pitch, length, etc.) as input, “processing” (or storing) the input in an interconnection network of hidden units, and outputting a representation of the next note, which is fed back into the net as a new input. Thus, the net can go on, “producing” an entire piece of music, note by note, always deriving the next note from the beginning input and previously computed notes.

This can only work if the weights of the net are already trained *before* starting to “compose”. The net has to “listen” to a collection of musical pieces again and again till it detects certain regularities in the data. The existence of those regularities is a necessary condition for this to yield satisfying results. For example, such

¹⁹keep in mind that F is not known

²⁰Of course, the general structure behind the mapping of F should not change to give the net a fair chance.

a regularity could be achieved by choosing musical pieces of only one composer, who has a very typical style. This in turn implies that the net is not really creative, it is merely imitating the style of this particular composer (for further details see [14]).

3.4 Solving Systems of Linear Equations

Numerous practical applications, such as, for example, ray tracing or aircraft control systems, rely on solving systems of linear equations efficiently. There are various algorithms for accomplishing this task, including Gaussian elimination or iterative techniques, such as Gauss-Seidel or the Jacobi method. It is not easy, however, to parallelize these algorithms.

The algorithm of Huang (see [4] for details) represents a completely different approach for directly solving determined systems of linear equations. Its fundamental steps involve matrix-vector multiplication, the dot product of two vectors, and vector-vector multiplication. The algorithm will thus perform well on any hardware that supports this kind of calculations as basic operations. It can be shown, and I want to do that for matrix-vector multiplication, that neural nets are among those hardware architectures.

Suppose now, we have a perceptron with n inputs i_1, i_2, \dots, i_n and n linear outputs o_1, o_2, \dots, o_n . The 2 layers are fully connected, and w_{ij} stands for the weight from input unit j to output unit i . Since the activation function f is in this case (for *linear* output units) the identity function, this yields:

$$o_i = \sum_{j=1}^n w_{ij} i_j, \quad \forall i = 1, \dots, n.$$

If we write the weights w_{ij} as a matrix \mathcal{W} , the inputs as a vector \vec{i} and the outputs as a vector \vec{o} , then we get:

$$\vec{o} = \mathcal{W}\vec{i}.$$

This means that computing the outputs of a perceptron given its inputs, which definitely is a basic operation in connectionist hardware, corresponds directly to matrix-vector multiplication.

Similarly, every computation in Huang's algorithm can be related to a basic neural net operation, so it is easy to implement this algorithm using neural hardware [4]. Since, as already mentioned in the section on the relationship between neural nets and computer architecture, the computations within any of the layers of a neural net can (and do!) execute concurrently, this implementation will be parallel *per definition*.

4 Desirable Future Directions

I want to start this final section on desirable future directions with pointing into a direction that is, to me, not at all desirable. I mean the creation of an “artificial being” capable of thinking, sensing, and reasoning, just as humans, only that this “being” would be perfect in the sense that it would not make the same mistakes that we do. This sounds to me more like a horror scenario, and I hope that scientific research will never degrade to be an attempt to “improve creation”. Besides, I am convinced that no artificial neural network will ever be able to have feelings, be intuitive, or reveal *true* creativity, because it takes more than an ever so adequate and precise model of the brain to really *live*.

Rather, neural nets can be a blessing, if applied properly. Here, I think of blind people being able to enjoy books by listening to an electronic voice which is reading the text aloud and which is realized with the help of a neural net, or of deaf people being able to communicate more easily with the assistance of a neural net, a machine which translates spoken words into written expressions displayable on a computer screen.

So far, little is known about how the brain really works. Considering the enormous power of the brain, knowing a little bit more in this respect should be rather promising, especially as far as constructing powerful connectionist computers is concerned. And once again, these neurocomputers should serve as artificial assistants, not as “artificial cobeings”. If it were possible in the future to build better models, trained with more appropriate learning algorithms, showing more biological fidelity, then artificial neural nets really could lead to new results, new ideas, and new insights that would clarify the big mystery of the neural computations our brain performs. It might even be possible that neural nets could help in finding the reasons for mental disorders and potentially indicate a way to cure them.

Finally, many connectionists have the dream of a neural net that could “program itself” completely independently, just by learning a bunch of examples. This would involve finding the optimal network topology (not too large to avoid “overfitting” and not too small to allow enough flexibility for the task to be learned), optimal learning parameters (like the learning rate), and, of course, optimal weights (which characterize the connections between the units in the network). Up to now, this dream is far from being realizable, and the search for network parameters is the probably most difficult and most critical part associated with the training of an artificial neural net.

In summary, there is enough room for further progress in the field, and a large amount of hard work and scientific research needs to be done in the future.

5 References

- [1] CATER, J. P., *Successfully Using Peak Learning Rates of 10 (and greater) in Backpropagation Networks with the Heuristic Learning Algorithm*, Austin, TX
- [2] FAHLMAN, S. E., *Faster-Learning Variations on Back-Propagation: An Empirical Study*, in: *Proceedings of the 1988 Connectionist Models Summer School*, Carnegie Mellon, pp. 38 – 51, 1988
- [3] FAHLMAN, S. E., LEBIERE, C., *The Cascade-Correlation Learning Architecture*, Technical Report # CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1990
- [4] FORBES, A. B., MANSFIELD, A. J., *Neural implementation of a method for solving systems of linear equations*, *Network* 1 (1990), pp. 217 – 229, IOP Publishing Ltd, 1990
- [5] FUKUSHIMA, K., *A Neural Network for Visual Pattern Recognition*, *IEEE Computer*, Vol. 21, no. 3, pp. 65 – 75, March 1988
- [6] FUKUSHIMA, K., MIYAKE, S., ITO, T., *Neocognitron: A Neural Network for a Mechanism of Visual Pattern Recognition*, *IEEE Transactions on Systems, Man, and Cybernetics* 13(5), pp. 826 – 834, September/October 1983
- [7] GERSHENFELD, N. A., WEIGEND, A. S., *The Future of Time Series*, in: *Time Series Prediction: Forecasting the Future and Understanding the Past*, WEIGEND, A. S., GERSHENFELD, N. A. (Eds.), pp. 1 – 70, Addison-Wesley, Reading, MA, 1993
- [8] HANSEN, J. V., MESSIER, W. F. JR., *Artificial Neural Networks: Foundations and Application to a Decision Problem*, in: *Expert Systems with Applications*, Vol. 3, pp. 135 – 141, Pergamon Press plc, 1991
- [9] HECHT-NIELSEN, R., *Neurocomputing: Picking the Human Brain*, *IEEE Spectrum*, 25 (3), pp. 36 – 41, March 1988
- [10] LEIGHTON, R. R., CONRATH, B. C., *The Autoregressive Backpropagation Learning Algorithm*, The MITRE Corporation, McLean, VA, Preprint 1991

- [11] LIPPMANN, R. P., *An Introduction to Computing with Neural Nets*, in: LAU, C. G. Y. (Ed.), *Neural Networks: Theoretical Foundations and Analysis*, pp. 5 – 23, IEEE Press, New York, 1992
- [12] LIPPMANN, R. P., MOODY, J. E., TOURETZKY, D. S. (Eds.), *Advances in Neural Information Processing Systems 3*, Morgan Kaufmann Publishers, San Mateo, CA, 1991
- [13] MØLLER, M. F., *A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning*, Computer Science Department, University of Aarhus, Denmark, Preprint 11/13/1990
- [14] MOZER, M. C., *Connectionist Music Composition Based on Melodic, Stylistic, and Psychophysical Constraints*, (Technical Report CU–CS–495–90), Boulder, CO: University of Colorado, Department of Computer Science, 1990
- [15] OTWELL, K., *Incremental Backpropagation Learning from Novelty-Based Orthogonalization*, in: *Proceedings of the International Joint Conference on Neural Networks*, Washington, DC, Volume I, Theory Track, Neural and Cognitive Sciences Track, pp. 561 – 564, Lawrence Erlbaum Associates (Publishers), 1990
- [16] RIEDMILLER, M., BRAUN, H., *A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm*, in: *Proceedings of the 1993 IEEE International Conference on Neural Networks*, San Francisco, California, Vol. I, pp. 586 – 591, 1993
- [17] ROHWER, R., *The “Moving Targets” Training Algorithm*, Centre for Speech Technology Research, University of Edinburgh, Scotland, in: *Lecture Notes in Computer Science 412, Proceedings of the EURASIP Workshop on Neural Networks, Sesimbra, Portugal*, pp. 100 – 109, Springer–Verlag, Berlin, New York, London, Paris, 1990
- [18] RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J., *Learning Internal Representations by Error Propagation*, ICS Report 8506, Sept. 1985, Institute for Cognitive Science, University of California, San Diego, published in: *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Vol. I, RUMELHART, D. E., MCCLELLAND, J. L. (Eds.), Cambridge, MA, MIT Press, pp. 318 - 362, 1986

- [19] RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J., *Learning Representations by Backpropagating Errors*, *Nature* 323: pp. 533 – 536, 1986
- [20] SANCHEZ-SINENCIO, E., LAU, C. G. Y. (Eds.), *Artificial Neural Networks: Paradigms, Applications, and Hardware Implementations*, IEEE Press, New York, 1992
- [21] SILVA, F. M., ALMEIDA, L. B., *Acceleration Techniques for the Backpropagation Algorithm*, in: *Lecture Notes in Computer Science 412, Proceedings of the EURASIP Workshop on Neural Networks, Sesimbra, Portugal*, pp. 110 – 119, Springer-Verlag, Berlin, New York, London, Paris, 1990
- [22] SMOLENSKY, P., *On the proper treatment of connectionism*, in: *Behavioral and Brain Sciences*, 11, Cambridge University Press, pp. 1 – 74, 1988
- [23] VEMURI, V., *Artificial Neural Networks: An Introduction*, in: VEMURI, V. (ed.), *Artificial Neural Networks: Theoretical Concepts*, Computer Society Press, pp. 1 – 12, 1988
- [24] WEIGEND, A. S., HUBERMAN, B. A., RUMELHART, D. E., *Predicting Sunspots and Exchange Rates with Connectionist Networks*, in: CASDAGLI, M., EUBANK, S. (Eds.), *Nonlinear Modeling and Forecasting, SFI Studies in the Sciences of Complexity*, Proc. Vol. XII, pp. 395 – 432, Addison-Wesley, 1992
- [25] WEIGEND, A. S., Review of the Book *Introduction to the Theory of Neural Computation* by HERTZ, J. A., KROGH, A. S., & PALMER, R. G., Elsevier, *Artificial Intelligence* 62 (1993) pp. 93 – 111, Elsevier Science Publishers B. V., 1993
- [26] WINSTON, P. H., *Artificial Intelligence*, Addison-Wesley Publishing Company, Reading, MA, 3rd Edition, 1992
- [27] YANG, J., HONAVAR, V., *Experiments with the Cascade-Correlation Algorithm*, Technical Report # 91-16, Department of Computer Science, Iowa State University, Ames, Iowa, July 1991