PS Einführung in die Bioinformatik SS2001

NC-001-2001 May 30, 2001

Sensitivity Analysis Of Neurons In ANN

Report

Sebastian Gumpold
sgumpold@cosy.sbg.ac.at

Clemens Bauer
cbauer@cosy.sbg.ac.at

Annemarie Mayer
ammayer@cosy.sbg.ac.at

Academic Supervisor Mag. Dipl.-Ing. Schwaiger Roland

Department of Computer Science
University of Salzburg

Correspondence to:

Universität Salzburg
Institut für Computerwissenschaften und Systemanalyse
Jakob–Haringer–Straße 2
A–5020 Salzburg
Austria

Sensitivity Analysis Of Neurons In ANN

Sebastian Gumpold, Annemarie Mayer and Clemens Bauer
Department of Computer Science
University of Salzburg
A–5020 Salzburg, Austria
{sgumpold}, {ammayer}, {cbauer}@cosy.sbg.ac.at

Abstract

In this paper we present methods to determine the optimal size and structure of a Feed Forward Neural Network (FFNN) for a specific problem.
Using these methods a network's complexity can often be minimized by pruning hidden neurons and/or connections without loss of function. We will also show how the optimal structure of an ANN can be reached by adding hidden neurons to an existing network structure. As a third way to optimize a neural net, evolutionary design is discussed.
One specific aim of this paper is to explain how to (automatically) determine the "optimal" structure of a FFNN to people who are currently learning about neural networks.
Some of the methods dealed with in our paper we also implemented in a neural net visualisation program called NetVis.

Contents

Chapter 1

Introduction

1.1 A short Introduction to ANN

> A Neural Network is an interconnected assembly of simple processing elements, *units* or *nodes*, whose functionality is loosely based on the animal neuron. The processing ability of the network is stored in the inter-unit connection strengths, or *weights*, obtained by a process of adaptation to, or *learning* from, a set of training patterns. [1]

In our context we are only dealing with *Feed Forward Neural Networks* (FFNN), which will be discussed in detail in the following subsection.

1.1.1 Feed Forward Neural Networks (FFNN)

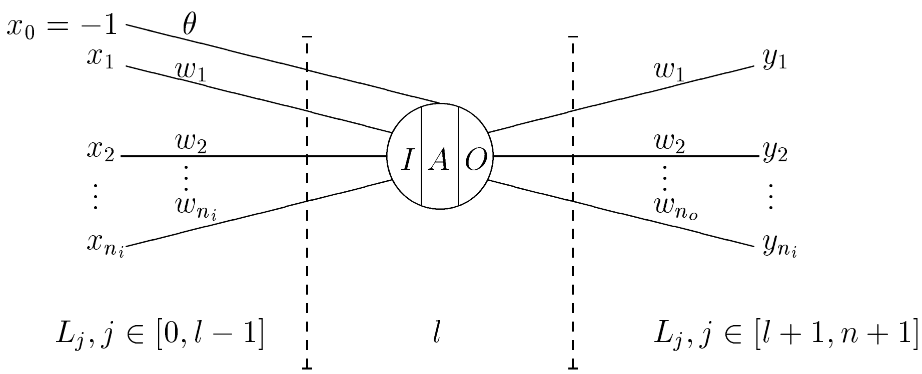
An ANN consists of a finite set of neurons, which are organized in layers:

- one input layer
- n hidden layer(s), where $n \in \mathbb{N}_0$
- one output layer

The input data is presented to the input neurons. All neurons perform specific operations on the data and transfer the results to the next layer (except the output layer, as it is the last layer in the network). In our case the input neuron's operation is the *id-function.*
The connections among the neurons are weighted, that means, values are changed by some factor. This goes on until the output layer is reached, and the “result” of the computation is expressed by the numerical values that are presented by the output neurons.
In a FFNN a neuron in layer l can only get input from neurons which are situated in layer(s) L_j $(0 \leq L_j \leq (l-1))$, where layer 0 denotes the input layer. And the output produced by the neuron can only be forwarded to neurons which are in layer(s) L_j $((l+1) \leq L_j \leq (n+1))$, where layer (n+1) is the output layer.

Let us consider a single neuron in the hidden layer l. This neuron has n_i input connections from neurons which are situated in the previous layer(s) L_j $(0 \leq L_j \leq (l-1))$. All those connections are weighted with the weights w_1 to w_{n_i}.

Computation is performed in the following way:



The Input Function I: The summed input that is presented to our neuron is:

$$I_{simple} = w_1 x_1 + w_2 x_2 + \cdots + w_{n_i} x_{n_i} \tag{1.1}$$

where n_i is the number of input connections, x_i is the signal of the ith input neuron, and w_i is the weight of the ith connection.
As an addition, we have a constant weight θ (the *threshold*), which has the value -1. Including θ, I can also be written as:

$$I = \sum_{k=0}^{n_i} w_k x_k \tag{1.2}$$

The Activation Function A: Then, an Activation Function is applied on the weighted sum as seen in 1.2. Mostly, the activation function is the sigmoid function denoted by

$$A = \frac{1}{1 + e^{-(I_{simple} + \theta)}} = \frac{1}{1 + e^{-I}} \tag{1.3}$$

where θ denotes the threshold.

The Output Function O: Normally, the output function is the identical function $O(x) = x$, that means, it has no further effect on the value computed by the activation function.

Finally, the value computed by our neuron is forwarded to all other neurons in layer(s) L_j $((l+1) \leq L_j \leq (n+1))$ which are connected to our neuron. The output connections y_1 to y_{n_o} are weighted similar to the input connections with the weights w_1 to w_{n_o}.

1.1.2 The Backpropagation Learning Algorithm

Although there are many different kinds of learning rules used by ANN we concentrate on the *Backpropagation Algorithm*, because it is the most widely used learning algorithm for FFNN.
Backpropagation is a training method that uses the *Generalized Delta Rule*. Its basic idea is to perform a gradient descent on the total squared error of the network output (considered as a function of the weights). It was first described by *Werbos* and made popular by *Rumelhart*'s, *Hinton*'s and *Williams*' paper [2].

ANN learning generally occurs by applying a list of patterns (a *patternset*) on the network. We can separate this set in

1. a *training* patternset, which is used to train the network
2. a *validation* patternset, to evaluate the training progress during training
3. a *test* patternset, to test the performance of the net after training

This is called *cross validation*.
The pattern of input values is presented to the input nodes, and the computed values (which are expressed by the output nodes) are compared to the pattern of output values. If those values differ for an output neuron and the desired output, we can compute the squared error E for a single node

$$E = \sum_{p} \frac{1}{2}(t_p - y)^2 \tag{1.4}$$

where p is one specific pattern in the patternset, t is the target function and y is the 'result function' of the node (the final output of one node, depending on the input, activation and output function as mentioned in 1.1.1).
Now, for computing the error for a layer of nodes, we have to sum the error E_p (for a single pattern) over the number n of nodes

$$E_p = \frac{1}{2}\sum_{j=1}^{n}(t_p^j - y^j)^2 \tag{1.5}$$

As the result function depends on the weights w_k (see 1.2), we can analyze the weight change in the output layer:

$$\Delta w_k^j = \alpha\sigma'(a^j)(t^j - y^j)x_k^j \tag{1.6}$$

where superscript j denotes which node is being described. The term $(t^j - y^j)$ represents a measure of the error on the jth node. The term $\sigma'(a^j)$ relates to how quickly (rate of change) the activation function can change the output (and hence the error). If this is large, we can expect a rapid change in error for a given change in activation. The factor of x_k^j is related to the amount that the kth input has affected the activation. If it is large, the weight of the kth input should be changed in order to get a large change in error.
For the ith node in a hidden layer, the formula looks like this:

$$\delta^i = \sigma'(a^k)\sum_{j \in I_i} \delta^j w_i^j \tag{1.7}$$

where I_i is the set of nodes which take an input from the hidden node k, and the term δ^k is an abbreviation for $\sigma'(a^k)(t^k - y^k)$.
As a conclusion, we can give a “pseudo code for the backpropagation algorithm:

```
repeat
   for each training pattern
      present the pattern at the input layer
      apply the target pattern to the output layer
      for each output node
         calculate and reduce the error according to 1.6
      end for
      for each hidden node
         calculate and reduce the error according to 1.7
         propagating from the output nodes back towards this node
      end for
   end for
until the error is ‘‘acceptably low’’
      (or the network does not converge)
```

1.2 Overview to Sensitivity Analysis

If a Neural Network is welltrained (the error function converges and the network is not “overtrained”), then the net should be able to solve a specific problem. In other words, the underlying trend in the data is captured by the network [1].
If we now want to know why the network recognizes the problem structure it is trained on, we need to perform *Sensitivity Analysis* on it. This helps us to analyze the inner structure and working method of the ANN, and further, to manually or automatically apply a better structure to it.
And, as *Hashem* states it in [3]:

> In general, one of the key factors that affect the success of process modeling, is the ability to extract information about the model structure and the relationships between its inputs and outputs from the trained network. Such information is essential for model validation and for process optimization and control. Moreover, in some cases where the original process is not well understood, this information can be employed as a basis for the analysis of the process and in determining the most significant factors that affect it.

1.3 Methods Overview

1.3.1 Pruning Methods

According to [4], five connection removal methods are presented.

[1] see [1], chapter 4, p.8

1. (a) The simplest method is to select the connections with weights near to zero ($w_i \approx 0$) and to erase them.

   (b) As an addition, if the connection's average contribution is added to the bias (= threshold) of the neuron that receives input from the erased connection.

2. The *smallest variance method* proposed by J. Sietsma and R.J.F. Dow [5] applies the idea of the "smallest contribution variance"

$$min(\sigma_P(\frac{1}{P}\sum_{p=1}^{P} c^p))$$

   where c is the output of the connection, P is the size of the training set, p is a single pattern and σ is the contribution variance. Then the mean output of the connection to be erased is added to the bias of the receiving neuron.

3. E.D. Karnin [6] determines the sensitivity s of a connection by:

$$s = \sum_{n=1}^{N} (\Delta w_n)^2 \frac{w_n}{\eta(w_n - w_0)} \quad \text{if } w_n \neq w_0\text{, otherwise} \quad s = 0 \qquad (1.8)$$

   where w_n is the weight in the current training epoch n, w_0 the initial weight and Δw_n the weight change in the n-th epoch. N is the number of training epochs, and η is a factor that specifies the "importance" of the change between the initial weight and the current weight in the current training epoch. If η is high, a little weight change has a big effect on the sensitivity value s.
   The connections with the lowest sensitivity s are pruned.

4. M.C. Mozer and P. Smolensky [7] weight removal method is called *skeletonization*. If a neuron is removed the error induced by this removal is computed by multiplying the neurons output with an *additional strength*.
   Setting an additional strength to zero is the same as removing a neuron. If the derivative of the error function to these additional strengths are small, the neuron is removed.

5. The 5th sensitivity measure is part of a pruning method called *autoprune* [8]. There the probability that a weight becomes zero is tested statistically. If this probability is high, then the connection will be removed.
   An extension to this pruning method can be found in [9]. The socalled pruning method *λ-prune* estimates how many neurons should be removed at each step.

1.3.2 Growing Methods

Growing Methods (or Constructive Methods) start with a minimal network structure (only input and output neurons) and then add neurons according to specific methods to gain the "optimal" network size. According to [10], methods are summarized in the following way:

1. The Upstart Algorithm [11]
   The first step is to train the minimal network until no further error reduction can be reached. Then it is determined for which patterns the output is incorrectly responding. The attempt is to add "daughter units" to the incorrectly responding output unit, which are supposed to correct the input of the output unit. Again, the new network is trained, and this cycle goes on until the error is low enough.

2. Cascade Correlation [12]
   This algorithm adds a hidden unit (the *candidate unit*) to a trained, but not "big enough" network in a way that it gets input from all input units and all (until then) existing hidden units and delivers its output to all output units. The basic idea is to equalize the output of the candidate unit and the error of the actual output units for a specific pattern p. After the connections from the input and hidden units to the candidate unit are trained, they are "frozen" (that means, they can not be changed during further training), and then the connections from the candidate unit to the output units are trained.
   The cascade-correlation algorithm produces very deep networks (that is, they have got a large number of hidden layers, but usually very few neurons in each of them).

3. The Tiling Algorithm [13]
   This algorithm adds a new layer containing an output unit (or a set of output units) to the "top" (the former output layer) of the existing net. These units are either connected only to the former output units and the input units, or to additional "ancillary units".

4. Dynamic Node Creation [14]
   This method is more empirical than the others, as it adds a new hidden neuron (if the previous rate decrease of the net's error falls below a certain value) with randomized initial weights connecting between the input and output layers. It is left to the user to adjust certain parameters, such as the minimum period between node additions, when to stop adding nodes, or what too slow a rate is.

1.3.3 Hybrid Methods

We also searched on the web for some documents about hybrid methods, but in all papers we found, "hybrid" meaned: grow net, then prune net (without detailed description of e.g. when the best time to stop growing and begin pruning is).

Chapter 2

Experimenting with SNNS

2.1 The XOR Problem

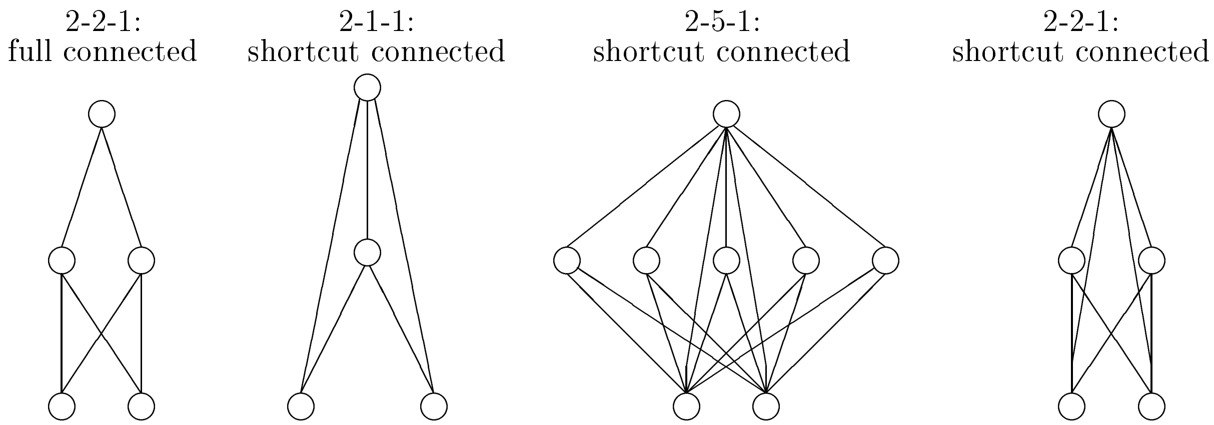
2.1.1 First Steps

1. Starting SNNS
2. Creating an XOR network

   First, we chose “bignet” to create a feedworward network.
   In the opening dialog box we defined the planes for our network, such that we had two neurons in both the input layer and the hidden layer (x=1, y=2) and one neuron in the output layer (x=1, y=1). Then we selected “full connection” to connect each neuron to every neuron in the following layer.
   Finally, we created our net by selecting “create net”.
3. Displaying the network

   The command “display” opens a window showing the neurons in the specific layers. To also display the connections, we had to set the option “links on” under “setup”.
4. Training the network

   As a starting point, we loaded the pattern file `xor.pat`. Under the option “control” we entered the number of cycles that the network should be trained with the patternset. As training function, we selected “Std. Backpropagation”. Then we started training the network.



After experimenting for a while we had to admit that our net did not converge after 10.000 cycles.
So, we tried another net structure, consisting of two neurons in the input layer, one neuron in both the hidden and the output layer and shortcut connections. This structure converged after 2000 cycles.
As we could not get our first net structure learning, we tried to add hidden neurons to it. With five hidden neurons and shortcut connections, the error was under 0.10 after 7000 cycles.
Inspired that way, we tried our original 2-2-1 structure with shortcut connections. To our surprise the error was under 0.05 after 7500 cycles.

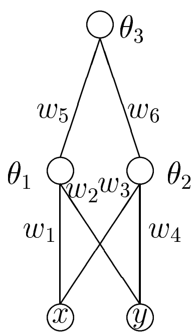
5. Trying to find out why our first XOR net did not work

   We realized that the our first 2-2-1 XOR net (with full connection) did not converge to the fact of we were using the sigmoid function as transfer function. So we repeated the experiment with the same net structure, but using a heaviside function as transfer function this time. The experiment worked and the network's error was near to zero after 2500 training cycles.

2.1.2 Calculating An XOR Net

As we had big problems training a simple XOR net with two input and two hidden neurons, one output neuron and full connection, we tried to find out which weight values would be the best. We assumed a heaviside function as activation function.

Our net (with description):



Our equations were as follows:

For the first node in the hidden layer, the input function is

$$I_1 = x * w_1 + y * w_2 - \theta_1$$

where x and y denote the input values represented by the two input neurons. Its activation function is the heaviside function specified as follows:

$$A_1 = \begin{cases} 0 & \text{if } I_1 > k_1 \\ 1 & \text{if } I_1 \leq k_1 \end{cases}$$

Accordingly, these formulae are written for the second hidden node:

$$I_2 = x * w_3 + y * w_4 - \theta_2$$

$$A_2 = \begin{cases} 0 & \text{if } I_2 > k_2 \\ 1 & \text{if } I_2 \leq k_2 \end{cases}$$

For the output node, the calculations look like this:

$$I_3 = A_1 * w_5 + A_2 * w_6 - \theta_3$$

$$A_3 = \begin{cases} 0 & \text{if } I_3 > k_3 \\ 1 & \text{if } I_3 \leq k_3 \end{cases}$$

As θ_i can always be added on k_i, we perform the following renaming:

$$\theta_1 + k_1 = k_1^*, \quad \theta_2 + k_2 = k_2^*, \quad \theta_3 + k_3 = k_3^*.$$

We now distinguish among two cases of calculation:
The "zero-cases" where $x = y = 0$ and $x = y = 1$ and the net output should be zero, and the "one-cases" where either $x = 0$ and $y = 1$ or $x = 1$ and $y = 0$ and the net output is expected to be one.

- The Zero-Cases:
  In the first case, we set as a condition that $A_1 = A_2 = 0$. Then we have $I_1 > k_1$ and $I_2 > k_2$. For the first case ($x = y = 0$), we get

  $$0 > k_1^* \ , \quad 0 > k_2^* \ ,$$

  and for the second case ($x = y = 1$)

  $$w_1 + w_2 > k_1^* \ , \quad w_3 + w_4 > k_2^* \ .$$

- The One-Cases:
  Here, the condition is that $A_1 \neq A_2$. For the first case ($x = 0$ and $y = 1$) we suppose that $A_1 = 1$ and $A_2 = 0$. This means that $I_1 \leq k_1$ and $I_2 > k_2$, and thus

  $$w_2 \leq k_1^* \ , \quad w_4 > k_2^* \ .$$

  In the second case ($x = 1$ and $y = 0$), where $A_1 = 0$ and $A_2 = 1$, we get

  $$w_1 > k_1^* \ , \quad w_3 \leq k_2^* \ .$$

- The output node weights:
  The output node checks if $A_1 = A_2$, or $A_1 \neq A_2$. For the first case ($A_3 = 0$, and $I_3 > k_3$) we get

  $$0 > k_3^* \ ,$$

  and for the cases that $A_1 = 1$ or $A_2 = 1$ ($A_3 = 1$ and $I_3 \leq k_3$)

  $$w_5 \leq k_3^* \ , \quad w_6 \leq k_3^* \ .$$

- Conclusions:
Considering the formulae $w_1 + w_2 > k_1^*$, $w_2 \le k_1^*$, $w_1 > k_1^*$, and $k_1^* < 0$, it follows that

$$w_1 > 0 \;, \quad w_2 \le k_1^* \;,$$

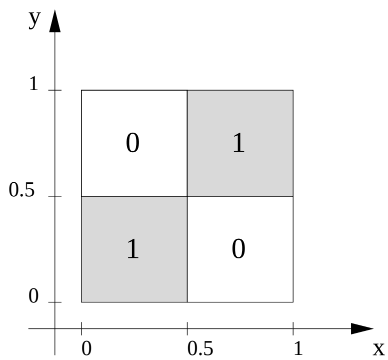
and accordingly

$$w_3 \le k_2^* \;, \quad w_4 > 0 \;.$$

2.2 The "Four Quadrant Problem"

2.2.1 Description

The "Four Quadrant Problem" indicates the value of the quadrant in which the point (x, y) lies, in a coordinate system which is divided into four quadrants.

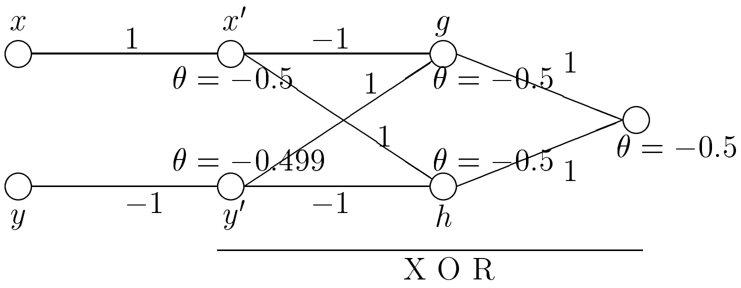


2.2.2 Implementation

First, we tried to generate a net with only one hidden layer and heaviside activation functions to solve our problem. But this net did not converge. We suppose that the heaviside function is not able to extract enough information out of the input values to represent all cases in one hidden layer.
So we built another net with 2 hidden layers. We first determined how the net has to be structured, which weights are optimal for the connections and which values our θ's should have.

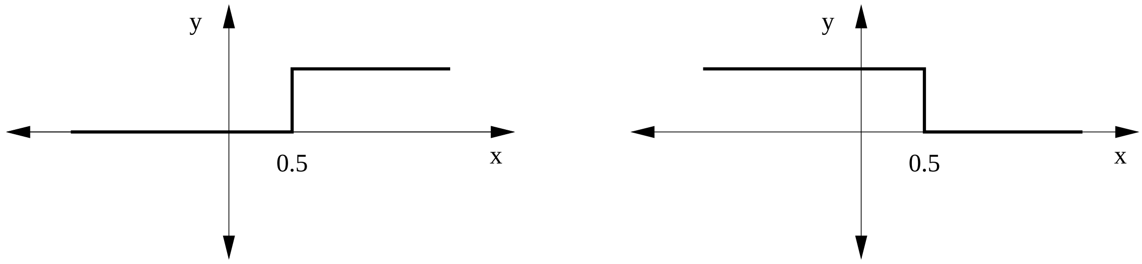
We decided that the net structure is 2-2-2-1, with two hidden neurons in the first hidden layer which only receive input from one input neuron each and transfer the float input values to integer.

The net we built to solve the "Four Quadrant Problem" has the following structure:



This net solves the "Four Quadrant Problem".

The connection from x to x' performs a *stepup function* on the input x. The connection from y to y' performs a *stepdown function* on the input y. We had to set x's θ to -0.5 and y's θ to -4.9999.



"stepup" "stepdown"

Chapter 3

Implementing a simple pruning method

3.1 Description

For a description of this method, see 1b on page 7.

The program in which we implemented this algorithm is called *netVis*[1].

We decided that connections which got pruned should not be displayed in the visualisation of the neural net (after pruning), but we implemented a list, which contains all pruned connections. With a click on an item of the list the selected (already pruned) connection is displayed as a bold, red line in the net structure.
The error sum is displayed before and after pruning (if possible).

A possible test case for this implementation is to use a net from "ALGAE"[2], to prune it with *netVis* and test it again in *ALGAE*.

3.2 The algorithm

1. Search for the average (connection-) weights

   $$\frac{1}{P}\sum_{p=1}^{P} c^p < x$$

   and mark every connection found as "TO ERASE".

2. "Feed" the neural net with patterns and calculate the "θ-Output" ("transmission value") of the connections.

3. Add the θ-Output to the bias of the node at which the connection ends.

[1]A program which displays the structure of a neural net.
[2]A program that simulates algae using neural nets.

3.3 Experimenting with XOR

...

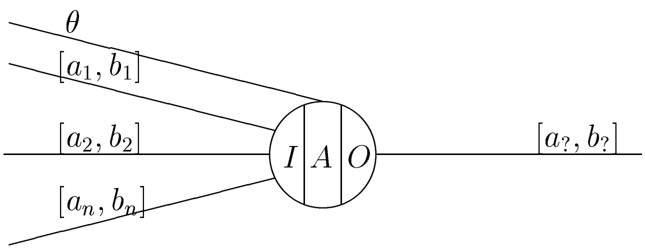
3.4 Experimenting with a neural net from ALGAE

...

Chapter 4

Analysing ranges and frequencies

4.1 Introducing ranges



Suppose we have a hidden unit in a specific ANN with a sigmoid activation function. As we know, only input values in the range of $[-5, 5]$ are significantly changed by this function. We can therefore improve the performance of our net by applying the significant range of the activation function to the range of the input function.

A formula for determining these ranges is presented in [15].

- If we are concerned with an input neuron (whose activation function is the identity function), then the presynaptic and postsynaptic ranges are the same. They equal the ranges of the input values presented to the net.

- In case of a hidden or an output neuron n (whose activation function is the sigmoid function), we can describe its presynaptic input range by:

$$[a''_k, b''_k] = \begin{cases} [a'_k w_k, b'_k w_k] & \Leftrightarrow w_k \geq 0, \\ [b'_k w_k, a'_k w_k] & \Leftrightarrow w_k < 0. \end{cases} \tag{4.1}$$

  where $[a'_k, b'_k]$ is the postsynaptic range of the kth neuron connected to n, w_k is the weight of this connection (from neuron k to n) and $[a''_k, b''_k]$ is the presynaptic input range of neuron n.

- The presynaptic range of n is

$$[a, b], where \begin{cases} a & = \sum a_l'' - \theta \\ b & = \sum b_l'' - \theta \end{cases}$$

- The postsynaptic range of n (sigmoid activation function σ) is

$$[a', b'] = [\sigma(a), \sigma(b)].$$

4.2 Introducing frequencies

As an addition to our thoughts above, we can improve the analysis by including frequencies into our considerations.
Imagine we have a range of input values among $[-1, 1]$, but most of the values that are presented to the network are among $[0, 0.5]$. How would this influence the pre-/postsynaptic ranges, and frequencies? In the best case, we could precise the postsynaptic range of values to a "significant" range which would be much smaller.

For the next equations we assume that the values' frequencies can be described by a function $f(x)$, where x is the value and $f(x)$ describes its frequency. This function is defined on the interval $[a, b]$ in which input values are distributed. Further, a percentage of values of over 90% should lie in the subinterval $[a^*, b^*]$.

An example for such a function could be the Gaussian curve $\phi(x) := \frac{1}{\sqrt{2\pi}\sigma} \cdot exp\,[-\frac{1}{2}(\frac{x-\mu}{\sigma})^2]$ where μ is the most frequent value and σ describes the standard deviation. 95.5% of all values lie in the interval $[-2\sigma, 2\sigma]$.

We will now do an attempt to describe the frequencies of the neurons in a network. Assumed is a set of frequency functions $f_i(x)$ (see above) describing the freuquencies of the input values that are presented to specific input neurons i, and intervals $[a_i, b_i]$ which descibe the ranges.

- For every input node, the presynaptic function equals the postsynaptic function $f_i(x)$ due to the fact that the input nodes' activation function is the identity function. We can define f_i over the interval $[a_i, b_i]$.

- Now we concentrate on a hidden neuron n receiving input from the input units. To determine the presynaptic range and frequency, we have to apply n's input function on the postsynaptic ranges and frequency functions of the input neurons (multiplied with the connection weights). The input function of this hidden neuron n is the sum of all postsynaptic values (here: ranges/frequencies) multiplied with the connection weights w_i (from the ith input node to hidden node n).
  First, we determine the presynaptic range $[a, b]$ as described in 4.1. Alternatively, we could replace the $[a_l'', b_l'']$ by the "significant ranges" $[a_l^*, b_l^*]$ to get the "significant

presynaptic range" $[a^*, b^*]$. Then we perform the input function on the weighted sum of frequency functions to get the presynaptic frequency function for the neuron n:

$$f_n(x) = \sum_i [f_i(x) \cdot w_i] - \theta.$$

- The postsynaptic range of n (with a sigmoid activation function σ) can then be simply obtained by applying σ on either $[a, b]$ or $[a^*, b^*]$, and the postsynaptic frequency function by applying σ on $f_n(x)$.

4.3 example

Appendix A

Protocols

A.1 2001/03/21

1. Responsible person for protocol: Annemarie Mayer

2. Participating persons: Sebastian Gumpold, Annemarie Mayer

3. Topics talked about:

   - What is a neural network?
   - What is sensitivity analysis?
   - Which methods can be used to perform SA?

4. New assignments

   - Introduction

5. Open questions

   - Is there a possibility to analyse only single neurons?
   - Under which aspects shall we order the different methods?
   - Shall we develop our own methods?
   - Do we have to treat this topic only theoretically?
   - ...

A.2 2001/03/28

1. Responsible person for protocol: Sebastian Gumpold

2. Participating persons: Annemarie Mayer, Sebastian Gumpold

3. Topics talked about:

   - Feedforward Neural Networks (FFNN)
   - Abstract

- Introduction to ANNs
- Discussion of methods to determine a network's optimal size

4. New assignments

- 'Evolve' Introduction

5. Open questions

- Should we search for further methods (pruning, growing, evolution) on the web?
- Shall we find/'invent' algorithms for automating the NN design process?
- ...

A.3 2001/03/31

1. Responsible person for protocol: Sebastian Gumpold
2. Participating persons: Annemarie Mayer, Sebastian Gumpold
3. Topics talked about:

- Examples with the SNNS (especially the XOR-Problem)
- Search for new methods/sources concerning our topic
- Changes in Text (as suggested by Prof. Schwaiger)
- Changes in Bibliography (adding URLs where possible)

4. New assignments

- Gurney's paper: URL
- Neurex (next time)
- Gorman und Sejnowski's paper (next time) ...

5. Open questions

- ...

A.4 2001/04/06

1. Responsible person for protocol: Annemarie Mayer
2. Participating persons: Mag. DI Roland Schwaiger, Sebastian Gumpold, Annemarie Mayer, Clemens Bauer
3. Topics talked about:

- Errors in the last version of the paper
- ...

4. New assignments

    - Calculate if our first net would have worked with a heaviside function
    - Implement the pruning method 1.b in netVis
    - ...

5. Open questions

    - ...

A.5 2001/04/26

1. Responsible person for protocol: Sebastian Gumpold

2. Participating persons: Mag. DI Roland Schwaiger, Sebastian Gumpold, Annemarie Mayer, Clemens Bauer

3. Topics talked about:

    - ...

4. New assignments

    - Simulate the "Quadrantenproblem" in SNNS (also try pruning some nodes)
    - Find documents about:
        - Fusion of growing and pruning (documents which show hybrid beginnings)
        - Are hybrid methods used in Robotics?
        - Appliance of growing/pruning in "real world problems"
    - Study new document (Burascano) as an addition to the Neurex paper
    - ...

5. Open questions

    - ...

A.6 200/05/10

1. Responsible person for protocol: Annemarie Mayer

2. Participating persons: Mag. DI Roland Schwaiger, Sebastian Gumpold, Annemarie Mayer, Clemens Bauer

3. Topics talked about:

    - what is the "Four Quadrant Problem"
    - its implementation in SNNS

4. New assignments

- Simulate the “Four Quadrant Problem” in SNNS (also try pruning some nodes)
- read paper: “Evolution of Cubic Spline Transfer Functions in a Multi Agent System”
- Calculate the “Four Quadrant Problem” (just like “XOR”)
- finish the program

5. Open questions

   - ...

A.7 200/05/17

1. Responsible person for protocol: Sebastian Gumpold
2. Participating persons: Mag. DI Roland Schwaiger, Sebastian Gumpold, Annemarie Mayer, Clemens Bauer
3. Topics talked about:
   - the solution of the “Four Quadrant Problem”
   - implementing the first pruning algorithm in *netVis*
4. New assignments
   - reflect about ranges and frequencies
   - finish the program
5. Open questions
   - ...

Bibliography

[1] K. Gurney, *An Introduction to Neural Networks.* 1 Gunpowder Square, London EC4A 3DE, UK: UCL Press, 1998. URL: ftp://ftp.shef.ac.uk/pub/uni/academic/N-Q/psology/gurney/psnotes.tar.gz.

[2] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back propagating errors," *Nature 323*, pp. 533–536, 1986.

[3] S. Hashem, "Sensitivity Analysis for Feedforward Artificial Neural Networks with Differentiable Activation Functions." URL: lancair.emsl.pnl.gov:2080/proj/neuron//papers/hashem.ijcnn92.ps.

[4] G. Thimm and E. Fiesler, "Pruning Of Neural Networks," Tech. Rep. IDIAP-RR 97-03, Dalle Molle Institute for Perceptive Artificial Intelligence, Martigny, Valais, Switzerland, February 1997.

[5] J. Sietsma and R. Dow, "Creating artificial neural networks that generalize," *Neural Networks, vol.4, no.1*, pp. 67–69, 1991.

[6] E. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE Transactions on Neural Networks, vol.1, no.2*, pp. 239–242, June 1990.

[7] M. C. Mozer and P. Smolensky, "Using relevance to reduce network size automatically," *Connection Science, vol.1, no.1*, pp. 3–16, 1989.

[8] W. Finnoff, F. Hergert, and H. G. Zimmermann, "Improving model selection by nonconvergent methods," *Neural Networks, vol.6*, pp. 771–783, 1993.

[9] L. Prechelt, "Adaptive parameter pruning in neural networks," Tech. Rep. 95-009, International Computer Science Institute, Berkeley, CA, March 1995.

[10] F. Smieja, "Neural network constructive algorithms: Trading generalization for learning efficiency?." URL: borneo.gmd.de/pub/as/janus/pre-6.ps.

[11] M. Frean, "The upstart algorithm: A method for constructing and training feedforward neural networks," *Neural Computation, vol.2*, pp. 198–209, 1990.

[12] S. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," *Advances in Neural Information Processing Systems 2*, pp. 524–532, 1990.

[13] M. Mezard and J.-P. Nadal, "Learning in feedforward layered networks: The tiling algorithm," *Journal of Physics A, 22(12)*, pp. 2191–2203, 1989.

[14] T. Ash, "Dynamic node creation in backpropagation networks," *Connection Science, vol.1, no.4*, pp. 365–375, 1989.

[15] R. Schwaiger and K. Lang, "Evolution of cubic-spline transfer functions in a multi agent system." Department of Computer Science, University of Salzburg.