Forward Modelling of Resistivity Logs using Neural Networks

P. Demmenie

August 31, 1995

Abstract

In the oil industry, one uses the difference in electrical conductivity between hydrocarbons (oil and gas) and water to determine the hydrocarbon saturation of rocks. To this end, a measuring device is lowered in a well bore to record a so-called *resistivity log*. Due to environmental effects this resistivity log differs from the resistivity of the formation, the *true resistivity*. To invert the measured log to the true resistivity one uses an iterative forward modelling process, involving the numerical solution of differential equations. Although the current modelling algorithms have significantly improved in speed comparing to a few years ago, they are still not fast enough for inversion on a well site. Therefore, we have investigated the feasibility of using neural networks to perform the forward modelling process. Once trained, neural networks are very fast in producing output to certain input.

The one-way mapping between the earth models (true resistivity model and environmental conditions) and the tool response can be learned by a "standard" fully connected net. However, problems arise from the high number of inputs that is needed to represent the earth model (≈ 450) and the high number of connections between the input layer and the hidden layer of the net (≈ 6750). The generalization performance of these nets is not sufficient for the purpose of resistivity log inversion.

We have studied two preprocessing methods to reduce the number of inputs: principal component analysis and the wavelet transform. This preprocessing of the input is quite successful. We have also studied methods to reduce the number of connections in the net by using locally connected nets instead of fully connected nets. We have found an architecture which proves to be quite efficient for this mapping: a convolutional-regression net. This type of net is based on the locally connected nets and shared weights. Due to these shared weights the hidden layer performs a convolution of the input layer. The convolution kernel, the set of shared weights, is learned by the net itself. The weight sharing reduces the number of connections in the net and these constraints improve the generalization ability of the net.

This convolutional-regression net is approximately 100 times faster than the forward model that is used at KSEPL nowadays. The performance of the net is measured in the average relative error between the network outut and the forward model output. The neural net can be used as a fast forward model, when the average relative error lies below 5 %. The convolutional-regression net achieves an accuracy of approximately 8 % on a resistivity log coming from a "real" oil well. Further improvements in accuracy can be achieved by using a more representative training set. Even with less accuracy the neural net could be used as initial start for the forward modelling process.

Preface

This report is the master thesis of Pamela Demmenie for graduating from the Rijks Universiteit Leiden (RUL) at the department of Computer Science. It describes a project that was performed at the Koninklijke Shell Exploratie en Produktie Laboratorium (KSEPL) in Rijswijk from December 1994 until August 1995. The project was supervised by Dr. Guozhong An (KSEPL) and Dr. Ida Sprinkhuizen-Kuyper (RUL).

The report is divided into six chapters. The first chapter provides a background on forward modelling and a short introduction in neural networks. The second chapter describes all the methods we have used in the experiments. These methods involve input reductions and architecture constraints. Chapter 3 and 4 describe the actual experiments for data without and with invasion. In chapter 5 we present the results of the trained neural networks on realistic logging data and in chapter 6 we summarize the conclusions of this project.

All neural net simulations were run on Sparc 10 and Sparc 20 (Unix) stations and should eventually be run on an IBM R600 workstation (the approximation times were measured on this type of workstation). We used the Xerion simulator, versions 3.1 and 4.0. We have also experimented with other simulators: Stuttgart Neural Network Simulator (SNNS) and Aspirin/MIGRAINES. The advantages and disadvantages of these network simulators are outlined in appendix A.

Acknowledgements
First of all I would like to thank my supervisors Guozhong An and Ida Sprinkhuizen-Kuyper for their support and ideas during this project. Of course the project would not have been completed without the help of Niels van Dijk and Leon Hoffman, who have provided the data that we have been working with and helped me in my understanding of the forward modelling process. I also like to thank the students from KSEPL and from the university for their support and company during the nine months I have been working on the project. And last but not least everlasting gratitude go to Pim Bussink for word 2 on page 42.

Contents

List of Tables

List of Figures

1 Forward Modelling of Resistivity Logs

1.1 Resistivity Logging

A formation consists of several layers (beds) of rock, which contain pores. The rock pores can be filled with water or hydrocarbons (oil and gas). Rock and hydrocarbons do not conduct electric currents, whereas the formation water does. One reason for measuring the resistivity is to determine the hydrocarbon saturation within the rocks. The hydrocarbon saturation is an indication for the presence of oil. A simplified expression of this quantitative aspect is exemplified by Archie's equation

$$S_w = \frac{F}{n}\frac{R_w}{R_t} \tag{1}$$

Here R_w is the resistivity of the water in the rock pores, F is the formation factor generally assumed to be derivable from a knowledge of the rock resistivity, S_w is the water saturation (percentage of pore space occupied by water), R_t is the measured rock resistivity and n is an empirically determined saturation exponent. Hydrocarbon saturation S_h is equal to $1 - S_w$ (Moran 1985).

1.1.1 Logging tools

Most of the physics behind the resistivity logging techniques can be found in (Moran 1985). In this section we will discuss the basic ideas behind these techniques.

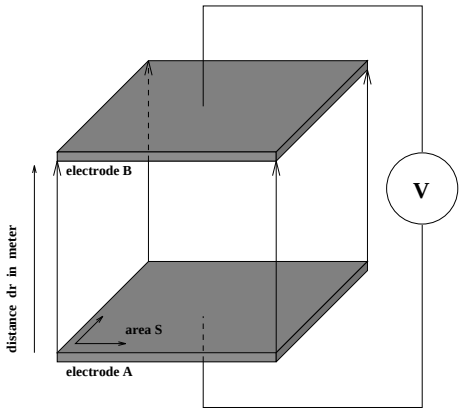

Figure 1: *Laboratory resistivity measuring apparatus with unguarded planar electrodes.*

A sheet of material whose resistivity is to be determined is placed between and in contact with electrodes A and B, both of area S as is shown in

Figure 1. A voltage V is applied to the electrodes, resulting in a current I. If I is distributed uniformly over the area S and is zero outside, an application of Ohm's law for material thickness dr (m) and resistivity ρ (Ωm) yields

$$\Delta V = \frac{I}{S}\,\rho\,dr \tag{2}$$

The resistivity ρ is found by measuring the voltage drop between the electrodes. In practice, the current I will not be uniformly distributed over the discs, but will tend to be maximum at the edges and minimum in the centre. To improve on this scheme, one splits the disc so that one has a small central disc of area S_0 carrying a current I_0 surrounded by the remainder of the discs held at the same potential V as S_0. This will result in nearly constant current density over S_0. The measured current I_0 will be "focused". The resistivity is now given more accurately than before by

$$\rho = \frac{S_0}{dr}\frac{\Delta V}{I_0} \tag{3}$$

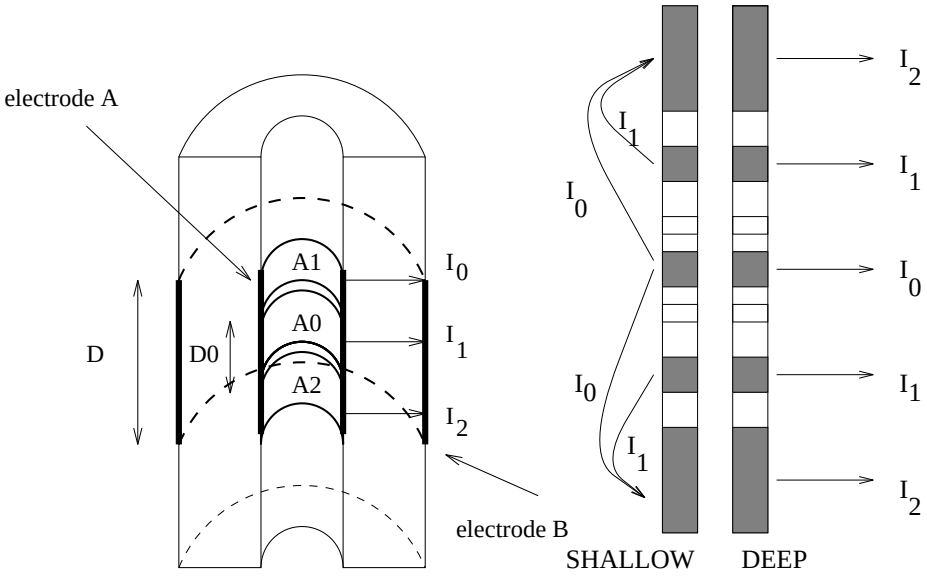

Figure 2: *Modification of the laboratory apparatus using cylindrical electrodes instead of planar electrodes and the Dual-Laterolog.*

This idea is the basic ingredient of the Laterolog. The real Laterolog uses cylindrical electrodes instead of planar electrodes as indicated in Figure 2 (left). The Dual Laterolog consists of a Shallow (Pseudo) Laterolog and a Deep Laterolog. The Deep Laterolog has its current return electrode B remotely at the surface, which results in currents as shown in Figure 3. The

Deep Laterolog reads far into the formation. The Shallow Laterolog has its current return electrode placed above and below the electrode A, which makes the currents bend back to the tool as shown in Figure 3. This Laterolog reads the formation close to the tool, which makes it more sensitive to the invaded zone (see Section 1.1.2).
Both measurements are made simultaneously by using different frequencies; a relatively high frequency for the Shallow-Laterolog and a very low frequency for the Deep Laterolog.

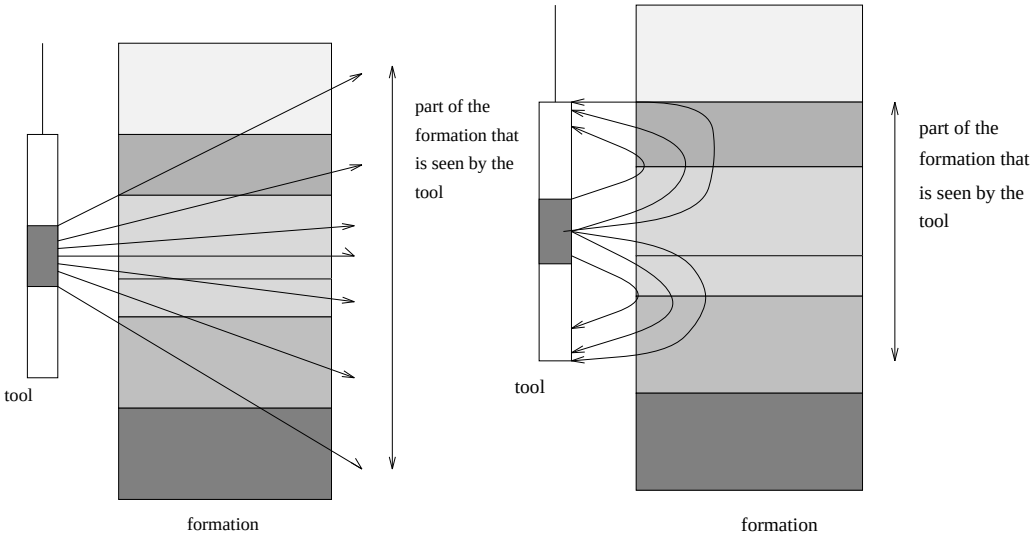

Figure 3: *Input to the Deep Laterolog (left) and the Shallow Laterolog (right). The Deep Laterolog gets information from a larger part of the formation.*

1.1.2 Environmental effects

Several anomalies, like invasion, dipping beds, washed-out bore holes and very resistive formations, are encountered when measuring the resistivity of the formation. These anomalies have an effect on the tool response, which are difficult to model. Situations in which these enviromental effects occur are shown in Figure 4 and more detailed descriptions can be found in (Gianzero 1977), (Asquith 1982), (Chemali, Gianzero & Strickland 1983) and (Chemali, Gianzero & Su 1988).
Wells are drilled with rotary bits. Special drilling mud is used as circulating fluid. The mud removes cuttings from the well bore, lubricates and cools the drill bit and helps maintaining an excess of bore hole pressure over formation pressure, which prevents blow-outs.

This pressure difference forces some of the drilling fluid to invade porous and permeable formation. In this process of invasion solid particles (clay minerals from the drilling mud) are trapped on the side of the bore hole and form a so-called mudcake. The part of the formation which is invaded by mud filtrate is called the invaded zone.

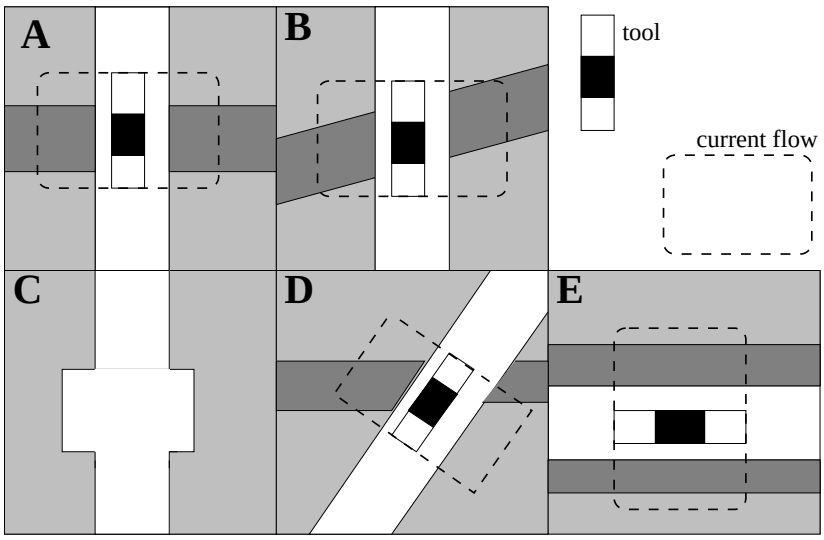

Figure 4: *Environmental effects: (A) ideal situation, (B) dipping formation, (C) caved bore holes, (D) deviated bore holes and (E) horizontal bore holes.*

The size of the bore hole and the mudcake resistivity influence the measured resistivity. The smoothing of anomalies on the log by the bore hole is quite effective in moderately saline to very saline mud, and less effective in fresh mud. The size of the bore hole and the resistivity of the drilling fluid are taken fixed in our study.

In dipping formations the Dual Laterolog-curves vary slowly across bed boundaries. The apparent bed thickness is increased in predictable proportions and the error due to shoulder bed effect (the influence of the beds adjacent to the current bed) is different from the non-dipping case.

In caved bore holes (wash-out) the Shallow Laterolog is sensitive to the effect of the increased hole diameter. Anomalous readings of the Deep Laterolog occur only at abrupt changes in hole diameter.

Very resistive formations and formations with high resistivity contrasts between beds also affect the tool readings. Formations like limestone and dolomite can have resistivities over 2000 Ωm, while most formations have a resistivity between 1 and 70 Ωm.

The tool response is essentially immune to a small eccentering of the tool. Only the Shallow Laterolog exhibits some sensitivity to eccentering in large bore holes.

The overall effect of an elliptical bore hole is that it produces characteristic responses which lie between those obtained in two circular holes with diameters equal to the major and minor axes of the elliptical hole.
The shoulder bed correction required for the Deep Laterolog is much less important in conductive mud than it is in non-conductive mud. The Shallow Laterolog has much less shoulder bed effect than the Deep Laterolog, especially for bed thicknesses above ten feet.
Deviated bore holes and horizontal bore holes also give different tool readings. Compare the current flow in the ideal case (Figure 4) and these types of bore holes.

1.2 Inversion by Forward Modelling

1.2.1 Inversion

The found resistivity logs, Shallow- and Deep-log, have to be inverted to the true resistivity of the formation. This is done by an iterative forward modelling process, which is sketched in Figure 5. The actual field logs are first corrected with chartbook corrections (tornado charts for example; for bore hole size, invasion, etc.) Then an initial guess for the formation model is made. This trial model includes a description of the bore hole and formation geometry and "parameter values" — numbers assigned to variables such as bore hole diameter and bedding dip, thickness and resistivity (Anderson & Barber 1990). Then the tool physics is used to compute an expected log, which is compared with the actual field log. If the match is not good enough, the initial trial model is altered and the calculation repeated. This process is iterated until the two logs match satisfactorily.
The process consists of two steps: the function approximation from the guessed formation model to the expected Deep- and Shallow-log (so-called Forward Modelling) and a matching procedure. The latter procedure implies a minimalization of a matching error between the two logs. This can be done by hand or using sophisticated algorithms like the least-squares and maximum entropy methods (Anderson & Barber 1990).

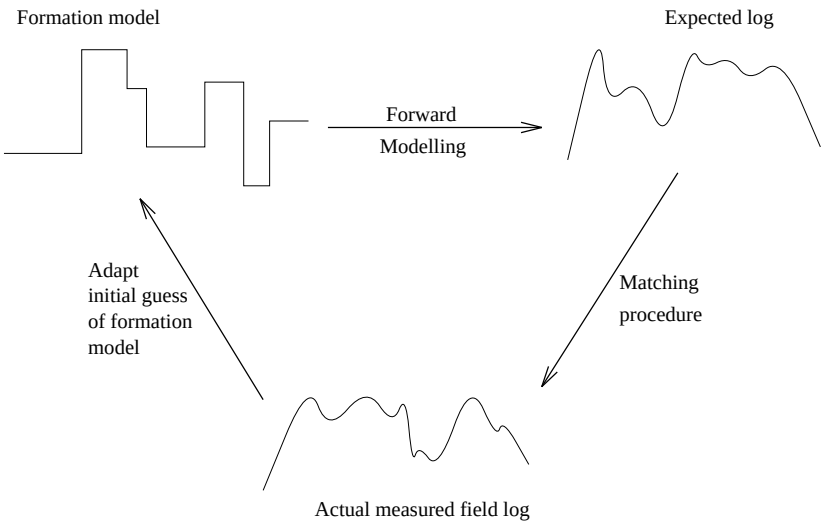

Figure 5: *The inversion process and forward modelling of resistivity logs.*

1.2.2 Forward Modelling

The forward modelling part of the inversion is the most time-consuming part of the process. To compute the Laterolog response for a given electrode array

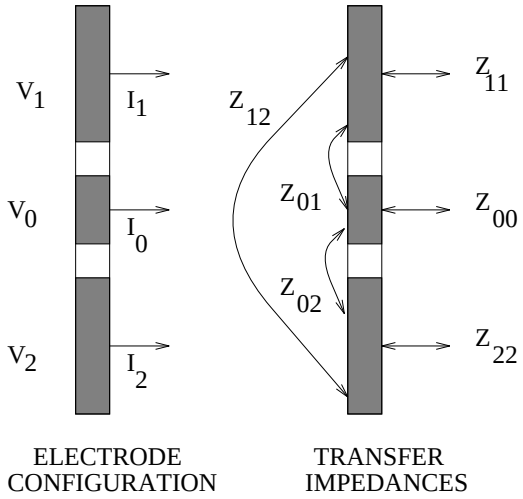

Figure 6: *Transfer impedances.*

and distribution of resistivities, it is sufficient to determine the associated transfer impedances. The transfer impedance Z_{ij} is equal to $\frac{V_j}{I_i}$, where V_j is the voltage measured at part j of the electrode configuration and I_i is the current emitted by part i. A specific electrode configuration and its transfer

impedances are shown in Figure 6. For the computation of the response in simplified models (concentric cylindrical boundaries or plane boundaries) one can use analytical approaches, but with extended electrodes for example, the problem can only be handled by numerical methods. In the case of dipping layers, the problem is of such complexity that no results have as yet been published (Moran 1985). The determination of the transfer impedances Z_{ij} involves the solution of Laplace's equation in two space variables under certain boundary conditions.
There are a number of methods that can be used to solve these boundary value problems, we mention the Finite Element Method, the Boundary element or finite-difference technique and a Hybrid Method (Gianzero, Lin & Su 1985).
The *Finite Element Method* is a numerical method based on an energy principle (Chemali et al. 1983). It can be shown that Laplace's equation is a direct consequence of minimizing the total energy of the system.
The *Hybrid Method* is a separation of variables approach where the radial (horizontal) dependence is treated numerically and the axial (vertical) dependence analytically. It combines the mode concept in wave-guide theory with the Finite Element Method. This method, employed by Gianzero, has been able to simulate a 100 foot log with 25 beds in less than twelve minutes on an IBM 3081, which is approximately 8 times faster than the Finite Element Method (Gianzero et al. 1985).

1.3 Neural Networks

In this section we will only describe the basic ingredients of neural networks. A good introduction can be found in (Haykin 1994).
A Neural Network consists of a number of layers that consist of nodes (neurons). A neuron receives input from the neurons it is connected to and can in its turn serve as input to other neurons. A connection between node i of a certain layer and a node j of another layer is characterized by a weight w_{ji}. The total input u_j for node j is

$$u_j = \sum_{i=1}^{n} w_{ji} x_i \tag{4}$$

where x_i is the activation of an input node i and the summation runs over all n nodes that node j receives input from.

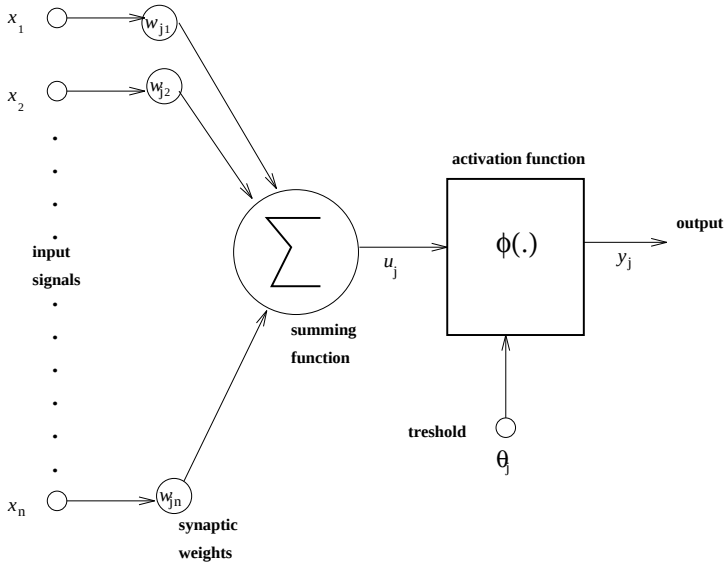

Figure 7: *A diagram of neuron j.*

Now this node uses an activation function to determine its own activation and output

$$y_j = \phi(u_j - \theta_j) \tag{5}$$

Here ϕ is a certain activation function, which is usually a sigmoid function for the hidden nodes and a linear function for the output node in regression problems, and θ_j is a certain threshold for node j. Figure 7 shows a visualization of this process.
This procedure of determining the input and the output of a node is done for all nodes, except the input nodes. The input nodes receive their activation directly from outside (from a file for example). In this case the input is some kind of representation of the formation model.
The time that is needed to determine the output of the neural net for a certain input can be expressed in the number of connections (weights) of the net. The operations done by the net are weight multiplications, summations and calculating the activation.
Determining the output of the net will not take much time. However, the process of making the net learn the problem, training, takes considerably more time. When using neural networks, the following aspects are important for the training time and the generalization ability. A net is said to generalize well if it produces (nearly) correct output for input that was not used during its training.

- The architecture of the net. Large nets (high number of weights) learn slowly and usually do not generalize well. This is due to the fact that the neural net "remembers" its training examples if it has too much freedom (too many weights).

- The training method. Several training methods for supervised learning exist, for example back-propagation, conjugate gradient, steepest descent, momentum descent. For some of these methods the tuning of certain parameters is very important and difficult. Supervised learning means that the net produces some output (the actual response) and corrects its behaviour according to the correct output (desired response). For multilayer feedforward networks (networks that have connections directed from the input to the output), the most widely used algorithm is the back-propagation algorithm. There are two phases in the BP-learning. In the first phase, the forward phase, the input signals propagate through the network layer by layer, eventually producing some response at the output of the network. The actual response so produced is compared to the desired response. The error signals generated by this comparison are then propagated backwards through the net in the second, the backward phase. More information on training methods can be found in (Haykin 1994).

- The training set. The training set should be representative for the problem, otherwise the net is not able to learn the problem or to generalize well. With conjugate gradient each training example is evaluated during training (batch training), so a large training set causes a long training time. More on the creation of the training set can be found in Section 1.3.3.

1.3.1 Network architecture

The network architecture is a description of the layers, the number of nodes per layer and the connections between the layers; it describes what the net looks like. For example a fully connected neural net is a net in which the nodes of one layer receive input from all nodes in the previous layer. This is the most common architecture, but it leads to a large number of weights. Another method is a locally connected network, which is described later when we turn to receptive fields and convolutional networks.

The architecture determines the number of weights and has certain implications for the learning and generalization ability of the net.

1.3.2 The Learning Method

A net uses a learning method to minimalize the error in the net. The training set consists of P patterns (x_p, d_p). Here, x_p is the input pattern p and d_p is the desired output for this input pattern. For a certain input x_p the network calculates an output a_p. The error is given by the quadratic sum

of the difference of this output and the desired output d_p:

$$E = \sum_{p=1}^{P} (d_p - a_p)^2 \tag{6}$$

Here, P is the number of examples. The idea behind the learning methods is that the weights are adapted during training so that E is minimalized. This is done by a down hill technique, gradient descent. The weights are adapted in the direction with the steepest descent

$$\begin{pmatrix} \text{Weight} \\ \text{correction} \\ \Delta w_{ji} \end{pmatrix} = \begin{pmatrix} \text{learning-rate} \\ \text{parameter} \\ \eta \end{pmatrix} \cdot \begin{pmatrix} \text{local} \\ \text{gradient} \end{pmatrix} \cdot \begin{pmatrix} \text{input signal} \\ \text{of neuron } j \\ \text{to neuron } i \end{pmatrix} \tag{7}$$

The weight correction depends on a learning parameter η, the local gradient and the input signal of neuron i to neuron j. If E no longer decreases the training stops.
We have used the conjugate gradient (CG) method, which is an adaptation of the normal gradient descent method. It avoids the zigzag path followed by the gradient descent method, by incorporating an intricate relationship between the direction and the gradient vectors. The CG method is the most convenient algorithm, because it needs no tuning parameters and it is faster than normal back-propagation if the size of the training set is not too large. We have used the Xerion (version 3.1 and 4.0) simulator to train the networks (for more details on neural network simulators see Appendix A).

1.3.3 The training set

The input has to be presented to the net, so the physical model must be transformed into a set of numbers that function as activations of the input layer. How this is done, is discussed in the next chapter. Now we will focus on the *scaling*.
The input to the net must be scaled so the values lie more or less in the range $[-1, 1]$. As we have seen the net calculates a weighted input. If two inputs have a very high ratio (for example $i_1 = 1$ and $i_2 = 1000$) the weights also have a large variation. So if the inputs do not lie in a small range, the weights will in general be far apart as well. This will slow down the training, because a larger weight space has to be searched for an optimum set of weights. Another point of view is that large input values will have more influence on the activation of the nodes they are connected to. In this way we have already build some prior knowledge into the net. To avoid this, we scale all the inputs to a small range. When certain inputs are important, the net can learn that itself. More on this subject of scaling can be found in Section 2.1.3.

A training set should be sufficiently large. Although there is no general prescription of how large a training set should be, there are some "rules", like the following from Baum and Hassler (Haykin 1994). A network will almost certainly provide generalization (see next section), provided that the following two conditions are met:

1. The fraction of errors made on the training set is less that $\frac{\epsilon}{2}$.

2. The number of examples, P used in training is

$$P \geq \frac{32W}{\epsilon} \ln\left(\frac{32M}{\epsilon}\right) \tag{8}$$

   Here, W is the total number of weights and M is the total number of hidden nodes. This formula provides a distribution-free, worst-case estimation for the size of the training set for a single-layer neural network that is sufficient for a good generalization.

A training set should also be representative. This means that the examples in the training set are randomly generated and distributed over the whole input space.

1.3.4 Generalization

Although a neural net can learn any input-output mapping, its applicability is determined by its ability to predict outputs to inputs it has not seen during training, which is called generalization. Generalization is influenced by three factors:

- the size and representativeness of the training set,
- the architecture of the network,
- the physical complexity of the problem at hand.

With too few examples the net just memorizes the training set and exhibits poor generalization. If the number of examples is more than the number of weights, the net will generalize better. Widrow's rule of thumb (Haykin 1994) comes from equation 8 and states that in practice we need a training set size of approximately 10 times the number of weights when the error on the training set is 10 %.
In our project, we will compare different architectures on one specific training set. The architecture that gives the best results, in terms of training error, generalization (testing) error and complexity (number of weights), will be trained on a larger training set.

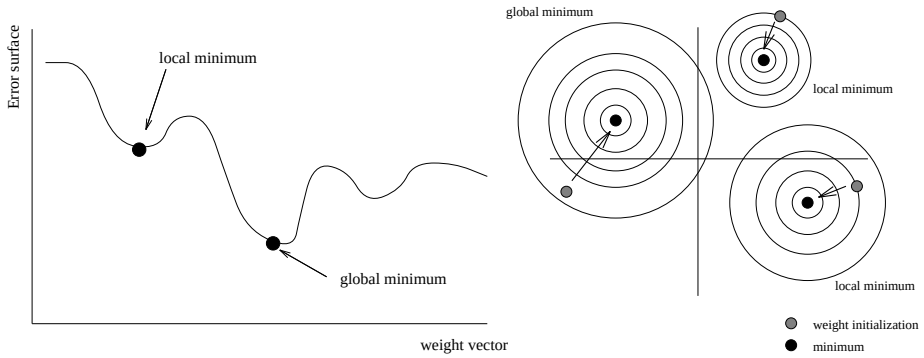

Figure 8: *Local minima in the error surface.*

1.3.5 Local minima

In Figure 8 (left) the error surface, belonging to specific weight vectors, is shown. The learning algorithm that we use is basically a hill-descending technique, which may cause the algorithm to get trapped in a local minimum in the error surface, although we are interested in the global minimum. The algorithm gets trapped, because it cannot find a direction which makes the error smaller than the previous one. But somewhere else in the weight space there exists another set of synaptic weights for which the cost function is smaller than the local minimum in which the network is stuck. One method to avoid local minima is retraining the neural net with different weight initializations. This process is shown in Figure 8 (right); with different weight initialization the net will converge to different minima.

1.4 Forward Modelling using a Neural Network

The goal of this project is to obtain a good approximation of the Deep- and the Shallow-log. The formation is described by a number of beds. For each bed a number of radial zones are given and for each radial zone the resistivity (Ωm) and its size (inch) is given. The first radial zone is the bore hole with its radius and the resistivity of the drilling fluid, the next radial zone describes the invasion (if there is any) and the last radial zone describes the true resistivity. All this can be described with a model like shown in Figure 9 (the corresponding formation is shown on the right side).
The tool response consists of two continuous logs (the Shallow-log and the Deep-log) like shown in Figure 10 and Table 1. The model shown in Figure 9 contains 80 beds. The first bed has 3 radial zones and it starts at minus infinity (this is the same for all models, indicating an infinite shoulder bed). The first radial zone of this bed (the bore hole) has a radius of 4.25 inch and the mud resistivity is 0.05 Ωm. The second zone has a radius (dxo) of 45 inch and a resistivity (Rxo) of 1.90 Ωm. And finally the third zone has a resistivity (Rt) of 27 Ωm. The last radial zone has a radius of infinity

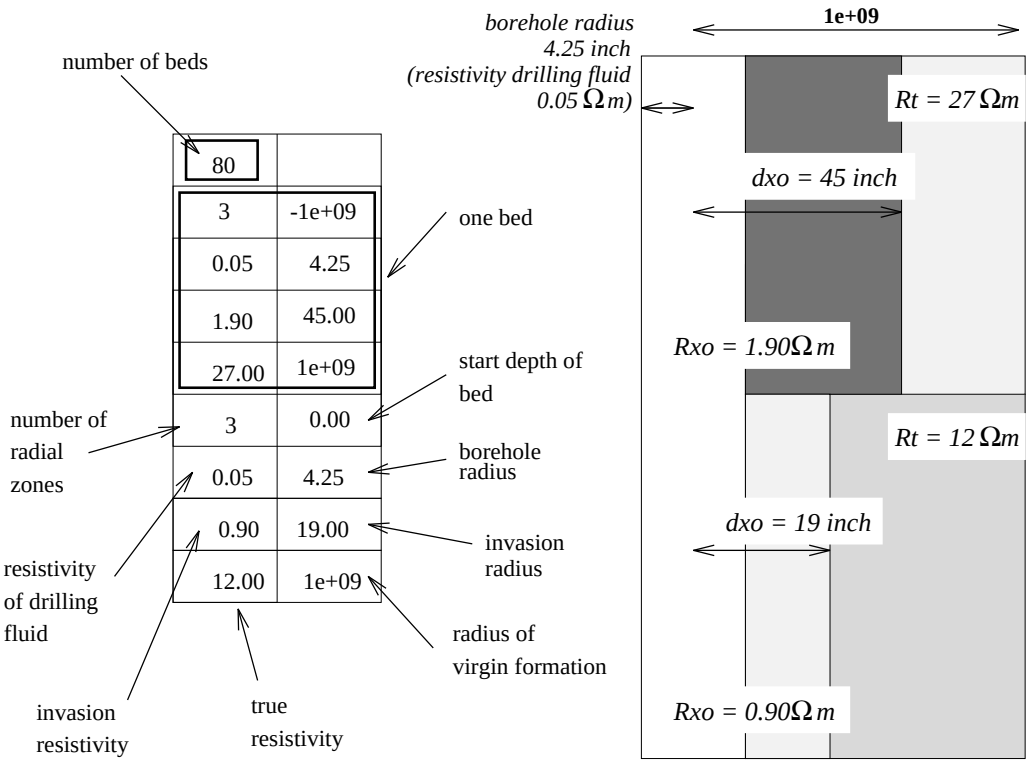

Figure 9: *A formation and its model.*

(otherwise there would be another radial zone).
The neural net can be used as a fast forward model when the average relative error between the forward model output d and the neural network output a lies below 5 %.

Table 1: *The tool response.*

depth	**LLd**	**LLs**
0.0	.110610E+02	.330318E+01
0.2	.108760E+02	.325301E+01
0.4	.106962E+02	.320733E+01
0.6	.105092E+02	.316235E+01
⋮	⋮	⋮

The earth models in this project are created by assigning random numbers within a certain range for the parameters *Rt*, *Rxo* and *dxo*. The bore hole radius and resistivity of the drilling fluid are fixed to 4.25 inch and 0.05 Ωm respectively. The tool responses to these models are calculated with the forward model that is used at KSEPL.

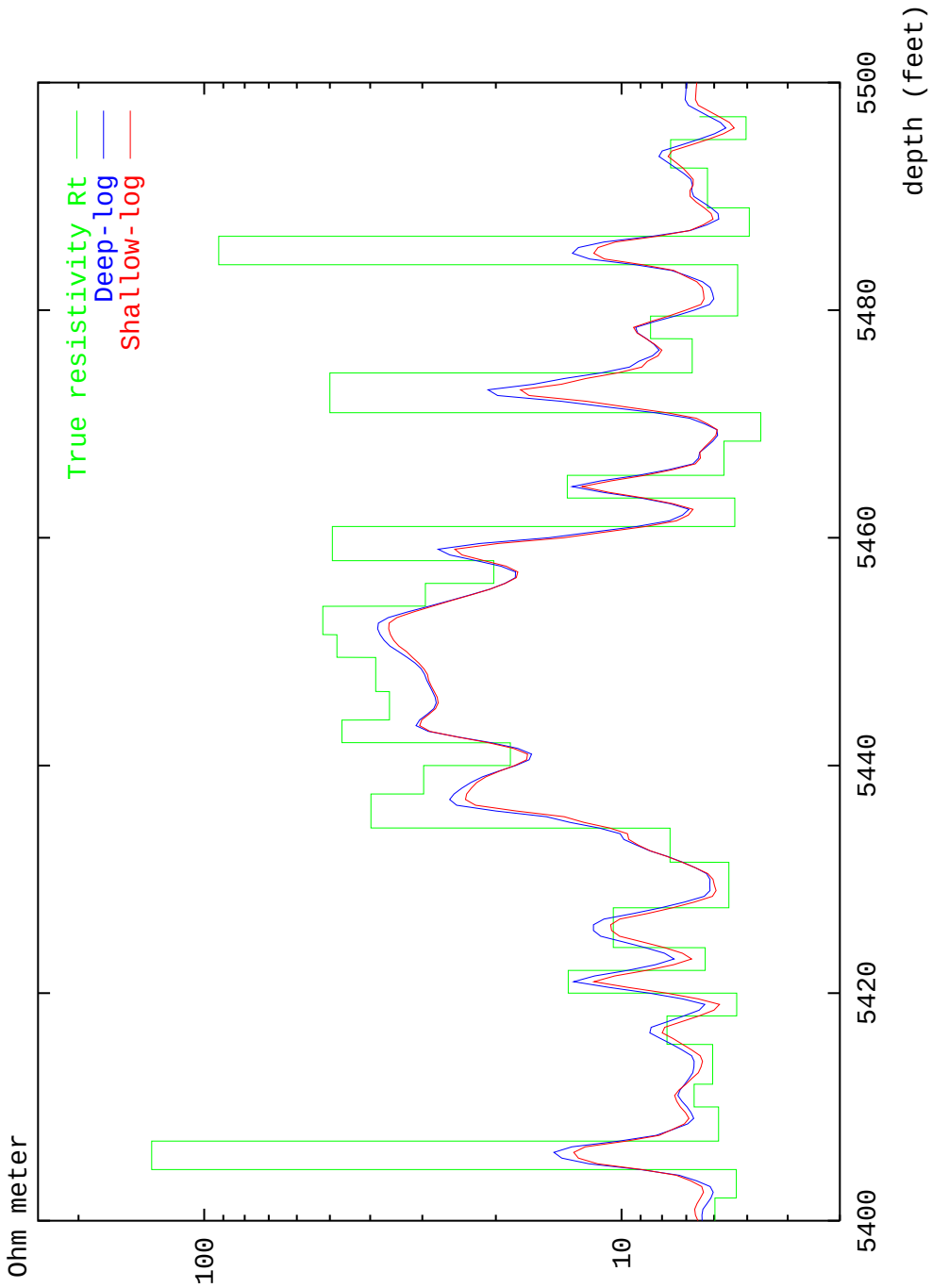

Figure 10: *Part of a model with input signal Rt and output signals Deep-log (LLd) and Shallow-log (LLs). The input signals Rxo and dxo are omitted.*

2 Input representation and architecture design

As described in Section 1.3, the training time mainly depends on the size of the net and the size of the training set. To decrease the training time and increase the generalization capability, we like to keep the net as small as possible. This can be achieved by choosing a compact input representation and by decreasing the number of free variables (weights) in the net. The input representation has to be compact and appropriate to the problem.
To facilitate the learning process, we employ two methods. Firstly we preprocess the input and secondly we force certain constraints on the network architecture. The purpose of the first method is to create an intermediate representation of the input which simplifies the problem for the neural network while making only a small computational overhead. The second method contributes to facilitating learning only if the resulting structure reflects the designer's a priori knowledge of the problem. Otherwise the network is a priori biased towards wrong solutions.

2.1 Input representation

We are looking for a compact and appropriate input representation. Given an earth model, described by $n \times (2 + r \times 2) + 1$ values (for n beds of r radial

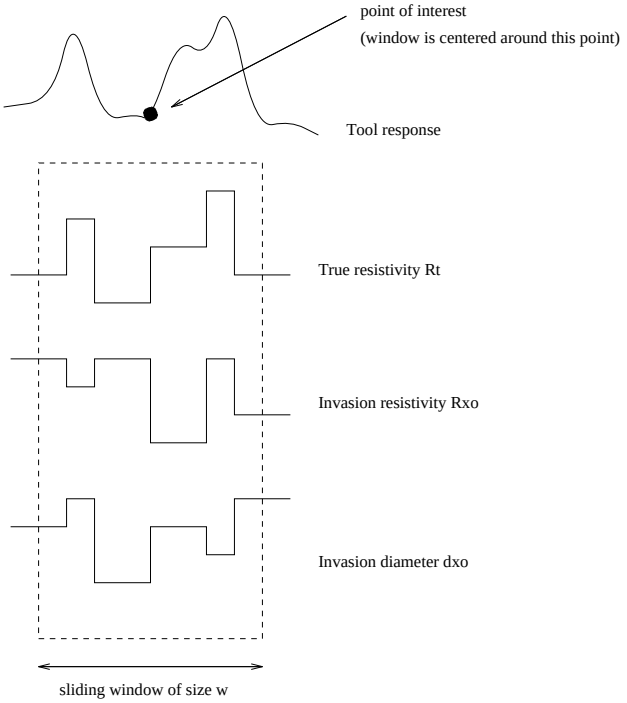

Figure 11: *Example of sliding window input representation. A sliding window of size w is placed around the point of interest along the input models.*

zones), the net should be able to produce the tool response at any depth. The tool response at a specific depth is used as the target of the neural net (the target is also called the desired output). For one formation we have a tool response of m feet, sampled every t feet. This produces $\frac{m}{t} + 1$ targets per log. But what do we use as input to produce this target? We assume that a part of the formation, centered around a certain depth, is responsible for the tool response at that depth. This part of the formation is called a sliding window. The sliding window approach is shown in Figure 11.
The sliding window is described in two ways: by discretizing the formation model in the sliding window and by describing the beds that lie in the window.
The first method is quite straight forward. It samples the part of the formation that lies in the window, without using knowledge about the input or relations between inputs.
The second method looks more like the original model description and uses the fact that the formation is described by beds. In the model each bed is described by a number of values, which could be seen as attributes of that bed. In the attributes approach we describe the beds that lie in a window, centered around the point of interest by a number of features.

2.1.1 Discretized sliding window

When there is invasion, see Section 1.1.2, the formation model contains three variables Rt, Rxo and dxo. When there is no invasion the formation model contains only the variable Rt. We use a sliding window of fixed size w, which is placed along the input logs around the point of interest. The models are sampled within this window with a sampling period s, resulting in $\frac{w}{s} + 1$ inputs when there is no invasion and 3 times this number when there is invasion.
The following aspects should be taken into account in determining the size of the sliding window:

- The size of the tool. The currents flowing from the tool penetrate the formation. The currents of the Shallow Laterolog penetrate the formation and return to the top and bottom of the tool. The part of the formation that the tool receives information from, is at least as large as the tool itself. The tool is approximately 30 ft, which gives an indication that the window size should also be at least 30 ft.

- The type of target log (Deep or Shallow). As shown in Figure 3, the Deep Laterolog receives information from a larger part of the formation. The currents of the Shallow Laterolog return to the tool itself and do not penetrate the formation much. This indicates that the window size for the Deep-log should be larger than for the Shallow-log

- The size of the beds. What happens if the window is smaller than the largest bed in the formation is shown in Figure 12. The sliding window is located more than once in the same bed, producing the same input, but possibly not the same target. Now the neural net has to learn $f(x) = y_1$ and $f(x) = y_2$, making the the problem non-deterministic. Conflicting examples make it very difficult for the net to learn the problem, because it adapts its weights to reproduce two different targets for one input.

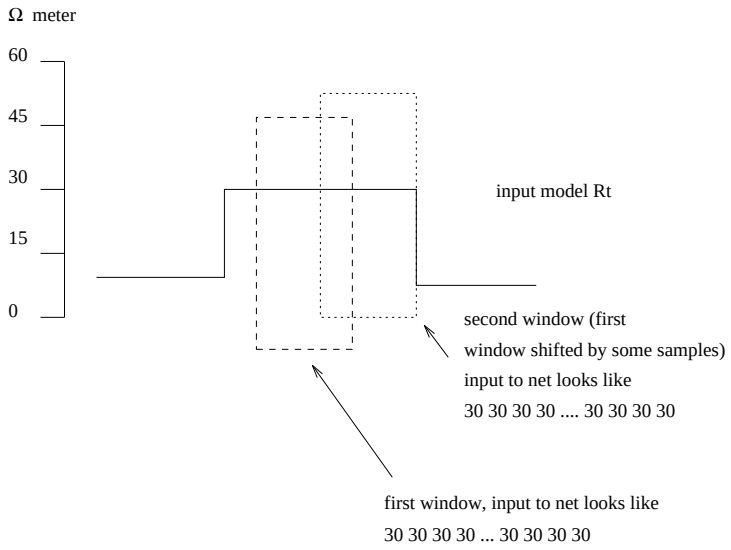

Figure 12: *When the sliding window is smaller than the largest bed in the model, the input to the net will be the same for the sketched situation, although the target differs a lot.*

We expect the net that is trained on the Shallow-log will perform better, because it needs a smaller window than the Deep-log. The sampling period of the target log determines the number of examples we obtain per log. In real applications one takes two logging points per feet, but for efficiency reasons, we have taken five logging points per feet. The sampling period in the sliding window is important for the resolution of the bed boundaries. When we use a sampling period of s feet, we can describe a bed boundary with s feet resolution. When we want to have at least the same accuracy as the target log, we should use the same sampling period for the sliding window as for the target log. In this case that would mean using a sampling period of 0.2 feet.

This input representation is not very compact. When we use a sliding window of 25.4 feet and a sampling period of 0.2 feet, we have 384 inputs (128 per variable and 3 variables when there is invasion).

2.1.2 Attributes

The input model describes a number of bed boundaries, each described by a number of radial zones. The neural net requires the number of inputs to be fixed for each sample. We use a fixed size window and describe the beds that occur within this window. If the window contains less than the fixed number of beds, we add "default" beds. These beds function as infinite shoulder beds. The order in which the beds are presented to the neural net is important. The location in the input of the bed that has most influence on the target signal for example (probably the bed in the center of the window) should be fixed (in our case it is presented first). Then the beds adjacent to this bed are presented and the beds adjacent to those beds and so on.
Each bed is described by a number of attributes (also called features). The contrast between two beds is defined as v_1/v_2 and the difference as $v_1 - v_2$ for values v_1 and v_2. The attributes we use are:

1. the true resistivity of the bed (Rt);
2. the invasion resistivity of the bed (Rxo);
3. the invasion radius of the bed (dxo);
4. the inverse distance to the logging point (points close to the bed boundary are considered to be more important than points that lie further away);
5. the contrast between the true resistivity of this bed and the bed that lies below this bed;
6. the contrast between the invasion resistivity of this bed and the bed that lies below this bed;
7. the contrast between the invasion radius of this bed and the bed that lies below this bed;
8. the difference between the true resistivity of this bed and the bed that lies below this bed;
9. the difference between the invasion resistivity of this bed and the bed that lies below this bed;
10. the difference between the invasion radius of this bed and the bed that lies below this bed.

The default beds have no contrast (1) and no difference (0) with their adjacent beds. The inverse distance to the logging point is 0 for the default beds (the beds continue to infinity). If we describe n beds in the chosen window, this results in $10n$ inputs.

This approach is more compact than the discretized sliding window approach, but it is difficult to choose appropriate features that will facilitate the learning process. We assume the contrasts and differences are important and describe the problem well. If this is not the case, the learning will not be facilitated. It could even make it difficult for the neural net to learn the problem, when these features are not describing the problem well.

2.1.3 Input and output scaling

As discussed in the first chapter, the input should be approximately scaled to the domain $[-1, +1]$. For the true resistivity and the tool response we use a combination of logarithmic and normalization scaling and for the other variables a normalization scaling. The true resistivity (and the tool response) can range between 1 and 2000 Ωm. To reduce this range, we use a logarithmic scaling. The range of the other variables, the invasion resistivity and the invasion radius, are much smaller and therefore we do not use the logarithmic scaling on these variables. The scalings take the following form

$$x'_{\log} = \ln(x) \tag{9}$$

$$x'_{\text{norm}} = \frac{x - \mu}{\alpha \times \sigma} \tag{10}$$

where, x is the original input or output value, μ the mean and σ the standard deviation of the variable x. The factor α is applied to make the scaled range even smaller (in the experiments $\alpha = 2$).
A normalized Gaussian distribution is shown in Figure 13. From this figure we find that for $\alpha = 1$, 68.27 % of the values (of this specific distribution) lie between -1 and +1. When we use $\alpha = 2$ we find that 95.45 % of the values lie between -1 and +1.

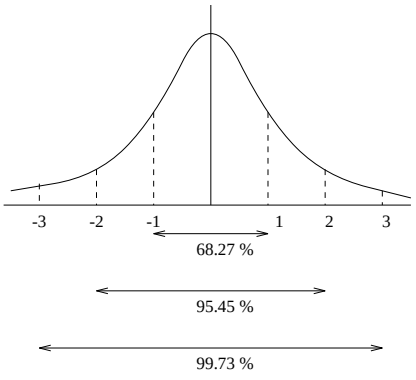

Figure 13: *Standard normal distribution.*

In the attribute description approach only a normalized scaling method is applied.

This combined scaling method of the tool response has an attractive consequence for the minimalization of the network error. For more details on the minimalization of the network error see Section 2.4.

2.2 Preprocessing

In this section we will describe the preprocessing methods we have used in the project. The purpose of preprocessing is to find an intermediate input representation that facilitates the learning process. Another advantage of preprocessing is that we can reduce the number of inputs. The preprocessing was only applied to the input that was created by the discretized sliding window approach.

We can view the input in two ways. Firstly as three N-dimensional vectors $\boldsymbol{Rt}$, $\boldsymbol{Rxo}$ and $\boldsymbol{dxo}$ and secondly as three functions of the depth x, $Rt(x)$, $Rxo(x)$ and $dxo(x)$. The latter is actually also a vector, because the functions are equally sampled within an interval $[1, N]$. In the first case we project the input to an M-dimensional subspace spanned by the principle components of the input. In the other case we use a set of orthogonal basis functions, the *Haar* wavelets, to project the input.

The disadvantage of input reduction is the loss of information. Hopefully, the information that is lost has a negligible influence on the learning and generalization of the net.

2.2.1 Principal Component Analysis

The following description of the principal component analysis is taken from (Hertz, Krogh & Palmer 1991) .

A common method from statistics for analyzing data is *principal component analysis* (PCA), also known as the Karhunen-Loève transform in communication theory. The aim is to find a set of M orthogonal basisvectors (eigenvectors) that account for as much as possible of the data's variance. Projecting the data from their original N-dimensional space onto the M-dimensional subspace spanned by these vectors performs a dimensionality reduction that retains most of the intrinsic information in the data.

The first principal component is taken to be along the direction with the maximum variance. The second principal component is constrained to lie in the subspace perpendicular to the first. Within that subspace it is taken along the direction with the maximum variance. Then the third principal component is taken in the maximum variance direction in the subspace perpendicular to the first two, and so on. An example is shown in Figure 14. In general it can be shown that the kth principal component direction is along an eigenvector direction belonging to the kth largest eigenvalue of the

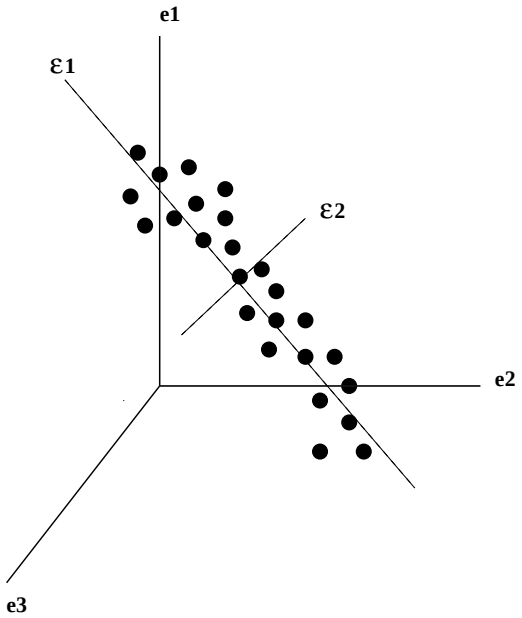

Figure 14: *Most of the variation of the data lies in the direction of* ϵ_1.

covariance matrix. This matrix is calculated for P patterns by

$$\text{Covariance}(i,j) = \frac{\sum_p (x_i^p - \overline{x_i})(x_j^p - \overline{x_j})}{P} \tag{11}$$

Here, x_i^p is input i of pattern p, x_j^p is input j of pattern p and $\overline{x_i}$ and $\overline{x_j}$ are the means of input i and input j respectively. Then the matrix is diagonalized and the eigenvalues are calculated.
The original input vector $\boldsymbol{x}$ (***Rt***, ***Rxo*** or ***dxo***) is written as

$$\boldsymbol{x} = x_1\boldsymbol{e}_1 + x_2\boldsymbol{e}_2 + ... + x_N\boldsymbol{e}_N \tag{12}$$

where $\boldsymbol{e}_1$, ..., $\boldsymbol{e}_N$ are the original basisvectors as shown in Figure 14. We can also write the input vector in another orthogonal set of basisvectors, the eigenvectors $\boldsymbol{\epsilon}_1, ..., \boldsymbol{\epsilon}_N$

$$\boldsymbol{x} = (\boldsymbol{\epsilon}_1 \cdot \boldsymbol{x})\boldsymbol{\epsilon}_1 + (\boldsymbol{\epsilon}_2 \cdot \boldsymbol{x})\boldsymbol{\epsilon}_2 + ... + (\boldsymbol{\epsilon}_M \cdot \boldsymbol{x})\boldsymbol{\epsilon}_M + ... + (\boldsymbol{\epsilon}_N \cdot \boldsymbol{x})\boldsymbol{\epsilon}_N \tag{13}$$

To reduce the size of the vectors from N to M, we project this vector to

$$\boldsymbol{x}' = (\boldsymbol{\epsilon}_1 \cdot \boldsymbol{x})\boldsymbol{\epsilon}_1 + (\boldsymbol{\epsilon}_2 \cdot \boldsymbol{x})\boldsymbol{\epsilon}_2 + ... + (\boldsymbol{\epsilon}_M \cdot \boldsymbol{x})\boldsymbol{\epsilon}_M \tag{14}$$

Instead of using N values $x_1, x_2, ..., x_N$ from the original input vector we use M values $\boldsymbol{\epsilon}_1 \cdot \boldsymbol{x}, \boldsymbol{\epsilon}_2 \cdot \boldsymbol{x}, ..., \boldsymbol{\epsilon}_M \cdot \boldsymbol{x}$ of the projected input vector as inputs for the neural net.

To calculate the percentage of information that is lost by this projection, we first have to write $\boldsymbol{x}'$ in the $\boldsymbol{e}_i$ basisvectors:

$$\boldsymbol{x}' = a_1\boldsymbol{e}_1 + a_2\boldsymbol{e}_2 + ... + a_N\boldsymbol{e}_N \tag{15}$$

The values a_i are calculated by

$$a_i = \sum_{n=1}^{N} x_n \sum_{m=1}^{M} \epsilon_{mn}\epsilon_{mi} \tag{16}$$

The information that is lost by this projection can be calculated (for P patterns in the input file) by

$$\text{Loss of information} = \frac{1}{P}\sum_{p=1}^{P} \frac{||\boldsymbol{x}'^p - \boldsymbol{x}^p||}{||\boldsymbol{x}^p||} \tag{17}$$

2.2.2 Wavelet transform

We can write any function $f(x)$ as a weighted combination of other orthogonal basisfunctions $f_i(x)$

$$f(x) = \sum_i c_i f_i(x) \tag{18}$$

Instead of using the values of $f(x)$ (as inputs), which may be infinite when $f(x)$ is a continuous function, we use the coefficients c_i. We choose functions $f_i(x)$ with properties, that make it easier to manipulate with those functions than with the original function $f(x)$.
A commonly known method is the Fourier transform, where the orthogonal basisfunctions are $\sin(ax)$ and $\cos(ax)$. These basisfunctions allow you to describe the function on different frequency levels.
Our input signals $Rt(x)$, $Rxo(x)$ and $dxo(x)$ have a very special shape: they are all rectangular "functions". The Fourier transform is not appropriate in this case, because we would need an infinite number of coefficients and basisfunctions to correctly model the discrete transitions. There is however another interesting set of orthogonal basisfunctions, called wavelets. We are especially interested in the simplest wavelets, the so-called Haar wavelets. These wavelets are blockfunctions as shown in Figure 16. What is so interesting about wavelets is that the input can be described on different detail levels. All coefficients, except the first, describe a specific property over a part of the window (varying in size and location). The property they describe is the difference between the average value over the first and second half of their part of the window. The first coefficient describes the average over the whole window.

We will first give a short introduction on wavelets and how the coefficients are calculated (taken from (Strang 1989)) and then we will explain why the wavelets are so useful in our project.

A wavelet is defined by

$$W(x) = \sum_k (-1)^k c_{1-k} \phi(2x - k) \tag{19}$$

Here, k is taken symmetrically around zero. The scaling function ϕ is defined by

$$\phi(x) = \sum c_k \phi(2x - k) \tag{20}$$

under conditions

$$\int \phi dx = 1 \tag{21}$$

$$\sum_k c_k = 2 \tag{22}$$

The Haar wavelet is the simplest wavelet. For this wavelet we choose $c_0 = 1$ and $c_1 = 1$. The scaling function ϕ for these coefficients is a blockfunction defined by

$$\phi(x) = \begin{cases} 1 & 0 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{23}$$

The wavelet as shown in Figure 15, is described by

$$W_{\text{Haar}}(x) = \phi(2x) - \phi(2x - 1) \tag{24}$$

We start with a vector $(f_1, f_2, ..., f_N)$, which are $N = 2^j$ equally sampled

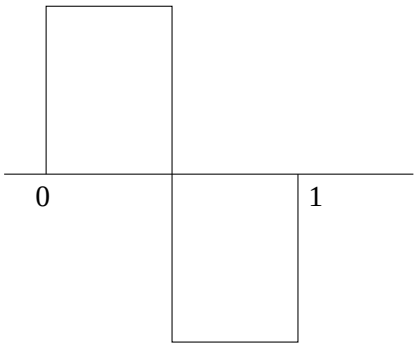

Figure 15: *Haar wavelet from box function* $W_{Haar}(x) = \phi(2x) - \phi(2x - 1)$.

values of a function $f(x)$ on a unit interval. This vector will be approximated by the sum of different weighted block functions. In Figure 16 the first 8 block functions, belonging to the levels 0, 1 and 2, are given. The projected input vector is described by the coefficients that correspond to the

weighing of these block functions.

How are these coefficients calculated? On a vector $\boldsymbol{x}$ of 2^j values we perform two operations $L : R^{2^j} \rightarrow R^{2^{j-1}}$ and $H : R^{2^j} \rightarrow R^{2^{j-1}}$. The first operation, $L(\boldsymbol{x})$, calculates the mean and the second, $H(\boldsymbol{x})$, calculates the difference.

$$L(\boldsymbol{x}) = k \times \begin{pmatrix} x_1 + x_2 \\ x_3 + x_4 \\ \vdots \\ x_{N-1} + x_N \end{pmatrix} \qquad H(\boldsymbol{x}) = k \times \begin{pmatrix} x_1 - x_2 \\ x_3 - x_4 \\ \vdots \\ x_{N-1} - x_N \end{pmatrix} \tag{25}$$

Here, usually $k = \frac{1}{2}$ in decomposition and $k = 1$ in reconstruction, but we could also use $k = \frac{1}{2}\sqrt{2}$ in both the decomposition and reconstruction, which has the advantage of normalizing the wavelets at every scale (this is done in the experiments described later).

The vectors produced by $L(\boldsymbol{x})$ and $H(\boldsymbol{x})$ are both half the size of the original vector. The coefficients found by $H(\boldsymbol{x})$ are the coefficients on the finest detail level, level $j-1$. To find the coefficients at the next detail level, we perform the operations $L(\boldsymbol{x})$ and $H(\boldsymbol{x})$ recursively on the vector produced by $L(\boldsymbol{x})$ at the previous level. This continues until we reach level 0.
The projected input consists of the coefficients per detail level found by $H(\boldsymbol{x})$ and the coefficient at level 0 found by $L(\boldsymbol{x})$. So, if we name L_i the average operator at level i and H_i the difference operator at level i, the new input vector is constructed by $(L_0, H_0, H_1, ..., H_{j-1})$.
On level i there are 2^i coefficients produced by H_i, the difference operator. On level 0 we have an extra coefficient coming from L_0. The total number of inputs is

$$N = 1 + 2^0 + 2^1 + ... + 2^{j-1} = 2^j \tag{26}$$

which is equal to the size of the original input vector.

We will now present an example for a vector of length 2^3, $\boldsymbol{x} = (1, 3, 2, 2, 5, 3, 8, 0)$. In the first step on level $j = 2$, we get for $k = \frac{1}{2}$

$$L(\boldsymbol{x}) = \begin{pmatrix} 2 \\ 2 \\ 4 \\ 4 \end{pmatrix} \quad \text{and} \quad H(\boldsymbol{x}) = \begin{pmatrix} -1 \\ 0 \\ 1 \\ 4 \end{pmatrix} \tag{27}$$

We continue on level $j = 1$ with $L(\boldsymbol{x}) = \boldsymbol{x}'$ as input vector

$$L(\boldsymbol{x}') = \begin{pmatrix} 2 \\ 4 \end{pmatrix} \quad \text{and} \quad H(\boldsymbol{x}') = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{28}$$

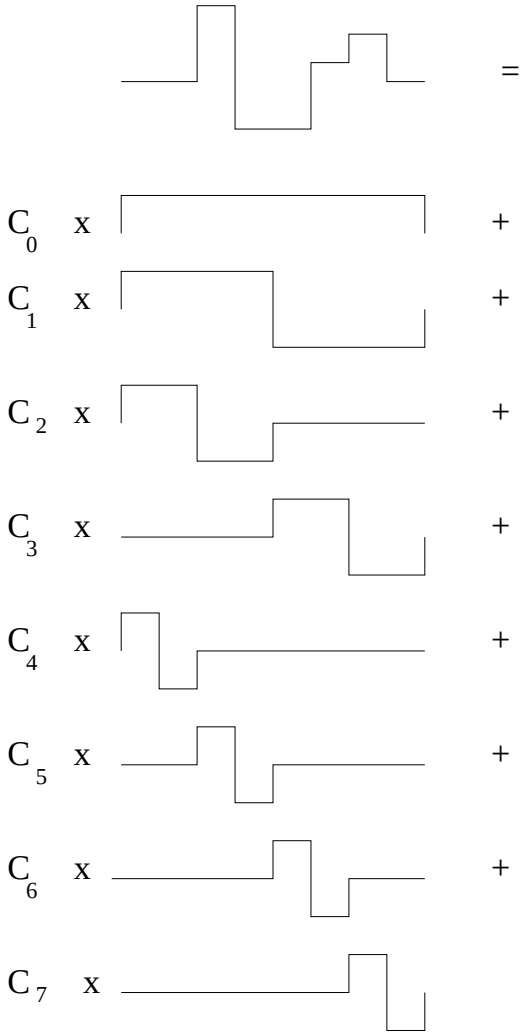

Figure 16: *The input signal is written as a weighted combination of rectangular block functions. The coefficients c_i are used as inputs to the neural net.*

And finally on level $j = 0$ we find for $L(\boldsymbol{x}') = \boldsymbol{x}''$

$$L(\boldsymbol{x}'') = \begin{pmatrix} 3 \end{pmatrix} \quad \text{and} \quad H(\boldsymbol{x}'') = \begin{pmatrix} -1 \end{pmatrix} \tag{29}$$

The vector of wavelet coefficients, $f'(x)$ is $(L(\boldsymbol{x}''), H(\boldsymbol{x}''), H(\boldsymbol{x}'), H(\boldsymbol{x}))$

$$f(x) = \begin{pmatrix} 1 \\ 3 \\ 2 \\ 2 \\ 5 \\ 3 \\ 8 \\ 0 \end{pmatrix} \quad \text{and} \quad f'(x) = \begin{pmatrix} 3 \\ -1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 1 \\ 4 \end{pmatrix} \tag{30}$$

What is so interesting about the wavelet coefficients, is that they describe a very local area of the input. We expect that the information in the center of the window is more important than the information on the edges. We know exactly which coefficients are corresponding to the edges and which to the center of the window and we can remove (some of) the coefficients that are corresponding to the edges, when we are not interested in this information. Coefficient c_4 from Figure 16, for example, describes the transitions in the first $\frac{1}{4}$ of the window (size w) with steps of size $w/8$. If we remove this coefficient, the transitions in the first quarter of the window can only be described by steps of $w/4$ of coefficient c_2. In this case we loose resolution on the first quarter of the window.
If we have an input coming from a sliding window of $w = 25.4$ feet, sampled every 0.2 feet, we have 128 input values and 7 detail levels 0 to 6. On the finer detail levels (5 and 6) we remove a number of coefficients on the edges of the window. We loose resolution at the edges, but we retain the same accuracy as the original input in the center. In the center the transitions are described by steps of 0.2 feet and on the edges by steps of 0.8 feet (this is detail level 4).
In Figure 17 is shown what happens when a number of coefficients on the edges is removed. The transitions in the center are described very accurately, but on the edges we use larger steps than in the original input (a transition is then approximated by steps instead of one transition).

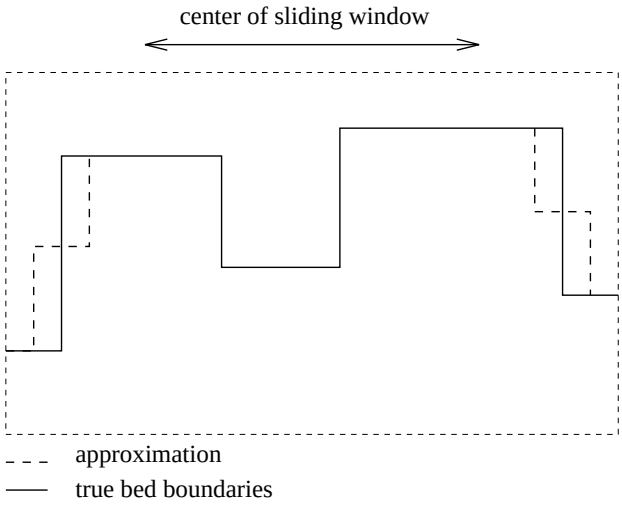

Figure 17: *Boundaries outside the center of the window are not described accurately.*

The coefficients are removed from the edges of the window, so all the detailed information in the center is preserved. If, for example, we remove 20 coefficients on each side on level 6 and 10 coefficients on each side on level 5, we removed 7.8 feet on level 6 and also 7.8 feet at level 5. This means that at these parts of the window we have a resolution of 0.8 feet and in the center 9.8 feet we have a resolution of 0.2 feet.
How much information is lost by this operation depends on the location of the bed boundaries and the sharpness of the transitions. A resistivity contrast is approximated by small steps instead of one transition. A hard measure for the loss of information is difficult to give. Whether the approximation affects the generalization performance in a negative way will show during training and testing.

2.3 Architecture constraints

The second method to reduce the complexity of the net, as we had already mentioned in the beginning of this section, is forcing certain constraints on the architecture. The advantage of this method is that no information is lost, like in the preprocessing methods. We can also reflect our prior knowledge about the input in the design of the net. This facilitates the learning process and hopefully will improve the generalization of the net. By using locally connected neurons (*receptive fields)*, the net contains much less weights than the fully connected nets. We can even further reduce the number of weights, by forcing some of the weights to be equal. This is further investigated in Section 2.3.4. In this section we will discuss the advantages and disadvantages of various fully and locally connected network architectures.

2.3.1 Fully connected nets

A common network architecture is a fully connected net as shown on the left side of Figure 18 (in this figure only the connections to the first hidden node are drawn). All nodes of one layer are connected to all nodes of the previous layer. The number of nodes of the hidden layer is very important. This number should not be too small, otherwise the net is not able to learn the problem. But it should also not be too large, otherwise the net has too much freedom and it will not generalize. More hidden layers can be added to improve on the training and generalization results. Usually one hidden layer is enough to learn a problem, but sometimes an extra layer helps to combine the features found by the first layer. It provides a more global view of the input.

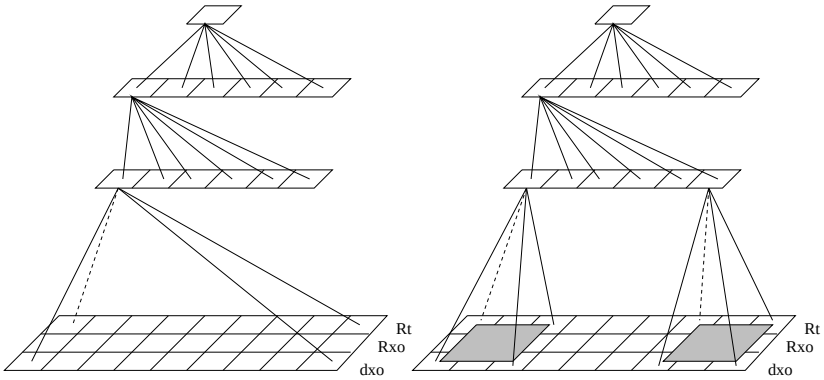

Figure 18: *Fully connected (left) and locally connected (right) neural nets.*

2.3.2 Locally connected nets

The input has a strong local structure, so it simplifies the problem by using so-called receptive fields. It may be easier for the neural net if a neuron only sees a part of the input and not all the inputs as in the fully connected nets. The neuron specialized on its part of the input and can be used as a local feature detector. We add an extra hidden layer in order to combine the local features properly. The part of the input a neuron is connected to, is called a receptive field. Usually all receptive fields have the same size and are only shifted in space (or time) with a fixed step (fixed overlap). The weight kernel of the first receptive field that is connected to the first neuron is shown in Figure 19. We can constrain the weight kernels for the various receptive fields to be the same, this is called *weight sharing* and it is used in the convolutional networks. The advantage of weight sharing is the decrease in the number of weights and freedom (and therefore complexity of the neural net). The decrease of freedom might improve the generalization ability of the net, but if the freedom is reduced too much, the net overall performance may decrease. The motivation for weight sharing is that we expect that a particular meaningful feature can occur at different times (or locations) in the input. An example of a locally connected net is shown in Figure 18 (right).

2.3.3 Symmetry constraints

In our models we have no dipping layers and no deviated bore holes. In this case the tool readings are assumed to be symmetric. This means that a signal and its mirror image, as shown in Figure 20, give the same tool response. This should also hold for the neural net. The response of the neural net (in a hidden node at the first hidden layer) to a signal $\boldsymbol{x}$ is $f(\boldsymbol{x}, \boldsymbol{w})$, for a certain weight vector $\boldsymbol{w}$. The mirror image of $\boldsymbol{x} = (x_1, x_2, ..., x_N)$ is

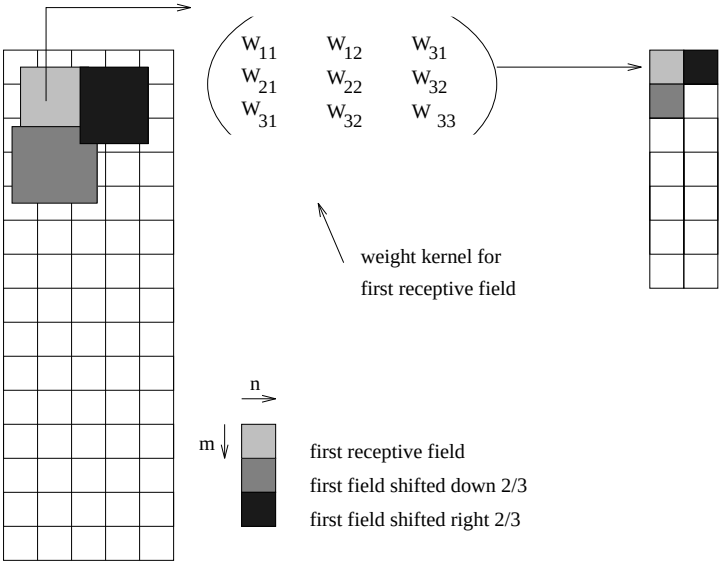

Figure 19: *Receptive fields which overlap* $\frac{1}{3}$.

$\boldsymbol{x}' = (x_N, ..., x_2, x_1)$. The response of the neural net to this input is $f(\boldsymbol{x}', \boldsymbol{w})$.

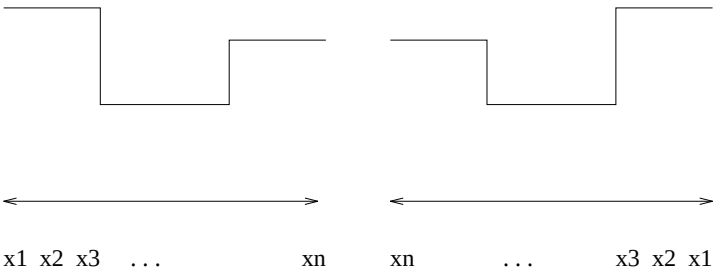

Figure 20: *A discretized input signal and its mirror image.*

We will now investigate what the implications are for the architectures of the nets when we force $f(\boldsymbol{x}, \boldsymbol{w}) \equiv f(\boldsymbol{x}', \boldsymbol{w})$, given

$$f_j(\boldsymbol{x}, \boldsymbol{w}) = \sum_i x_i w_{ji} \tag{31}$$

Here, j stands for hidden node j. Since the following equations should hold for any hidden node j in the first hidden layer, the index j is omitted.

For the fully connected nets the following equation should hold

$$\begin{aligned} \forall \boldsymbol{x} \in \Re^N : f(\boldsymbol{x}, \boldsymbol{w}) &= x_1 w_1 + x_2 w_2 + ... + x_{N-1} w_{N-1} + x_N w_N \\ &\equiv f(\boldsymbol{x}', \boldsymbol{w}) \\ &= x_N w_1 + x_{N-1} w_2 + ... + x_2 w_{N-1} + x_1 w_N \end{aligned} \tag{32}$$

This is only true when $w_i = w_{N+1-i}$. This constraint can be build into the net by forcing the weights coming from input node i to be equal to the weights coming from input node $N+1-i$. These constraints are shown in

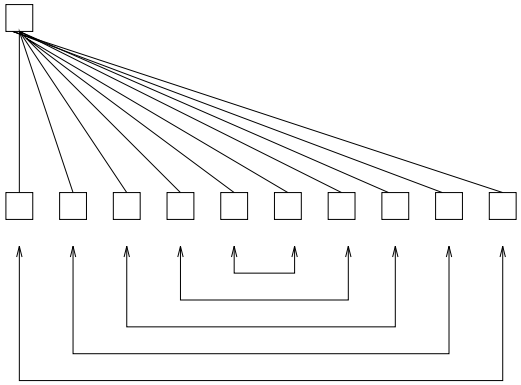

Figure 21: *Symmetry constraints on fully connected net.*

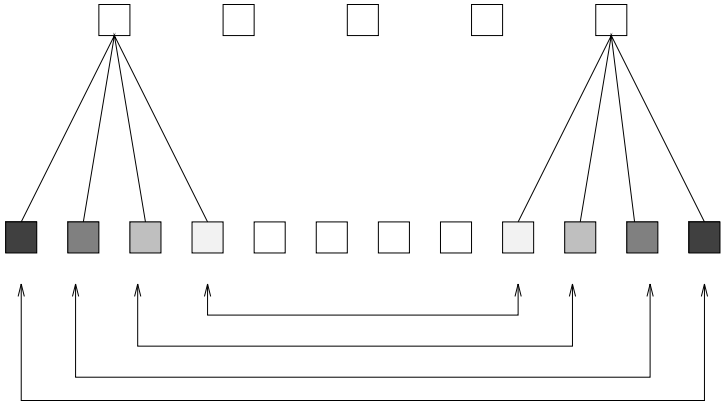

Figure 22: *Symmetry constraints on locally connected net. The receptive fields are constrained symmetrically (receptive field f is constrained to field $F-f+1$ for F fields).*

Figure 21. The same constraints can be used for the locally connected net as shown in Figure 22. In the locally connected nets the receptive fields are symmetrically constrained. This means that receptive field f is constrained to receptive field $F-f+1$, where F indicates the total number of receptive fields. The fields are also internally constrained symmetrically as shown in Figure 22.

For the nets that are trained on wavelet coefficients, we can also find some constraints. The wavelet vector of an input $\boldsymbol{x} = (x_1, x_2, ..., x_N)$ $(N = 2^j)$ on

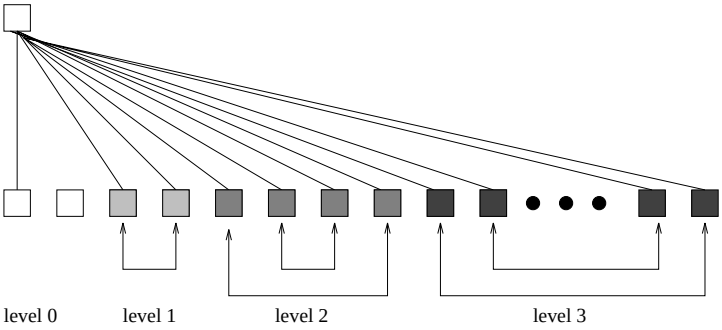

Figure 23: *Symmetry constraints on wavelet nets. The coefficients are constrained per detail level.*

a certain level is constructed by

$$L(\boldsymbol{x}) = k \times \begin{pmatrix} x_1 + x_2 \\ x_3 + x_4 \\ \vdots \\ x_{N-1} + x_N \end{pmatrix} \qquad H(\boldsymbol{x}) = k \times \begin{pmatrix} x_1 - x_2 \\ x_3 - x_4 \\ \vdots \\ x_{N-1} - x_N \end{pmatrix} \tag{33}$$

The coefficients on this level come from $H(\boldsymbol{x})$. The coefficients for the mirror image of $\boldsymbol{x}$ are calculated by

$$L(\boldsymbol{x}') = k \times \begin{pmatrix} x_N + x_{N-1} \\ \vdots \\ x_4 + x_3 \\ x_2 + x_1 \end{pmatrix} \qquad H(\boldsymbol{x}') = k \times \begin{pmatrix} x_N - x_{N-1} \\ \vdots \\ x_4 - x_3 \\ x_2 - x_1 \end{pmatrix} \tag{34}$$

The coefficients on higher levels are calculated recursively on the vector constructed by $L(\boldsymbol{x})$. On every level we find that $L(\boldsymbol{x}')$ is the mirror image of $L(\boldsymbol{x})$. The response of the net to $H(\boldsymbol{x})$ and the response of the net to $H(\boldsymbol{x}')$ should be the same on every detail level and for every possible input signal $\boldsymbol{x}$. For the 2^j wavelet coefficients on level $j > 0$ the following equation should hold for any wavelet vector $\boldsymbol{y} = H(\boldsymbol{x})$ and $\boldsymbol{y}' = H(\boldsymbol{x}')$

$$\begin{aligned} \forall \boldsymbol{y} \in \Re^N : f(\boldsymbol{y}, \boldsymbol{w}) &= y_1 w_{k+1} + y_2 w_{k+2} + ... + y_{2^j} w_{k+1+2^j} \\ &\equiv f(\boldsymbol{y}', \boldsymbol{w}) \\ &= -y_1 w_{k+1} - y_2 w_{k+2} - ... - y_{2^j} w_{k+1+2^j} \end{aligned} \tag{35}$$

Here, k indicates how many weights are used for the previous levels, which is $N = 2^j$. On level 0 we find $w_1 = w_1$ and $w_2 = -w_2 = 0$. The first weight corresponds to the coefficient that describes the average over the whole window and the second weight corresponds to the coefficient that

describes the difference between the average over the first half of the window and the average over the second half of the window. The tool readings should be symmetric so for the tool it does not matter whether this difference is positive or negative.
On level j we find $w_{2^j+i} = -w_{2^j+1-i}$ for $i = 0, 1, ..., 2^{j-1}$. The resulting network is shown in Figure 23.

2.3.4 Convolutional network

A neuron that is locally connected extracts local features. Sometimes we are not concerned about the location of these features, but only about the features themselves (for example in character recognition).

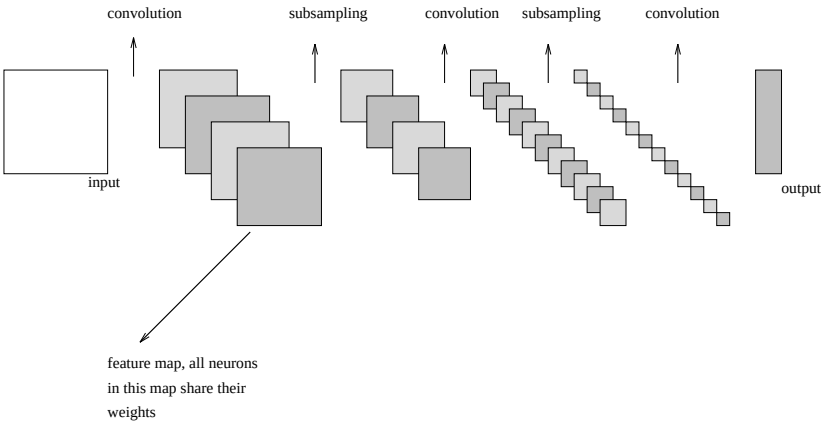

Figure 24: *Convolutional network.*

The set of weights, $\boldsymbol{w}$, belonging to a receptive field, makes it possible for a neuron in the first hidden layer to detect a specific feature at the location of the receptive field. We can use the same set of weights for all the receptive fields. This enables the first hidden layer to detect that specific feature, that was first only detected by the first hidden neuron, anywhere in the total input. The neurons in the first hidden layer that use this set of weights to detect a specific feature, are called a feature map.
The hidden layer is now able to detect one feature. We can add more feature maps to make it possible to detect more features. Each feature map consists of a number of neurons that make use of the same set of weights (weight sharing). This set of weights, as shown in Figure 19, is used as a convolution kernel. The convolution, performed by one feature map, is defined by the sum

$$y_j = \sum_{k}^{k+n-1} \sum_{l}^{l+m-1} w_{jkl} x_{kl} \tag{36}$$

where x_{kl} is the input pixel at location (k, l) and w_{jkl} is the weight between this input pixel and neuron j. The indices k and l indicate the left upper corner of the receptive field and $n \times m$ indicate the size of the receptive field. The convolution layers are alternated by so-called subsampling layers. These subsampling layers are like the convolution layers: they also make use of receptive fields and shared weights. The overlap of the receptive fields is maximal (replacement of one pixel) at the convolution layers and not at all at the subsampling layers. (When the receptive fields do not overlap maximal this is usually called subsampling). In the subsampling layer the spatial resolution of the feature maps, generated by the convolution layers, is reduced. Due to this reduction in resolution, this layer provides some degree of translational and rotational invariance.
This type of network is called a *convolutional network* and is shown in Figure 24.

2.3.5 Time Delayed network

Convolutional networks are used in two-dimensional problems, like character recognition. The one-dimensional version is called a *time delayed network*. The principles are more or less the same. A time delayed network is used in applications like speech recognition (Waibel, Hanazawa, Hinton, Shikano & Lang 1989).
The idea is that the response at a certain point of time (or depth) depends on previous inputs with a certain delay (hence the name time delayed). In this type of net there are no subsampling layers, only convolution layers. This is because we do not want the translational and rotational invariance, that is provided by the subsampling layers. The loss of time resolution by the first layer is partially compensated by an increase in the number of features in the next convolution layer. This is called bi-pyramidal scaling (Guyon 1991). The architecture for a typical time delayed network is shown in Figure 25.

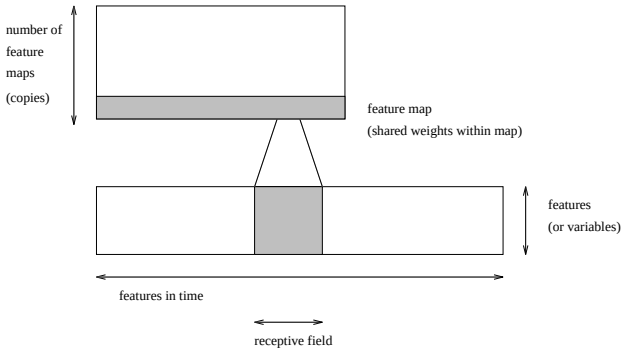

Figure 25: *Time delayed neural network.*

2.3.6 Convolutional-regression network

The design of the convolutional-regression network is based upon both the convolutional networks and the time delayed networks. The idea behind this

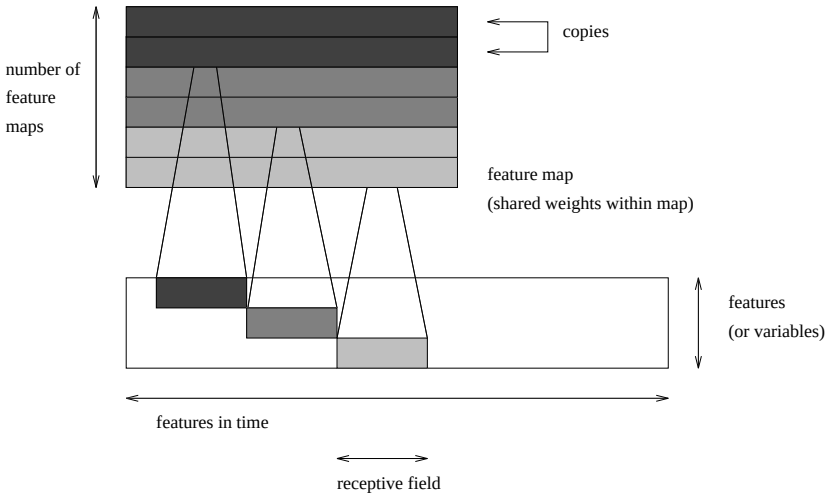

Figure 26: *Convolutional-regression network.*

type of network is that the network output depends on a number of inputs around the point of interest. So it depends on inputs above and below the logging point. For this net we use the input representation produced by the (uniform) discretized sliding window approach.

The feature maps in the first hidden layer can be connected to all or only some of the features (variables) in the input layer. This is like the original convolutional networks, but in contrast with the time-delayed networks. In the latter the feature maps are always connected to all the features of the previous layer. All other layers of the convolutional-regression network are fully connected. So this net only contains one convolution layer (and this is different from both the convolutional and time-delayed architectures). We only use one convolution layer, because we want to retain the spatial information of the input. The architecture is shown in Figure 26. In this example the feature maps are connected to only a number of the input features.

The receptive fields in the first layer do not overlap maximal. So actually this is not a convolution, but subsampling. The advantage of subsampling over convolution is that the number of hidden nodes (and the number of weights) that is needed is small (the number of hidden nodes is equal to the number of receptive fields). A disadvantage is the loss of spatial resolution.

A feature map from the first hidden layer detects a feature in the input. The second hidden layer determines the importance of the location of that feature. So the spatial information is preserved in the second (fully connected) hidden layer.

2.4 Error function

A neural network can minimalize any error function. The most common error function used is the sum square error, which is, for P patterns of the training set

$$E_{\text{Net}} = \sum_{p=1}^{P} (d_p - a_p)^2 \tag{37}$$

Here, d_p is the desired output of pattern p and a_p is the actual output of pattern p.

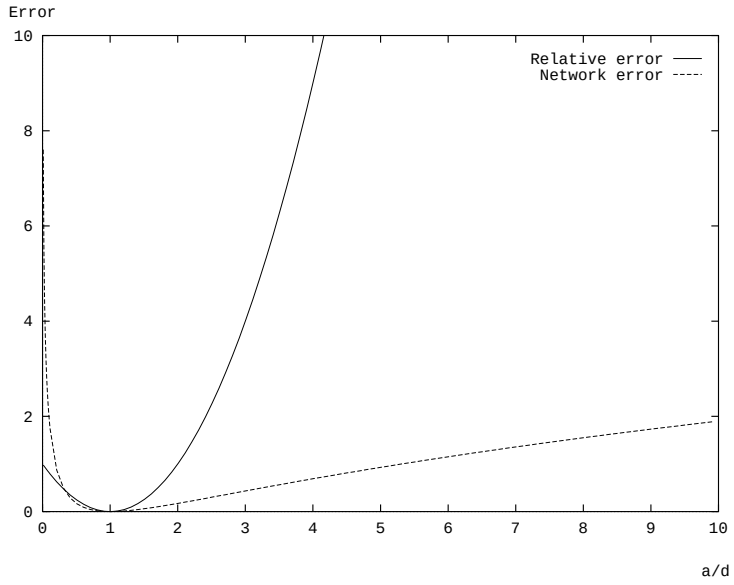

Figure 27: *The network and relative error for fixed proportions d/a.*

In this project the performance of the neural net is measured in the relative error between the desired and the actual output, averaged over P patterns

$$E_{\text{Rel}} = \sqrt{\frac{1}{P} \sum_{p=1}^{P} \left(\frac{d_p - a_p}{d_p} \right)^2} \tag{38}$$

The neural net error for one specific pattern p can be written as

$$E_{\text{Net}}(p) = d_p^2 \left(1 - \frac{a_p}{d_p} \right)^2 \tag{39}$$

This means that for the same proportion a/d, a pattern p produces a higher error when the desired output is large. The net emphasizes the learning of patterns with a high desired output. This is not what we want, because a small desired output is more likely to have a high relative error than a high target value. Thus we preferred the error E_{Rel} as given in eq. 38, which overcomes this disadvantage.
Before presenting the patterns to the net, we have scaled the input and the output. The desired and actual outputs are scaled by

$$t' = \frac{\ln(t) - \mu}{\sigma} \tag{40}$$

The consequence of this scaling is that the network is minimalizing the following error

$$E_{\mathrm{Net}} = \sum_{p=1}^{P} \left(\frac{\ln(d_p) - \mu}{\sigma} - \frac{\ln(a) - \mu}{\sigma} \right)^2 \tag{41}$$

$$= \frac{1}{\sigma^2} \sum_{p}^{P} \ln^2 \frac{d_p}{a_p} \tag{42}$$

which is almost the same as minimalizing the relative error, especially when the proportion d/a is almost equal to 1. Actually the net is minimalizing the proportion d/a. For fixed proportions d/a the network error E_{Net} and relative error E_{Rel} are drawn in Figure 27.
An overestimation ($a = d + k$) and an underestimation ($a = d - k$) must give the same error signal, as shown by the relative error in Figure 27. The neural net, with the error function as mentioned in eq. 41, favours $a = d - k$ over $a = d + k$, because the former produces a lower error signal.
It does, however, emphasize the learning of small target values over larger target values, which is also the case for the relative error.

It is very important to choose an appropriate error criterion. In this project we want to minimalize the relative error. When we had used another scaling method (not logarithmic), the network error function should have been adapted for better performance. With this scaling however the sum square error function suffices.

3 Forward modelling without mud invasion

Forward modelling in the presence of mud invasion is very complex. The measured resistivity is heavily affected by the invasion resistivity (Rxo), although this depends on the invasion radius (dxo). Before investigating this complex situation, we first look at the case without mud invasion. Here we only have one variable (Rt). The bore hole radius and drilling fluid resistivity are fixed to 4.25 inch and 0.05 Ωm, respectively. With only one variable the input space of earth models is not too large, which allows us to first experiment with the input representation and the network architecture. When we have found the most appropriate representation and architecture, we can use this in the case with invasion.

We have taken small bed sizes that range between 1 and 5 feet. Each earth model consists of a small log of 50 feet long and 15 beds, with a total of 47 models (2350 feet) for training and 33 models (1650 feet) for testing. The true resistivity ranges between 1 and 70 Ωm. These parameters are summarized in Table 2.

For the experimentation with the input representation and network architecture we have used a small training set of 4 earth models and an equally sized test set of 4 models. For these small tests no absolute errors are included, since they are only used for comparison. The performance is measured in the average relative error of the training set and the test set. A net performs well when both errors are small and comparable.

Table 2: *Parameters for earth models without invasion.*

Fixed parameters		
Bore hole radius	4.25	inch
Drilling fluid resistivity	0.05	Ωm
Variables		
Bed size	1, 2, 3, 4 or 5	feet
True resistivity	1, 2, 3, ..., 70	Ωm
Data		
Training set	47	models
Test set	33	models
Per model	50	feet
	15	beds
	251	examples

3.1 Experimenting with different scaling methods

The input and output are approximately scaled to the range $[-1, +1]$ (see Section 2.1.3). This is done by normalization of the input and output

$$z' = \frac{z - \mu}{\sigma} \tag{43}$$

where μ is the mean and σ is the standard deviation of the variable z on the training set. This type of scaling can also be done in combination with a logarithmic scaling. In that case the mean and standard deviation of $\ln(z)$ is used. The mean and standard deviations for the input and output are given in the following table

	μ	σ
Rt	29.00	16.67
logarithmic scaled Rt	3.30	0.80
LLd or LLs	30.00	16.67
logarithmic scaled LLd or LLs	3.00	0.83

We tested a fully connected net of 25 inputs (sliding window of 5 feet) and one hidden layer of 5 nodes on 1004 examples (4 models) and found that the logarithmic scaling in combination with the normalization gave the best results. The worst results are found when no logarithmic scaling was applied, so the normalization on itself is not sufficient. When no logarithmic scaling was applied, the neural net is minimizing the absolute error between the desired and actual output. We measure the performance in the relative error and a minimalization of the absolute error does not necessary mean a minimalization of the relative error. The logaritmic scaling overcomes this problem. Due to this scaling the proportion is minimalized (see Section 2.4), which is almost the same as minimalizing the relative error.
We will use this combined scaling method in the other tests.

3.2 Experimenting with the network architecture

Now we started experimenting with the network architecture. The following experiments are done for a window size of 5 feet. The tests are done for both the Deep- and the Shallow-log with 4 training models (1004 examples).

- Variation of number of hidden nodes (5, 10, 15, 20, 25). We found that increasing the number of nodes up to 15 improve the results, but that nets with 15 nodes and more gave similar results.

- Variation of number of hidden layers (1 or 2). Two layers of both 15 nodes or one hidden layer of 15 nodes gave comparable results.

- Adding connections of the input layer directly to the output layer (25 extra connections). Again the results for these extra connections and the normal fully connected net were comparable.

When the results are comparable we choose for the net with the fewest number of weights. These nets most likely have the best generalization and the smallest training time. The best architecture found by these experiments is a normal fully connected net, with one hidden layer of 15 nodes.

3.3 Experimenting with the size of the sliding window

The scaling method and the network architecture are fixed, only the window size is varied in the following tests. We tried window sizes of 5, 10, 15 and 30 feet. The smallest window size we can try is 5 feet (see Section 2.1.1). Again the nets are trained on 4 models (1004 examples), on both the Deep- and the Shallow-log. In both cases, a window size of 15 feet gives the best results. Although the results are slightly better for a window size of 30 feet, this does not outweigh the longer training time due to this high number of weights (2281 versus 1156).
The Shallow-log approximation is better than the Deep-log for the same window size. We already expected this, because the Deep Laterolog "sees" a larger part of the formation and therefore needs a larger window size. This was already shown in Figure 3 in Section 1.1.1.

3.4 Summary and results

The architecture we found by the previous tests is a fully connected neural net with one hidden layer of 15 nodes and a window size of 15 feet (75 inputs). This net is trained on 47 models with a total length of 2350 feet and tested on 33 models with a total length of 1650 feet.

Table 3: *Average relative error for earth models without invasion (1).*

Shallow-log	Training set	1.7 %
	Test set	2.2 %
Deep-log	Training set	5.1 %
	Test set	6.0 %

As shown in Table 3, the average relative error is below 5 % for the approximation of the Shallow-log. The generalization ability of the net, that is trained on the Shallow-log, is quite good (the test error and the training error are comparable).
However, the average relative error may not be an appropriate error criterion, since it is possible that a few points with a high relative error have a relatively large contribution to the average relative error. Therefore, we also calculate the percentage of the output that has a relative error below 5 % (called "correct") as shown in Table 4. From this table we find that only

Table 4: *Performance on earth models without invasion (1).*

Shallow-log	Training set	98 % correct
	Test set	96 % correct
Deep-log	Training set	62 % correct
	Test set	61 % correct

4 % of the approximation of the Shallow-log has a relative error above 5 %. And this is very close to the error on the training set. Therefore, we can say the learning and generalization ability of the net are quite good.

The approximation of the Deep-log, however, causes more problems. This is shown in Figure 28. The Deep-log differs substantially from the true resistivity profile, in contrast to the Shallow-log. This is caused by the effect of the shoulder beds, which is less profound in the case of the Shallow-log (see Section 1.1.2).
The output the neural net gives for this model is shown in Figure 29. In this figure one sees that the output given by the net looks more like the resistivity profile (like the Shallow-log) and therefore has quite a large relative error with its target value (the Deep-log). An improvement could be made by using more models similar to this one in the training set.

We conclude that our input representation (sliding window and scaling) is appropriate for the problem. A fully connected neural net with one hidden layer is sufficient to learn the input-output mapping and to provide good generalization over data it has not seen before. In the following chapter we will abandon the fully connected nets and look for more complex architectures to improve on the results and to handle invasion.

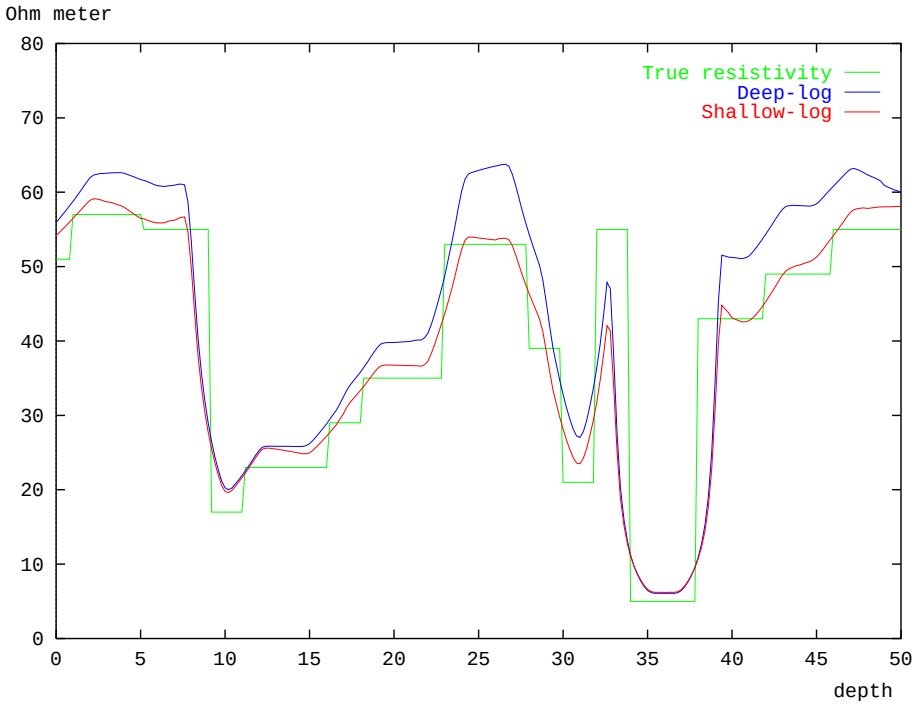

Figure 28: *Model that causes difficulties in approximating the Deep-log.*

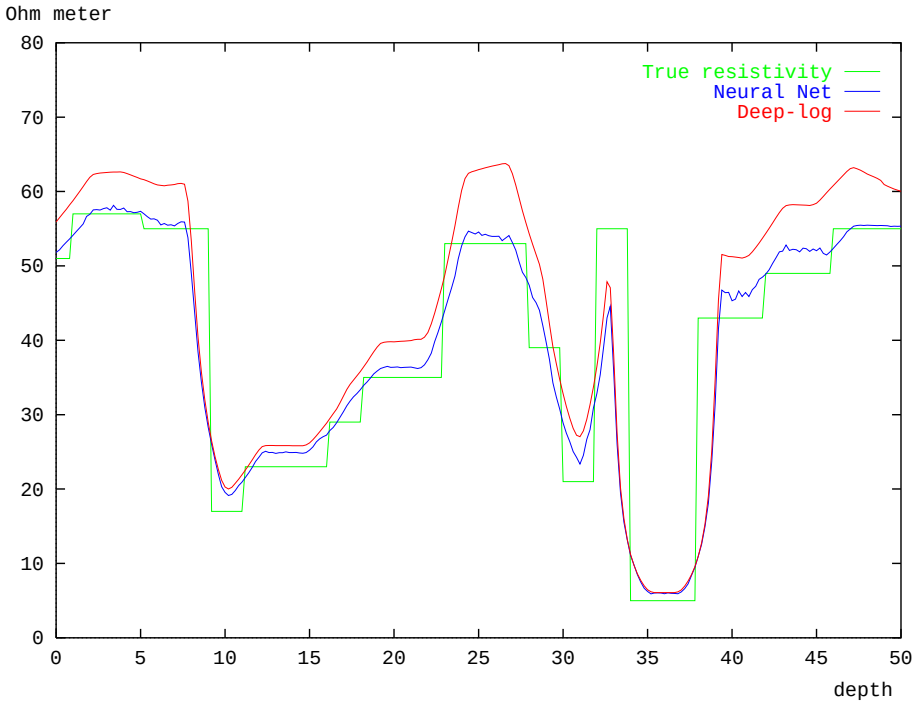

Figure 29: *Response of the neural net to the difficult model shown above.*

4 Forward modelling with mud invasion

Now we have found that it is possible to learn the mapping between the earth models and the tool response, we are going to look at a more difficult (and interesting) problem. In the new earth models the layers of the formation are invaded by the drilling fluid. This means we also have to take Rxo, the invasion resistivity, and dxo, the invasion radius, in account. Again the bore hole radius and the resistivity of the drilling fluid are fixed. The input space of earth models, constructed from randomly chosen combinations of the three variables Rt, Rxo and dxo, is much larger than in the previous case, when there was only one variable (Rt). We expect the network needs more examples to learn the problem.

We first perform a number of tests on a small training set in order to get an idea about how well the chosen input representation and architecture perform on an (also small) test set.

Table 5: *Parameters for earth models with invasion.*

Fixed parameters		
Bore hole radius	4.25	inch
Drilling fluid resistivity	0.05	Ωm
Variables		
Bed size	1, 2, 3, ..., 20	feet
True resistivity	2, 3, 4, ..., 71	Ωm
Invasion resistivity	0.5, 0.7, ..., 2.5	Ωm
Invasion radius	8, 9, ..., 50	inch
Data		
Training set	A, B, C, D, E, F (M, N, O, P, Q, R and S)	models
Validation set	G, H and I	models
Test set	J, K and L	models
Per model	± 1000	feet
	80	beds
	± 5000	examples

We use bed sizes between 1 and 20 feet, which is more realistic than in the previous tests, where we used beds between 1 and 5 feet. Each earth model is approximately 1000 feet long and consists of 80 beds. There is a total of 19 models for training, testing and validation as shown in Table 5. During training one calculates the error on a set of examples that is not used during training, the so-called validation set. When the error on this set starts to increase, the training is stopped. In the small tests we have not used a

validation set. The training was stopped, when the network converged to a minimum. This means the nets are overtrained: the error on the training set is minimal, but the error on the test set is not. In these experiments we only use the Shallow-log. We have found in the previous tests, without invasion, that the mapping between the earth models and the Shallow-log was easier to learn than the mapping between the earth models and the Deep-log. If a net finds it difficult to learn the Shallow-log, we expect it would have even more difficulty in learning the Deep-log.
In the next sections some of the results are given in duplicate in order to get a better comparison between the various tests. We have used one model (A) of 4601 sample points as training set and one model (F) of 4876 sample points as test (generalization) set.

4.1 Scaling of the parameters

The true resistivity Rt and the Shallow- and Deep-log (LLs and LLd) are scaled as in the previous tests. The scaling methods we have used here are summarized in the following table

Variable	**Domain**		**Scaling**	**Mean**	**Deviation**
Rt	2, 3, ..., 71	Ωm	log + norm	3.30	1.67
Rxo	0.5, 0.7, ..., 2.3	Ωm	norm	1.30	0.59
dxo	8, 9, ..., 50	inch	norm	29.00	25.00
LLd			log + norm	1.40	1.67
LLs			log + norm	1.40	1.67

In the following sections we use tables to describe the performed tests. The following table entries are used to describe a test:

- **connections**: The type of connections. This is fully or locally connected (fully connected by default).
- **architecture**: A description of the net per layer seperated by a ".". For each layer the number of nodes is given. This number can be split into the number of values per variable or feature map. A layer of 3×128 for example means we have 128 values per variable. A layer of 6×27 means we have 6 feature maps of 27 nodes each.
- **Number of weights**: The number of weights in the net is given.
- **window size**: The used size of the sliding window is given in feet (default value is 25.4 feet).
- **Sampling period**: The used sampling period is given in feet (default value is 0.2 feet).

- **Sampling**: The type of sampling. This can be uniform (default value), non-uniform or none.
- **Number of epochs**: The number of epochs that was needed to train the net.
- **Training error**: The average relative error on model A.
- **Generalization error**: The average relative error on model F.

4.2 Experimenting with different input representations

In this more complex situation we have to look at three variables per sampling point. This results in a high number of inputs for the neural net. Because of the results in the experiments without invasion, we first looked at the input representation coming from a uniform sampled sliding window in Test 1. A window size of 25.4 feet and a sampling period of 0.2 feet gives us 3×128 inputs, which is quite high. But we also looked at an input description by attributes in Test 2 as described in Section 2.1.2. The results are given in Table 6.

Since the sliding window representation is giving better results than the attributes description, we will now look for methods to reduce the complexity of the net. This can be done by reducing the number of inputs or by reducing the number of weights.
The generalization performance of the attributes input representation is very bad. The net is heavily overtrained, since the error on the test set was only 41.8 % after 183 epochs (training error at that time was 11.4 %).

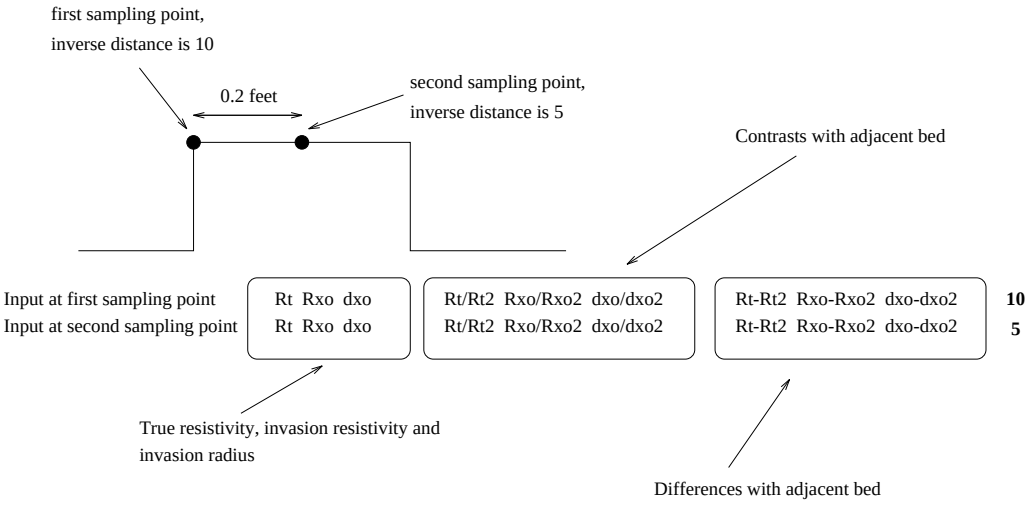

Figure 30: *Small difference in input with attributes input representation.*

Table 6: *Comparing discretized sliding window against attributes representation.*

Representation	**Test 1**	**Test 2**
architecture	$3 \times 128.10.1$	100.15.15.1
number of weights	3861	1771
window size	25.4 feet	30 feet
sampling period	0.2 feet	
sampling	Uniform	
input description	Discretized sliding	Attributes
	window	(10 beds)
number of epochs	3060	3484
training error	3.1 %	4.2 %
generalization error	70.0 %	$>$ 100 %

Table 7: *Comparing uniform sampling methods with different sampling periods.*

Sampling 1	**Test 1**	**Test 4**
architecture	$3 \times 128.10.1$	$3 \times 100.10.1$
number of weights	3861	3021
window size	25.4 feet	29.7 feet
sampling period	0.2 feet	0.3 feet
number of epochs	3060	3634
training error	3.1 %	2.9 %
generalization error	70.0 %	67.3 %

Table 8: *Comparing uniform and non- uniform sampling methods.*

Sampling 2	**Test 3**	**Test 5**
architecture	$3 \times 128.10.10.1$	$3 \times 64.15.15.1$
number of weights	3971	3151
window size	25.4 feet	29.4 feet
sampling period	0.2 feet	
sampling	Uniform	Non-uniform
number of epochs	6429	4404
training error	1.1 %	1.0 %
generalization error	65.2 %	46.4 %

A possible cause of the bad performance on the attributes is that the net is fed with a number of almost similar examples. Only the inverse distance for each bed differs in those cases, as shown in Figure 30. The inverse distance becomes the most important value in the input. This can also be seen from the weights. Most weights are quite small, ranging from -0.5 to 0.5, but the weights belonging to the connections from the inverse distance- inputs to the hidden layer are large ($\pm$ 1.5).
It is important to choose appropriate attributes, but finding these attributes is very difficult. The network is actually pushed in some kind of direction by using attributes. This can make the learning easier, but it can just as easy make the learning more difficult.

4.3 Experimenting with input reduction methods

The number of inputs coming from a uniform sampled sliding window is very high. For example, a window of 29.8 feet and a sampling period of 0.2 feet produces 3×150 inputs. We would like to use a hidden layer of 15 nodes and maybe even two hidden layers of 15 nodes. The former gives us 6781 weights and the latter 7021 weights. To train such a large net we need approximately 70000 examples (Widrow's Rule of Thumb to take about 10 times the number of free variables). Each example consists of 451 values (3×150 inputs and one target), so we would need to store approximately 31570000 values to train this net. This large set will slow down the training and causes storage and memory problems.
It is for this reason that we looked at several ways to reduce the number of inputs for a fixed windowsize. The disadvantage of input reduction is the fact that we loose information.

4.3.1 Using different sampling methods

The number of inputs depends on the used window size and the used sampling period. We could use a smaller sliding window than 25.4 feet, but this is not an attractive option, because we suspect the tool readings are affected by a window of at least the size of the tool (see Section 2.1.1).
Another option is to use a coarser sampling period, for example 0.3 feet (Test 4) instead of 0.2 feet. In a sliding window of 29.7 feet this results in 3×100 inputs, instead of the 3×150 inputs with a 0.2 feet sampling period in a sliding window of 29.8 feet.
Another approach is to use a non-uniform sampling method. This approach is based on the fact that the physical tool receives most of its information from the center and less from the sides. We have chosen an heuristic non-uniform sampling method in Test 5 as shown in Figure 31, which results in only 3×64 inputs per window (instead of 3×150). In Test 3 we use the same net as in Test 1, but with two hidden layers. The results for these

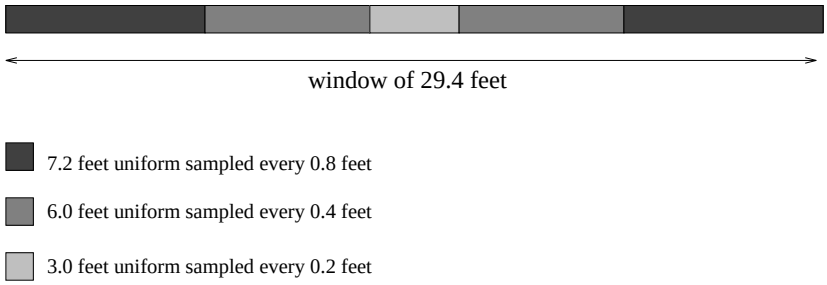

Figure 31: *Non-uniform sampling in sliding window.*

tests are shown in Table 7 and Table 8.

When a uniform sampling method is used, it it possible to use a coarser sampling period (0.3 feet instead of 0.2 feet), because the training and testing results are similar. The bed boundaries are described less accurately when a coarser sampling period is used. But from these tests we find that it does not influence the performance in a negative way.
With the non-uniform sampling method we only loose accuracy at the edges, because the center of the window is sampled with the same sampling period as in the original input. The net that is trained on this input representation performs better than the net that was trained on the original input.
Both methods, coarser and non-uniform sampling, are appropriate as methods to reduce the inputs.

4.3.2 Reducing the input by projection to principal components

We use the principal component analysis as described in Section 2.2.1. The eigenvectors are calculated for an input file consisting of six models. The models are not alike internally, so it is better to use a set of models. Otherwise the eigenvectors will be appropriate for one model, but not for another. We found that it does not matter whether you use six models or more, so we have chosen for the models A, M, N, P, Q and R (these models are very dissimilar).
Both vectors (original N-dimensional and projected M-dimensional) are rescaled with the inverse normalization scaling of equation 9, before the LOI is calculated. Due to this scaling, the loss of information for $M = 0$ is not 100 %. The scaled vector consists of only zero's, but the rescaled vector is everywhere equal to the mean of the input values. This rescaling is not absolutely necessary, but we wanted to calculate the loss of information on the original input.

In Test 7 the original input vector consists of 3×64 inputs, constructed by

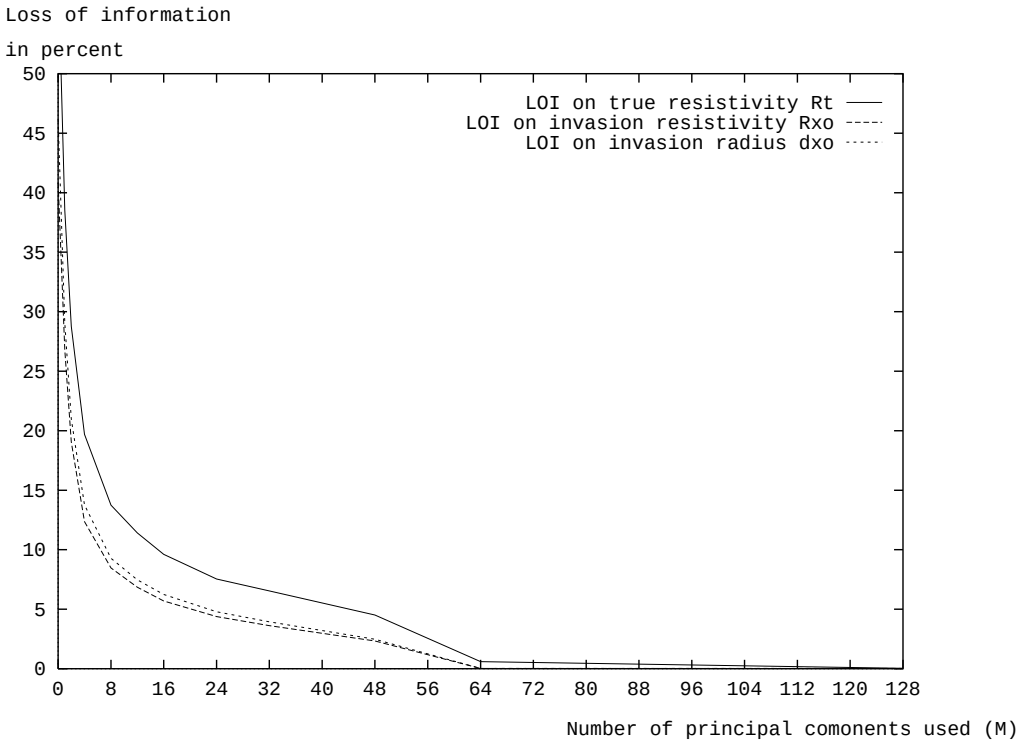

Figure 32: *Loss of information per variable for Test 6. Original input consists of* 3×128 *inputs, coming from a uniform sampled sliding window.*

Table 9: *Comparing uniform sampled input without and with projection to principal components.*

PCA 1	**Test 3**	**Test 6**
architecture	$3 \times 128.10.10.1$	$3 \times 32.10.10.1$
number of weights	3971	1091
PCA	No	Yes
		LOI 7.4 % on Rt
		LOI 4.7 % on Rxo
		LOI 4.4 % on dxo
number of epochs	6429	5193
training error	1.1 %	1.7 %
generalization error	65.2 %	52.3 %

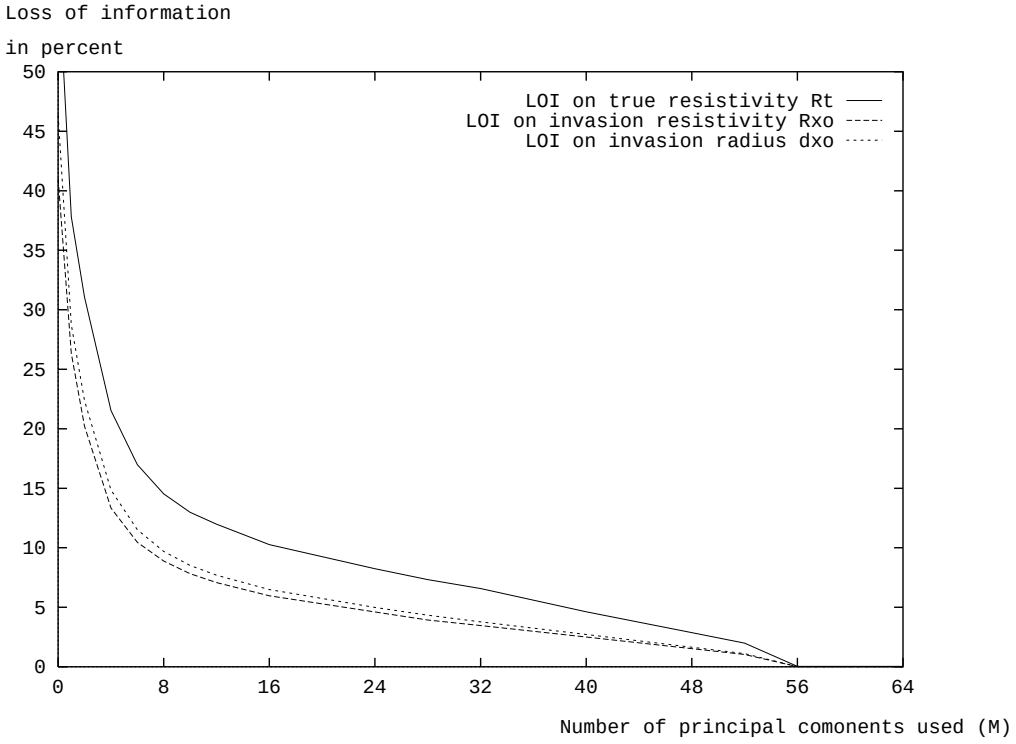

Figure 33: *Loss of information per variable for Test 7. Original input consists of* 3×64 *inputs, coming from a non-uniform sampled sliding window.*

Table 10: *Comparing non-uniform sampled input without and with projection to principal components.*

PCA 2	**Test 5**	**Test 7**
architecture	$3 \times 64.15.15.1$	$3 \times 32.15.15.1$
number of weights	3151	1711
window size	29.4 feet	29.4 feet
sampling period		
sampling	Non-Uniform	Non-Uniform
PCA	No	Yes
		LOI 7.2 % on Rt
		LOI 4.2 % on Rxo
		LOI 3.9 % on dxo
number of epochs	4404	3896
training error	1.0 %	1.4 %
generalization error	46.4 %	45.6 %

a non-uniform sampled sliding window as in Test 5. The LOI for various values of M is shown in Figure 33.

The LOI, for both tests, is calculated for $P = 1520$ patterns coming from models J, O and S.
In the following tables, Table 9 and Table 10, the LOI is given per variable for the training and test set.

In both tests we find that the training and generalization error for the tests with and without PCA are comparable. This means the principal component analysis is a good method to reduce the number of inputs. In the first test we achieve an input reduction of 75 % and in the second test 50 %. The weights are reduced with 73 % and 46 % respectively.

4.3.3 Reducing the inputs by removing wavelet coefficients

In the following tests we use wavelet coefficients as inputs instead of the original inputs. We do not know how much coefficients can be removed on the finer detail levels so that not too much information is lost. Therefore, we run several tests with different reductions. In the first test, Test 8, no coefficients are removed at all and in the last test, Test 11, we remove all the coefficients on the finest detail level and some on higher detail levels. The results are given in Table 11 and Table 12.
The first interesting aspect is that the net learns better when coefficients are removed. During training the weight correction for a hidden node is calculated as indicated by equation 7. It depends on the learning parameter, the local gradient and most important on the input signal of node j. So when input node j is 0, there is no weight correction. This is probably the cause of the bad performance of the net in Test 8.
The wavelet coefficients are 0 when there is no bed transition at that spot, because the difference between the first half and the second half of (a part of) the window is 0. Most of the coefficients on higher detail levels are zero, because there are only a few bed boundaries in the window. Sometimes a bed boundary is not even detected as shown in Figure 34.

On the finest detail level this problem is quite severe, because these coefficients only see a very small part of the input and this small part is not likely to contain a bed boundary. Whenever the bed boundary is not detected or when there is no bed boundary at all, the coefficients are zero and the weight from this coefficient to the hidden node is not updated. In other words, the hidden node does not “learn” from this coefficient.

Table 11: *Comparing input representations with different number of wavelet coefficients (1).*

Haar 1	**Test 8**	**Test 9**
architecture	3 × 128.10.1	3 × 68.15.1
number of weights	3861	3091
coefficient reduction	none	2 × 20 on level 6
		2 × 10 on level 5
number of epochs	2125	4303
training error	4.1 %	0.8 %
generalization error	63.5 %	51.5 %

Table 12: *Comparing input representations with different number of wavelet coefficients (2).*

Haar 2	**Test 10**	**Test 11**
architecture	3 × 48.10.1	3 × 34.15.1
number of weights	2191	1561
coefficient reduction	2 × 25 on level 6	2 × 32 on level 6
	2 × 12 on level 5	2 × 12 on level 5
	2 × 3 on level 4	2 × 3 on level 4
number of epochs	4479	3848
training error	1.0 %	0.9 %
generalization error	50.3 %	56.4 %

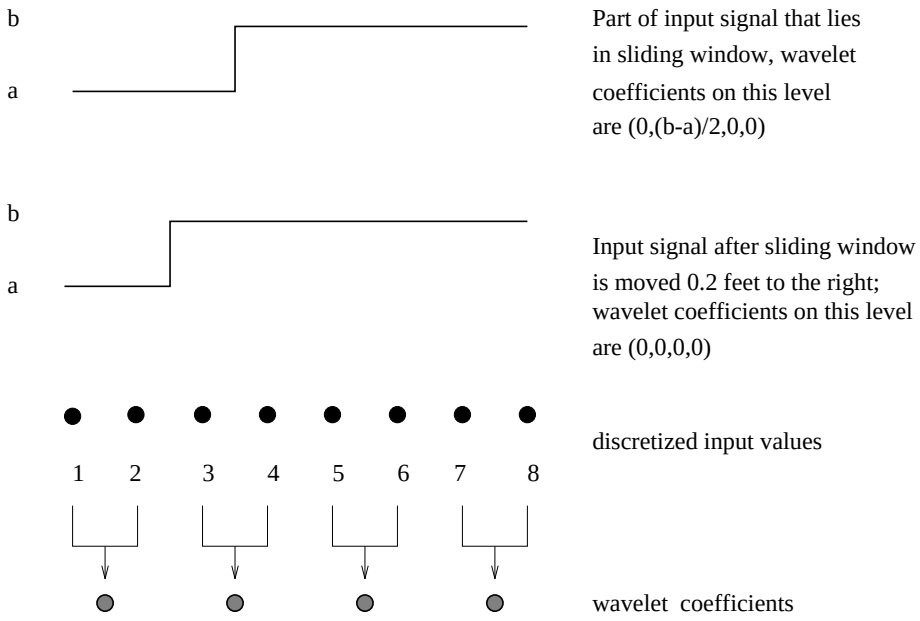

Figure 34: *Not all transitions are detected by wavelet coefficients.*

There are approximately 2 bed boundaries per pattern. Not all boundaries are detected by the coefficients on the finest detail level, but when it is not detected by coefficient c_i it will be detected by coefficient c_{i-1} after moving the sliding window. So on average 50 % of the boundaries is detected on the finest detail level. The training set consists of 4601 patterns, so this means the weight coming from coefficient c_i to hidden node h_j is updated approximately 72 times (4601 patterns, 2 boundaries per pattern from which 50 % is detected and 64 coefficients). A usual training takes about 2000 to 3000 updates per weight.
In Test 9, 10 and 11 we remove a number of the coefficients on the finest detail level and also from higher levels and we see that the performance of these nets is much better than the performance of the net in Test 8.
When we compare the results to the net trained in Test 1, we see that the nets have better training and generalization errors, although the number of inputs is reduced from 3×128 to 3×68, 3×48 and 3×34. An input reduction of 73 % is possible, without reducing the net performance. One explanation for the better generalization performance, is that the nets contain much less weights. The number of training examples in proportion to the number of weights is higher in these tests than in Test 1. But it could also be possible that the net finds this input representation facilitates the training and generalization more than the original discretized sliding window. The improvement in generalization is better than for the principal component analysis of Test 6 and 7.

4.4 Creating a more representative training set

Our training set is not very representative, because the examples of adjacent logging points look very similar. This is caused by the fact that we use a dense sampling period for the target log. We use this dense sampling period because we want as much examples per log as possible (the data set is limited). There are two methods we employ to increase the representativeness of the training set: firstly by using a coarser sampling period for the target log and secondly by using "difficult" (non-similar) examples.

When we use a sampling period of 3 feet, one model of 1000 feet only produces about 300 examples. This is what we did in Test 12. To create a training set of the same size as our original training set we use 15 models with this sampling period. This training set is more representative, because it contains more different examples. We expect this will make it more difficult for the net to learn the problem. The network has to find a more general solution to fit all the examples. This improves the generalization of the network.

After training the net, we have looked at the approximation of the net to an arbitrary log. We found that the highest relative errors occur around areas of low resistivity and areas with high resistivity contrasts. The training set would be more representative when it contained more examples like this. Actually this is some kind of weighing of the examples. By offering more "difficult" examples, these examples will get more attention in the minimalization process. In Test 13 we use a training set of 4600 examples, coming from difficult parts of log A, P and S and 1436 coarser sampled examples of the other parts of log A (with a total of 6036 examples as training set). The results for these tests are given in Table 13 and Table 14.

As we can see from these tests, both methods improve the generalization performance of the net. The training, however, becomes more complicated. It is very important to create a training set that contains a large number of different examples that look like examples that are found in reality.

Table 13: *Comparing training set of target logs and training set of coarser sampled target logs.*

Training set 1	**Test 1**	**Test 12**
architecture	3 × 128.10.1	3 × 128.10.1
number of weights	3861	3861
training set	model A	mixture of 15 models
number of epochs	3060	3023
training error	3.1 %	5.6 %
generalization error	70.0 %	26.9 %

Table 14: *Comparing training set of target logs and training set of difficult parts of target logs.*

Training set 2	**Test 1**	**Test 13**
architecture	3 × 128.10.1	3 × 128.10.1
number of weights	3861	3861
training set	model A	difficult examples of
		models A, P and S
number of epochs	3060	1042
training error	3.1 %	9.8 %
generalization error	70.0 %	30.3 %

4.5 Intermediate results input representations

We select some of the nets from the previous tests to train them on a large training set. The training set for the Shallow-log is constructed from the models A, B, C, D, E and F. The target logs are sampled every 0.2 feet, so the training set contains 28331 examples. We use the models G, H and I as validation set (14153 examples) and the models J, K, L, M, N, O, P, Q, R and S as test set (47485 examples).
In Figure 35 the results for three nets are shown. The first six models are the models the net is trained on, the other models are used for validation and testing. The first net, **Non-uniform**, is a net with a non-uniform sampled input and PCA reduction from 3×64 inputs to 35 inputs for Rt and 20 inputs for both Rxo and dxo. The second net, **Haar** from Test 9, is a net with Haar coefficients as inputs. The inputs are reduced by removing some of the coefficients from 3×128 inputs to 3×68 inputs. And the third net, **Uniform** from Test 4, is a uniform sampled input with a window of 29.7 feet and a sampling period of 0.3 feet, resulting in 3×100 inputs.
In Figure 35 the percentage of the log that has a relative error below 5 % is shown.
As can be seen in Figure 35, the non-uniform sampled input gives the best results on the training set, but they all perform equally over the validation and testing set.

The methods to reduce the number of inputs work very well. The networks are able to learn the problem even when a large number of the inputs is removed. The generalization performance, however, is not sufficient and we have also lost information. In the next sections we will investigate certain architectural constraints in order to improve on the generalization performance.

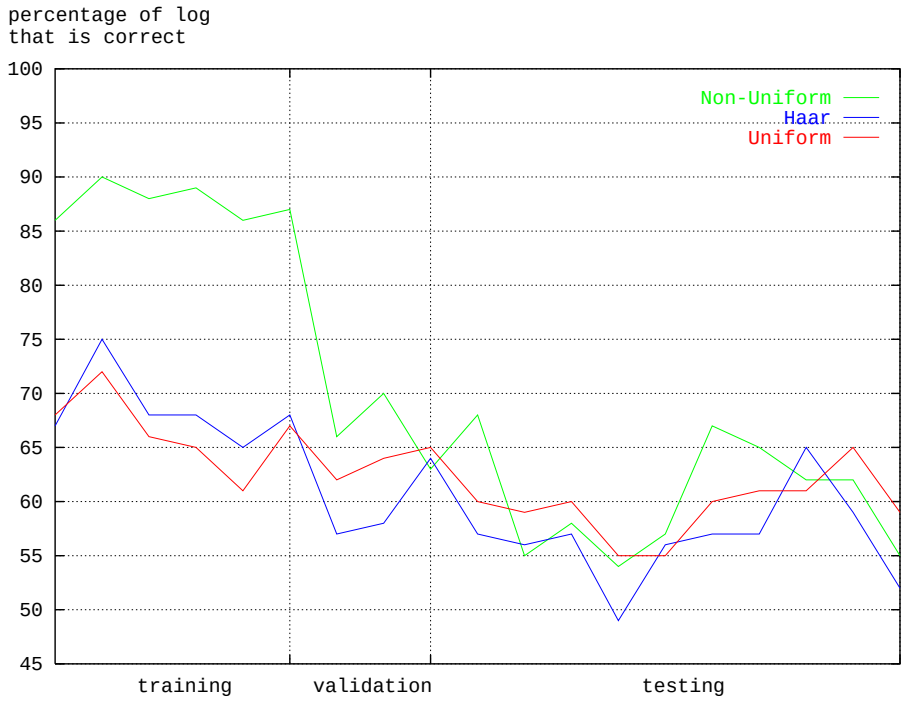

Figure 35: *Intermediate results of training three different neural nets on 6 models.*

4.6 Experimenting with architecture constraints

In the previous tests we only used fully connected nets. Now we will look at locally connected nets and other connection constraints. An attractive method to reduce the number of weights is by forcing certain weights to be equal. Of course we have to be very careful when we use this type of constraint, because it could also make the learning more difficult and decrease the generalization performance. This occurs when the weight sharing is forcing the net to find a solution that does not fit the problem. It will only help the training and generalization when the weight sharing is a logical consequence of the problem.

4.6.1 Experimenting with fully connected nets

As we have seen, the fully connected nets can be trained quite well when a sufficient number of (different) examples is given. We train different fully connected nets with uniform sampled inputs and one or two hidden layers in Test 1 and Test 3. The results of these tests are given in Table 15.
We find that two hidden layers improve the training results, but it is not feasible to train a net with that many weights on a large training set.

4.6.2 Experimenting with locally connected nets

The advantage of locally connected nets over fully connected nets is that they need much less weights and the neurons in the hidden layer can specialize on their part of the input. The size and overlap of the receptive fields are things to be determined. We have tried two locally connected nets in Test 15 and Test 16. The results are given in Table 16.
The results are better when we use small receptive fields and a sufficient number of hidden nodes on the second hidden layer. When we compare the net from Test 16 to the fully connected net from Test 3, we have obtained a weight reduction of 63 %. Although the training error is slightly worse, the generalization, which is more important, is a lot better. We find that locally connected nets are more attractive (in this problem) than the fully connected nets.

4.6.3 Using symmetry constraints

In the following tests we have build the symmetry constraints, as described in Section 2.3.3, in the fully connected net (Test 17 and Table 17), the locally connected net (Test 18 and Table 18) and the wavelet nets (Test 19 and Table 19).

Table 15: *Comparing different fully connected nets.*

Fully connected	**Test 1**	**Test 3**
architecture	$3 \times 128.10.1$	$3 \times 128.10.10.1$
number of weights	3861	3971
number of epochs	3060	6429
training error	3.1 %	1.1 %
generalization error	70.0 %	65.2 %

Table 16: *Comparing different locally connected nets.*

Locally connected	**Test 15**	**Test 16**
connections	Locally connected	Locally connected
architecture	$3 \times 128.12.5.1$	$3 \times 128.19.15.1$
number of weights	1523	1475
receptive field size	7.8 feet	3.8 feet
receptive field overlap	80 %	70 %
number of epochs	2679	5267
training error	7.5 %	2.1 %
generalization error	61.7 %	48.9 %

Table 17: *Comparing fully connected nets without and with symmetry constraints.*

Symmetry 1	**Test 1**	**Test 17**
architecture	$3 \times 128.10.1$	$3 \times 128.10.1$
number of weights	3861	1941
symmetry constraints	No	Yes
number of epochs	3060	655
training error	3.1 %	6.6 %
generalization error	70.0 %	37.7 %

In all tests the number of weights is reduced approximately 50 %. This makes the training more difficult for the net, but it improves the generalization performance. This idea of symmetry constraints is worked out further in the convolutional-regression nets.

4.6.4 Experimenting with convolutional regression nets

In the convolutional-regression nets we constrain all receptive fields to share the same weights. The size and the overlap of the fields determine the accuracy in spatial resolution. The second hidden layer combines the activations from the first hidden layer and determines the location of the feature in the input.

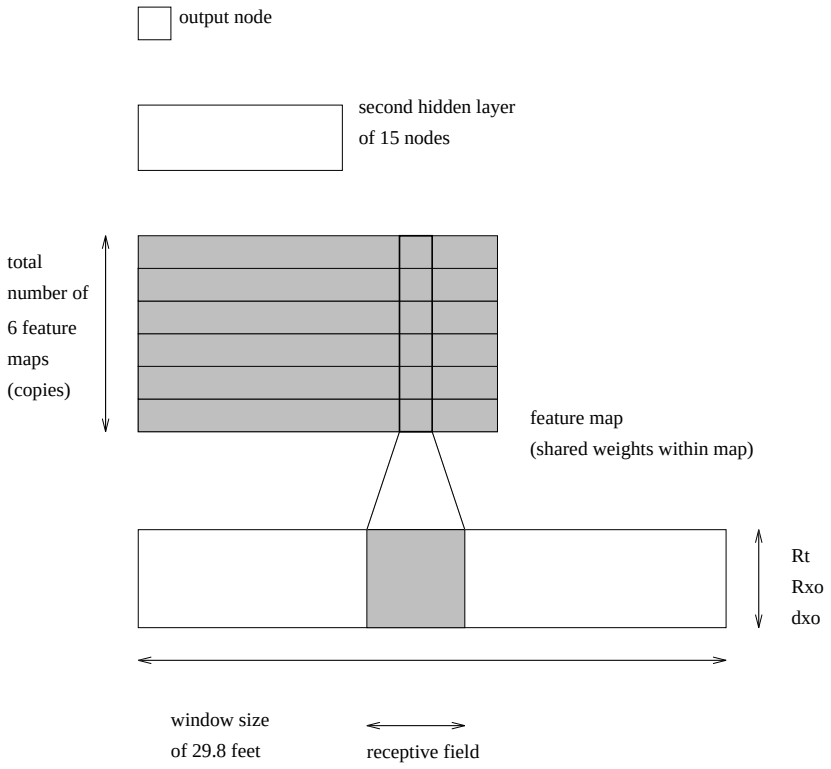

Figure 36: *Convolutional-regression net. The feature maps in the first hidden layer are connected to all the variables in the input layer.*

The number of feature maps determine how many features can be detected. We should not use too few feature maps, otherwise the net has difficulty in learning the problem and in generalization. The number of weights, however, depend upon the number of feature maps used (for 15 hidden nodes in

Table 18: *Comparing locally connected nets without and with symmetry constraints.*

Symmetry 2	**Test 16**	**Test 18**
connections	Locally connected	Locally connected
architecture	$3 \times 128.19.15.1$	$3 \times 128.19.15.1$
number of weights	1475	905
receptive field size	3.8 feet	3.8 feet
receptive field overlap	70 %	70 %
symmetry constraints	No	Yes
number of epochs	5267	5536
training error	2.1 %	2.3 %
generalization error	48.9 %	37.8 %

Table 19: *Comparing "wavelet" net without and with symmetry constraints.*

Symmetry 3	**Test 9**	**Test 19**
architecture	$3 \times 68.15.1$	$3 \times 67.15.1$
number of weights	3091	1561
remarks	Wavelet coefficients	Wavelet coefficients
	Reduction 20,10	Reduction 20,10
symmetry constraints	No	Yes
number of epochs	4303	3137
training error	0.8 %	3.6 %
generalization error	51.5 %	46.6 %

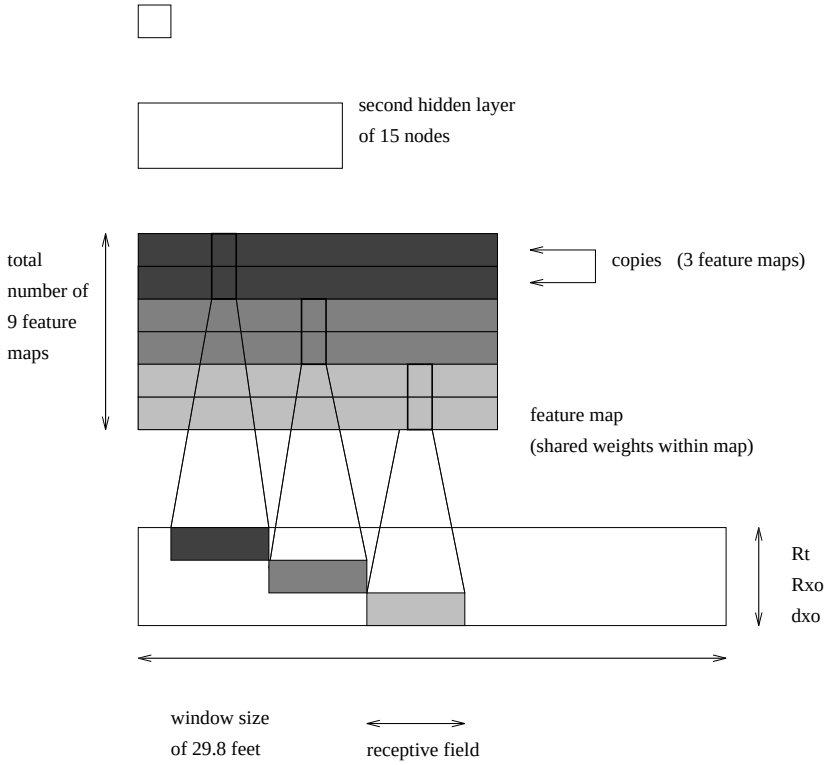

Figure 37: *Convolutional-regression net. The first hidden layer consists of three sets of feature maps. Each set consists of three maps and is connected to one of the variables in the input layer.*

the second hidden layer one feature map of k nodes increases the number of weights by $k \times 15$). A trade-off should be made between the size and overlap of the fields, the number of feature maps to use and the resulting number of weights in the net. In Test 20, 21 and 22 the hidden layer was connected to all the variables of the input layer as shown in Figure 36. In Test 23, on the other hand, the first hidden layer was split into three variable maps, each containing three feature maps. Each variable map was connected to its corresponding variable in the input layer. This is shown in Figure 37. The results for these tests are given in Table 20 and Table 21. For all these tets the nets are locally connected and the window size is 29.8 feet. The convolutional-regression nets perform very well on both the training and the test set. The generalization performance of these nets is significantly better than for all the nets we have tried before. When we compare the results from Test 21 with the results from Test 3 for example (Test 3 is a fully connected net with two layers), we have a weight reduction of 29 % and the generalization is approximately 2.5 times better.

We have investigated the features that are detected by the first hidden layer. In Figure 39 the activations of the first hidden layer at a certain depth are plotted. These activations come from Test 21.
From these activations we can see the first hidden layer is actually detecting the bed boundaries. It smooths the input signal and the activations are constant for parts of the input signal that are constant.

Table 20: *Results for convolutional- regression nets (1).*

Convolutional 1	**Test 20**	**Test 21**
architecture	$3 \times 60.6 \times 9.5.1$	$3 \times 150.6 \times 27.15.1$
number of weights	503	2827
sampling period	0.5 feet	0.2 feet
receptive field size	5.5 feet	3.8 feet
receptive field overlap	75 %	75 %
figure	Figure 36	Figure 36
number of epochs	3316	2865
training error	3.9 %	0.6 %
generalization error	34.7 %	25.5 %

Table 21: *Results for convolutional- regression nets (2).*

Convolutional 2	**Test 22**	**Test 23**
architecture	$3 \times 100.6 \times 16.15.1$	$3 \times 150.9 \times 27.15.1$
number of weights	1927	3865
sampling period	0.3 feet	0.2 feet
receptive field size	7.2 feet	3.8 feet
receptive field overlap	80 %	75 %
figure	Figure 36	Figure 37
number of epochs	4081	3689
training error	1.6 %	0.5 %
generalization error	26.3 %	28.6 %

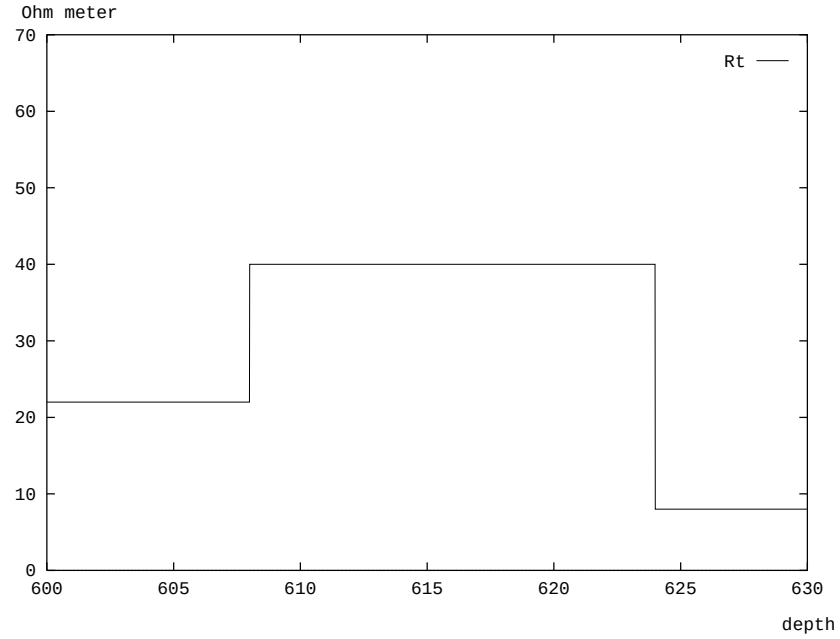

Figure 38: *True resistivity profile of model A at depth 615 feet.*

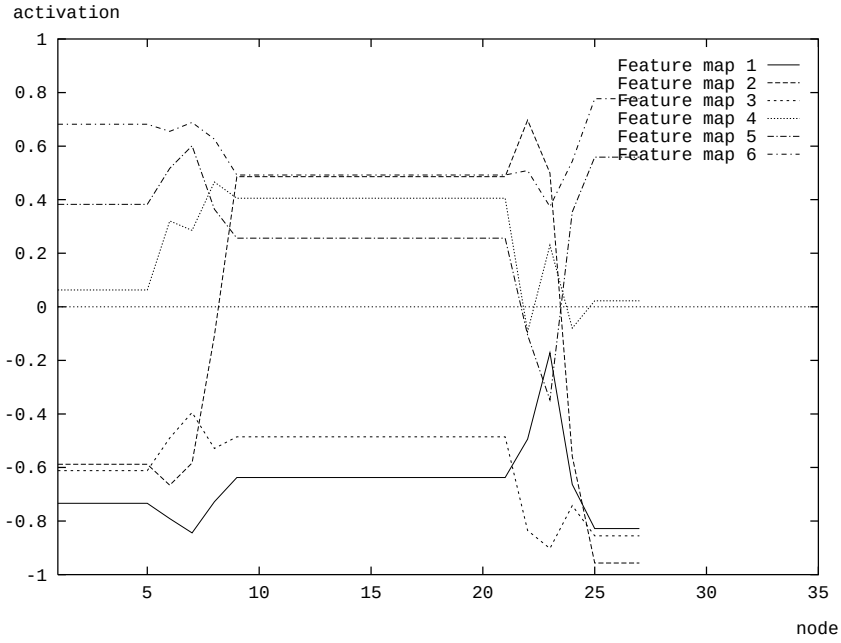

Figure 39: *Activation of first hidden layer per feature map for Test 21. These activations are for model A at depth 615 feet. This layer consists of 6 feature maps of 27 nodes each.*

4.7 Intermediate results architecture design

We train a convolutional-regression net on a larger set of models. The training set, validation set and test set is equal to the one used for the previous networks (28331, 14153 and 47485 patterns respectively). The net that is used is shown in Figure 36. The performance in comparison with the previous results is shown in Figure 40.
As one can see, the generalization performance of the convolutional-regression net is much better than for the previous nets.
We are going to use this type of net for the approximation of both the Deep- and the Shallow-log.

4.8 Summary and results

The nets with the best performance (training and generalization error) are the convolutional-regression nets. We train a net on the Shallow-log and one on the Deep-log. We use the convolutional-regression net from Test 21. This net has a sliding window of 29.8 feet, sampled every 0.2 feet (3×150 inputs), 27 receptive fields of 3.8 feet with 75 % overlap and 6 feature maps. We perform the following experiments:

- **Shallow-log (1)**: We train a convolutional-regression net on earth models with invasion (models A, B, C, D, E and F). The target logs are sampled every 0.2 feet. The results are given in Table 22.

Table 22: *Average relative error and performance for earth models with invasion (Shallow-log).*

	average relative error	**percentage of set predicted correct**	**number of patterns**
Training set	4.3 %	88 % correct	28331
Validation set	5.1 %	82 % correct	14153
Test set	6.2 %	77 % correct	47485

- **Shallow-log (2)**: After this training we add models without invasion and some models with invasion and restart the training. For this purpose we use the whole training set (see Table 5) (the target logs are now sampled every 1.0 feet) and 40 models without invasion (target logs wer sampled every 0.4 feet). The results are shown in Table 23. All models contain either invasion or not. Therefore, the performance is split into *invasion* and *no invasion*.

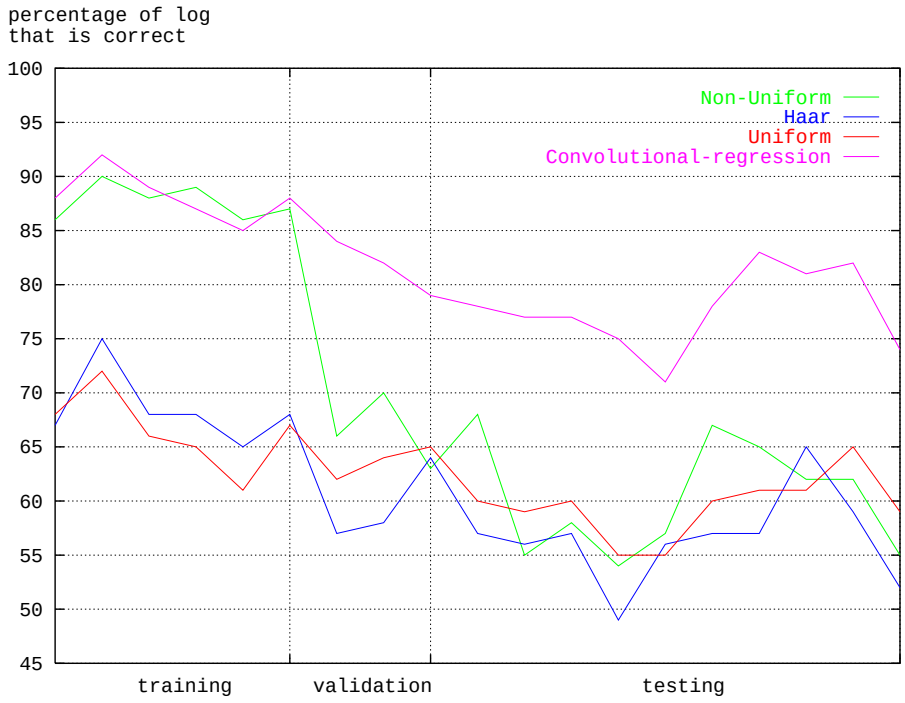

Figure 40: *Intermediate results (2) of training different neural nets on 6 models.*

Table 23: *Average relative error and performance for earth models with and without invasion (Shallow-log).*

		average relative error	**percentage of set predicted correct**	**number of patterns**
invasion	Training set	5.3 %	76 % correct	61688
	Validation set	5.5 %	75 % correct	14153
	Test set	5.6 %	74 % correct	14128
no invasion	Training set	5.6 %	77 % correct	10040
	Test set	5.6 %	74 % correct	10040

- **Deep-log**: We train a convolutional-regression net on both models with and without invasion. We use the same training set as for the Shallow-log. The results are given in Table 24.

Table 24: *Average relative error and performance for earth models with and without invasion (Deep-log).*

		average relative error	**percentage of set predicted correct**	**number of patterns**
invasion	Training set	8.4 %	52 % correct	61688
	Validation set	8.8 %	51 % correct	14153
	Test set	9.0 %	50 % correct	14128
no invasion	Training set	7.0 %	56 % correct	10040
	Test set	6.9 %	58 % correct	10040

The last two nets are not optimal in a sense that they are not trained from scratch. The net that was trained on the Shallow- log was optimal for models with invasion. Then we added models without invasion and new models with invasion. The net that was trained on the Deep-log used the optimal set of weights from the first net that was trained on the Shallow-log (this was done because the Shallow-log and Deep-log look very similar in practice) and continued its training on new models with and without invasion. Better results can be achieved by retraining the nets (random initialization of the weights) and directly using a large training set.

5 The neural network as fast forward model

In this section we present the neural networks that can be used as fast forward models. The accuracy is not yet sufficient, but can be improved by training on a more representative training set. For each net the "input domain" is given. This is a description of the models the net is trained on. For earth models similar to these training models, the net gives an approximation with an accuracy that is also given in this description. When an earth model lies outside this domain, one cannot expect the net to perform optimal.
For each net the best and worst test result is given. These results consist of (a part of) the earth model (only for models with invasion), the approximation of the log by the neural net and a cumulative error plot. In this cumulative error plot the percentage of the log that has a relative error below R % (for $R = 0...25$) is given. In all plots the invasion radius is given in 0.1 inch and the resistivities in Ωm.

5.1 Application to earth models without invasion

A description of this net can be found in Chapter 3. The net is fully connected. The input domain for the net that is trained on the Deep- and the Shallow-log, here called NoInvasion, is given in the following table:

net	NoInvasion	
tool response	Deep-log and Shallow-log	
size sliding window	15	feet
size training set	11797	examples
range Rt	1 ... 70	Ωm
range bed size	1 ... 5	feet
speed	400	pnt / sec
accuracy Deep-log	2.2	%
accuracy Shallow-log	6.0	%

In Figures 41 and 42 the worst and best test results are shown for the Deep-log approximation of the net NoInvasion. From the cumulative error plots for these approximations one can see that the deviation in performance is quite large. In the worst case only 8 % of the log has a relative error below 5 %, but in the best case this is 100 %.
In Figures 43 and 44 the worst and best test results are shown for the Shallow-log approximation of the net NoInvasion. The deviation in performance is much less than in the case of the Deep-log. In the worst case 75 % of the log has a relative error below 5 % and in the best case this is 100 %.

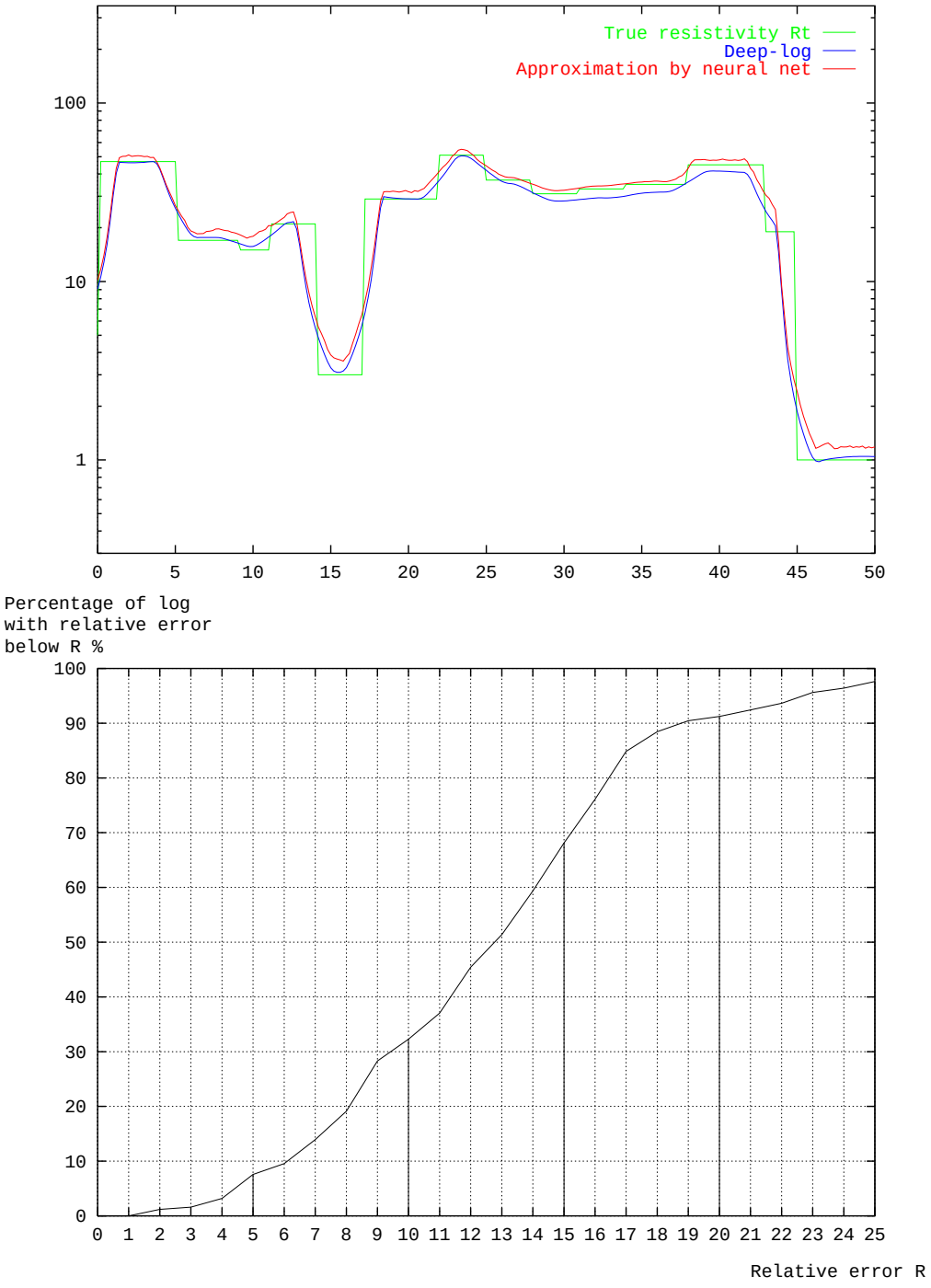

Figure 41: *Worst case neural net approximation of Deep-log. Average relative error is 14.2 %.*

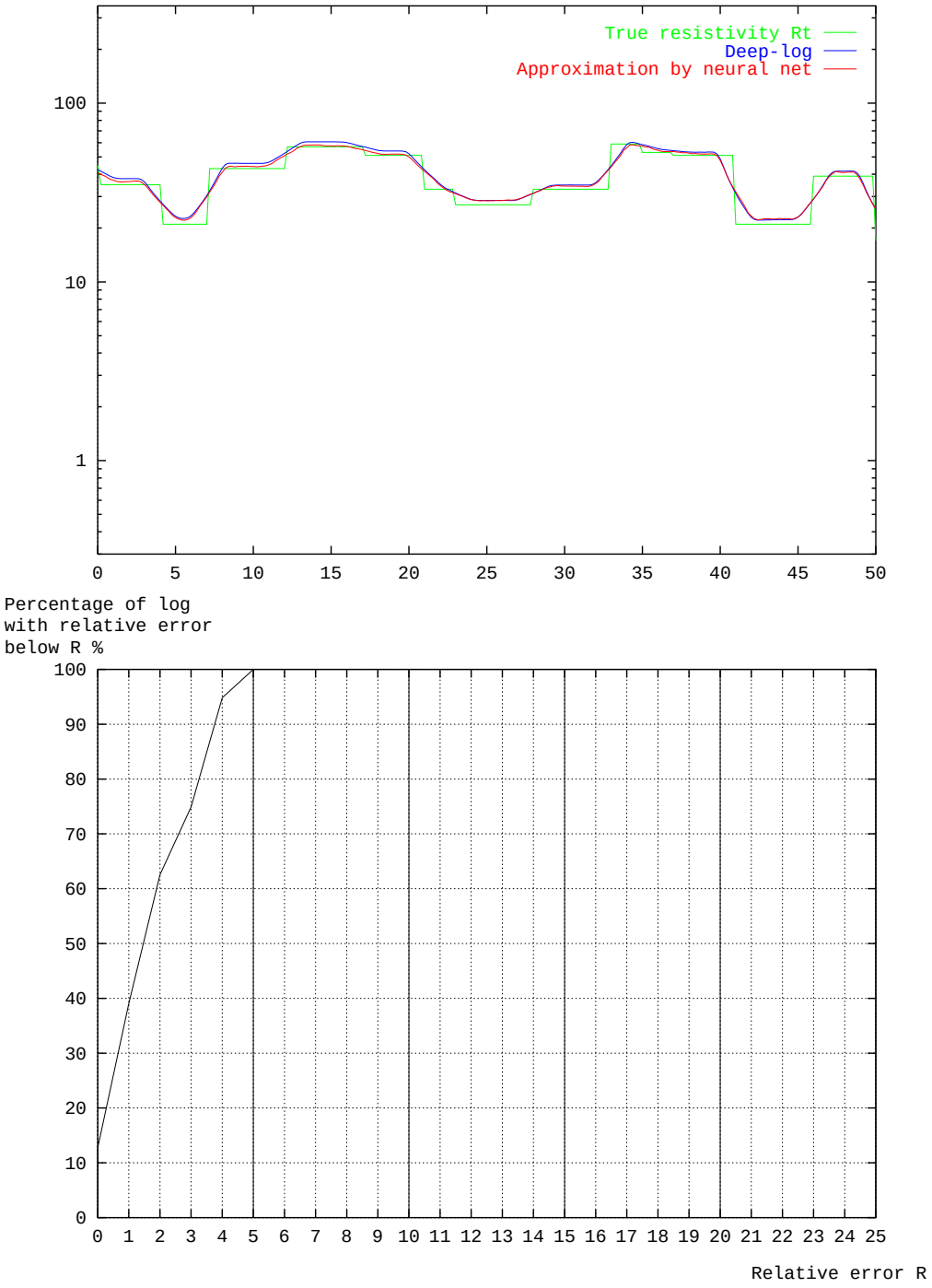

Figure 42: *Best case neural net approximation of Deep-log. Average relative error is 2.6 %.*

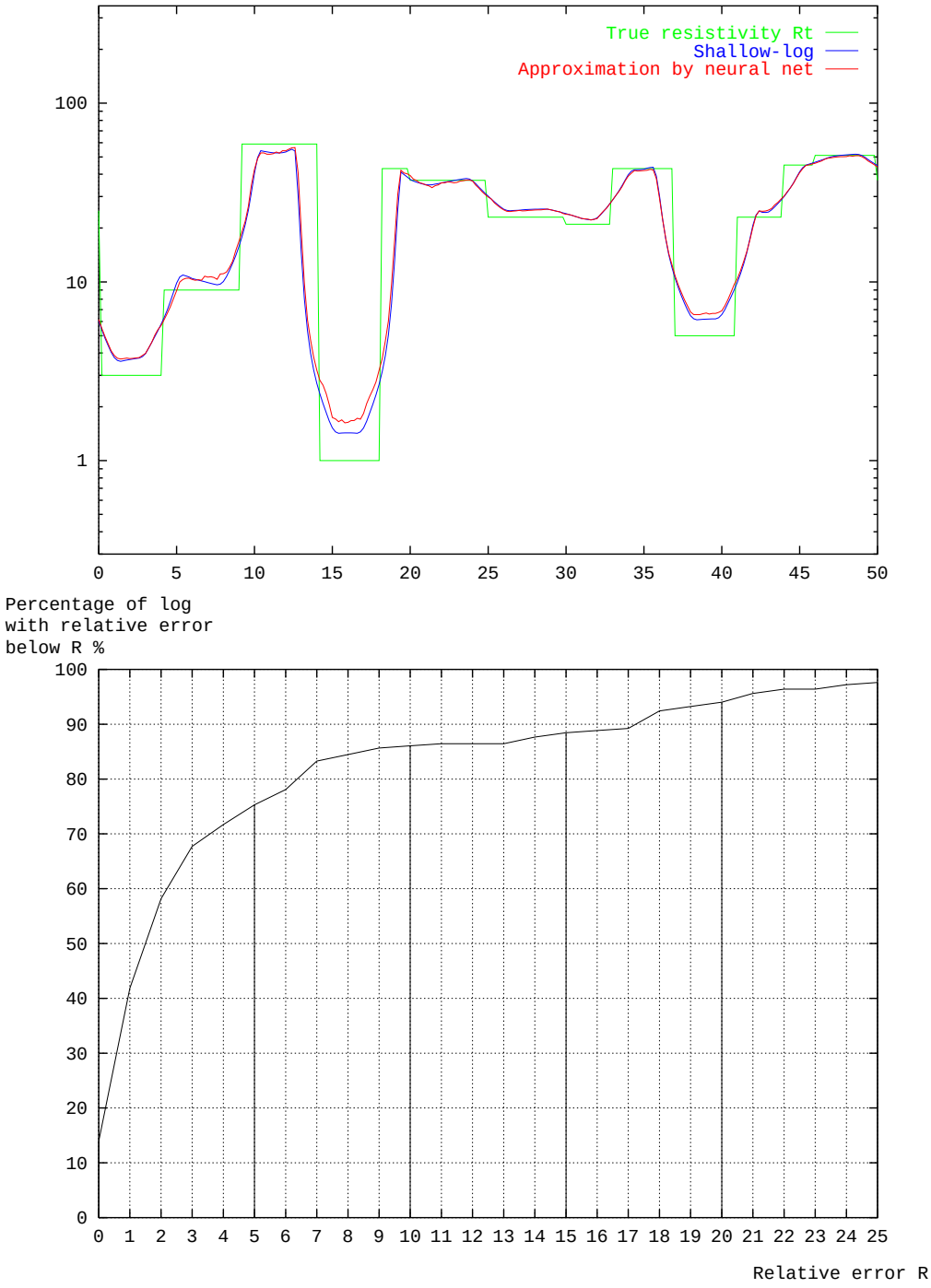

Figure 43: *Worst case neural net approximation of Shallow-log. Average relative error is 8.5 %.*

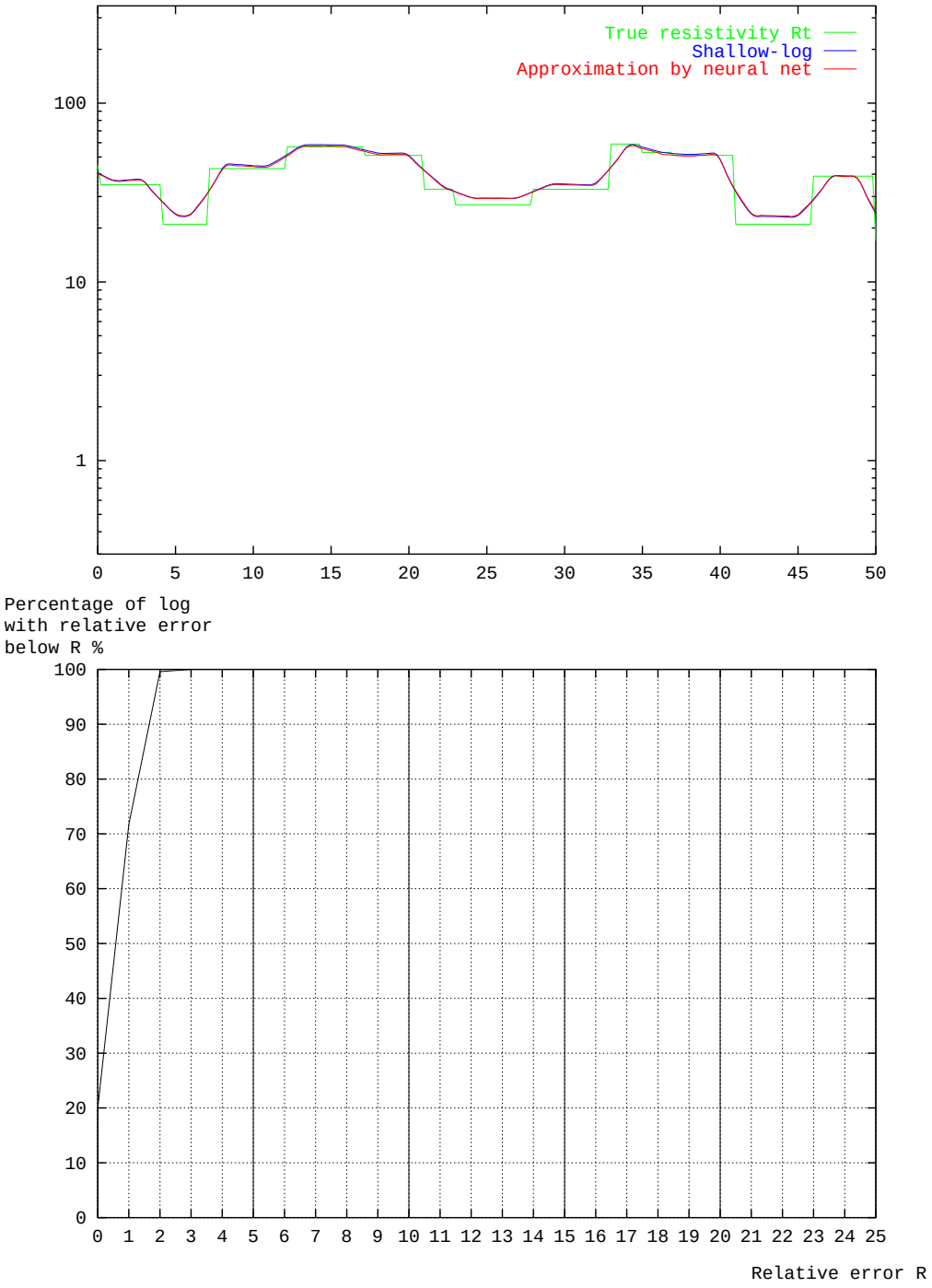

Figure 44: *Best case neural net approximation of Shallow-log. Average relative error is 1.2 %.*

5.2 Application to earth models with invasion

A description of this net can be found in Chapter 4. This is a convolutional-regression net. The following net only produces an approximation of the Shallow-log. It is trained on earth models as described in the following table:

net	Invasion	
tool response	Shallow-log	
size sliding window	30	feet
size training set	28331	examples
range Rt	1 ... 71	Ωm
range Rxo	0.5 ... 2.5	Ωm
range dxo	8 ... 50	inch
range bed size	1 ... 20	feet
speed	90	pnt / sec
accuracy Shallow-log	6.2	%

In Figure 45 two of the test models are shown. The model at the top is difficult for the neural net and the model at the bottom relatively easy. In Figures 46 and 47 the worst and best test results for the Shallow-log approximation of the net Invasion are shown.
In the cumulative error plot of the first approximation is shown that 70 % of the log is approximated with a relative error below 5 %. In the cumulative error plot of the second approximation one can see that in the best case 83 % of the log is approximated with a relative error below 5 %.
This network is optimal for the earth models it was trained on.

5.3 Application to realistic earth model

The domain of earth models the convolutional-regression net FM is trained on, are a combination of the previous models without and with invasion. A remark should be made that the value Rxo was taken to be 1.1 Ωm when dxo was 4.25 inch. In Figure 48 a formation model is shown that comes from a real oil well. This model is more realistic. It does not lie in the input domain, because it contains both parts with and without invasion. In our training set each model only contained either invasion or not. The beds are relatively small. The Deep-log and Shallow-log look very similar for this model, this is also different from the models the net is trained on. All in all this model looks quite different from the models in the training set and we do not expect the net to be optimal for this model.

A description of the earth models the net was trained on is given in the following table:

net	FM	
tool response	Deep-log and Shallow-log	
size sliding window	30	feet
size training set	61688	examples
range Rt	1 ... 71	Ωm
range Rxo	0.5 ... 2.5	Ωm
range dxo	4.25, 8 ... 50	inch
range bed size	1 ... 20	feet
speed	85	pnt / sec
accuracy Shallow-log	8.0	%
accuracy Deep-log	9.0	%

The approximation of the Deep-log is shown in Figure 49 and the approximation of the Shallow-log is shown in Figure 50. As we can see from the cumulative error plots, the relative error between the neural net response and the Deep-log is below 5 % for 67 % of the log. For 88 % of the log the relative error lies below 10 %.
The Shallow-log approximation is much better. Here 77 % of the log has a relative error below 5 %.

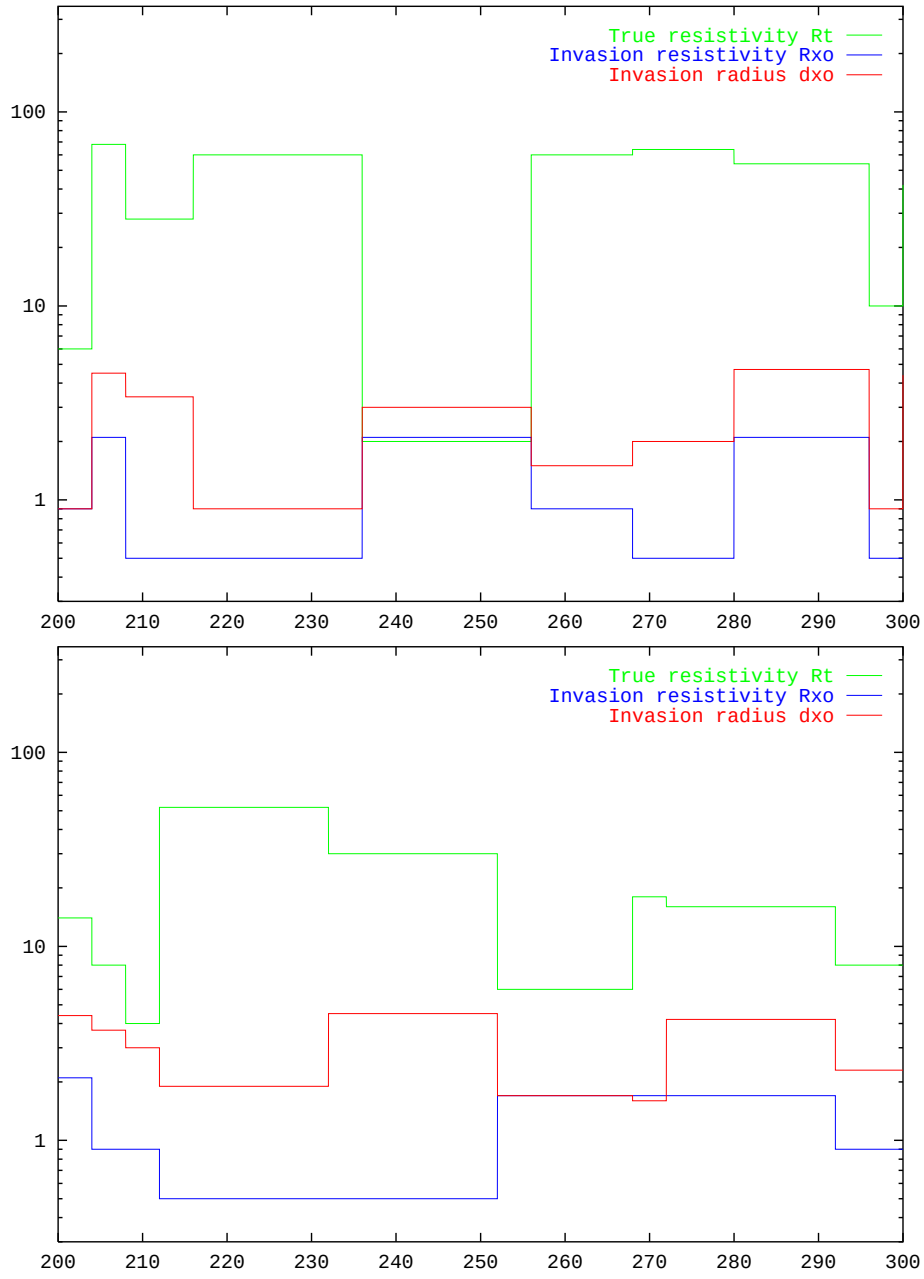

Figure 45: *Examples of earth models used for the net Invasion.*

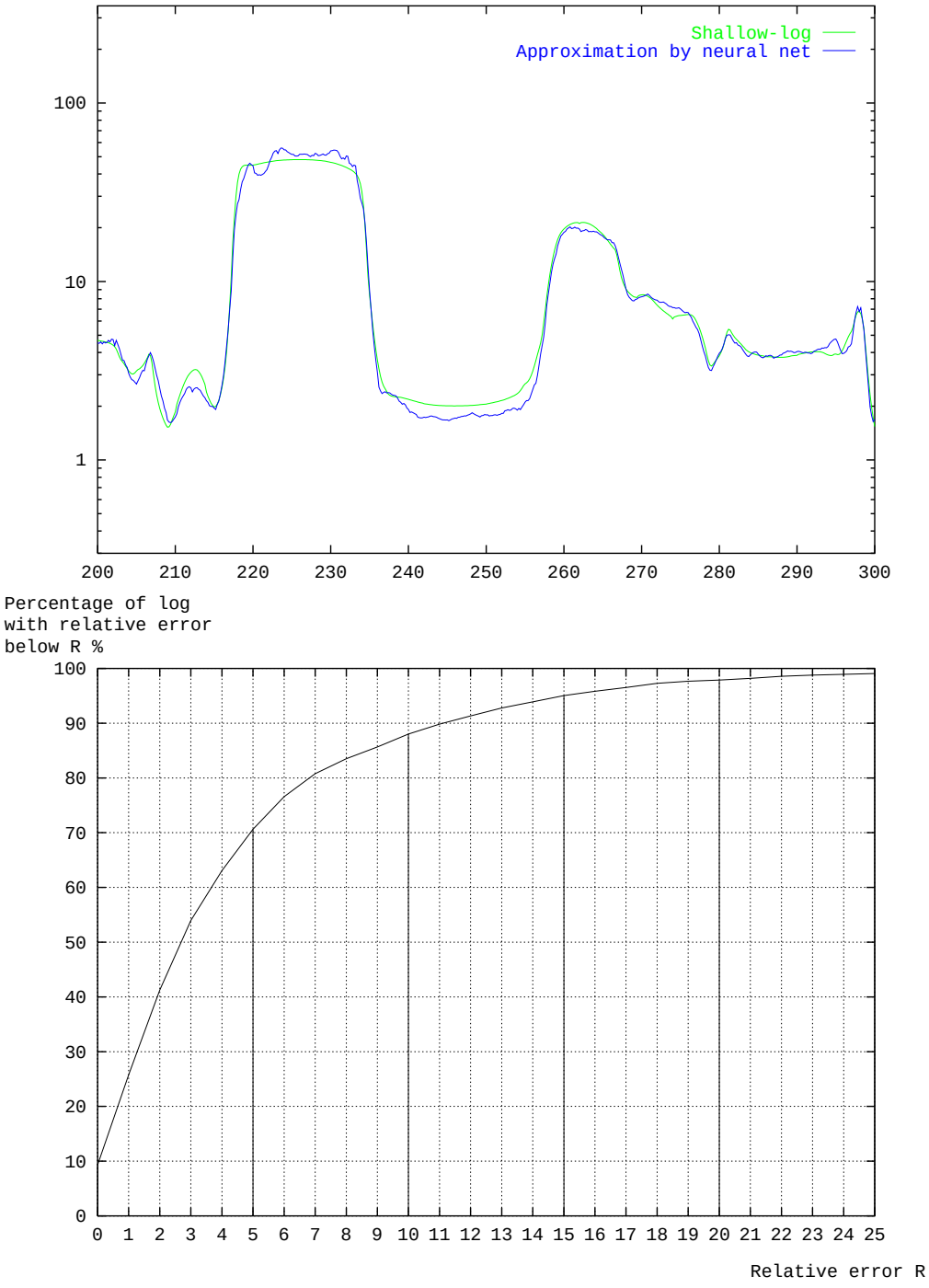

Figure 46: *Worst case neural net approximation of Shallow-log. Average relative error is 7.5 %.*

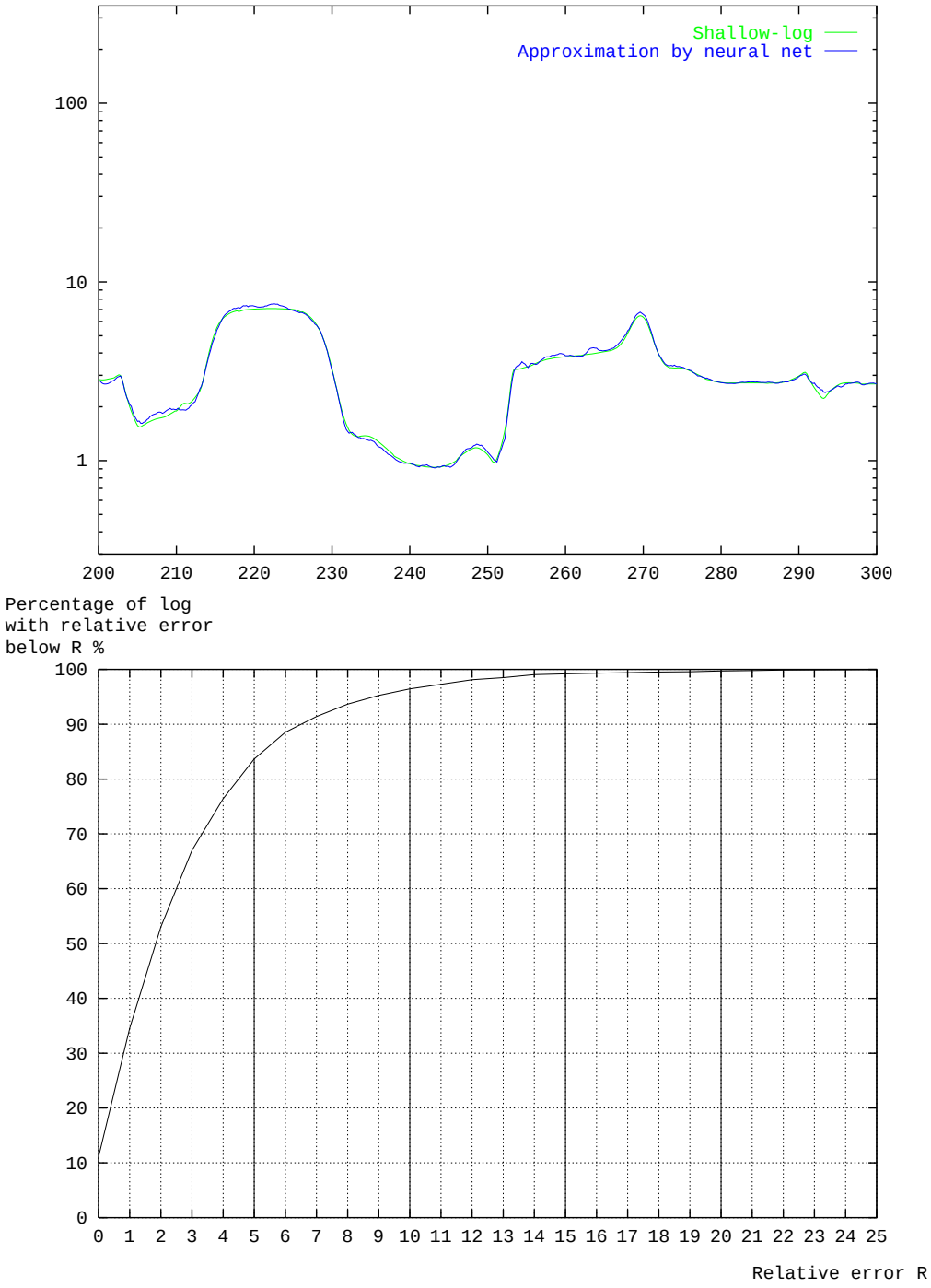

Figure 47: *Best case neural net approximation of Shallow-log. Average relative error is 4.5 %.*

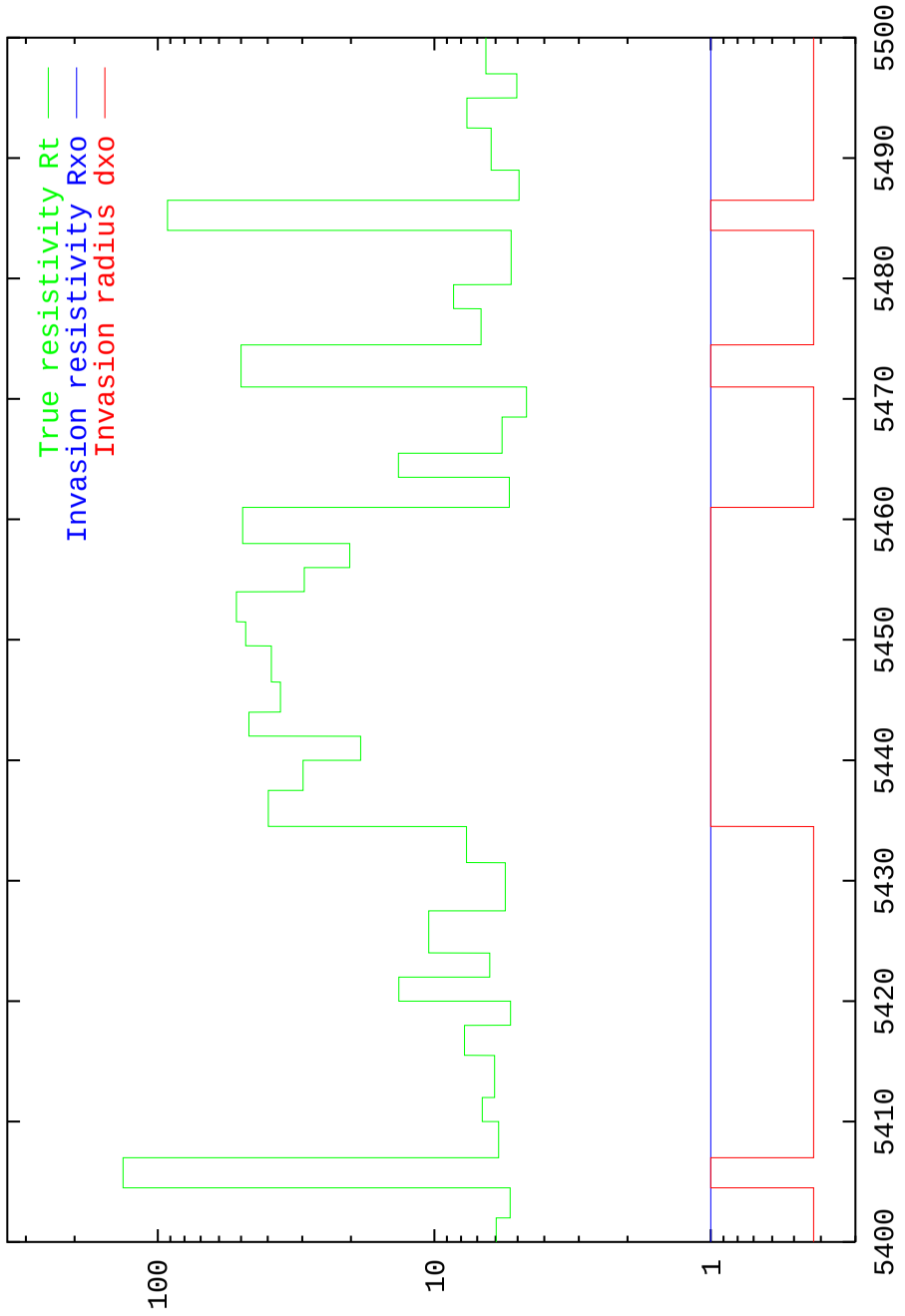

Figure 48: *A (realistic) earth model.*

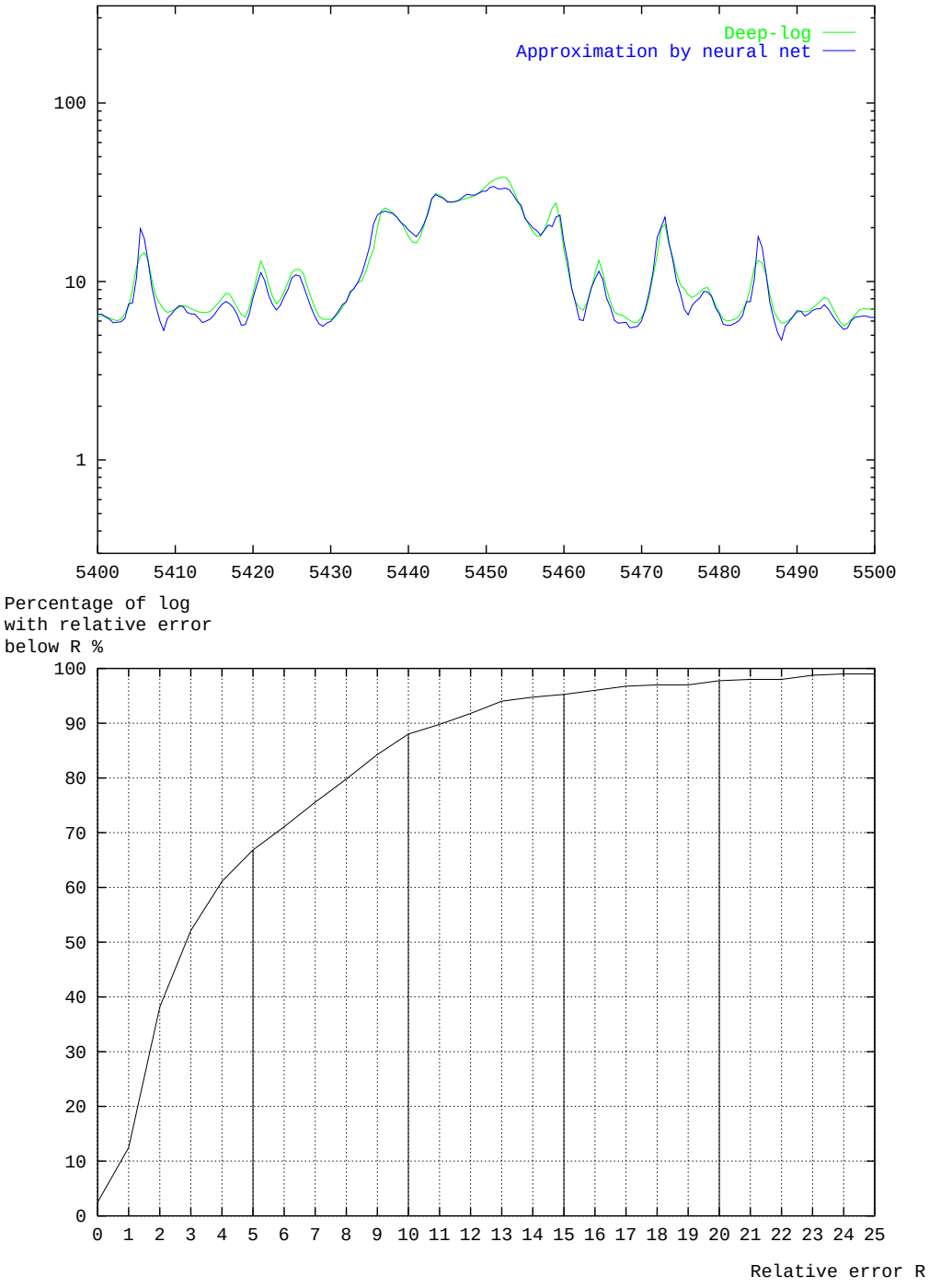

Figure 49: *Neural net approximation of Deep-log. Average relative error is 7.6 %.*

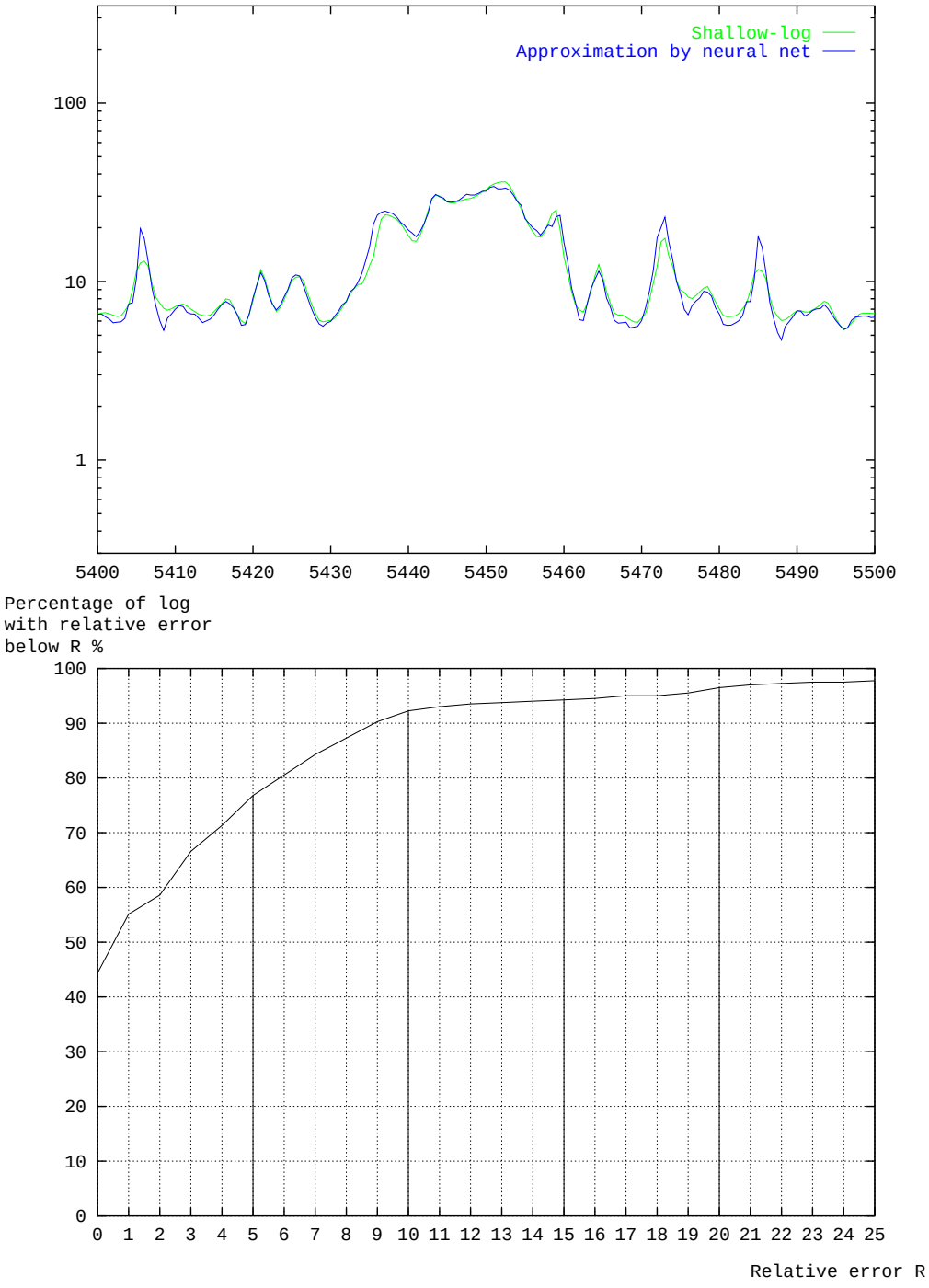

Figure 50: *Neural net approximation of Shallow-log. Average relative error is 8.3 %.*

6 Conclusions

In this section we present the conclusions following from the experiments we have performed and the methods we have investigated. The conclusions are split into three parts

1. A final conclusion about the goal of this project: investigating whether it is feasible to use a neural network in the forward modelling process.
2. Conclusions about the used methods. This involves the input representation, the techniques that are used to reduce the number of inputs and the architectural constraints.
3. Conclusions about the use of the neural network in training and testing. These conclusions are useful in further investigation and in the use of the trained network in other applications.

6.1 Neural network as fast forward model?

In the first part of the project we only used earth models without invasion and in the second part we only used earth models with invasion. For both types of earth models neural nets were trained with the following results:

- **No invasion**: We use a "standard" fully connected net with one hidden layer containing 15 nodes, a sliding window of 15 feet and a sampling period of 0.2 feet (75 inputs). The network has more trouble in learning the Deep-log than the Shallow-log. This is caused by the shoulder bed effect, which is more pronounced in the Deep-log than in the Shallow-log. The performance is given in the following table

Target log	**Data set**	**percentage of log with relative error below 5 %.**	**average relative error**
Shallow-log	Training set	98 %	1.7 %
	Test set	96 %	2.2 %
Deep-log	Training set	62 %	5.1 %
	Test set	61 %	6.0 %

- **Invasion**: We trained one convolutional-regression net on the Shallow-log. The net uses a sliding window of 29.8 feet, sampled every 0.2 feet, 27 receptive fields of 3.8 feet with 75 % overlap and 6 feature maps. The performance is given in the following table

Target log	**Data set**	**percentage of log with relative error below 5 %.**	**average relative error**
Shallow-log	Training set	88 %	4.3 %
	Validation set	82 %	5.1 %
	Test set	77 %	6.2 %

- **Mixed invasion and no invasion**: We used the same convolutional-regression net as in the previous tests with a training set consisting of earth models with and without invasion (but not mixed). This time we trained the net on both the Deep- and the Shallow-log. These nets are not optimal, because first the net was minimalized on models with invasion and then we added more models with and without invasion. In the following table the performance of the nets are given for the models with and without invasion

Target log		**Data set**	**percentage of log with relative error below 5 %.**	**average relative error**
Shallow-log	invasion	Training set	76 %	5.3 %
		Validation set	75 %	5.5 %
		Test set	74 %	5.6 %
	no invasion	Training set	77 %	5.6 %
		Test set	74 %	5.6 %
Deep-log	invasion	Training set	52 %	8.4 %
		Validation set	51 %	8.8 %
		Test set	50 %	9.0 %
	no invasion	Training set	56 %	7.0 %
		Test set	58 %	6.9 %

- **Real logging data**: Real earth models contain both layers with and without invasion. The trained convolutional-regression networks from the *mixed invasion and no invasion* part were tested on a realistic earth model. The performance of these nets on the real logging data is

Target log	percentage of log with relative error below 5 %.	average relative error
Shallow-log	77 %	8.3 %
Deep-log	67 %	7.6 %

  The network performance can be improved by using more models like this (partly with and partly without invasion) and by choosing the invasion radius between 4 and 25 inch instead of 8 and 50 inch.

- **Approximation time**: The goal of the project is to create a faster forward model. The neural network is approximately 100 times faster than the forward model that is presently used at KSEPL (both times measured on an IBM R6000 workstation). This could still be improved by optimization of the neural net calculations.

Although the accuracy of the approximation still needs some improvement, the network is a lot faster than the forward model that is used now. We conclude that it is feasible to use a neural network in the forward modelling process. Even with less accuracy the neural network could be used in the first iterations of the forward modelling process. In that way a fairly good initial guess can be made very quickly. Then one can use the more accurate forward model.

6.2 Methods

In the first part of the project we only used fully connected nets with varying number of layers, hidden nodes and variations in the size of the sliding window. More interesting results are found in the second part of the project, where we used earth models with invasion.

6.2.1 Input representation

We experimented with two input representation. The first, the discretized sliding window approach, gave the best results (training and generalization) and has been used in further tests:

- **Discretized sliding window**: We used a uniform sampled sliding window as input to the neural net. The number of inputs coming from

this window can be quite high, especially in the case with invasion. The training result was good, but the network was not able to generalize well.

- **Attributes:** Again we used a fixed size sliding window, but now we describe the beds that occur in this window. Each bed was described by a number of attributes. The problem, however, was not described well by these attributes. To find appropriate attributes is a difficult task.

6.2.2 Input reduction

All preprocessing methods we used were successful. We achieved a high input reduction without loss of performance (measured in training and generalization results):

- **Sampling method**: The sliding window can be sampled uniform and non-uniform. In the first case we experimented with different sampling periods. A coarse sampling period results in less inputs and the training and generalization results were comparable. The bed boundaries are described less accurately. The non-uniform sampling method worked very well. This method is based on the tool physics. The tool receives most of its information from the center of the window and less from the edges. Non-uniform sampling results in less inputs and the generalization performance of the net was better than for the net that used a uniform-sampled sliding window as input. Here we only loose accuracy at the edges of the window.

- **Principal components**: With the projection of the N-dimensional input vector to a M-dimensional vector, described by M principal components, an input reduction of 75 % was achieved. The training results were comparable and the network generalized better than the original discretized input, although approximately 7 % of the information of the input was lost (measured in the relative distance between the original and projected input).

- **Haar transform**: The advantage of transformation of the original input values to the wavelet coefficients is that a number of the coefficients can be removed. A reduction of 73 % of the number of inputs was achieved with good training and generalization results. By removing coefficients at the edges of the window, we loose some accuracy. The bed boundaries outside the center of the window are described less accurately.

6.2.3 Architecture design

- **Fully connected nets**: In the case without invasion a fully connected net can be used. The generalization performance and the training results are comparable. When the beds contain invasion, we need three parameters to describe the bed. This type of network does not generalize well. The number of connections in this net is very high, resulting in a long training time and storage and memory problems. Preprocessing methods can be used to reduce the number of inputs.

- **Locally connected nets**: A neural that is locally connected has much less weights and better generalization than the fully connected nets. The training time needed for these nets is shorter than for the fully connected nets. The size of the receptive fields and the overlap are difficult to determine. We choose, however, for small fields with large overlap. In this way it is easier for the net to determine the precise location of a feature that occurs in one of the fields. This net needs at least two hidden layers: one for detecting the local features and one for combining the found features.

- **Symmetry constraints**: An input signal and its mirror image give the same tool response and should therefore also give the same network response. This requirement leads to certain weight constraints in the fully, locally and wavelet nets. The number of weights is reduced, because certain weights are "shared" (equal). The reduction in the number of weights makes the training more difficult, but it improves the generalization performance in comparison with the same nets without these constraints.

- **Convolutional-regression nets**: In the convolutional-regression net all receptive fields share the same weights. One group of hidden nodes with shared weights is called a feature map. The reduction of freedom is compensated by an increase in the number of feature maps. This net performs a convolution on the input with a convolution kernel, the set of shared weights, that is has learned itself. The generalization performance of this type of net is better than for all the other nets we have tried.

6.3 Application of the convolutional-regression net

When the convolutional-regression net is going to be used in other applications, the following aspects are important:

- **Size of the sliding window**: The size of the sliding window depends on the application. One should have an idea of how much of the input log is responsible for the output.

- **Sampling period of sliding window**: The sampling period of the sliding window determines how accurate the input is described. A good indication is to use the same sampling period as for the target log.
- **Receptive fields**: The size and overlap of the receptive fields determine the accuracy of the location of the features in the input. Use small fields with high overlap. The number of weights, however, depend on the number of receptive fields. Choose the size of the fields and the overlap so that the number of weights is not too large.
- **Representative training set**: The training set is constructed from the target logs. Do not take a too dense sampling of the target log, otherwise most examples look too much alike. It is better to take a (very) coarse sampling period and use a large number of different logs. In this way the training set consists of a large number of very different examples. Create models that are likely to be found in reality.
- **Size training set**: The number of training examples that are needed in order to get good generalization performance depends upon the representativeness of the training set. When the training set contains a high number of different examples, one needs approximately 10 times the number of weights as examples.
- **Minimalization network error**: The error that the network is minimalizing can be adapted to the requirements of the approximation. With the combined logarithmic and normalization scaling the neural net is minimalizing the proportion d/a, where d is the desired output and a is the actual output.

The convolutional-regression net can easily be adapted for problems with more variables than *Rt*, *Rxo* and *dxo*. When there are for example dipping layers, the dip angle can be added as fourth variable (for each sampling point). This is only possible when there is a different dip angle for each bed in the formation. The scaling of a new variable needs new investigation. The bore hole radius and resistivity of the drilling fluid can also be added as variables, but like the dip angle, this should only be done when there are enough different values.

A Neural network simulators

In this project we used the Xerion network simulator versions 3.1 and 4.0 (van Camp 1993). We also tried other network simulators, Stuttgart Neural Network Simulator (SNNS) (Zell 1993) and Aspirin/MIGRAINES (Leighton 1992), but we found that Xerion provides more freedom in specifying architecture constraints. The Xerion simulator allows you to alter the network design on any level: layers, nodes and even connections. This is very useful for the implementation of the convolutional- regression nets, where we constrain certain weights to be equal.

The SNNS simulator is a beautiful graphical simulator and has a special option for time delayed neural nets (see Section 2.3.5). A layer is specified by its number of features (or variables) and its total delay length (number of nodes in sliding window). Layers can be fully or locally connected. Receptive fields are specified by their size (delay length) and always have maximum overlap (displacement of one node). All receptive fields have shared weights. It is, however, not possible to constrain arbitrary weights to be the same. And it is also not possible to change the overlap of the receptive fields. We present the inputs as a sequence within a sliding window, but for the SNNS simulator one should present the inputs at one specific logging point. The advantage of this is that the file that contains the data is much smaller than in our case. There are only 4 values per pattern (Rt, Rxo, dxo and the tool response) instead of $3 \times(\frac{w}{s}+1)$ for a window size w and a sampling period s. The delay length specifies how many inputs before this input affect the output. One cannot specify how many inputs after this input affect the output. So there are three main points why this simulator is not suitable for our purposes:

1. It is not possible to specify arbitrary weight constraints.
2. The overlap of the receptive fields is always maximal.
3. One can only specify how many inputs before (and not after) the current input affect the output.

It is not possible to specify convolutional-regression nets with this simulator.

Aspirin/MIGRAINES is a compiler. It compiles your code into a working neural network simulator (in C). In the source code the neural net is specified by its components. A component is described by the name of the layer, the size of the layer (in nodes) and connection information (how the layer is connected to other layers). Receptive fields with shared weights are called *Shared Tessellations* and one can specify the size of the fields and the overlap in the x- and y-direction. It is not possible, however, to constrain specific weights to be shared, which we needed in the symmetry experiments.

It is possible to create convolutional-regression nets with this simulator.

Although receptive fields are easier defined by the previous simulators, we have used the Xerion simulator. All hidden neurons in one feature map are connected by $n \times m$ links to their corresponding receptive field (of size $n \times m$). All these connections have to be specified, which can mount up to 1620 connections in the convolutional-regression net. Neurons can be constrained to have the same incoming links, which is what we did per feature map. Then the nodes in one feature map are constrained to have the same weights. Our Xerion topology files contained a total of 9951 lines. The reading of this file and the building of the net is quite slow, but it outweighs the lack of freedom of the other simulators.

References

Anderson, B. & Barber, T. (1990). Modelling electromagnetic tool response, *Oilfield Review* pp. 22–32. Well logging.

Asquith, G. (1982). Basic well log analysis for geologists.

Chemali, R., Gianzero, S. & Strickland, R. (1983). The shoulder bed effect on the dual laterolog and its variation with the resistivity of the borehole fluid, *SPWLA 24th annual logging symposium* .

Chemali, R., Gianzero, S. & Su, S. (1988). The dual laterolog in common complex situations, *SPWLA 29th annual logging symposium* .

Gianzero, S. (1977). Characteristic responses of resistivity tools in elliptical boreholes, *IEEE transactions on geoscience electronics* **GE-15**(4): 251–256.

Gianzero, S., Lin, Y. & Su, S. (1985). A new high-speed hybrid technique for simulation and inversion of resistivity logs.

Guyon, I. (1991). Neural networks and applications tutorial, *Physics Reports (Review section of Physics Letters) 207*(3–5): 215–259. Review Section of Physical Letters 207.

Haykin, S. (1994). *Neural Networks, A Comprehensive Foundation*, Macmillan College Publising Company, Inc.

Hertz, J., Krogh, A. & Palmer, R. (1991). *Introduction to the theory of neural computation*, Addison-Wesley Publishing Company.

Leighton, R. R. (1992). *The Aspirin/MIGRAINES Neural Network Software*, v6.0 edn.

Moran, J. (1985). Focused resistivity logs, *Developments in geophysical exploration methods* **6**: 225–260.

Strang, G. (1989). Wavelets and dilation equations: A brief introduction, *SIAM Review* **31**(4).

van Camp, D. (1993). *A Users Guide for The Xerion Neural Network Simulator*, ver3.1 & ver4.0 edn, University of Toronto.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K. & Lang, K. (1989). Phone recognition using time-delay neural networks, *IEEE Transactions on acoustics, speech and signal processing* **37**(3).

Zell, A. (1993). *SNNS Stuttgart Neural Network Simulator*, v3.1 edn, Institute for Parallel and Distributed High Performance Systems.