ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
Τμήμα Ηλεκτρολόγων & Ηλεκτρονικών Μηχανικών

UNIVERSITY of WEST ATTICA
FACULTY OF ENGINEERING
Department of Electrical & Electronics Engineering

Πρόγραμμα Μεταπτυχιακών Σπουδών
Διαδικτυωμένα Ηλεκτρονικά Συστήματα

Master of Science in
Internetworked Electronic Systems

MSc Thesis

Machine Learning Modeling in Service of Geophysics Analysis: Estimation of Oil Industry Parameters

Student: BOVIATSIS, Dimitrios, Registration Number IES-0030
MSc Thesis Supervisors: POTIRAKIS, Stylianos, Professor
TATLAS, Nicolas-Alexander, Assistant Professor

ATHENS-EGALEO, 2020

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
Τμήμα Ηλεκτρολόγων & Ηλεκτρονικών Μηχανικών

UNIVERSITY of WEST ATTICA
FACULTY OF ENGINEERING
Department of Electrical & Electronics Engineering

Πρόγραμμα Μεταπτυχιακών Σπουδών
Διαδικτυωμένα Ηλεκτρονικά Συστήματα

Master of Science in
Internetworked Electronic Systems

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Machine Learning Modelling in Service of Geophysics Analysis: Estimation of Oil Industry Parameters

Μεταπτυχιακός Φοιτητής: Δημήτριος Μποβιάτσης, ΑΜ: IES-0030
Επιβλέποντες: Στυλιανός Ποτηράκης, Καθηγητής
Νικόλαος-Αλέξανδρος Τάτλας, Επίκουρος Καθηγητής

ΑΙΓΑΛΕΩ, 2020

ABSTRACT

CSEM (Controlled Source Electromagnetics) has been used in parallel with seismic surveys for the past decade by the oil industry and is a development of older geophysical techniques that aim at the "mapping" of electrical properties, especially electrical resistance, of the subsoil. This thesis aims to study statistical well log data analysis methods, to accelerate the solution of the inverse problem which is required during CSEM data processing. The inverse problem determines the distribution of the electrical resistance of the subsoil, which would generate signals equal to the actual received signals at the receivers. Solving the inverse problem is a computationally demanding and time-consuming process (it takes several hours to days for a complete solution by computing clusters).

In particular, the possibility of providing reliable estimations of the ground electrical resistivity using sonic well-log data will be studied. These estimations will be used as initial conditions in the process of solving the inverse problem which, in turn, can significantly reduce the duration of the inversion while increasing its accuracy.

KEYWORDS: *AIC, CSEM, Data Analysis, Inverse Problem, Machine Learning, Neural Networks, Parameter Estimation, Resistivity, RMSE, System Identification*

ΠΕΡΙΛΗΨΗ

Η τεχνολογία CSEM (Controlled Source Electromagnetics) χρησιμοποιείται παράλληλα με σεισμικές μεθόδους, την τελευταία δεκαετία, από την πετρελαϊκή βιομηχανία και αποτελεί εξέλιξη παλιότερων γεωφυσικών τεχνικών οι οποίες στοχεύουν στην «χαρτογράφηση» των ηλεκτρικών ιδιοτήτων, ειδικά της λεκτρικής αντίστασης, του υπεδάφους. Η διπλωματική αποσκοπεί στην μελέτη στατιστικών μεθόδων ανάλυσης γεωφυσικών δεδομένων γεωτρήσεων προκειμένου να επιταχυνθεί η επίλυση του αντίστροφου προβλήματος που απαιτείται κατά την ανάλυση CSEM δεδομένων. Με το αντίστροφο πρόβλημα (inverse problem) υπολογίζεται η κατανομή της ηλεκτρικής αντίστασης του υπεδάφους η οποία θα δημιουργούσε στους δέκτες σήματα ίσα με τα πραγματικά λαμβανόμενα. Η επίλυση του αντιστρόφου προβλήματος είναι υπολογιστικά απαιτητική και χρονοβόρα διαδικασία (απαιτούνται αρκετές ώρες έως και ημέρες για την πλήρη επίλυση από υπολογιστικά clusters).

Συγκεκριμένα, θα μελετηθεί η δυνατότητα προσεγγιστικού υπολογισμού της ηλεκτρικής αντίστασης του εδάφους χωρίς την επίλυση του αντίστροφου προβλήματος. Ο προσεγγιστικές τιμές θα χρησιμοποιούνται ως αρχικές συνθήκες στην διαδικασία επίλυσης του αντιστρόφου προβλήματος. Θα εξεταστεί επίσης η πιθανή βελτίωση στην διάρκεια και ακρίβεια επίλυσης του αντίστροφου προβλήματος.

ΛΕΞΕΙΣ – ΚΛΕΙΔΙΑ: *AIC, CSEM, Data Analysis, Inverse Problem, Machine Learning, Neural Networks, Parameter Estimation, Resistivity, RMSE, System Identification*

ACKNOWLEDGEMENTS

In this piece of work, I would like to thank everyone, in general, for contributing to the completion of my work.

First, I would like to thank my advisors, Mr. **POTIRAKIS** Stylianos and Mr. **TATLAS** Nicolas-Alexander, who entrusted me with the study of this interesting subject.

Also, thanks to Mr. **VARDOULIAS** Georgios and **MVest Energy** partners, I was granted access to Norwegian well log data.

Special thanks to Mr. **PARASKEVAS** Spyridon, for his valuable help with R programming language and the discussions we made regarding Machine Learning prospects.

This work is dedicated to my family.

TABLE OF SYMBOLS -ACRONYMS-ABBREVIATIONS

Abbreviations	Full Description
AIC	Akaike Information Criterion
ANN	Artificial Neural Network
BIC	Bayesian Information Criterion
BP	Backpropagation
BPROP	Backpropagation for Batch training
CG	Conjugate Gradient
CGM	Conjugate Gradient Method
CSEM	Controlled Source Electro-Magnetic
EM	Electromagnetism
GA	Genetic Algorithm
GD	Gradient Descent
GOR	Gas Oil Ratio
LM	Lagrange Multiplier Test
LMS	Least Mean Square
MAE	Mean Absolute Error
MD	Measured Depth
MDT	Modular formation Dynamics Tester
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NN	Neural Network
PL	Production License
PVT	Pressure, Volume, Temperature
QN	Quasi-Newton method
RBF	Radial Basis Function
RBFN	Radial Basis Function Networks
RD	Deep Resistivity
RKB	Rotary Kelly Bushing
RL	Reinforcement Learning
RMSE	Root-Mean-Square Error

RPROP	Resilient Backpropagation
RSNNS	neural networks in R using the Stuttgart Neural Network Simulator (SNNS)
RSS	Residual Sum of Squares
SCG	Scaled Conjugate Gradient
SD	Steepest Descent Method
SGD	Stochastic Gradient Descent Method
SNNS	Stuttgart Neural Network Simulator (SNNS)
SVM	Support-Vector Machine
TVD	True Vertical Depth
VP	Compressional Velocity
VS	Shear Velocity

TABLE OF FIGURES

TABLE OF CONTENTS

INTRODUCTION:
Aims, Objectives and Research Questions – Thesis Outline

Statement of the Problem

CSEM (Controlled Source Electromagnetics) has recently been developed by the oil industry and is an evolution of older geophysical techniques that aim at the "mapping" of electrical properties, especially electrical resistance, of the subsoil.

This thesis aims to study methods that will help accelerate the solution of the inverse problem. The inverse problem determines the distribution of the electrical resistance of the subsoil, which would generate signals equal to the actual received signals at the receivers. Solving the inverse problem is a computationally demanding and time-consuming process.

What we do here is this: Can we predict the electrical resistivity from sonic well-logs which reflect the elastic properties of rocks?

Related Work

An approach has been suggested by Economou and Aleaei (1), in order to develop models capable of estimating regional subsurface resistivity in a well, based on information available from regional wells.

What this approach suggests is applying a multivariate non-linear method on data derived from wells. Some interesting remarks have been made, after studying this technical paper. These are:

- The performance index of the model used in the work is not fully understood. The determination coefficient (R^2) does not give the correct sense of the model error.
- Multiple network topologies, or other families of Machine Learning (ML) algorithms, are not tested to identify the best estimation model for the variable output. Also, no hyper-parameters are set, such as learning rate, etc.
- No resampling method is used to adapt the network to the available data, in order to be able to have an estimate of the performance of the algorithm for data that were not available during network training.

Purpose and Research Questions

In this thesis, besides reproducing the methodology of the work above, we will also:

- Deliver performance results to a prediction performance measure which can be exploited by the industry (RMSE, MSE, MAE).
- Test different neural network topologies.
- Perform parameter tuning of neural networks.

All this will be done under a properly selected re-sampling experiment, in order to ensure a satisfactory assessment of the performance of the algorithms, leading, in the end, at choosing the best. This work aims to improve the physical constrains, leading to an improvement in the final CSEM inversion result.

Approach and Methodology

This thesis focuses on using machine learning methods and algorithms to train models. A machine learning technique will be used, in an estimation process, to predict electrical resistivity from elastic parameters. Further an approach on how to evaluate the trained model quality, based on the predictions, will be proposed. This procedure is going to be followed throughout this thesis:

Based on research on an existing method, an iterative parameter estimation process will be implemented, to answer the aforementioned research questions. This process includes the determination of the quality criteria for available data, the implementation of metrics, the fine tuning of a neural network algorithm, as well as the optimization of the machine learning approach, to achieve the best possible results. Finally, the process and results will be critically reviewed, evaluated and compared to one another.

Target Group

This work is especially interesting for research in the area of machine learning, applied geophysics, etc. and more generally in the area of parameter estimation, by using combinations of different metrics and machine learning approaches.

The primary interest groups for this research are international companies, related with the oil industry, as knowledge obtained from a trained model can help during

parameter inversion process. Also, this work is of general interest, since it provides a robust method that can be used in estimating any parameter.

Outline

This thesis report is structured into six chapters. Chapter 1 provides a theoretical explanation of machine learning theory. After reviewing the methodology that will be followed in our research, in Chapter 2, the proposed parameter estimation method will be described, initially at a concise and then at a fully analytical level, in Chapter 3. Chapters 4 and 5 present, discuss and criticize the results of the implementation phase. In the last chapter, conclusion of the project is given and recommendations for further improvements are discussed.

CHAPTER 1:
Theoretical Framework – Field Survey

How quickly does an acoustic signal propagate through a material? How effective is this material at conducting electricity? These questions arise when dealing with a geophysical/geotechnical/geological problem. To be able to answer these questions, sufficient understanding of material physical properties is ultimately needed. Physical properties characterize how materials respond to various physical inputs. In this chapter, a short description of the physical properties of rocks will be given.

Physical Properties of Rocks and Hydrocarbons

Density

Density (ρ) is defined as the mass (m) contained within a material per unit volume (V):

$$\rho = mV$$

In the geophysical context, density is measured through a tool that sends gamma rays into a formation and detects those that are scattered back (2). Each rock type is defined by a range of density values (not a specific value). The densities for rocks, minerals, and other materials are summarized below (3).

Table 1 Densities for various materials

Material	Density Range ($g/c\,m^3$)
Air	0.001225
Water	1.00
Ice	0.917
Petroleum	0.60 - 0.90
Rocks	1.50 -7.60

Susceptibility

Magnetic susceptibility (κ) quantifies the magnetization ($\vec{M}$) that a rock or mineral experiences when it is subjected to an applied magnetic field ($\vec{H}$). This relationship takes the form:

$$\vec{M} = \kappa \vec{H}$$

Within the mineral grains comprising rocks, there are tiny magnetized volumes (magnetic domains). The direction and magnitude of each magnetic domain is defined by its magnetic dipole moment ($\vec{m}$). When the material is subjected to an external magnetic field, $\vec{H}$, the magnetic moments of these dipoles try to re-orient themselves along the direction of the field.

Seismic Velocity

Elastic energy propagates through the earth in different ways and at different speeds. Elastic waves are comprised of compressional (or P-waves) and shear (or S-waves). In compressional waves, the particle motion is in the direction of propagation. In shear waves, the particle motion is perpendicular to the direction of propagation. Understanding the velocity of these waves provide valuable information about the rocks and fluids through which they propagate. (4)

Elastic properties are used to define the physical deformation that matter experiences in response to an applied mechanical force. Seismic velocities define the speed at which various elastic deformations propagate through materials. Seismic velocities for a given material can be expressed explicitly in terms of its elastic properties, which are not covered in the scope of this thesis.

P-waves are compressional waves in which particle motion is in the direction of the wave propagation, while S-waves are shear waves in which the particle motion is perpendicular to the direction of wave propagation. The P and S-wave velocities are related to the elastic properties of the medium by the following expressions:

$$v_p = \sqrt{K + \frac{4\mu}{3}\rho}$$

$$v_s = \sqrt{\frac{\mu}{\rho}}$$

where K is the bulk modulus, μ is the shear modulus and ρ is the density. Symbols used to define P-wave and S-wave velocities, as well as the elastic properties on which they depend, are summarized in the following table:

Table 2 Seismic velocities and elastic properties summary

Property	Symbol	Units
P-Wave Velocity	V_p	$m/s \vee km/s$
S-Wave Velocity	V_s	$m/s \vee km/s$
Bulk Modulus(Incompressibility)	K	$Pa \vee GPa$
Shear Modulus (Rigidity)	M	$Pa \vee GPa$
Density	P	$kg/m\,3 \vee g/cm\,3$

P-waves and S-waves travel at different speeds depending on the media they are propagating through; P-waves travel faster than S-waves. P-waves are able to propagate through solids and fluids, while S-waves can only propagate through solid materials. A summary over the range of P-wave and S-wave velocities for common materials is given below:

Table 3 Seismic velocities for common materials. (3)

Material	P-wave (m/s)	S-wave (m/s)
Air	343	N/A
Water	1450 - 1500	N/A
Ice	3400 - 3800	1700 - 1900
Oil	1200 - 1250	N/A
Dry Sands	400 - 1200	100 - 500
Wet Sands	1500 - 2000	400 - 600

Saturated Shales and Clays	1100 - 2500	200 - 800
Porous and Saturated Sandstones	2000 - 3500	800 - 1800
Coal	2200 - 2700	1000 - 1400
Salt	4500 - 5500	2500 - 3100

Electrical Conductivity

Most of physical properties of oil are identical to those of water: density is of the same order of magnitude, the same magnetic permeability, almost the same elastic and radioactive properties, etc. Only the electrical conductivity, i.e. the ability to conduct an electrical current, clearly differentiates oil from water. Simply put, oil is resistant while the deposit water is conductive through the greater or lesser presence of dissolved salts. (5)

Electrical conductivity (σ) describes how easily electric currents can flow through a medium when subjected to an applied electric field. It defines the relationship between the density of electrical current density ($\vec{J}$) within a material, and the electric field ($\vec{E}$):

$$\vec{J} = \sigma\vec{E}$$

where σ is measured in Siemens per meter (S/m).

Electrical conductivity of rocks is associated with porosity and permeability of rocks, as well as the conductivity of liquids that flow through porous rocks. The inverse of conductivity, resistivity, has been a subject of discussion for a long time, since its virgin appearance in 1930, when scientists were searching the distribution of electrical resistivity in Earth's subsurface:

$$\rho = \frac{1}{\sigma}$$

Electrical resistivity, among other quantities and seismic data, are being implemented in hydrocarbon investigation in oil fields. It is measured in *ohmm*. Resistivity of a rock sample with can be expressed as:

$$R = \frac{\Delta E}{I}\frac{A}{L}$$

where: R = the resistivity of the sample (Ωm or Ohm.m)

ΔE = the potential difference applied across the sample (volts, V)

I = the current flowing through the sample (amperes, A)

A = the cross-sectional area of the sample perpendicular to the current flow (m^2

L = the length of the sample (m)

Looking at Table 4, it is evident that the resistivity of reservoir rocks depends upon the water or water-based mud occupying its porous space. (6)

Table 4 Components of reservoir rocks

Description	Resistivity
Rock matrix	High
Formation water	Low
Oil	High
Gas	High
Water-based mud	Low
Oil-based mud	High

Resistivity Transforms

Geophysicists process seismic data to obtain seismic velocities. Seismic velocities are derived from raw data, as a result of aligning seismic arrivals. Seismic measurements provide us with information about Earth's subsurface.

In the past, many scientists have tried to find transforms of elastic to electric parameters. However, getting from velocities to resistivities is not a trivial task; the underlying seismic wave propagation theory and EM wave propagation theory share no physical property. (7)

Archie's law is an empirical relationship between porosity and given by

$$\rho = \alpha \rho_f \varphi^{-m}$$

where ρ is the resistivity of the formation, ρ_f is the resistivity of the fluids saturating the formation, φ is the porosity of the formation, m is the cementation exponent and α is the tortuosity factor. Archie found this relationship empirically by analyzing laboratory resistivity measurements. Archie's equation is often used to predict resistivity from reservoir properties.

Archie's law postulates that the rock matrix is non-conductive, which is true for clear sands. For sandstones with clay minerals, this assumption is no longer true, as things get more complicated. (8)

Next, another empirical relation, Faust's equation, which gets us directly from velocity to resistivity. This equation relates seismic velocity with the true resistivity of the formation:

$$\frac{1}{V} = \Delta T = k \rho^{-\alpha} Z^{-b}$$

where Z is the true or vertical depth, V is the seismic velocity and ρ is the resistivity of the formation. This type of transform has been used in cases where the sonic logs were missing. However, it does not account for gas effects.

The approximation that will be used in this thesis is similar with Faust's relation.

Machine Learning

Definition

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Arthur Samuel was the first one to use the term "Machine Learning", in 1959, to define a science of getting computers to perform a task "*without being explicitly programmed"* to do so.

Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions. Machine learning algorithms are used in the applications of email filtering, detection of network intruders, and computer vision, where it is infeasible to develop an algorithm of specific instructions for performing the task. Machine learning is closely related to computational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a field of study within machine learning, and focuses on exploratory data analysis through unsupervised learning.

Types of Learning

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

Supervised learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. (9) In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way (see inductive bias).

Unsupervised learning

Unsupervised learning is a branch of machine learning that learns from test data that has not been labeled, classified or categorized. Instead of responding to feedback, unsupervised learning identifies commonalities in the data and reacts based on the presence or absence of such commonalities in each new piece of data. Alternatives include supervised learning and reinforcement learning.

Reinforcement learning

Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. The problem, due to its generality, is studied in many other disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics and genetic algorithms. In the operations research and control literature, reinforcement learning is called approximate dynamic programming, or neuro-dynamic programming. (10) (11).

Function Approximation

Generally, we can divide all function approximation tasks into classification tasks and regression tasks.

Classification Predictive Modeling

Classification predictive modeling is the task of approximating a mapping function (f) from input variables (X) to discrete output variables (y). The output variables are often called labels or categories. The mapping function predicts the class or category for a given observation. For example, an email of text can be classified as belonging to one of two classes: “spam“ and “not spam“.

Regression Predictive Modeling

Regression predictive modeling is the task of approximating a mapping function (f) from input variables (X) to a continuous output variable (y). A continuous output variable is a real-value, such as an integer or floating point value. These are often quantities, such as amounts and sizes.

Common Machine Learning Techniques

The number of machine learning algorithms has been growing rapidly during the last years. However, these algorithms often have similar characteristics and can be categorized. In this section, we will give a short review of the mathematical theory behind each algorithm.

Artificial neural networks

Artificial neural networks (ANNs), or connectionist systems, are computing systems vaguely inspired by the biological neural networks that constitute animal brains. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

Support vector machines

Support vector machines (SVMs), also known as support vector networks, are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.

Bayesian networks

A Bayesian network, belief network or directed acyclic graphical model is a probabilistic graphical model that represents a set of random variables and their conditional independence with a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms.

Genetic algorithms

A genetic algorithm (GA) is a search algorithm and heuristic technique that mimics the process of natural selection, using methods such as mutation and crossover to generate new genotypes in the hope of finding good solutions to a given problem.

Artificial Neural Networks

Neural Network Architectures

Neural networks (12) (13) (14) are powerful nonlinear regression techniques inspired by theories about how the brain works. The outcome of a Neural Network is modeled by an intermediary set of unobserved variables (called hidden variables, hidden units or neurons). These hidden units are linear combinations of the original predictors, but they are not estimated in a hierarchical fashion (15).

Each hidden unit is a linear combination of some or all of the predictor variables. However, this linear combination is typically transformed by a nonlinear function $g(\cdot)$, such as the logistic (i.e., sigmoidal) function. So, these units perform two functions: the combining function and the activation function:

$$h_k(x) = g\left(\beta_{0k} + \sum_{i=1}^{P} x_j \beta_{jk}\right), \text{ where}$$
$$g(u) = \frac{1}{1 + e^{-u}}$$

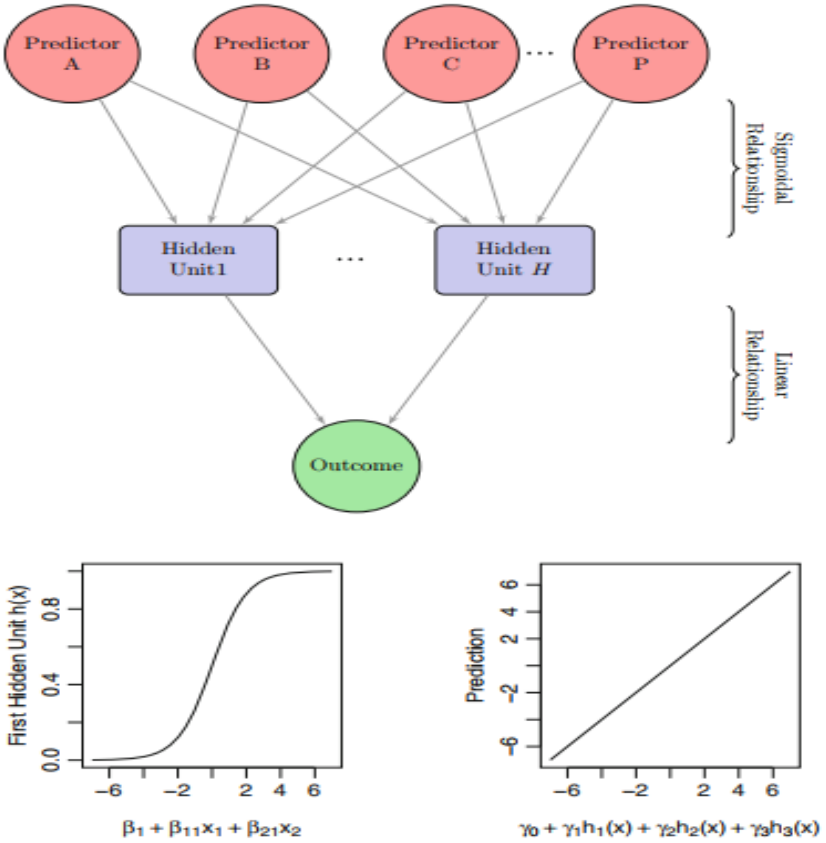


Figure 1 A diagram of a neural network with a single hidden layer. The hidden units are linear combinations of the predictors that have been transformed by a sigmoidal function. The output is modeled by a linear combination of the hidden units (15).

The β coefficients are similar to regression coefficients; coefficient β_{jk} is the effect of the jth predictor on the kth hidden unit. A neural network model usually involves multiple hidden units to model the outcome.

Once the number of hidden units is defined, each unit must be related to the outcome. Another linear combination connects the hidden units to the outcome:

$$f(x) = \gamma_0 + \sum_{k=1}^{H} \gamma_k h_k$$

In system modelling applications, the dynamic range of output data is usually greater than 1, thus the activation function of the output nodes is chosen to be linear.

For this type of network model and P predictors, there are a total of H(P + 1) + H + 1 total parameters being estimated, which quickly becomes large as P increases. For 3

predictors, a neural network model with four hidden units would estimate 21 parameters while a model with five hidden units would have 26 coefficients.

Number of Hidden Units and Layers

Generally speaking, it is better to have too many hidden units than too few. With too few hidden units, the model might not have enough flexibility to capture the nonlinearities in the data; with too many hidden units, the extra weights can be shrunk toward zero if appropriate regularization is used.

Typically, the number of hidden units is somewhere in the range of 5 to 100, with the number increasing with the number of inputs and number of training cases. It is most common to put down a reasonably large number of units and train them with regularization. Some researchers use cross-validation to estimate the optimal number, but this seems unnecessary if cross-validation is used to estimate the regularization parameter.

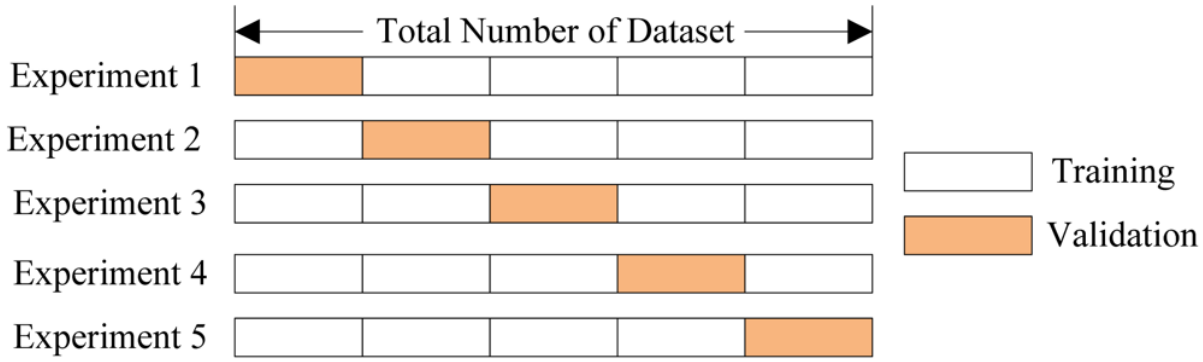


Figure 2 5-fold cross validation (15).

Choice of the number of hidden layers is guided by background knowledge and experimentation. Each layer extracts features of the input for regression or classification. Use of multiple hidden layers allows construction of hierarchical features at different levels of resolution.

Training Algorithms

Treating this model as a nonlinear regression model, the parameters are usually optimized to minimize the sum of the squared residuals. This can be a challenging numerical optimization problem (recall that there are no constraints on the parameters of this complex nonlinear model). The parameters are usually initialized to random values and then specialized algorithms for solving the equations are used. The back-propagation

algorithm (16) is a highly efficient methodology that works with derivatives to find the optimal parameters. However, it is common that a solution to this equation is not a global solution, meaning that we cannot guarantee that the resulting set of parameters are uniformly better than any other set.

Similar to other steepest descent-type algorithms, the back-propagation algorithm suffers from a slow convergence rate. Also, it may become trapped at local minima and can be sensible to other selectable parameters. Some modifications made in the basic back-propagation algorithm help improve some of these fallbacks.

Also, neural networks have a tendency to over-fit the relationship between the predictors and the response due to the large number of regression coefficients. To combat this issue, several different approaches have been proposed. First, the iterative algorithms for solving for the regression equations can be prematurely halted (17). This approach is referred to as early stopping and would stop the optimization procedure when some estimate of the error rate starts to increase (instead of some numerical tolerance to indicate that the parameter estimates or error rate are stable). However, there are obvious issues with this procedure. First, how do we estimate the model error? The apparent error rate can be highly optimistic and further splitting of the training set can be problematic. Also, since the measured error rate has some amount of uncertainty associated with it, how can we tell if it is truly increasing?

Another approach to moderating over-fitting is to use weight decay. Here, we add a penalty for large regression coefficients so that any large value must have a significant effect on the model errors to be tolerated. Formally, the optimization produced would try to minimize an alternative version of the sum of the squared errors:

$$\sum_{i=1}^{N}\left(y_i - f_i(x)\right)^2 + \lambda \sum_{k=1}^{H}\sum_{j=0}^{P}\beta_{jk}^2 + \lambda \sum_{k=0}^{H}\gamma_k^2$$

for a given value of $\lambda \geq 0$.

As the regularization value increases, the fitted model becomes smoother and less likely to over-fit the training set. Of course, the value of this parameter must be specified and, along with the number of hidden units, is a tuning parameter for the model. Reasonable values of λ range between 0 and 0.1. Also note that since the regression coefficients are being summed, they should be on the same scale; hence the predictors should be centered and scaled prior to modeling.

The structure of the model described here is the simplest neural network architecture: a single-layer feed-forward network. There are many other kinds, such as models where there are more than one layers of hidden units (i.e., there is a layer of hidden units that models the other hidden units). Also, other model architectures have loops going both directions between layers. Practitioners of these models may also

remove specific connections between objects to further optimize the model. There have also been several Bayesian approaches to neural networks (18). The Bayesian framework outlined in (18) for these models automatically incorporates regularization and automatic feature selection. This approach to neural networks is very powerful, but the computational aspects of the model become even more formidable. A model very similar to neural networks is self-organizing maps (19). This model can be used as an unsupervised, exploratory technique or in a supervised fashion for prediction (20).

Given the challenge of estimating a large number of parameters, the fitted model finds parameter estimates that are locally optimal; that is, the algorithm converges, but the resulting parameter estimates are unlikely to be the globally optimal estimates. Very often, different locally optimal solutions can produce models that are very different but have nearly equivalent performance. This model instability can sometimes hinder this model. As an alternative, several models can be created using different starting values and averaging the results of these model to produce a more stable prediction (21) (22) (23). Such model averaging often has a significantly positive effect on neural networks.

These models are often adversely affected by high correlation among the predictor variables (since they use gradients to optimize the model parameters). Two approaches for mitigating this issue is to pre-filter the predictors to remove the predictors that are associated with high correlations. Alternatively, a feature extraction technique, such as principal component analysis, can be used prior to modeling to eliminate correlations. One positive side effect of both these approaches is that fewer model terms need to be optimized, thus improving computation time.

In Figure 3, model averaged neural networks were used. Three different weight decay values were cross-validated (λ = 0.00, 0.01, 0.10) along with a single hidden layer with sizes ranging between 1 and 13 hidden units. The final predictions are the averages of five different neural networks created using different initial parameter values. The cross-validated RMSE profiles of these models are displayed. It is shown that, increasing the amount of weight decay.

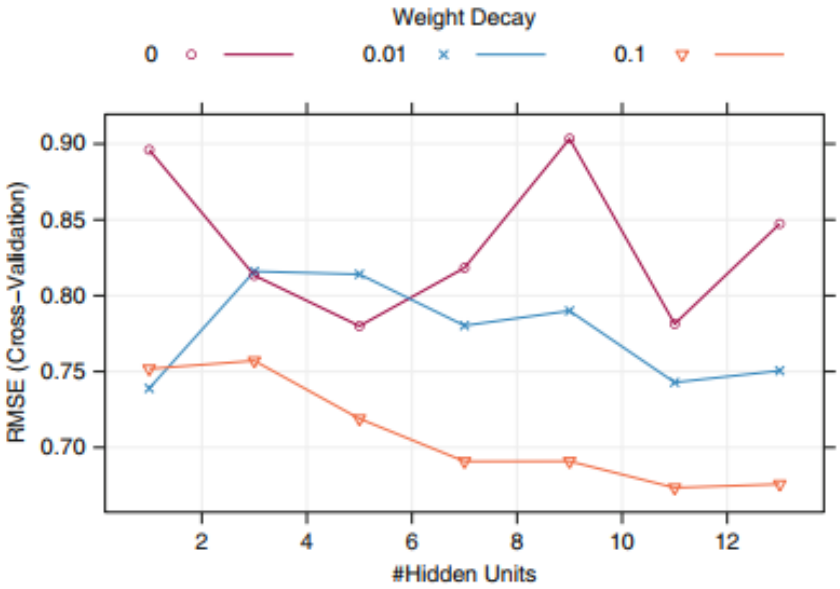


Figure 3 RMSE profiles for a neural network model. The best model utilizes λ = 0.1 and 11 hidden units (15).

clearly improved model performance, while more hidden units also reduce the model error. The optimal model used 11 hidden units with a total of 2,531 coefficients. Also, performance of the model is fairly stable for a high degree of regularization (i.e., λ = 0.1), so smaller models could also be effective for these data.

Multilayer Perceptrons and Back-propagation

The back-propagation algorithm has emerged as the key for the design of a special class of layered feedforward networks, multilayered perceptrons (MLPs). A multilayer perceptron has an input layer of nodes and an output layer of neurons, which act as an interface with the outside world.

In addition to these two layers, an MLP usually has one or more layers of hidden neurons. The hidden layer extracts important features contained in the input data.

The training of an MLP is usually accomplished by using a back-propagation (BP) algorithm, in two phases:

- Forward phase: During this phase the free parameters of the network are fixed and the input signal is propagated through the network in a “layer by layer” basis. The forward phase finished with the computation of an error signal:

$$e_i = d_i - y_i$$

where d_i is the desired response and y_i is the actual output produced by the network in response to the input x_i

- Backward phase: During this second phase, the error signal e_i is propagated through the network in the backward direction, hence the name of the algorithm. It is during this phase that adjustments are applied to the free parameters of the network so as to minimize the error function e_i in a statistical sense.

The two-pass procedure that has been described above, known as back-propagation, is also called the Delta Rule or Least Mean Square (LMS) method (24). It is closely related to the Gauss-Newton algorithm. The back-propagation learning algorithm is simple to implement and computationally efficient in that its complexity is linear in the synaptic weights of the network.

Online learning is a method of machine learning in which data becomes available in a sequential order and is used to update the best predictor for future data at each step, as opposed to batch learning techniques which generate the best predictor by learning on the entire training data set at once. Online learning is a common technique used in areas of machine learning where it is computationally infeasible to train over the entire dataset, requiring the need of out-of-core algorithms. It is also used in situations where it is necessary for the algorithm to dynamically adapt to new patterns in the data, or when the data itself is generated as a function of time, e.g., stock price prediction.

When applying the backpropagation algorithm, which is a typical gradient steepest descent method, decisions have to be made concerning the:

- learning rate, i.e. the step size or the magnitude of weight updating
- momentum, which is required for escaping the trapping in local minima.

An appropriate selection of learning rate is particularly important because the steepest descent method suffers from slow convergence and weak robustness. Convergence acceleration by taking a larger learning rate bears the danger of network oscillatory behaviour around the minimum. To avoid this, and still to take a larger learning rate, addition of a momentum parameter was recommended (16). The added term stabilizes the learning convergence. An alternative way is to implement an adaptive step size. (25)

Manipulating the parameters of the neuron transfer function can also improve both objectives, namely of learning velocity and of convergence stabilization, as proposed by (26). Back-propagation can be very slow, and for that reason is usually not the method of

choice. Second-order techniques such as Newton's method are not attractive here, because the second derivative matrix of the error function can be very large. Better approaches to fitting include conjugate gradients and variable metric methods. These avoid explicit computation of the second derivative matrix while still providing faster convergence.

Some simple heuristics to make better back-propagation learning are: (27)

- Use neurons with antisymmetric activation functions (e.g. hyperbolic tangent function) instead of nonsymmetric ones (e.g. logistic function).
- Shuffle the training examples after the presentation of each epoch; an epoch involved the presentation of the entire set of training examples to the network.
- Follow an easy-to-learn example with a difficult one.
- Preprocess the input data so as to remove the mean and decorrelate the data.
- Arrange the neurons in the different layers to learn at essentially the same rate. This may be attained by assigning a learning parameter to neurons in the last layer that is smaller than those at the front end
- Incorporate prior information into the network design whenever it is available

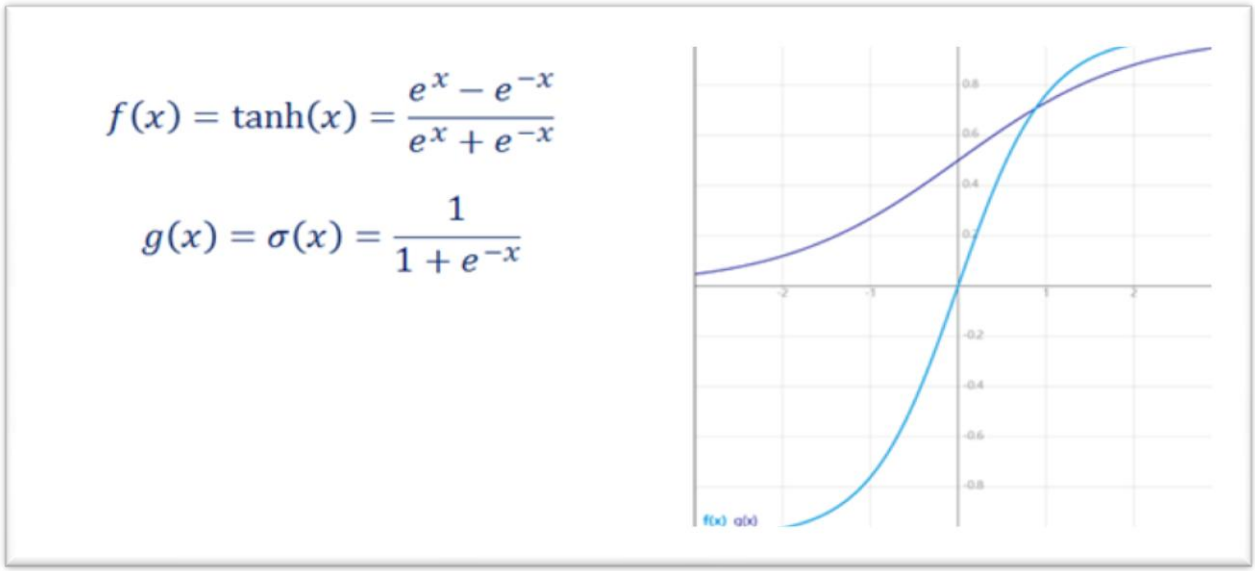


Figure 4 Hyperbolic Tangent and Logistic functions (15).

Radial Basis Functions

A second category of neural network modeling techniques is that of radial basis networks. In these single-hidden-layer networks, the nonlinear activation function is a Radial Basis Function (RBF). RBF networks have the advantage of being much simpler than the MLP networks, while acting as universal approximators of functions (28).

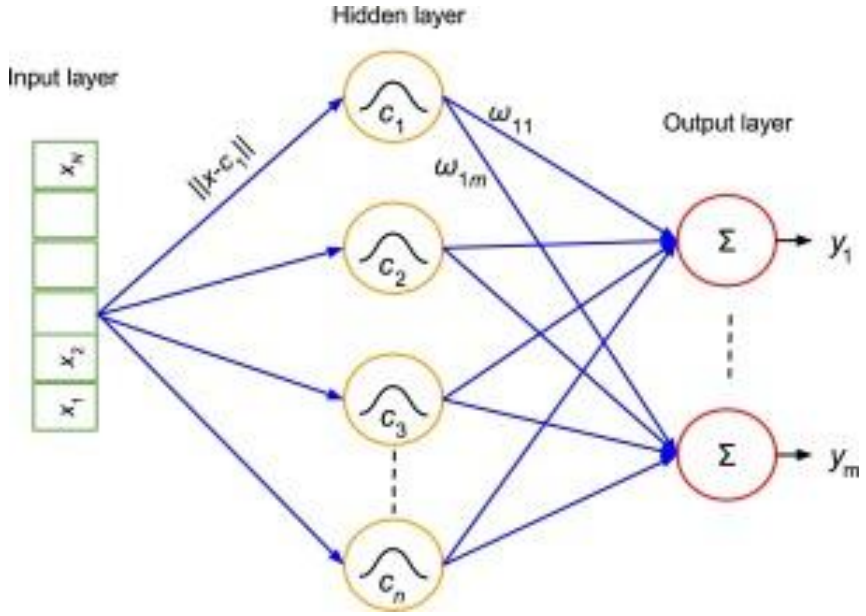


Figure 5 A*rchitecture of a radial basis function network. An input vector x is used as input to all radial basis functions, each with different parameters. The output of the network is a linear combination of the outputs from radial basis functions.*

RBF architecture is similar to that of MLPs. However, the output is computed as a linear combination of basis functions. The distance between the basis functions' centers and the input data is computed by the hidden layer. Each node in the hidden layer evaluates a RBF based on incoming inputs. The RBF can overcome the possibility of producing complex error surfaces, produced by back-propagation algorithm. In most applications of RBF networks, the Gaussian function is used as the hidden layer activation function.

CHAPTER 2:
Research Methodology

System Identification Problem

Constructing (mathematical) models from data is a prime problem in many scientific fields and many application areas. Many communities and cultures around the area have grown, with their own nomenclatures and their own "social lives". This has created a very rich, and somewhat confusing, plethora of methods and approaches for a problem. (29)

The System Identification problem has to do with estimating a model of a system based on observed input-output data. (30) Several ways to describe a system and to estimate such descriptions exist.

The procedure to determine a model of a dynamical system from observed input-output data involves three basic ingredients (31):

1. The input-output data
2. A set of candidate models (the model structure)
3. A criterion to select a particular model in the set, based on the information in the data (the identification method)

The identification process amounts to repeatedly selecting a model structure, computing the best model in the structure, and evaluating this model's properties to see if they are satisfactory. This methodology can be itemized as follows (32):

1. Design an experiment and collect input-output data from the process to be identified.
2. Examine the data. Polish it so as to remove trends and outliers, and select useful portions of the original data. Possibly apply filtering to enhance important frequency ranges.
3. Select and define a model structure (a set of candidate system descriptions) within which a model is to be found.
4. Compute the best model in the model structure according to the input-output data and a given criterion of fit.
5. Examine the obtained model's properties

6. If the model is good enough, then stop; otherwise go back to Step 3 to try another model set. Possibly also try other estimation methods (Step 4) or work further on the input-output data (Steps 1 and 2).

Design of the Experiment

We are going to try to infer relationships between measured inputs and outputs. First, we are going to need a finite number of inputs and outputs, let it be vector $\varphi(t)$

$$\varphi(t) = [y(t-1) \dots y(t-n_a)x(t-1) \dots x(t-n_b)]^T$$

where $y(k)$ are past outputs and $x(k)$ are past inputs. The dimension of the problem is $d = n_a + n_b$. The problem is to understand the relationship between the next output $y(t)$ and $u(t)$:

$$?\, y(t) \leftrightarrow x(t)?$$

This system will be modelled by a general model structure f, that implements an $f: R^d \rightarrow R$ mapping:

$$\hat{y_M} = f\left(x_i, \hat{\theta}\right)$$

where $\hat{y_M}$ is the output of the model, or simply stated, a guess of the output, given the parameter value $\hat{\theta}$.

Data Preparation

Normalization

Data normalization is a process of final data preparation for their direct use for network training. It includes the normalization of preprocessed data from their natural range to the network's operating range, so that the normalized data are strictly shaped to meet the requirements of the network input layer and are adapted to the nonlinearities of the neurons, so that their outputs should not cross the saturation limits.

Moreover, instead of linear normalization, nonlinear scaling or logarithmic scaling of input signals is used to moderate the possible nonlinearity problems during the network training. For instance, logarithmic transformation can squeeze the scale in the

region of large data values, and exponential scaling can expand the scale in the region of small data values, etc. But by far the most critical data preparation issue here is the risk of possible loss of critical information present within the acquired data. (33)

Upscaling

Upscaling is a process that is often used in geophysical data processing, aiming at averaging static and dynamic characteristics of a model (34). There are many upscaling methods, and each method is convenient for different parameters, so before upscaling begins, the most suitable upscaling method for the parameter must be decided.

Gaussian filtering is a usual method of noise removal. Along with upscaling methods, it can be used as a filtering technique, providing gentler smoothing and preserving edges better than a similarly sized mean filter (35).

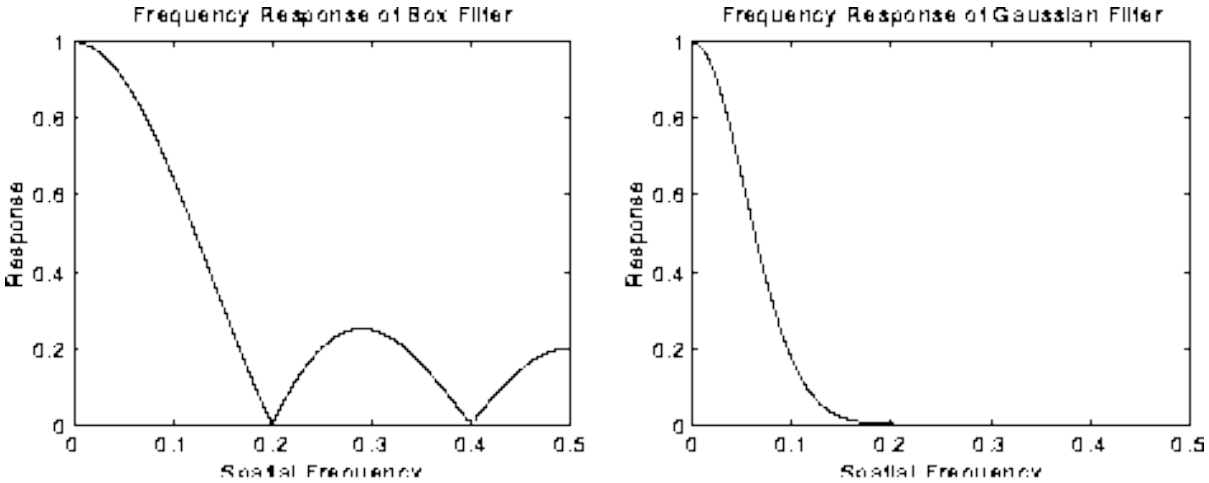


Figure 6 Frequency responses of 1-D mean filter with width 5 and Gaussian filter with $\sigma = 3$. A mean filter will exhibit oscillations in its frequency response (34).

As proposed by (1), we upscale the well data using a Gaussian filter. This way, the spatial frequencies are still present in the data after filtering. The size of the Gaussian filter is adjusted to the size of the depth interval step of the seismic-derived elastic data. The Gaussian upscaling filter equation we are using is:

$$G(x,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\sigma^2}}$$

where the parameter σ is the standard deviation of the distribution.

An example of upscaling can be seen in Figure 11, where noisy data (green) has been upscaled to a smoother dataline (blue).

Model Selection Strategies

The generalization performance of a learning method relates to its prediction capability on independent test data. Assessment of this performance is extremely important in practice, since it guides the choice of learning method or model, and gives us a measure of the quality of the ultimately chosen model. In this section we describe and illustrate the key methods for performance assessment, and show how they are used to select models.

Model selection is the task of selecting a statistical model from a set of candidate models, given data. The decision of the structure of the network is one of the most difficult questions in the literature on neural networks. Architecture selection involves choosing an appropriate number of hidden units and their connections. It is desirable to have an architecture with a few hidden units and connections.

Statistical procedures, such as hypothesis testing, information criteria and cross validation methods, can be used to guide model selection in neural networks. The most commonly used criteria are the Akaike (AIC) and Bayesian (BIC) information criteria. (15)

Hypothesis Testing

Since multilayer perceptron neural networks are nonlinear regression models, the standard procedures for testing parameter significance, like Wald-tests or LM-tests, apply, in principle. However, to perform these tests the (asymptotic) distribution of the network parameters is needed.

Information Criteria

The training set MSE is generally an underestimate of the test MSE. (MSE =RSS/n.) This is because when we fit a model to the training data using least squares, we specifically estimate the regression coefficients such that the training RSS (but not the test RSS) is as small as possible. In particular, the training error will decrease as more variables are included in the model, but the test error may not. Therefore, training set RSS and training set R2 cannot be used to select from among a set of models with different numbers of variables.

However, a number of techniques for adjusting the training error for the model size are available. These approaches can be used to select among a set of models with

different numbers of variables. We now consider four such approaches: Cp, Akaike information criterion (AIC), Bayesian information criterion (BIC), and adjusted R2.

Validation and Cross-Validation

Probably the simplest and most widely used method for estimating prediction error is cross-validation. As an alternative to the approaches just discussed, we can directly estimate the test error using the validation set and cross-validation methods. We can compute the validation set error or the cross-validation error for each model under consideration, and then select the model for which the resulting estimated test error is smallest. This procedure has an advantage relative to AIC, BIC, Cp, and adjusted R2, in that it provides a direct estimate of the test error, and makes fewer assumptions about the true underlying model. It can also be used in a wider range of model selection tasks, even in cases where it is hard to pinpoint the model degrees of freedom (e.g. the number of predictors in the model) or hard to estimate the error variance σ^2.

In the past, performing cross-validation was computationally prohibitive for many problems with large p and/or large n, and so AIC, BIC, Cp, and adjusted R2 were more attractive approaches for choosing among a set of models. However, nowadays with fast computers, the computations required to perform cross-validation are hardly ever an issue. Thus, cross-validation is a very attractive approach for selecting from among a number of models under consideration.

In cross-validation schemes, we use the existing data to identify settings for the model's parameters that yield the best and most realistic predictive performance (known as model tuning). Traditionally, this has been achieved by splitting the existing data into training and test sets. The training set is used to build and tune the model and the test set is used to estimate the model's predictive performance. Modern approaches to model building split the data into multiple training and testing sets, which have been shown to often find more optimal tuning parameters and give a more accurate representation of the model's predictive performance.

If a set of models appear to be more or less equally good, then we might as well choose the simplest model—that is, the model with the smallest number of predictors.

The Problem of Over-Fitting

There now exist many techniques that can learn the structure of a set of data so well that when the model is applied to the data on which the model was built, it correctly predicts every sample. In addition to learning the general patterns in the data, the model has also learned the characteristics of each sample's unique noise. This type of model is said to be over-fit and will usually have poor accuracy when predicting a new sample.

Model Tuning

Many models have important parameters which cannot be directly estimated from the data. This type of model parameter is referred to as a tuning parameter because there is no analytical formula available to calculate an appropriate value. Since many of these parameters control the complexity of the model, poor choices for the values can result in over-fitting. There are different approaches to searching for the best parameters. A general approach that can be applied to almost any model is to define a set of candidate values, generate reliable estimates of model utility across the candidate values, then choose the optimal settings (Figure 7). This way of performing hyperparameter optimization is called grid search.

Once a candidate set of parameter values has been selected, then we must obtain trustworthy estimates of model performance. The performance on the hold-out samples is then aggregated into a performance profile which is then used to determine the final tuning parameters. We then build a final model with all of the training data using the selected tuning parameters.

Other approaches such as genetic algorithms (36) or simplex search methods (37) can also find optimal tuning parameters.

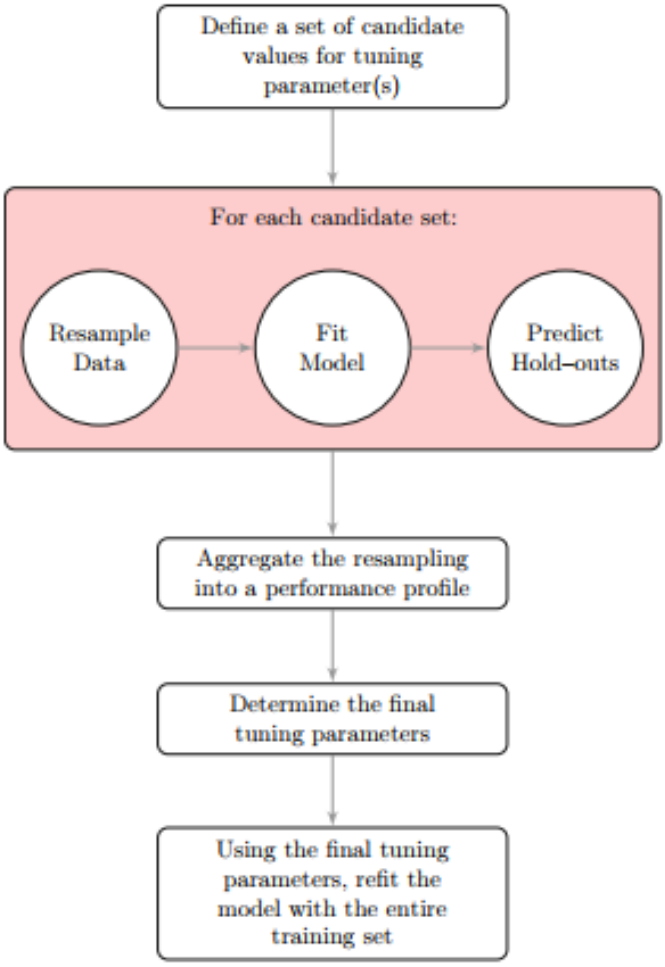


Figure 7 A schematic of the parameter tuning process (38).

Choosing Final Tuning Parameters

Once model performance has been quantified across sets of tuning parameters, there are several philosophies on how to choose the final settings. The simplest approach is to pick the settings associated with the numerically best performance estimates.

In general, it may be a good idea to favor simpler models over more complex ones and choosing the tuning parameters based on the numerically optimal value may lead to models that are overly complicated. Other schemes for choosing less complex models should be investigated as they might lead to simpler models that provide acceptable performance (relative to the numerically optimal settings).

Software Used

The software that is used in this research is MATLAB and seislib, for importing correctly the las2 files associated with well logs, and RStudio, which includes plenty of libraries to train neural networks.

CHAPTER 3:
The Proposed Approach: Foundation, Design, Development

Parameter Estimation Framework

The first step in any model building framework is to understand the data, which can most easily be done through a graph. For this purpose, we use scatter plots. These figures show the relationships between pairs of variables. These relationships can be linear or may exhibit curvature towards the extremes of the axis. We need to understand the characteristics of the predictors, as these characteristics may suggest important and necessary pre-processing.

After understanding the data, the next step is to build and evaluate a model on the data. A standard approach is to take a random sample of the data for model building and use the rest to understand model performance. The common terminology would be that some data are used as the model "training set" and some other are the "test" or "validation" set.

Now that we have defined the data used for model building and evaluation, we should decide how to measure performance of the model. For regression problems where we try to predict a numeric value, the residuals are important sources of information. Residuals are computed as the observed value minus the predicted value (i.e., $y_i - \hat{y}_i$). When predicting numeric values, the root mean squared error (RMSE) is commonly used to evaluate models. This metric is interpreted as how far, on average, the residuals are from zero.

At this point, we will try to mathematically define the relationship between the predictors and outcome, using neural networks. To do this, the training set is used to estimate the various values needed by the model equations. The test set will be used only when a few strong candidate models have been finalized (repeatedly using the test set in the model build process negates its utility as a final arbitrator of the models).

When working with the training set, one must be careful not to simply evaluate model performance using the same data used to build the model. If we simply re-predict the training set data, there is the potential to produce overly optimistic estimates of how well the model works, especially if the model is highly adaptable. An alternative approach for quantifying how well the model operates is to use resampling, where different subversions of the training data set are used to fit the model. Common resampling techniques are 5-fold or 10-fold cross-validation.

Neural networks is the technique that will be applied in this thesis, for the purpose of creating a relationship between the predictors and outcome. These networks

have tuning parameters, which are parameters that cannot be determined by the use of analytical equations. We have to try different values and use resampling to determine the appropriate values. Once the final values are found, a final neural network model will be fit using all the training set data and used for prediction.

Using cross-validation, we evaluate candidate values for the tuning parameters to create resampling profiles.

Tools

The main tool used for this research is RStudio. This program provides us with R, a strong open-source programming language. We use caret package (15) along with mxnet (38) and RSNNS (39) libraries, for neural network tuning and training.

Location of the Study Area

Our data come from the deep waters of the Norwegian Sea. The wells used in this study are located in Nyk High.

Three wells are used. These are: Vema well (6706/11-1, Dry well), Luva well (6707/10-1, gas discovery) and Haklang well (6707/10-2S, gas discovery). The characteristics of each well are given:

Vema well (40)

"Well 6706/11-1 was drilled on the Ægir prospect on the Vema Dome in the north of the Vøring basin. The main objectives for the well were to test the hydrocarbon potential of the Ægir prospect and to obtain data to allow for efficient exploration of the remaining licence area.

The dominant part of the sediments proved to be of Cretaceous age. Both the Nise Sandstone 1 and the Nise Sandstone 2 formations were developed. No significant amounts of hydrocarbons were encountered only small amounts of very dry gas in an MDT sample from 3740 m. Five cores were cut in the NISE Formation. The well was permanently plugged and abandoned as a dry hole."

Luva well (40)

"Well 6707/10-1 was the first well drilled in PL (Production License) 218 and the first in the Vøring Basin deepwater area. The well targeted a tilted fault block on the Nyk High

comprising Late Cretaceous sandstones with a pronounced flat spot. The main objectives of the well were to establish the presence, quality, and fluid content of the Campanian age Nise Formation. The underlying Santonian age Lysing Formation provided a secondary target.

Wire line logging and sampling confirmed a gas column from 2957.5 to 3113.5 m. A total of nine cores were cut over the interval 2967 - 3145 m with a total recovery of 171 m (96 %). Poor shows were recorded in cores throughout this reservoir section, down to the base of core 9, where moderate shows were seen in the interval 3140 - 3145 mBRT. One 27 m core (core 10) was cut in the secondary target (4118 - 4145 mBRT). This core had fair shows at the top. No shows were seen in the drilled cuttings. Altogether 10 MDT fluid samples were taken in the Nise Formation. Seven of these were taken from 2964 to 3086 m RKB in the reservoir zone, two samples were from 3114.5 m just below the gas-water contact, and one from further down at 3195 m. The MDT samples were contaminated with mud, but PVT flash to stock tank conditions gave a consistent gas gravity of 0.59 relative to air and a GOR in excess of 117000 m3/m3. The well was plugged and abandoned as a gas discovery with oil shows."

Haklang well (40)

"The 6707/10-2 S Haklang well was drilled south-east of the 6707/10-1 Luva Discovery on the Nyk High in the Northern Norwegian Sea. The objective of the Haklang well was to prove hydrocarbons in the Nise 1 sandstone. After completion of this well bore, a sidetrack well, 6707/10-2 A Haklang West flank, was planned to prove hydrocarbons in the Nise 2 sandstone.

The top of the reservoir was picked at 3150.5 m (3146.5 m TVD RKB). The reservoir showed good properties (Net/Gross: 0.73, porosity 0.25) and contained dry gas. The gas-water contact was defined at 3281.1 m (3274.6 m TVD RKB), giving a gas column of 128 m. Variable oil shows, as proven by cut and weakly colored fluorescence (light hydrocarbons), were observed on the cores from the gas-bearing reservoir section, otherwise no shows were recorded in the well.

A total of 142.6 m core was recovered in 8 cores from the interval 3158 to 3308 m in the Nise Formation reservoir. One MDT run for pressure points and fluid sampling was performed in the well. Fluid samples were obtained at 3165.6 m MD/3161.3 m TVD and 3282.9 m MD/3276.3 m TVD. Gas was the movable fluid on both sampling depths.

The well was plugged back and prepared for sidetracking on 13 October 2010. It is a gas discovery."

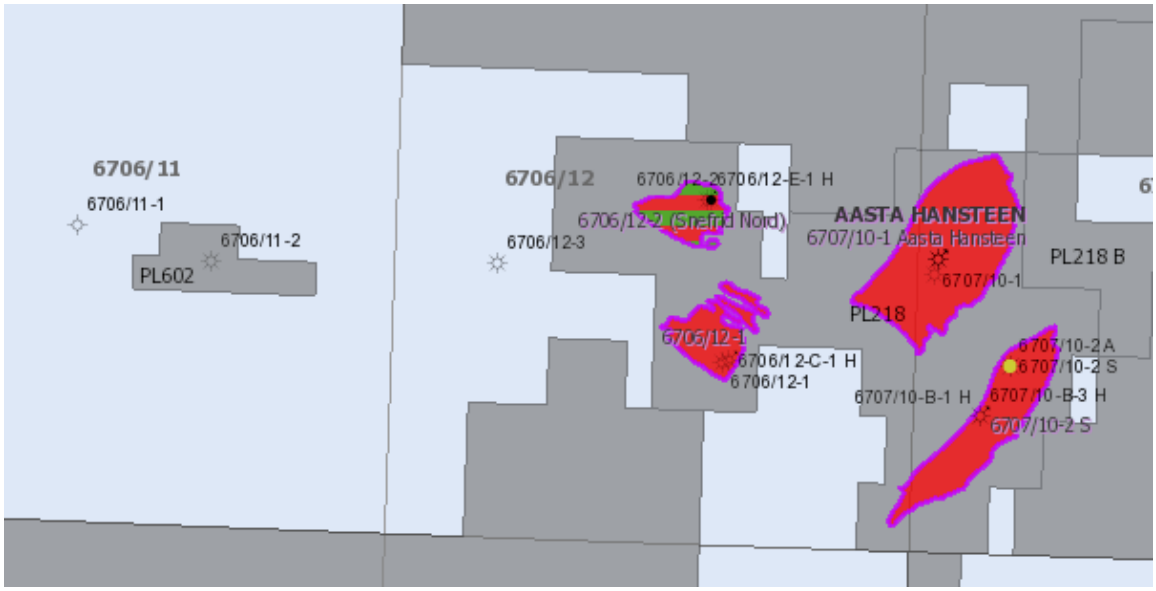


Figure 8 Aasta Hansteen field location. Wells 6706/11-1 (Vema), 6707/10-1 (Luva) and 6707/10-2S (Haklang) can be seen.

Well-log Data

For the three different wells available, we have logs for acoustic travel time (sonic + shear) and deep resistivity versus depth, which can be seen in Figure 9.

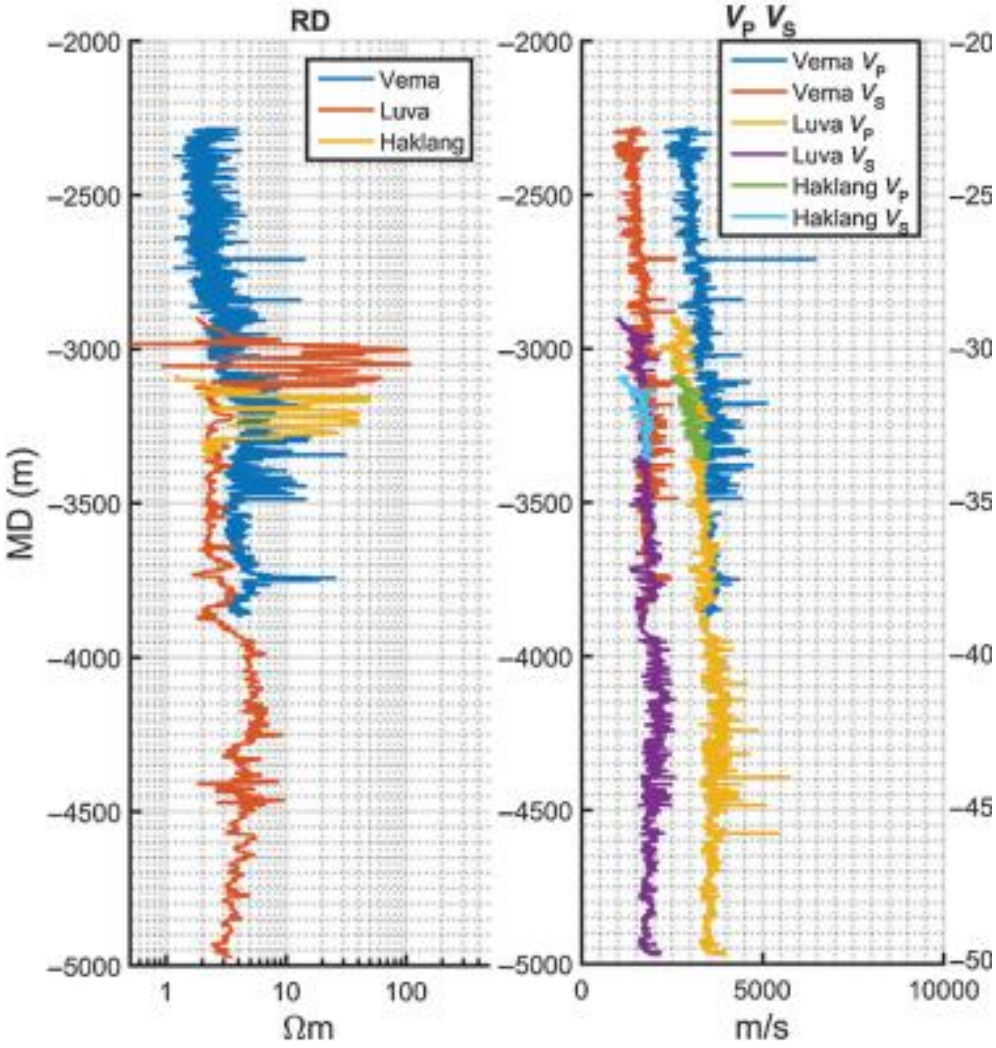


Figure 9 The figures indicate well log data in different wells. From the left figure, deep resistivity (RD) compressional (Vp) and shear velocity (Vs). MD is the studied depth.

We derive seismic velocities V_p and V_s, from sonic and shear transition times. A scatter plot matrix between data labels is shown in Figure 10. It can be seen that sonic velocity (x2_VP) and shear velocity (x3_VS) have the biggest linear relationship among the variables. This is due to the fact that they share the same Physics' laws.

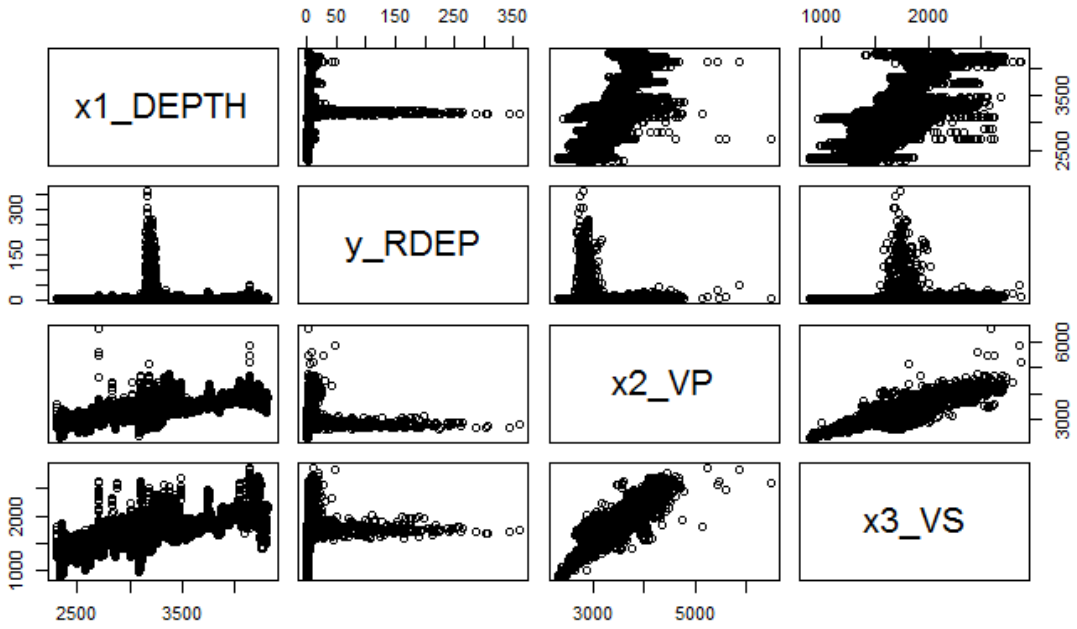


Figure 10 Scatterplot between data pairs.

We upscale the well data using a Gaussian filter, as suggested by (1). The size of the Gaussian filter is adjusted to filter a 5m interval (which is close to seismic resolution[1]). In Figure 11 upscaled data are plotted together with original data for comparison.

1

Well-log data are provided with a 10–30 cm resolution, whereas for seismic and CSEM data, 5–50m depth resolution is usually adequate.

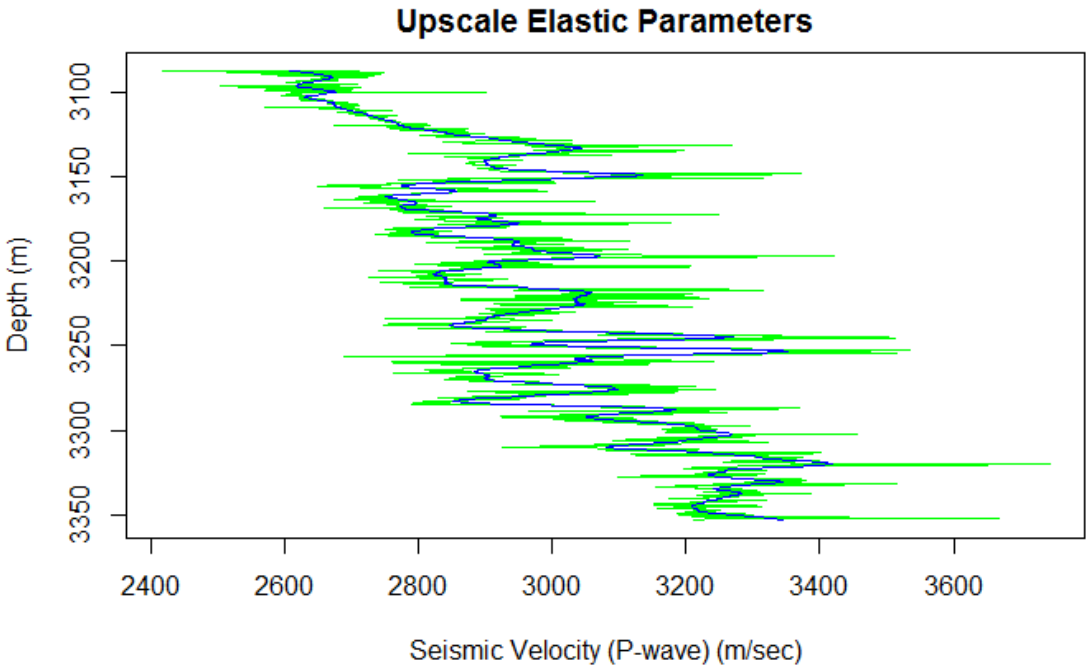


Figure 11 Sonic velocity vs Depth log. The blue line indicates data that have been upscaled.

We use the universal approximation theorem with sigmoid activation function $\sigma(.)$. A sigmoid function can be any continuous function with the property:

$$\sigma(\tau) \rightarrow \begin{cases} 1 ast \rightarrow +\infty \\ 0 ast \rightarrow -\infty \end{cases}$$

Cybenko proves that given a continuous function f, there exists an integer n and real constants w_i;a_{ij}; b_i, for which the sum

$$f_n(x_1, \ldots, x_m) = \sum_{i=1}^{n} w_i \sigma \left(\sum_{j=1}^{m} a_{ij} x_j + b_i \right)$$

can approximate the continuous function f to a given accuracy ε, so that

$$\|f - f_n\| < \varepsilon$$

Equation 4 can be approximated with a feed-forward neural network with a single hidden layer that contains a finite number of hidden neurons. Figure 12 shows the fully

connected multilayer perceptron (MLP), which is used to derive the elastic to electric parameters transform.

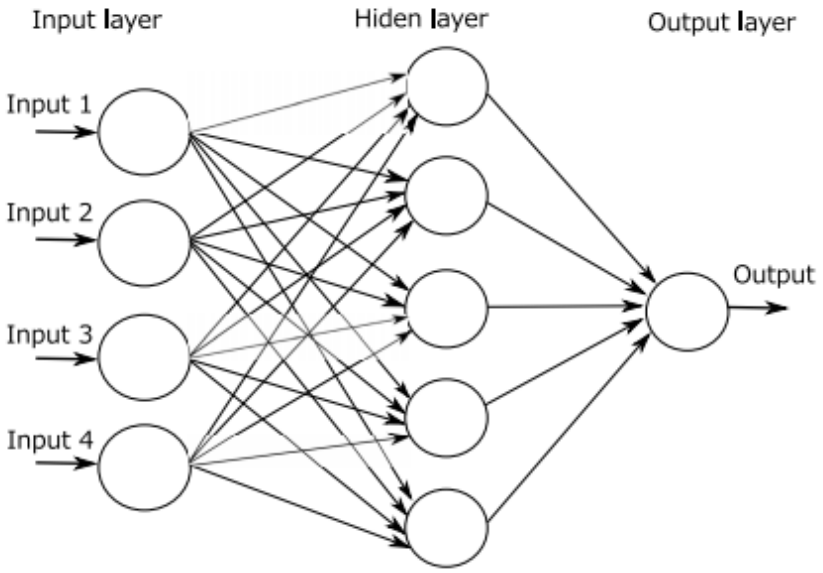


Figure 12 Diagram of the MLP used for deriving the elastic parameters to resistivity transform.

The hidden layer is using symmetric hyperbolic tangent activation function, whereas the output layer uses symmetric linear activation function. The number of perceptron in the hidden layer is calculated as the sum of input elastic parameters plus one.

Two additional layers, one before the MLP input and one after the MLP output, respectively, are connecting MLP input parameters to the actual well-log input parameters and the MLP output result to the output resistivity data. These layers are used to scale the well-log data to the [–1, 1] interval. The MLP models the log10 of the resistivity log, to achieve a less sparse well-log resistivity distribution in the [–1, 1] output interval after the normalization of the resistivity log values.

Learning Functions

The training of an MLP is accomplished by using the back-propagation algorithm. Training is the process of determining the optimal weights of a NN. This is accomplished by defining a performance function. The RMSE between the network's output and the desired target is usually chosen. The performance function is then minimized with respect to weights. The minimization is performed by calculating the gradient using

backpropagation technique, which can be done in batch or incremental styles. In this paper, we evaluate both training styles. The NNs can be trained using different training algorithms belonging to three backpropagation classes:

A. Steepest Descent (SD)
B. Quasi-Newton (QN)
C. Conjugate Gradient (CG)

Conjugate Gradient Methods (CGMs) are general purpose second order techniques that help minimize goal functions of several variables. A second order technique generally finds a better way to a (local) minimum than a first order technique, but at a higher computational cost. SCG has been shown to be considerably faster than standard backpropagation and other CGMs.

CHAPTER 4:
Implementation and Results

Paper Reproduction

In this part we will introduce a neural network to be used for resistivity prediction. We are going to reproduce the network from (1). To do that, we will first prepare our data, then we are going to construct a 1-hidden layer ANN with four neurons. We will then train the model using the back-propagation algorithm, as suggested by the paper. Finally, we will make some comments regarding the results.

Scaled Conjugate Gradient Algorithm (SCG)

In this first example, we used the SCG algorithm for the training of the MLP. SCG is a supervised, second order, learning algorithm for feedforward neural networks, and is a member of the class of conjugate gradient methods (CGM). This is also the proposed algorithm by (1). All of the algorithm-specific parameters are non-critical, i.e. they influence only the speed of convergence, not whether there will be success or not. (41)

During paper replication, we fit a neural network, with cross-validation and four hidden units, and recorded the average validation error for each turn. A 10-fold cross-validation process was run and repeated 10 times.

A plot between measured and predicted well log data for Luva well is show in Figure 13. The neural network succeeds in making a good estimation of the resistivity and the four-unit model does perform well, achieving an error of RMSE 9.04. R^2 is moderate, around 61%. It appears that this is not a reliable metric as it does not depict the true situation, regarding the performance of the algorithm.

Next, we fit more models, using cross-validation. The results of applying parameter tuning during the training process will be discussed in the next chapter.

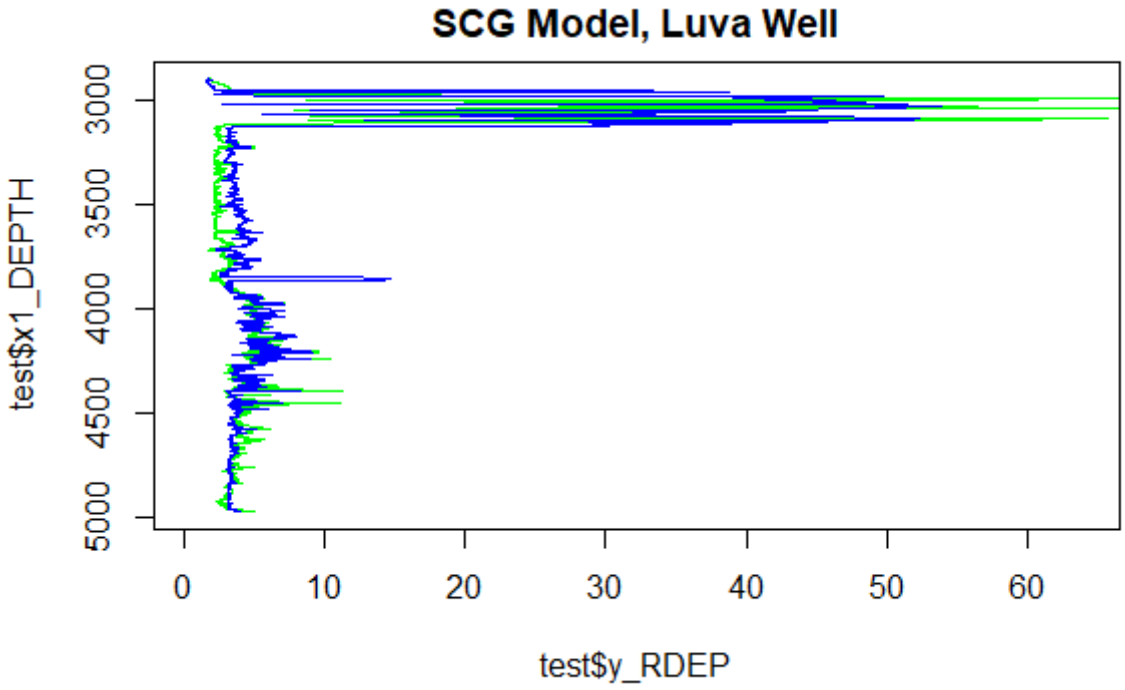


Figure 13 SCG: Well-log resistivity measured data (in green) and the predicted (in blue) vs Depth.

Implementation of More Algorithms

Stochastic Gradient Descent Algorithm (SGD)

In Figure 14, we repeated the experiment selecting SGD as the algorithm used in training of the ANN. GD method is also known as Steepest Descent. Gradient descent is a kind of iterative method that is provided with an initial point and it follows the negative of the gradient in order to move the point toward a critical point. This critical point is the required local minimum value.

We are concerned only with local optimization of system because global optimization is challenging to compute. It is an optimization algorithm, that approaches a local minima of a function by taking the steps iteratively and these steps are directly proportional to the negative of the gradient of a function at the current point. Gradient descent is popular for very large-scale optimization problems because it is easy to implement and it can handle "black box" functions and each iteration is cheaper than the other algorithms used for this purpose. Its major disadvantage is that it can take a long time to converge.

A learning rate of 0.01 was chosen initially. No momentum was used.

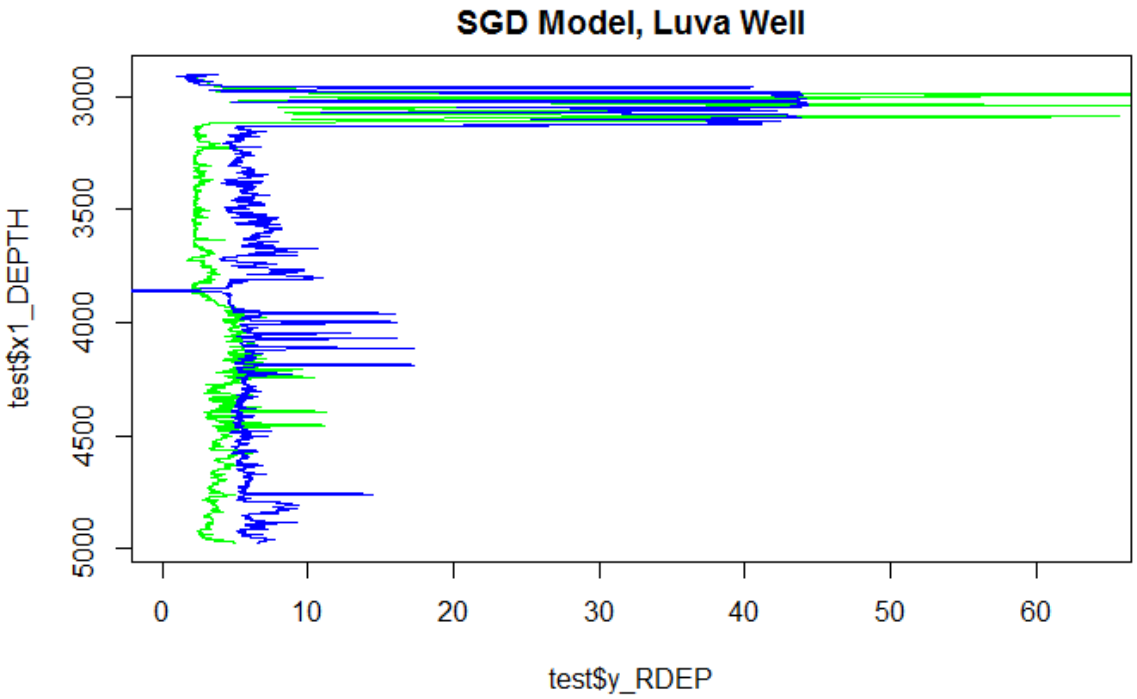


Figure 14 SGD: Well-log resistivity measured data (in green) and the predicted (in blue) vs Depth.

Backpropagation Algorithm with batch training mode (BPROP)

Training was performed using BPROP learning algorithm with batch update. The algorithm managed to converge faster than SGD. However, the results were not good. Luva well results are shown in Figure 15.

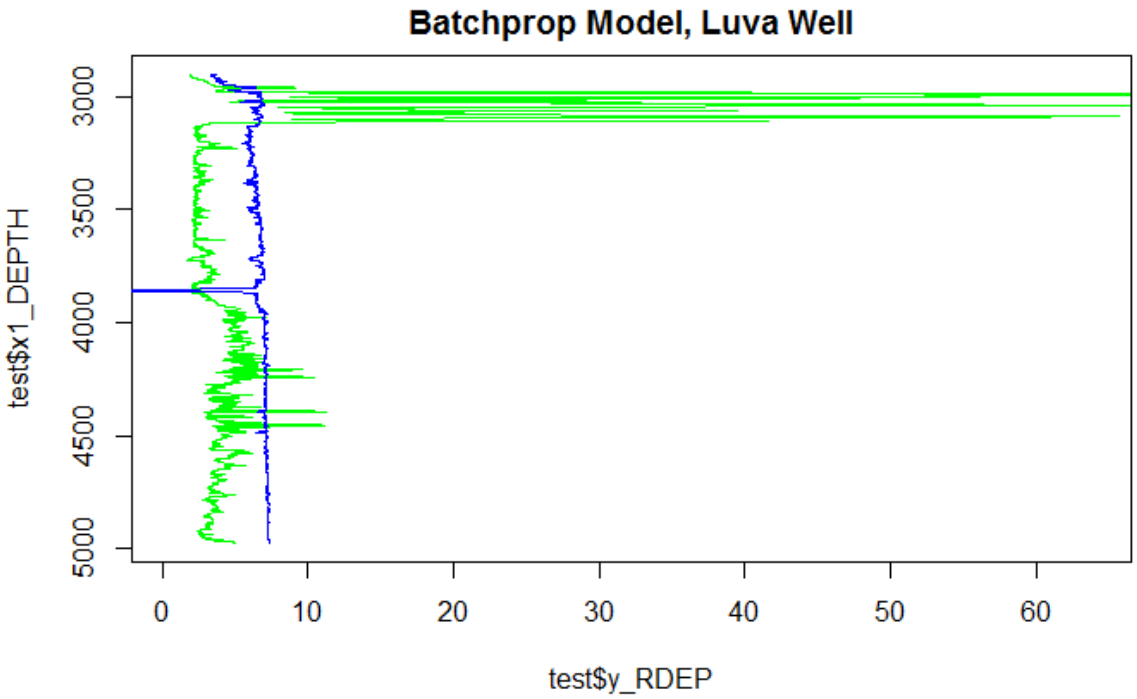


Figure 15 BPROP: Well-log resistivity measured data (in green) and the predicted (in blue) vs Depth.

Radial Basis Functions (RBF)

Training was performed using Radial Basis Functions learning. The results for Luva well are shown in Figure 16.

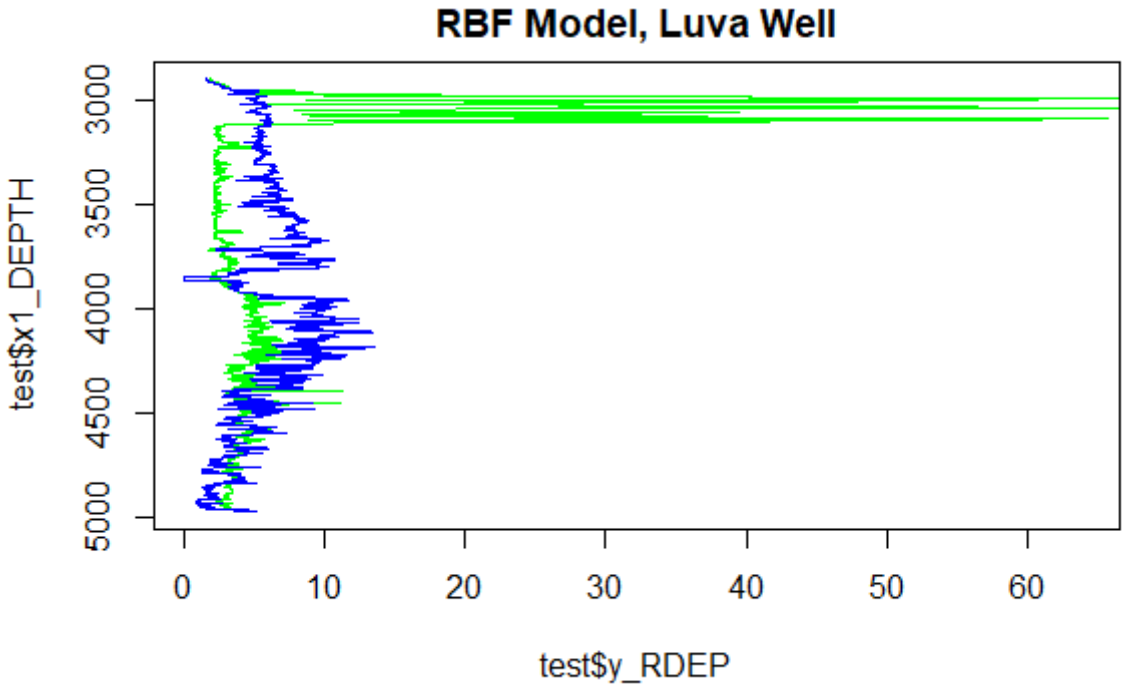


Figure 16 RBF: Well-log resistivity measured data (in green) and the predicted (in blue) vs Depth.

Resilient Propagation Algorithm (RPROP)

Rprop, short for resilient backpropagation, is a learning heuristic for supervised learning in feedforward artificial neural networks. This is a first-order optimization algorithm. This algorithm was created by Martin Riedmiller and Heinrich Braun in 1992.

Rprop is one of the fastest weight update mechanisms. RPROP is a batch update algorithm.

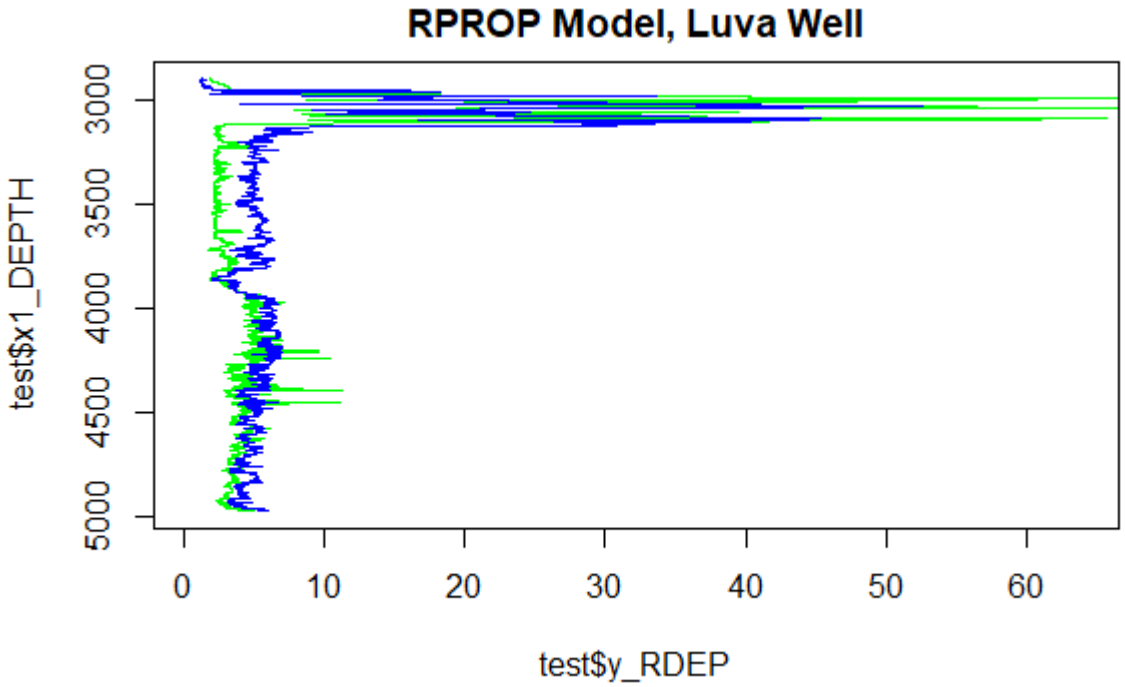


Figure 17 RPROP: Well-log resistivity measured data (in green) and the predicted (in blue) vs Depth.

Training with Multiple Network Topologies & Hyperparameter Tuning

Stochastic Gradient Descent Algorithm (SGD)

In Figure 18 we repeated the experiment selecting SGD as the algorithm used in training of the ANN. First, with no momentum, we applied a grid search procedure to find the optimal learning rate value for the algorithm. The learning rate defines how quickly a network updates its parameters. Low learning rate slows down the learning process but converges smoothly. Larger learning rate speeds up the learning but may not converge. Usually a decaying Learning rate is preferred. It appears that training is improved when increasing the learning rate. Therefore, we picked the highest value of 0.1.

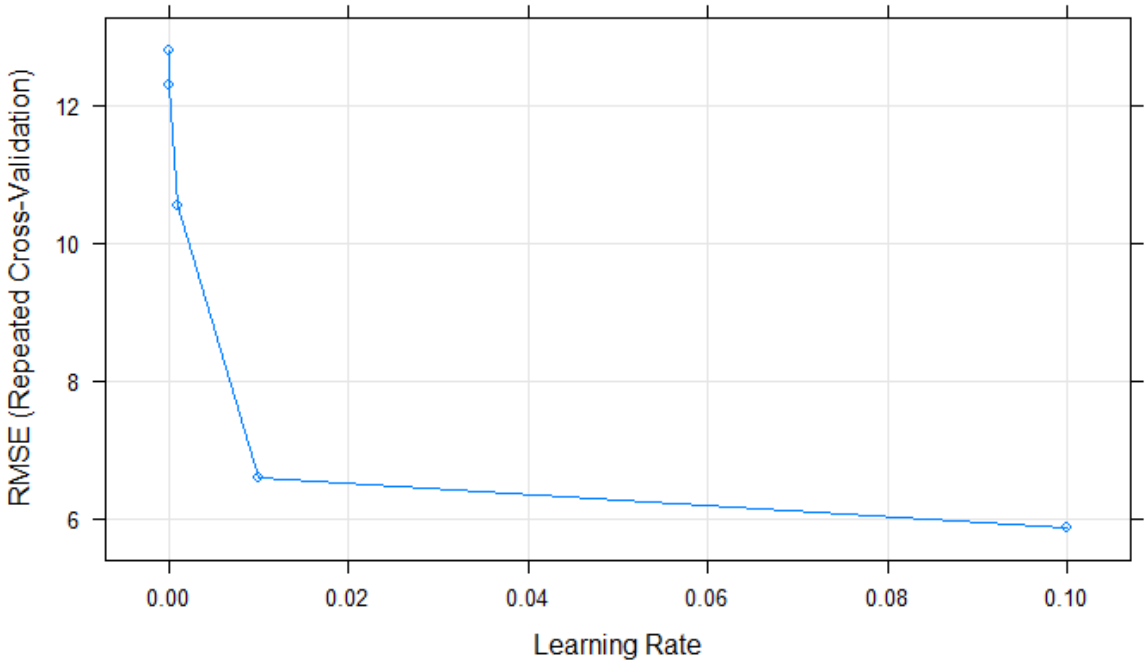
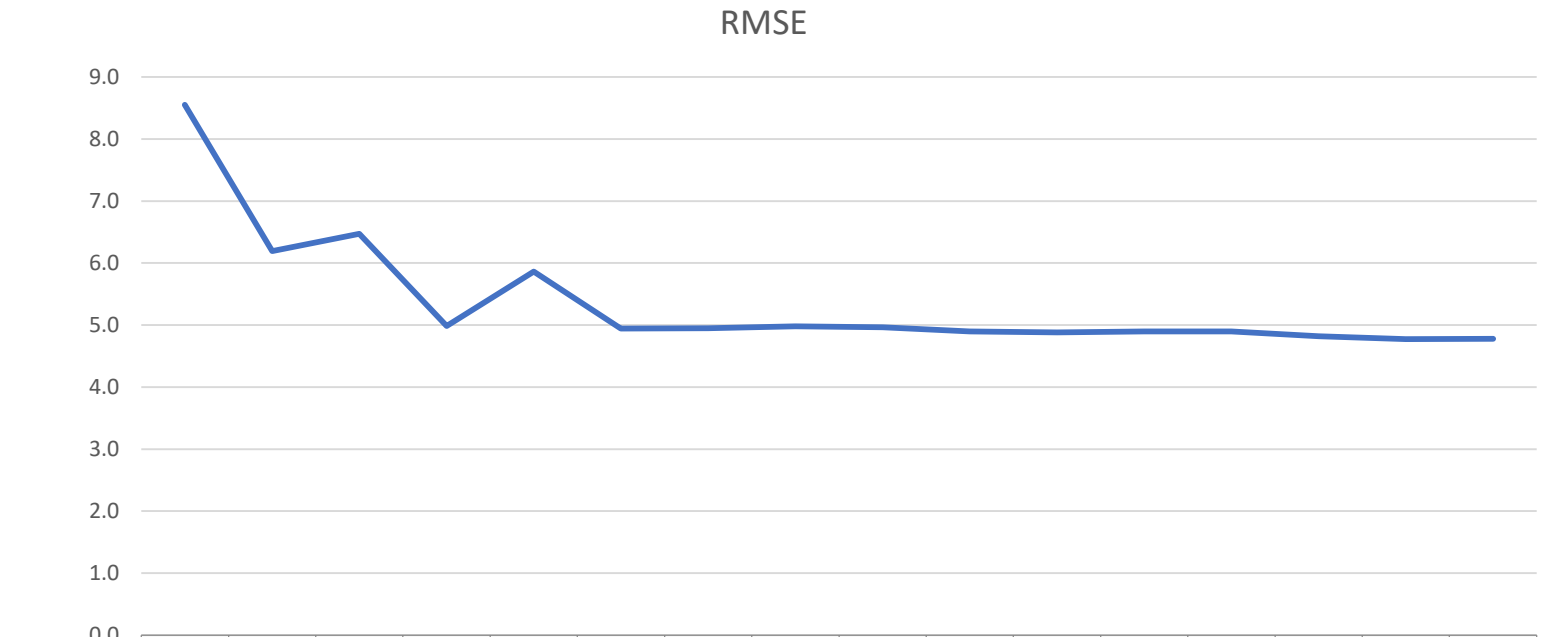


Figure 18 SGD: Plot of RMSE of resistivity for different learning rates. The performance of the algorithm is relatively stable for higher values, therefore a learning rate of 0.1 was picked.

Figure 19 shows the mean training RMSE cross-validated error for 0.1 learning rate, varying the number of hidden neurons parameter over a wide range. We start from 0 and reach an architecture of 10 hidden units, with a step of 1. The line search is continued

from 10 to 20 hidden units, incremented by 2. The value 0 stands for no hidden layer, and therefore is nearly equivalent to a linear model.

A number of five neurons is approximately optimal. AIC(2), AIC(4) and BIC criteria were also evaluated while varying the number of hidden units. The resulting charts are shown next.

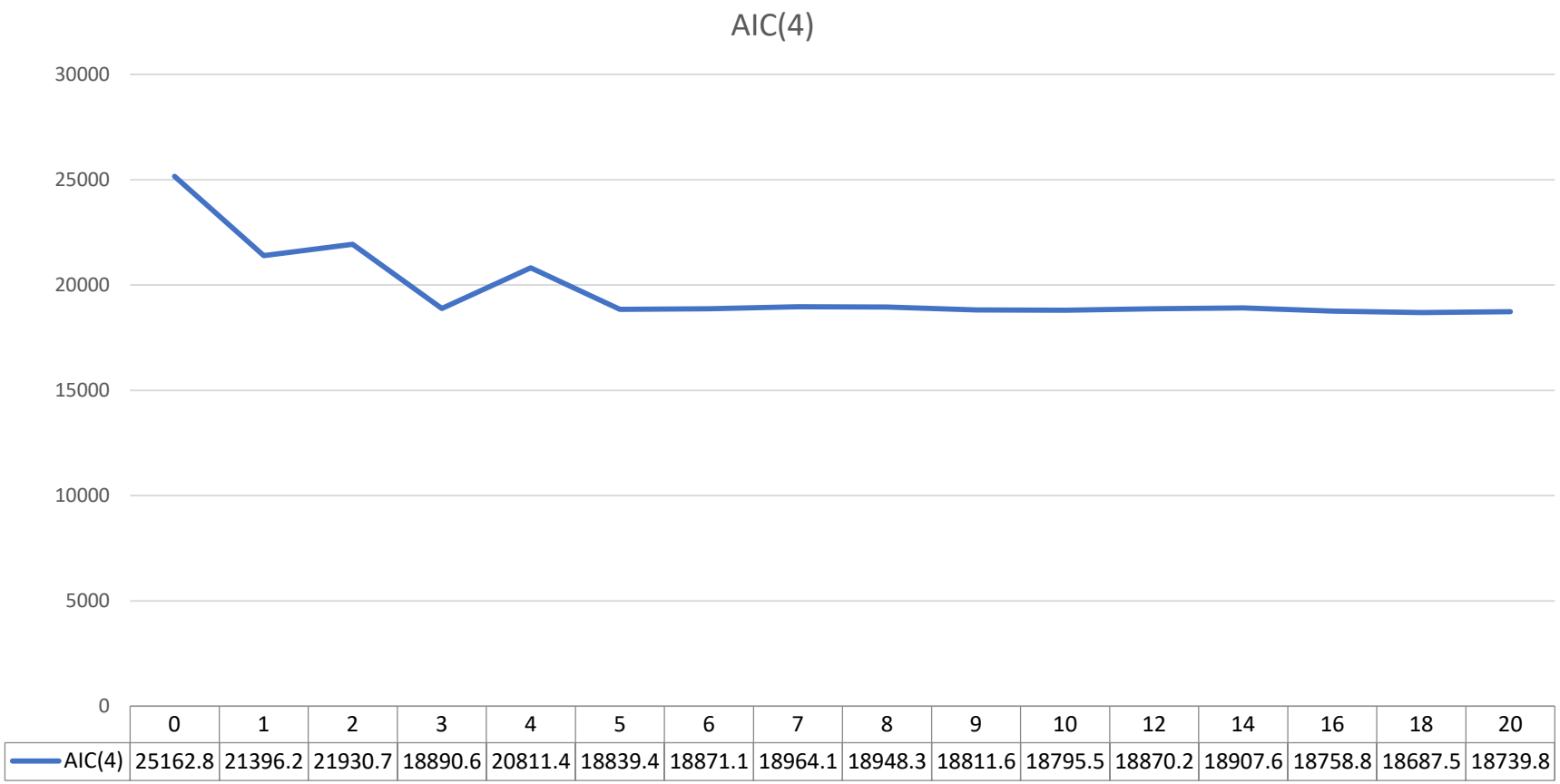


	0	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20
RMSE	8.5537	6.1924	6.4702	4.9836	5.8609	4.9449	4.9499	4.9808	4.9656	4.8997	4.8846	4.8991	4.898	4.8197	4.7742	4.7792

Figure 19 Tuning of Hidden Units parameter using cross-validation for SGD algorithm.

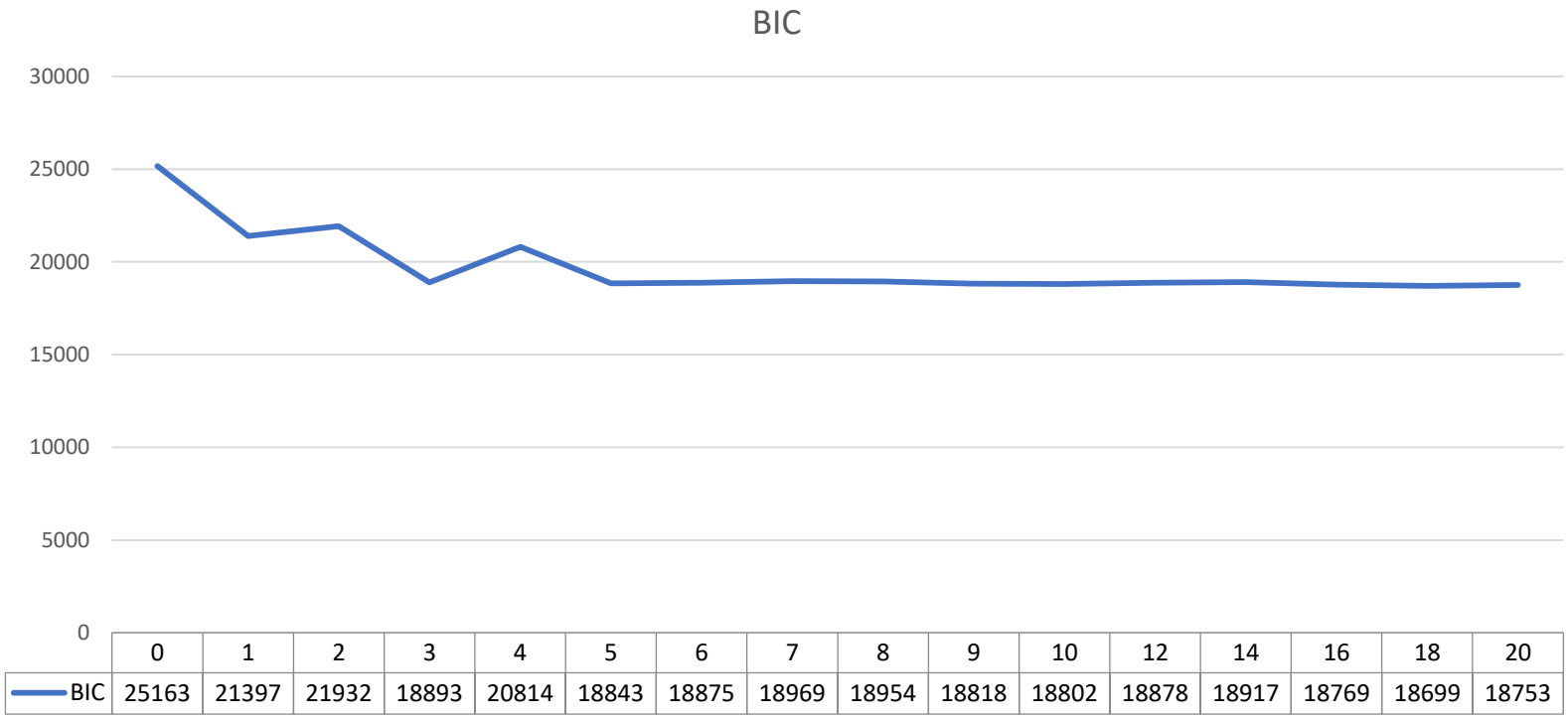
	0	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20
AIC(2)	25161	21384	21909	18859	20769	18787	18809	18892	18866	18720	18693	18748	18766	18597	18506	18538

Figure 20 Performance of SGD algorithm, according to AIC(2) information criterion.



	0	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20
AIC(4)	25162.8	21396.2	21930.7	18890.6	20811.4	18839.4	18871.1	18964.1	18948.3	18811.6	18795.5	18870.2	18907.6	18758.8	18687.5	18739.8

Figure 21 Performance of SGD algorithm, according to AIC(4) information criterion.



	0	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20
BIC	25163	21397	21932	18893	20814	18843	18875	18969	18954	18818	18802	18878	18917	18769	18699	18753

Figure 22 Performance of SGD algorithm, according to BIC information criterion.

Backpropagation Algorithm with batch training mode (BPROP)

Next, we repeated the experiment selecting BPROP as the method used for neural network training. According to (41), two learning parameters are used when choosing backpropagation with batch training method:

- η, which specifies the step width of the gradient descent and has typical values of 0.1 to 1.
- d_{max} which is the maximum difference $d_j = t_j - o_j$ between a teaching value t_j and an output o_j of an output unit which is tolerated, i.e. which is propagated back as $d_j = 0$. If values above 0.9 should be regarded as 1 and values below 0.1 as 0, then d_{max} should be set to 0.1. This prevents overtraining of the network.

In Table 5, we vary η parameter from 0.1 to 1, with a step of 0.1 and compare the training RMSEs. We pick the best value and see how well the network succeeds in estimating the resistivity for Luva Well. These results, along with results from other algorithms, will be compared and discussed furtherly in the next chapter.

Table 5 Training RMSE for varying learning parameter η (step width of gradient descent)

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
7.638	7.059	6.965	7.244	7.257	8.050	8.765	3.102e+17	5.257e+17	4.386e+17

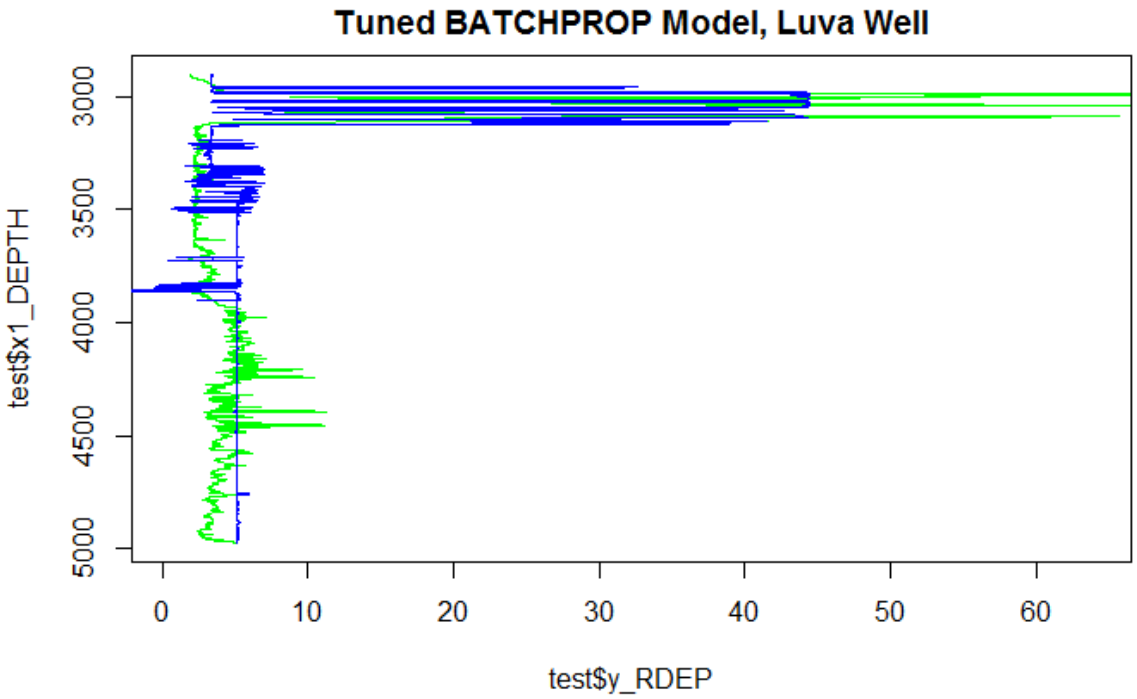


Figure 23 Plot of measured (green) vs predicted (blue) resistivity. The model used for the predictions is using BPROP algorithm with learning rate 0.3.

Scaled-Conjugate Gradient Algorithm (SCG)

We trained again using the SCG algorithm, varying the number of hidden units. The number of neurons was incremented sequentially, starting from three neurons, until it reached twenty neurons in the hidden layer. We noted the difference in execution time between the SGD and SCG algorithms. For the SCG algorithm, convergence happens in 91.5% of the time, compared to SGD. Resampling results between the two algorithms are similar. It should be noted that SCG is a batch learning method.

The results of tuning the number of hidden units for the SCG algorithm will be discussed in the next chapter.

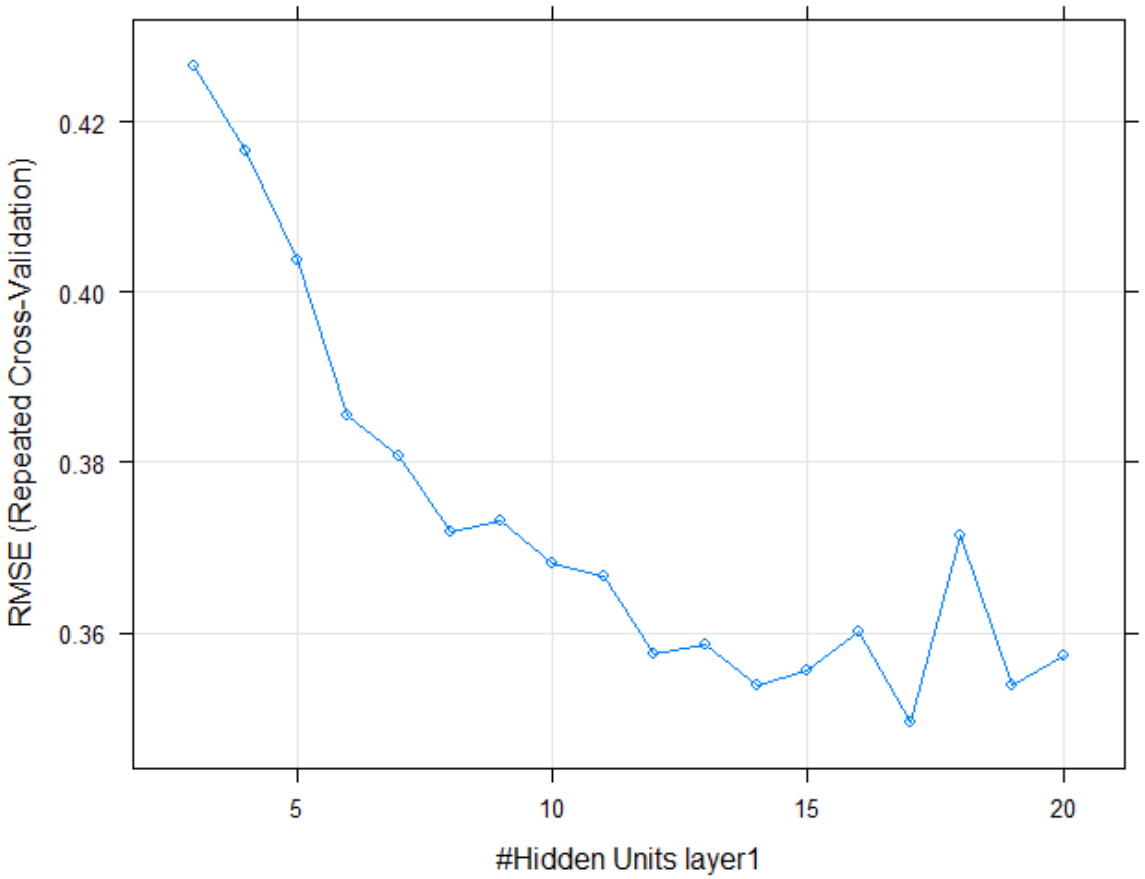


Figure 24 SCG method, tuning the size of the hidden layer. The best model was a network with seventeen (17) neurons.

Radial Basis Functions (RBF)

No significant improvement was noticed (Figure 25), repeating the procedure discussed above, for the RBF method. RMSE was used to select the optimal model using the smallest value. Some topologies had negative effects (high RMSE), whereas most of them remained at low-RMSE levels.

The final value used for the model was size of four (4) hidden units. Taking into account AIC criterion, which penalizes model complexity, leads us to using four neurons in the hidden layer. The results will be discussed in the next chapter extensively.

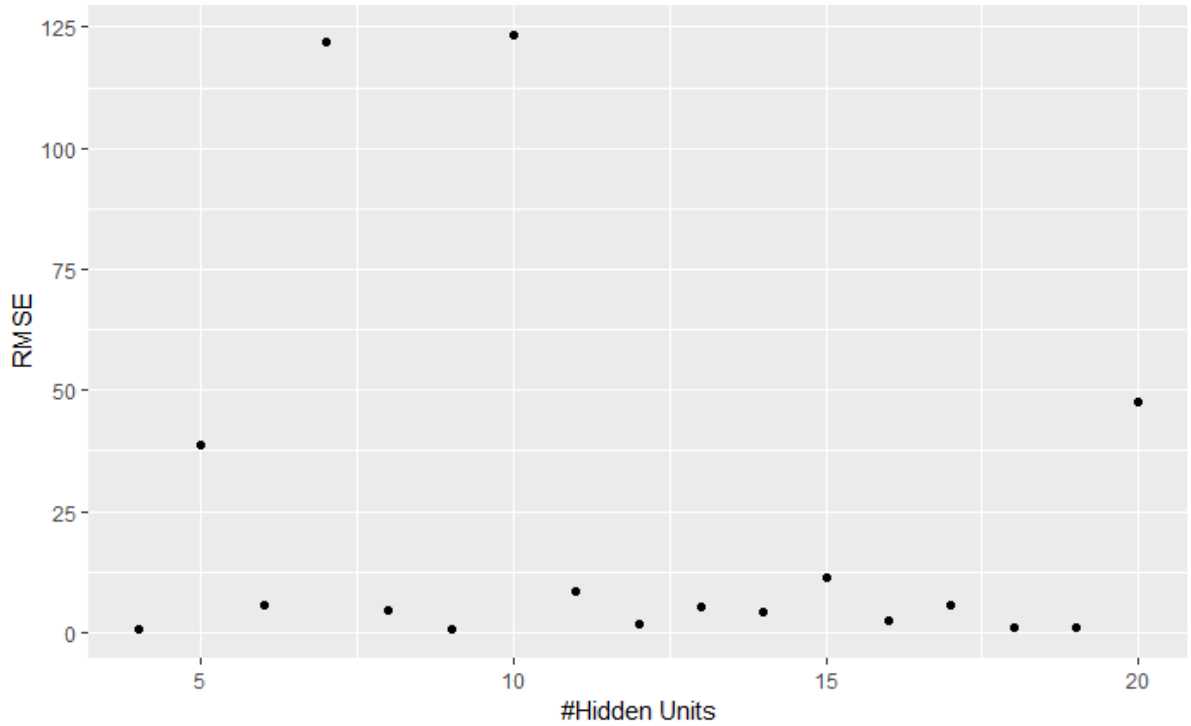


Figure 25 RBF method, tuning the size of the hidden layer. The best model was a network with four (4) neurons.

Resilient Propagation Algorithm (RPROP)

Training was repeated using RPROP algorithm. Tuning was applied to the number of hidden units parameter. The number of neurons was incremented sequentially, starting from three neurons, until it reached twenty neurons in the hidden layer.

The results of tuning the number of hidden units for the RPROP algorithm will be discussed in the next chapter.

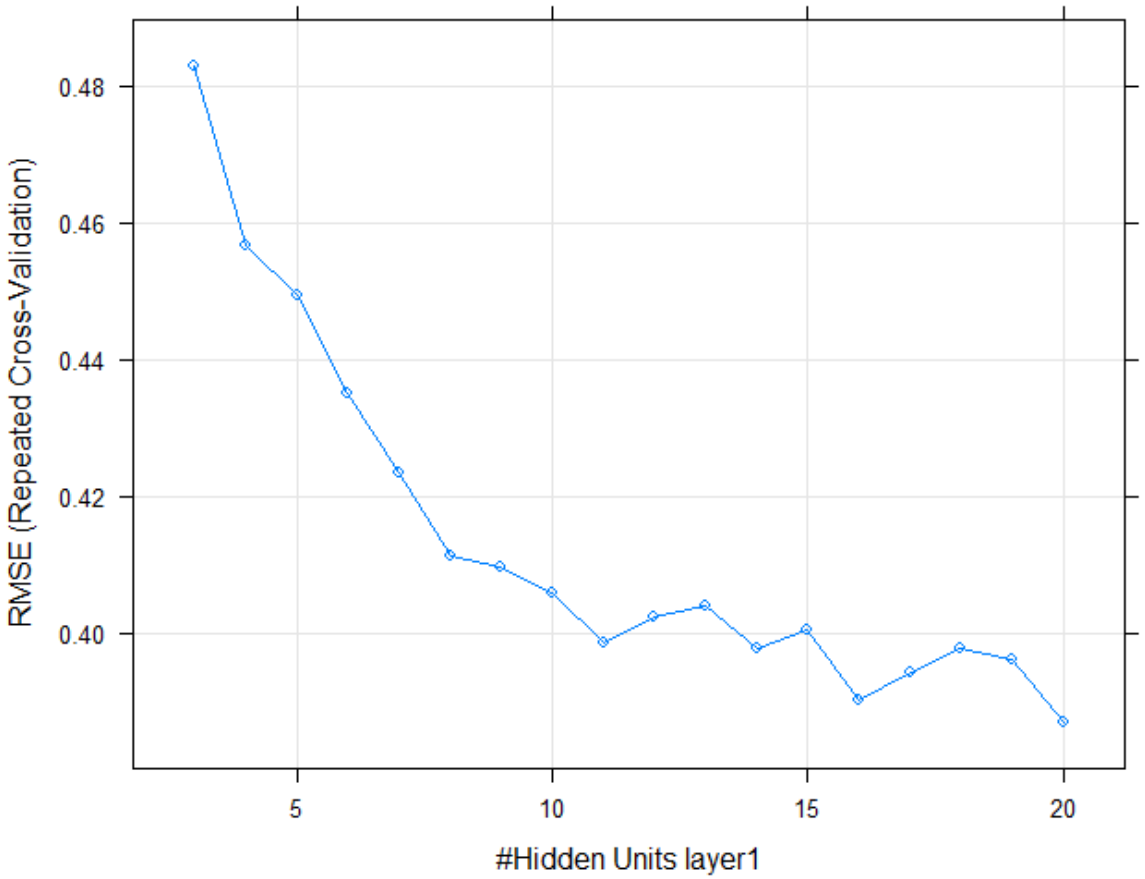


Figure 26 RPROP method, tuning the size of the hidden layer. The best model was a network with twenty (20) neurons.

CHAPTER 5:
Analysis and Discussion of the Results

This chapter presents the analysis and discussion of data gathered during neural network simulation. Responses from all neural networks, taken during simulation using different training algorithms and implementing hyperparameter tuning were examined, compared and evaluated to answer the research questions asked at the beginning of the study. As far as possible, data were tabulated and displayed through tables, with the aim of identifying and discerning any patterns that provided the best interpretation of the results of the study.

Comparison of Algorithms

After training with multiple algorithms, we had to compare them and see which one performs best. In order to do that, a test set was used. Our test set was made of well log data from Luva well. Measured data were plotted green, while predicted data appear with blue color.

Looking at figures 13 to 17, we take a first idea of the prediction performance of the algorithms. It appears that SCG, RPROP and SGD algorithms make good prediction of the "target" Luva well. RBF did not perform well and BATCHPROP did not manage to converge at all.

A comparison between the training performance of the algorithms can be seen in Figure 27, where the prediction errors for each algorithm are plotted. It can be seen that SCG outperforms every other algorithm. Notice that all algorithms in the figure derive from the same family of Supervised Learning with Batch Update algorithms. Despite the fact that the figure represents the simplest case (four hidden units, no tuning), SCG succeeds in giving us a decent first impression of its abilities

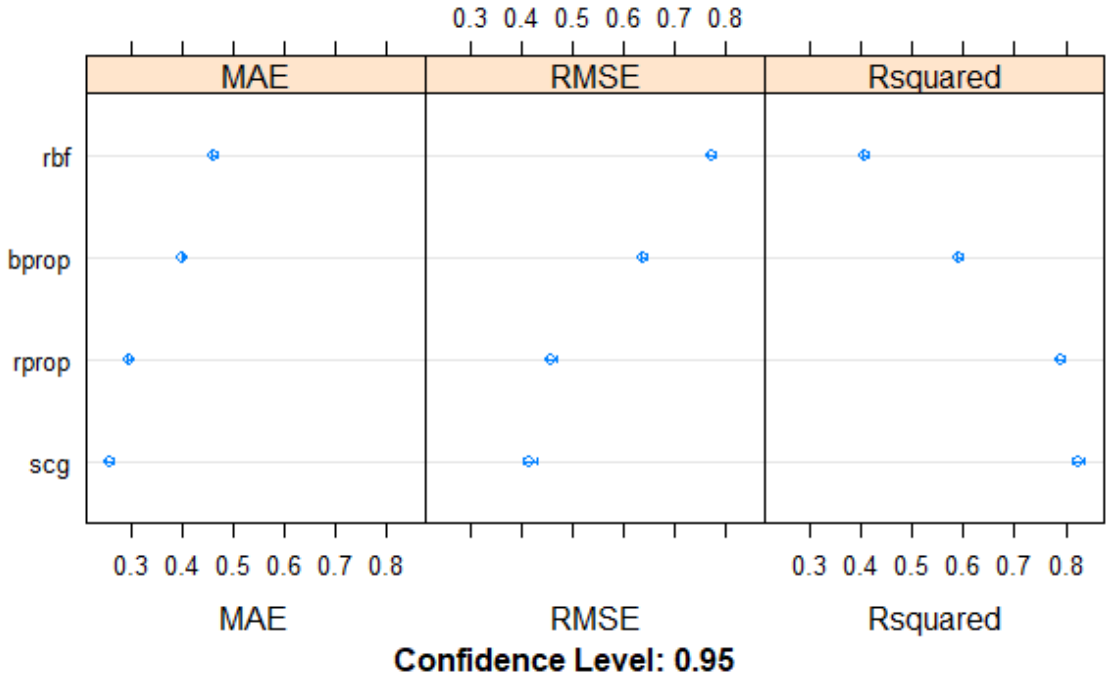


Figure 27 Comparison of training performance metrics between the algorithms used for training of the ANN.

Training with Multiple Network Topologies & Hyperparameter Tuning

Learning Rate

In Figure 18 we can see tuning of learning rate parameter for SGD algorithm. A simple grid search procedure was conducted. The training RMSE appeared to decrease when the learning rate was increased from 0.01 to 0.1. We decided not to tune further than 0.1, as the network stopped converging after this value.

In Table 5 we applied a linear grid search, varying the learning rate parameter from 0.1 to 1.0. Using cross-validation and RMSE, the value 0.3 was picked as the optimum learning rate for training with the BATCHPROP algorithm. The network produced using these learning rate manages to converge at the first depth steps but fails to provide us with useful results for Luva well, as depicted in Figure 23.

Hidden Units & Information Criteria

Taking a look at Figure 19, the network using eighteen (18) hidden units succeeds in performing best during training. The second and third candidates feature twenty (20) and sixteen (16) neurons respectively. However, taking a closer look at the results, we notice that a network using three (3) neurons can effectively compete the previous one, having a training RMSE of 4.9836. This is a 4.39% percentage difference compared to RMSE 4.7742 of the big network.

Next will look at the AIC(2) values, for the same algorithm. Figure 20 shows a similar trend with the RMSE figure. AIC(2) scores show us that the network with the smallest AIC(2) score has eighteen (18) neurons with a score of 18506. However, this score does not have a big difference with the rest network scores. This is a 1.91% percentage difference compared to AIC(2) 18506 value of the best network, compared with AIC(2) 18859 of the network with three (2) neurons.

Repeating the above comparison using the AIC(4) and BIC values (Figure 21, Figure 22) the difference, between the mean percentage of the "runners-up" and the best model is at 0.89% and 0.87% respectively.

We applied parameter tuning to find the optimal number of hidden units for SCG algorithm. This resulted in a total of nineteen (19) hidden units. The same procedure was applied for RPROP algorithm; twenty (20) hidden units were found to perform best. The results are show at Figure 24 and Figure 26. Training RMSE seems to decrease while we add more neurons in the hidden layer.

Tuning hidden units parameter for RBF network gave us a random pattern of RMSE, as seen in Figure 25. The best network consisted of a small number of neurons. However an R^2 value of 41.9% revealed poor performance of the network.

~~~~~~

In Figure 28, it can be seen that SCG still outperforms every other algorithm. We note that even after parameter tuning, RBF and BPROP algorithms did not make any significant improvement in performance. RPROP and SCG however showed some signs of improvement.

In Figure 29 the predicted resistivity of two tuned networks (SCG + RPROP algorithms) is include at the same plot versus measured resistivity. We observe that both algorithms succeed in making a fair prediction of resistivity.
~~~~~~

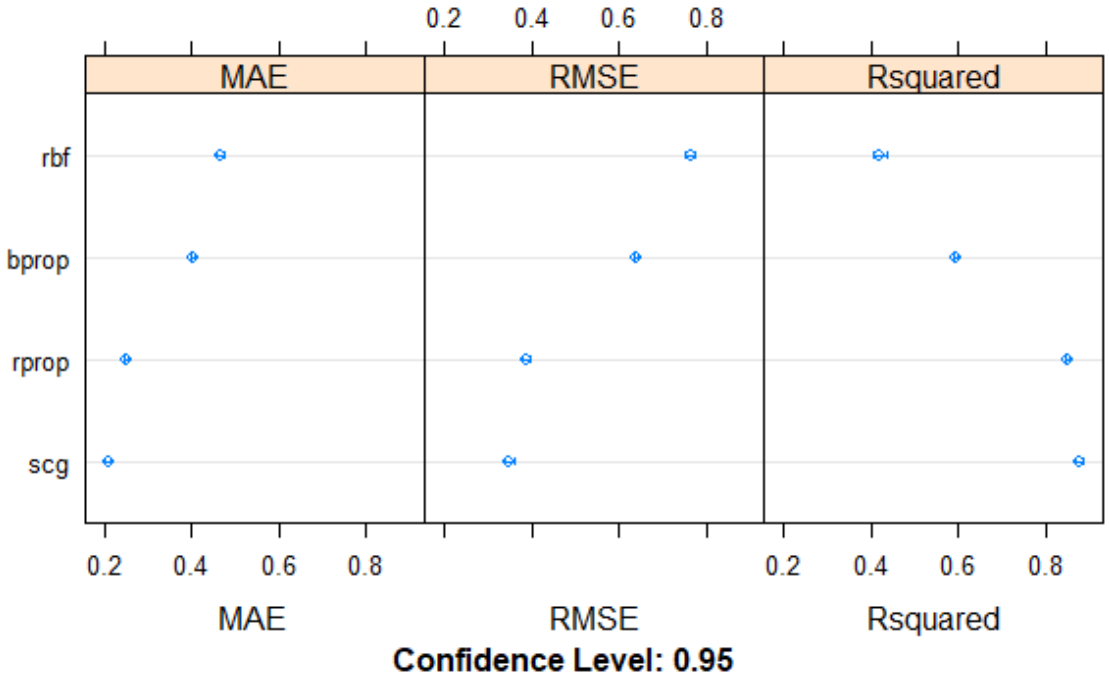


Figure 28 Comparison of training performance metrics between the tuned algorithms.

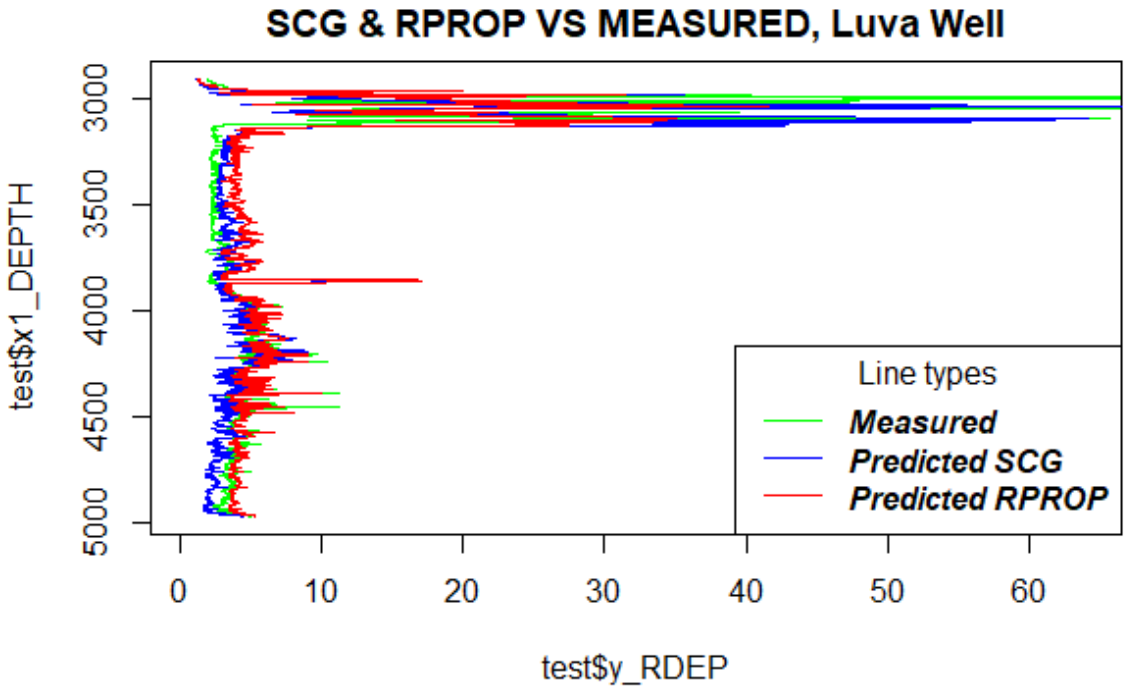


Figure 29 Plot of measured (green) vs predicted resistivity. [SCG (blue) network and RPROP (red) network]

CHAPTER 6:
Conclusions – Further Research

Conclusions

Artificial Neural Networks are models inspired by biological neural networks. They are commonly used for regression and classification problems but are comprised of many algorithms and variations for all manner of problem types. As part of this thesis, we were concerned with the more classic methods, such as Back-Propagation algorithms, Stochastic Gradient Descent and Radial Basis Function Networks (RBFN).

Some of the algorithms managed to converge while others failed. However, one must conduct various tests, including different algorithms, before finally deciding the one that performs best. As a learning strategy, one could fix either parameter at the value corresponding to the least constrained model, to ensure that the model is rich enough, and use cross-validation to choose the other parameter.

To summarize, with regards to neural network training, one has to select a number of parameters, such as the number of hidden units, the learning rate, etc. SCG algorithm is more flexible compared to other algorithms, as it does not depend on any other parameters for training.

The main pitfall when using RMSE as the objective function is obtaining good estimates of the error rate that are not subject to over-fitting. It was shown, through the application of information criteria such as AIC, that a model with many neurons does not always have to be preferred as the final model.

Further Research

Many algorithm families were not covered in this thesis. There are different ways an algorithm can model a problem, based on its interaction with the available data. Other powerful that can be used as an approach to similar problems include Boosting Techniques, Random Forests, AdaBoost, etc.

If the data is not very reliable due to noise or missing factors, learning is only one part of the model building procedure. If that is the case, we need to gather more information to assist the building process with prior knowledge about the specific task. This way, we battle uncertainty introduced by the data. This thesis evaluated resistivity predictions made from sonic and depth data originating from three (3) Norwegian wells. More parameters and more well can be examined in the future.

Neural network training practices that can be found in the world-wide web are countless. In addition to this, much progress has been made during the last decade, regarding the available computing power. It is relative easy to use this large computing power and feed all kinds of available data in a neural network.

By using R-Studio as our programming platform, we have a huge collection of libraries to assist us in machine learning processes. Also, R language is really easy to learn. This helps us prove the theory behind neural network training. As a next step, a program can be developed in Python language, that provides more capabilities to extend. Python is the most popular language used for Machine Learning (42).

REFERENCES

1. *Direct Elastic Properties to Resistivity Transform.* **Economou Dimitrios, Alaei, Behzad.** 5, 2016, Geophysics, Vol. 81, pp. IM109-IM118.

2. **Society of Petroleum Engineers.** Density logging. [Online] Jun 24, 2015. [Cited: Jan 12, 2019.] https://petrowiki.org/Density_logging.

3. **GeoSci Developers.** Seismic Velocity. [Online] 2017. [Cited: Jan 12, 2019.] https://gpg.geosci.xyz/content/physical_properties/physical_properties_density.html.

4. **Society of Petroleum Engineers.** Compressional and shear velocities. [Online] Jun 3, 2015. [Cited: Jan 12, 2019.] https://petrowiki.org/Compressional_and_shear_velocities.

5. **Sainso, Stephane.** *Electromagnetic Seabed Logging: A new tool for geoscientists.* s.l. : Springer, 2017.

6. **Glover, Dr. Paul W.J.** *Petrophysics MSc Course Notes.* [Online] http://homepages.see.leeds.ac.uk/~earpwjg/PG_EN/CD%20Contents/GGL-66565%20Petrophysics%20English/.

7. *Background resistivity model from seismic velocities.* **Werthmuller, Dieter, Ziolkowski, Anton and Wright, David.** 4, 2013, Geophysics, Vol. 78, pp. E213-E223.

8. **Wikipedia.** Archie's Law. [Online] Oct 6, 2018. [Cited: Jan 13, 2019.] https://en.wikipedia.org/wiki/Archie%27s_law.

9. **Russell, Stuart J. and Norvig, Peter.** *Artificial Intelligence: A Modern Approach.* Third Edition. s.l. : Prentice Hal, 2010.

10. **Bertsekas, Dimitri P.** *Dynamic Programming and Optimal Control: Approximate Dynamic Programming, Vol.II.* s.l. : Athena Scientific, 2012.

11. **Bertsekas, Dimitri P. and Tsitsiklis, John N.** *Neuro-Dynamic Programming.* s.l. : Athena Scientific, 1996.

12. **Bishop, C.** *Neural Networks for Pattern Recognition.* Oxford : Oxford University Press, 1995.

13. **Ripley, B.** *Pattern Recognition and Neural Networks.* s.l. : Cambridge University Press, 1996.

14. *Neural Networks.* **Titterington, M.** 1, 2010, Wiley Interdisciplinary Reviews: Computational Statistics, Vol. 2, pp. 1-8.

15. *Building Predictive Models in R Using the caret Package.* **Kuhn, Max.** 5, 2008, Journal of Statistical Software, Vol. 28.

16. **Rumelhart, D, Hinton, G and Williams, R.** Learning Internal Representations by Error Propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* s.l. : The MIT Press, 1986.

17. **Wang, C and Venkatesh, S.** Optimal Stopping and Effective Machine Complexity in Learning. *Advances in NIPS.* 1984, pp. 303–310.

18. **Neal, R.** *Bayesian Learning for Neural Networks.* s.l. : Springer-Verlag, 1996.

19. **Kohonen, T.** *Self-Organizing Maps.* s.l. : Springer, 1995.

20. *Supervised Kohonen Networks for Classification Problems.* **Melssen, W, Wehrens, R and Buydens, L.** 2, 2006, Chemometrics and Intelligent Laboratory Systems, Vol. 83, pp. 99-113.

21. **Perrone, M and L, Cooper.** When Networks Disagree: Ensemble Methods for Hybrid Neural Networks. [book auth.] R J Mammone. *Artificial Neural Networks for Speech and Vision," pp. 126–142.* London : Chapman & Hall, 1993, pp. 126-142.

22. **Ripley, B.** Statistical Ideas for Selecting Network Architectures. *Neural Networks: Artificial Intelligence and Industrial Applications.* 1995, pp. 183-190.

23. *Analysis of Decision Boundaries in Linearly Combined Neural Classifiers.* **Tumer, K and Ghosh, J.** 2, 1996, Pattern Recognition, Vol. 29, pp. 341–348.

24. **Widrow, B and Hoff, M.E.** Adaptive switching circuits. *IRE WESCON Convention Record.* 1960, pp. 96–104.

25. **Silva, FM and Almeida, LB.** Speeding-up backpropagation. *Advances of Neural Computers.* North Holland : Elsevier Science Publish , 1990, pp. 151-158.

26. *An Improved Learning Law for Backpropagation Networks.* **Zhou, S, Popovic, D and Schulz-Ekloff, G.** San Francisco : s.n., 1991. IEEE Int. Conf. on Neural Networks.

27. **Stergiopoulos, Stergios.** *Advanced Signal Processing Handbook: Theory and Implementation for Radar, Sonar, and Medical Imaging Real-Time Systems.* s.l. : CRC Press, 2018.

28. **Park, J and Sandberg, I W.** Universal Approximation Using Radial-Basis-Function Networks. *Neural Computation.* Summer, 1991, Vol. 3, 2, pp. 246-257.

29. **Lennart, Ljung.** Perspectiveson System on System Identification. [Online] [Cited: Feb 21, 2019.] http://users.isy.liu.se/en/rt/ljung/seoul2dvinew/seoul_ljung_ho.pdf.

30. **Electrical and Computer Engineering Northwestern University.** System Identification Toolbox: The Basic Steps of System Identification. [Online] Nov 4, 2003. [Cited: Feb 21, 2019.] http://www.ece.northwestern.edu/local-apps/matlabhelp/toolbox/ident/ch1sip9.html.

31. **Lennart, Ljung.** *System Identification: Theory for the User.* s.l. : Prentice Hall, 1987.

32. **Lahiri, Sandip Kumar.** *Multivariable predictive control : applications in industry.* s.l. : John Wiley & Sons Ltd, 2017.

33. **Palit, Ajoy K. and Popovic, Dobrivoje.** *Computational Intelligence in Time Series Forecasting.* s.l. : Springer-Verlag London, 2005.

34. **International, SPE.** Upscaling of grid properties in reservoir simulation. [Online] SPE International, June 12, 2015. [Cited: December 22, 2018.] https://petrowiki.org/Upscaling_of_grid_properties_in_reservoir_simulation.

35. **Fisher, Robert, et al., et al.** Image Processing Learning Resources. *Gaussian Smoothing.* [Online] January 2019, 22. https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm.

36. **Mitchell, M.** *An Introduction to Genetic Algorithms.* s.l. : MIT Press, 1998.

37. *The Nelder–Mead Simplex Procedure for Function Minimization.* **Olsson, D and L, Nelson.** 1, 1975, Technometrics, Vol. 17, pp. 45-51.

38. **Tianqi, Chen, Qiang, Kou and Tonh, He.** Package ‘mxnet’. [Online] April 6, 2018. [Cited: October 12, 2018.] https://s3.amazonaws.com/mxnet-prod/docs/R/mxnet-r-reference-manual.pdf.

39. **Bergmeir, Christoph.** Neural Networks using the Stuttgart Neural Network Simulator(SNNS). [Online] August 10, 2018. [Cited: October 12, 2018.] https://cran.r-project.org/web/packages/RSNNS/RSNNS.pdf.

40. NPD FactPages. [Online] December 15, 2018. http://factpages.npd.no/factpages/.

41. **University of Stuttgart; University of Tubingen.** Stuttgart Neural Network Simulator. [Online] [Cited: Feb 13, 2019.] http://www.it.uom.gr/teaching/SNNSv4.2.Manual.pdf.

42. **Patel, Prince.** Why Python is the most popular language used for Machine Learning. [Online] Udacity India, March 8, 2018. [Cited: December 21, 2018.] https://medium.com/@UdacityINDIA/why-use-python-for-machine-learning-e4b0b4457a77.

43. **Wikipedia.** Model selection. [Online] Wikipedia, March 7, 2019. [Cited: March 10, 2019.] https://en.wikipedia.org/wiki/Model_selection.

44. **Kuhn, Max and Johnson, Kjell.** Over-Fitting and Model Tuning. *Applied Predictive Modeling.* New York : Springer, 2013, pp. 61-88.

45. **James, Gareth, et al., et al.** *An Introduction to Statistical Learning with Applications in R.* s.l. : Springer, 2013.

46. **Hastie, Trevor, Tibshirani, Robert and Friedman, Jerome.** *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* California : Springer, 2008.

47. **Watson, Mark.** *Practical Artificial Intelligence Programming in Java, 3rd Edition.* s.l. : lulu.com, 2008.

ANNEX A: Paper Reproduction & Implementation of More Algorithms Results

SCG method

Time difference of 3.161498 mins

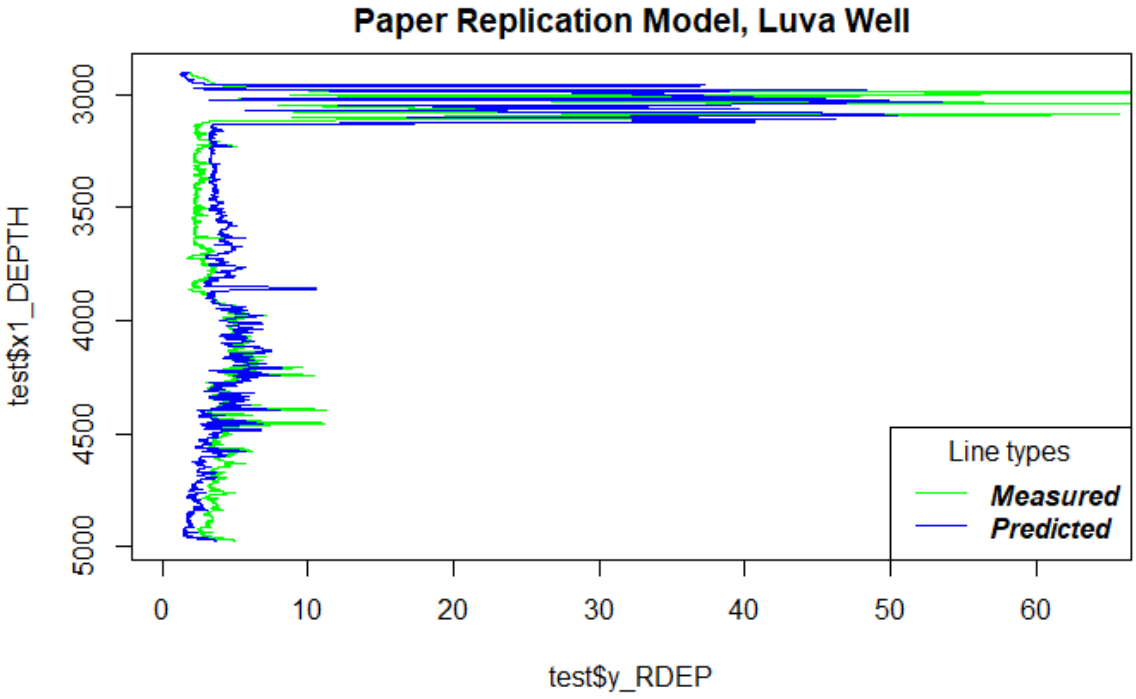
Resampling results:

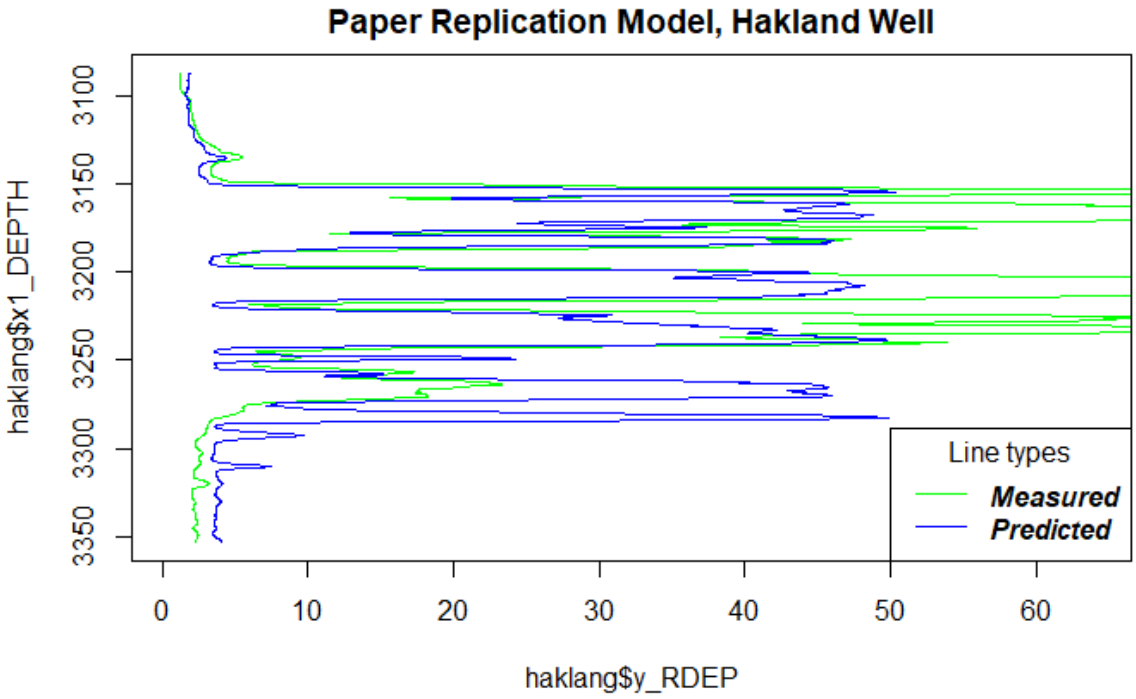
RMSE	Rsquared	MAE
0.4197269	0.8218319	0.2634368

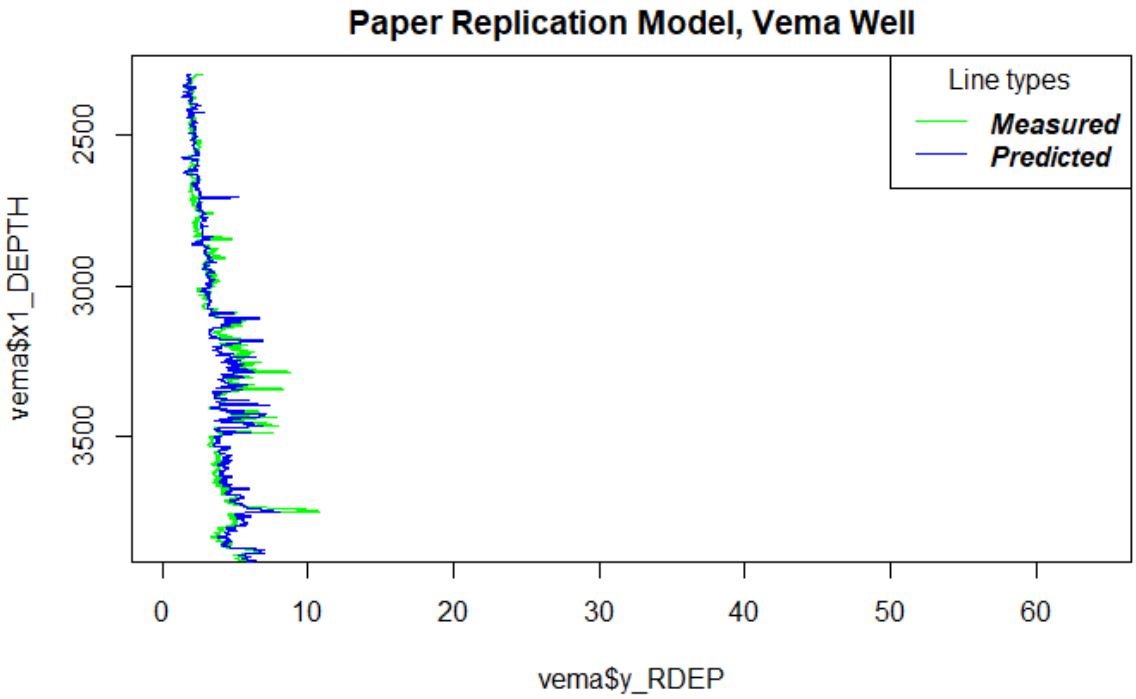
Tuning parameter 'layer1' was held constant at a value of 4

bestTune predictions:

	RMSE	Rsquared	MAE
luva	6.2633758	0.590215	2.305258
haklang	18.2095735	0.666655	10.06982
vema	0.7781473	0.848496	0.543986


Paper Replication Model, Luva Well
test$x1_DEPTH
3000
3500
4000
4500
5000
0
10
20
30
40
50
60
test$y_RDEP
Line types
Measured
Predicted


Paper Replication Model, Hakland Well
haklang$x1_DEPTH
3100
3150
3200
3250
3300
3350
0
10
20
30
40
50
60
haklang$y_RDEP
Line types
Measured
Predicted



BPROP method

Time difference of 1.848162 mins

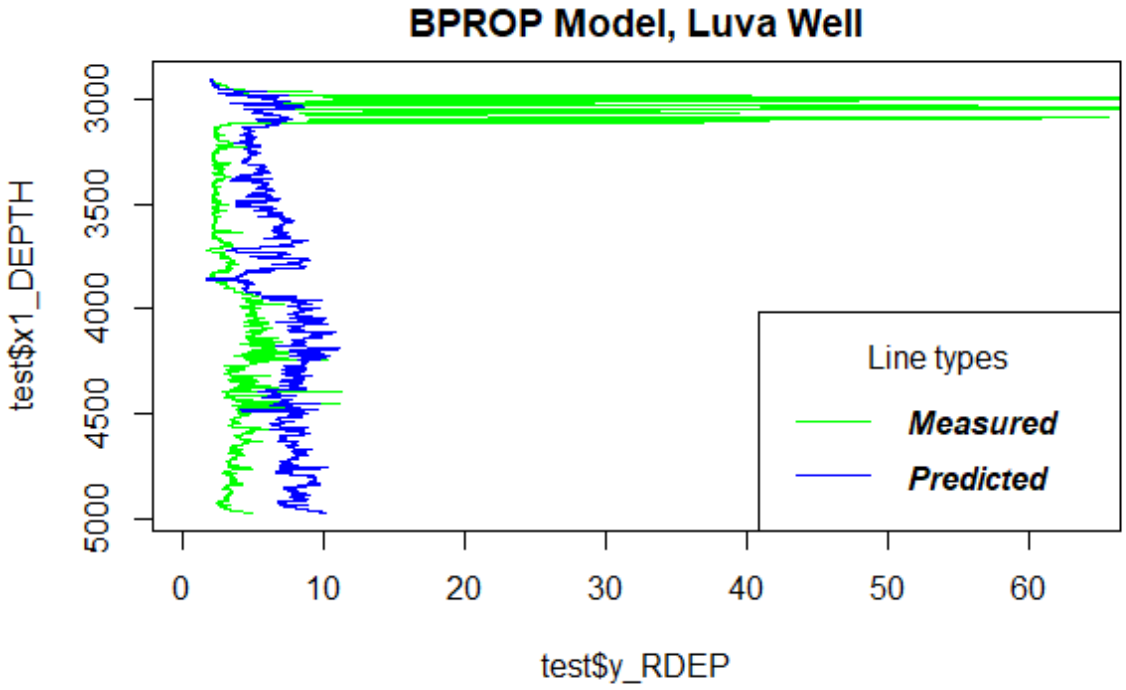
Resampling results:

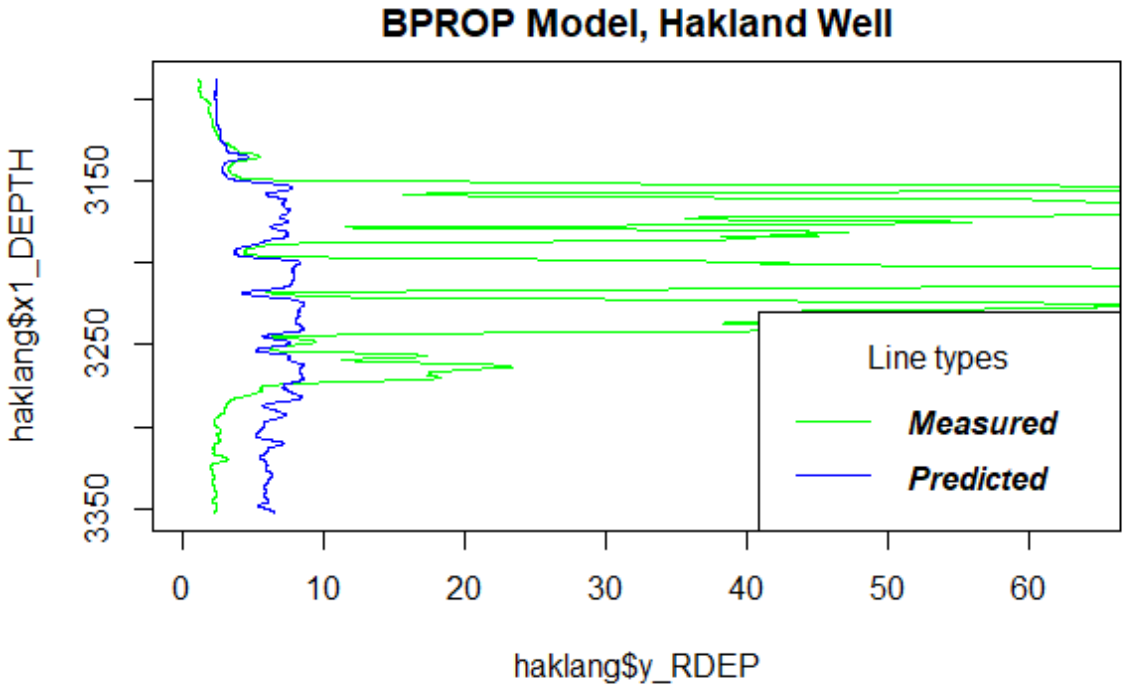
RMSE	Rsquared	MAE
0.6392809	0.5927361	0.4026228

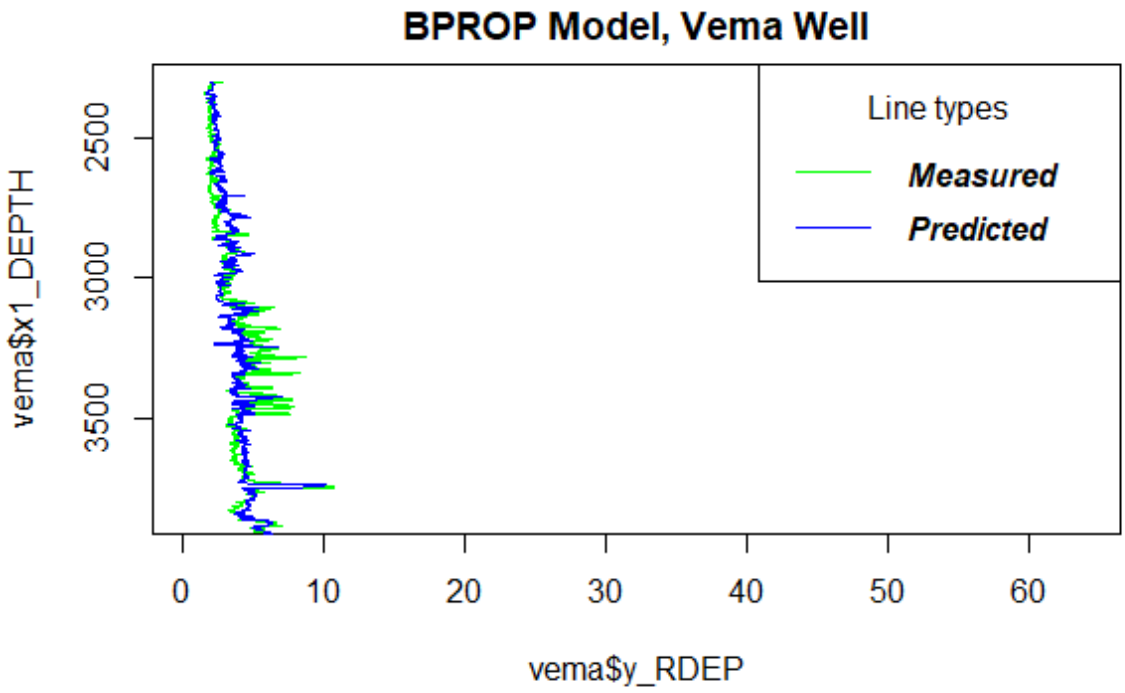
Tuning parameter 'layer1' was held constant at a value of 4

bestTune predictions:

	RMSE	Rsquared	MAE
luva	9.1863206	0.129248	5.205752
haklang	28.8204474	0.41616	15.97498
vema	1.238673	0.645446	0.84508


BPROP Model, Luva Well
test$x1_DEPTH
3000
3500
4000
4500
5000
0
10
20
30
40
50
60
test$y_RDEP
Line types
Measured
Predicted


BPROP Model, Hakland Well
haklang$x1_DEPTH
3150
3250
3350
0
10
20
30
40
50
60
haklang$y_RDEP
Line types
Measured
Predicted



RBF method

Time difference of 2.034333 mins

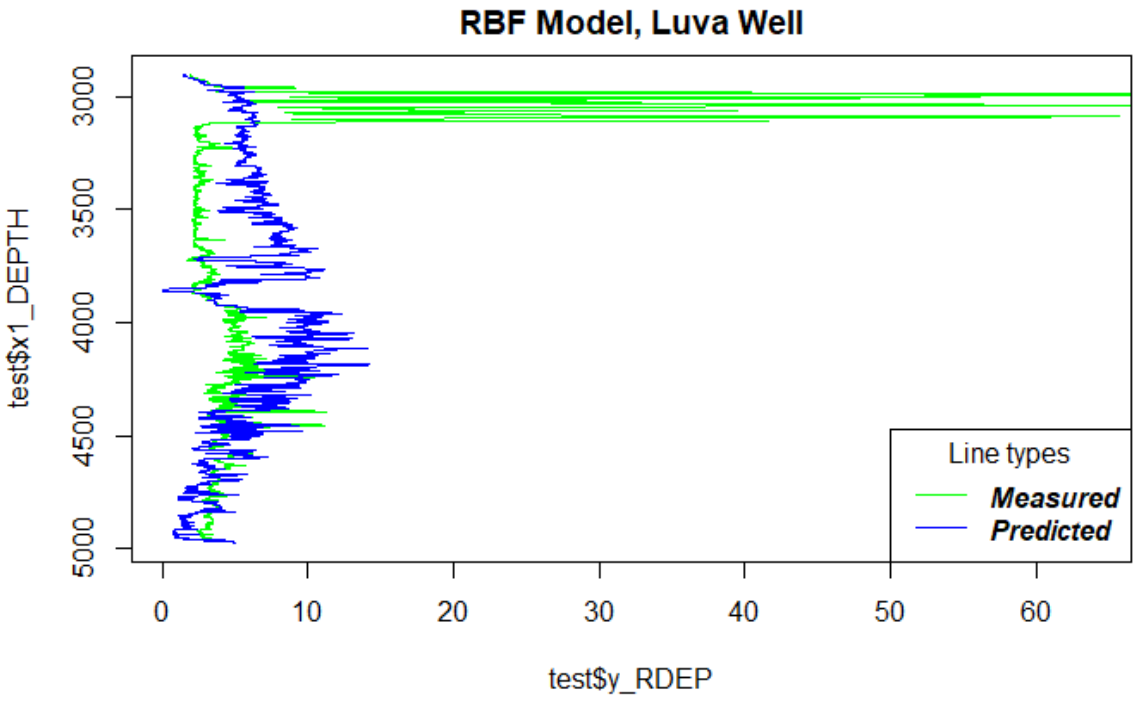
Resampling results:

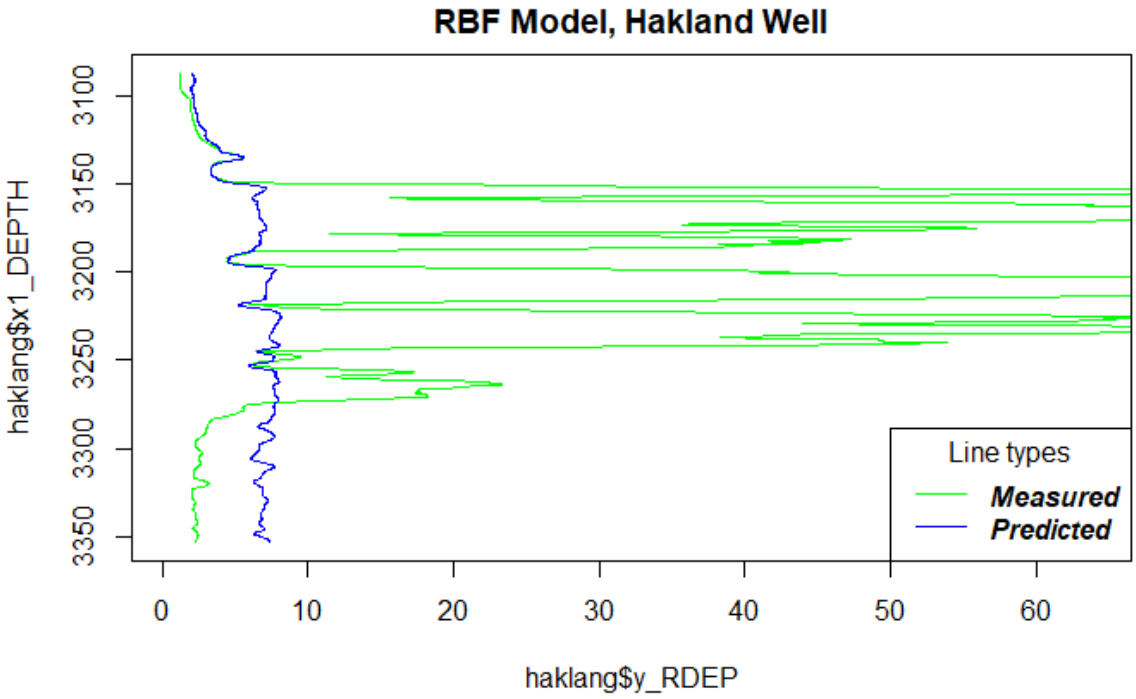
RMSE	Rsquared	MAE
0.7726364	0.4083333	0.464147

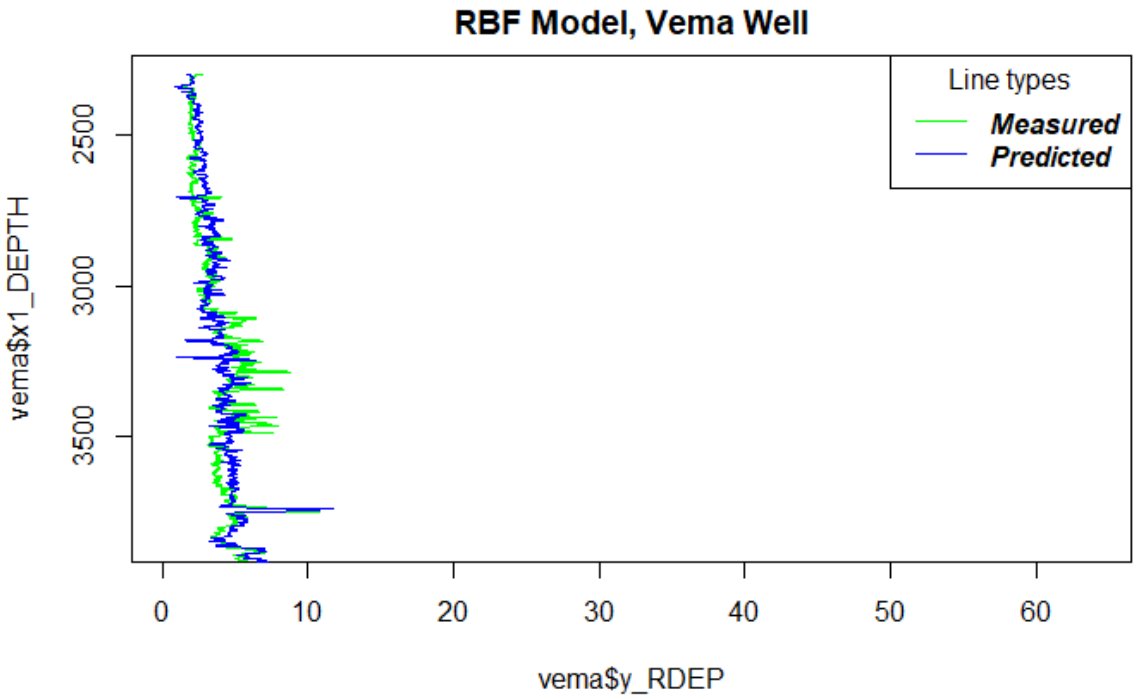
Tuning parameter 'size' was held constant at a value of 4

bestTune predictions:

	RMSE	Rsquared	MAE
luva	9.811762389	0.000984	4.446456
haklang	31.3582743	0.146027	17.32983
vema	1.2821642	0.609827	0.90166


RBF Model, Luva Well
test$x1_DEPTH
3000
3500
4000
4500
5000
0
10
20
30
40
50
60
test$y_RDEP
Line types
Measured
Predicted


RBF Model, Hakland Well
haklang$x1_DEPTH
3100
3150
3200
3250
3300
3350
0
10
20
30
40
50
60
haklang$y_RDEP
Line types
Measured
Predicted



RPROP method

Time difference of 1.591093 mins

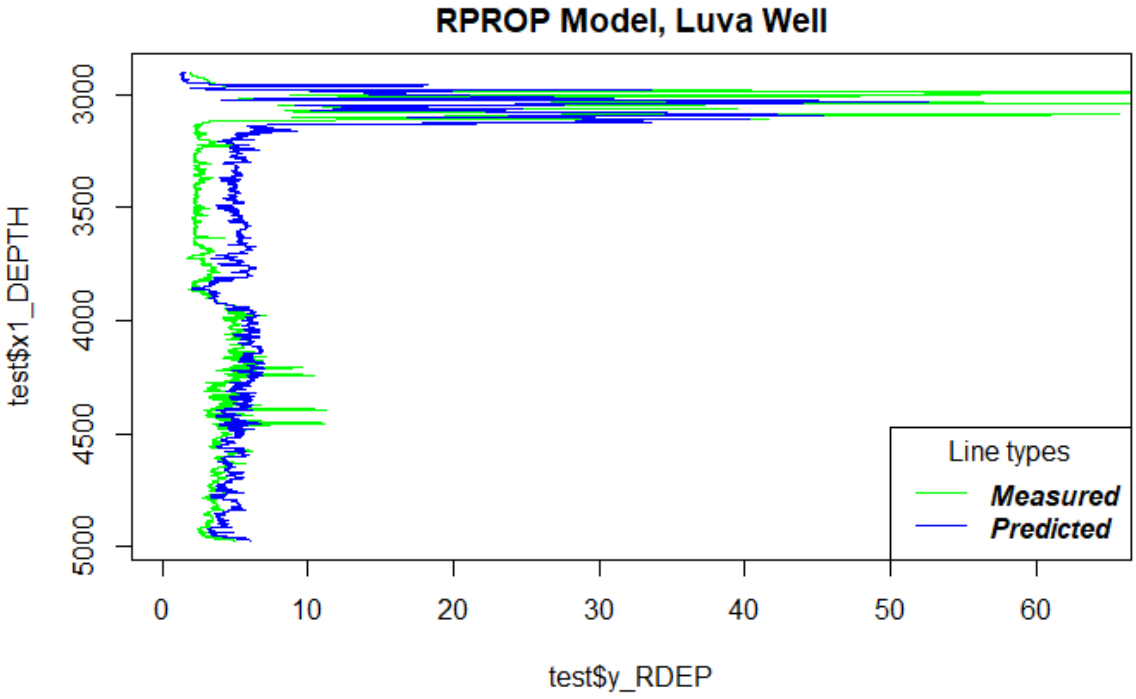
Resampling results:

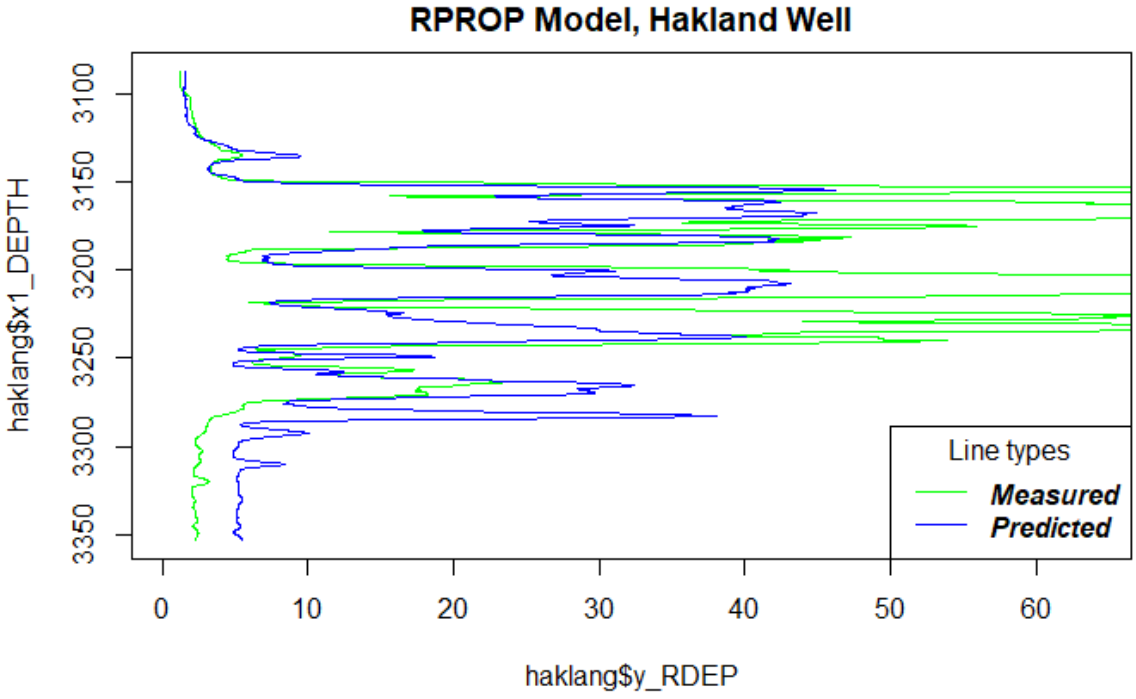
RMSE	Rsquared	MAE
0.4596228	0.7882741	0.29928

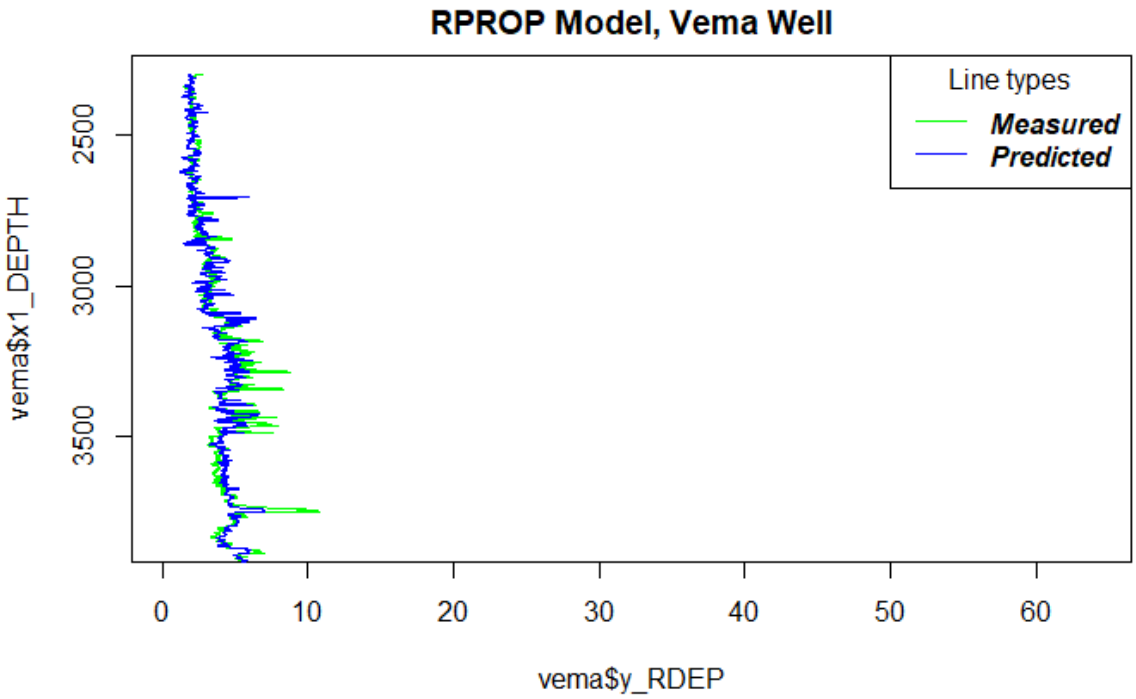
Tuning parameter 'layer1' was held constant at a value of 4

bestTune predictions:

	RMSE	Rsquared	MAE
luva	6.7316402	0.511982	2.622463
haklang	19.8633444	0.709383	10.89701
vema	0.9338216	0.794793	0.619898


RPROP Model, Luva Well
test$x1_DEPTH
3000
3500
4000
4500
5000
0
10
20
30
40
50
60
test$y_RDEP
Line types
Measured
Predicted


RPROP Model, Hakland Well
haklang$x1_DEPTH
3100
3150
3200
3250
3300
3350
0
10
20
30
40
50
60
haklang$y_RDEP
Line types
Measured
Predicted


RPROP Model, Vema Well
Line types
Measured
Predicted
vema$x1_DEPTH
2500
3000
3500
0
10
20
30
40
50
60
vema$y_RDEP

ANNEX B: Training with Multiple Network Topologies & Hyperparameter Tuning Results

BPROP method

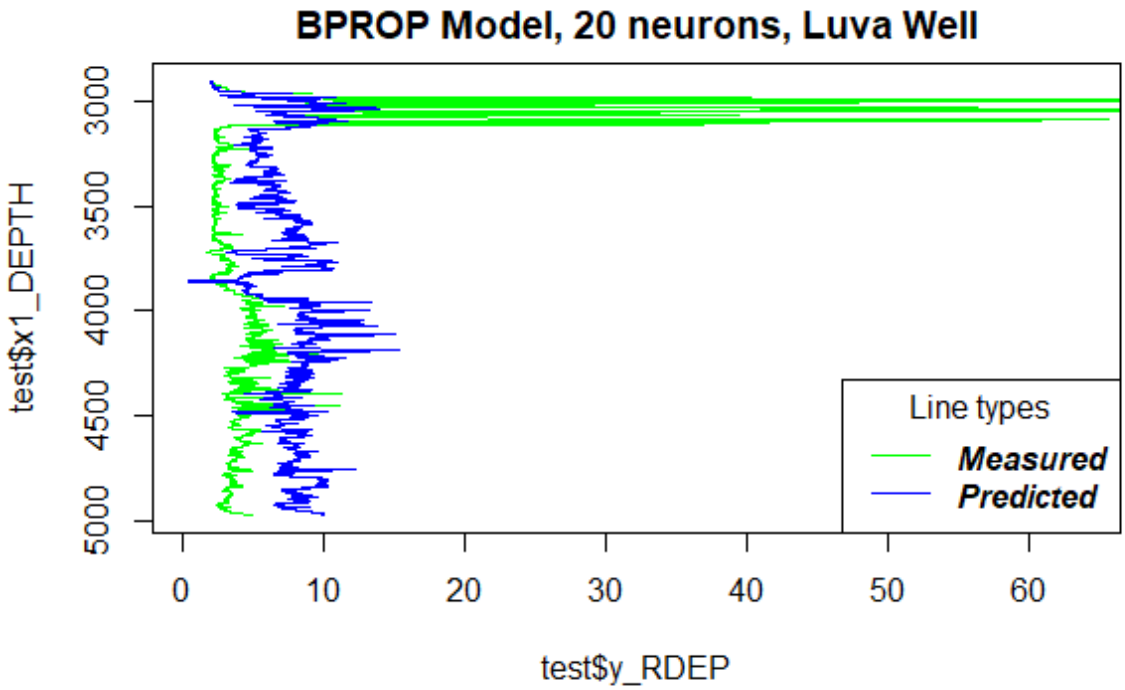
Resampling results across tuning parameters:

layer1	RMSE	Rsquared	MAE
3	0.727129	0.486219	0.410935
4	0.717978	0.498813	0.407355
5	0.713488	0.505259	0.406365
6	0.704399	0.516702	0.405269
7	0.698575	0.524922	0.405156
8	0.69457	0.52929	0.406275
9	0.69122	0.534002	0.406415
10	0.687134	0.539384	0.405506
11	0.68185	0.545352	0.406753
12	0.679654	0.548169	0.40692
13	0.677069	0.550985	0.407246
14	0.672151	0.556587	0.407986
15	0.670943	0.557669	0.408449
16	0.668897	0.559328	0.410376
17	0.668228	0.56004	0.410673
18	0.666952	0.561592	0.410435
19	0.663452	0.565324	0.410647
20	**0.662761**	**0.565564**	**0.412639**

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was layer1 = 20.

bestTune predictions:

	RMSE	Rsquared	MAE
luva	9.317063	0.076685	5.062602
haklang	29.50387	0.39526	16.27774
vema	1.194333	0.661014	0.8206



RPROP method

Resampling results across tuning parameters:

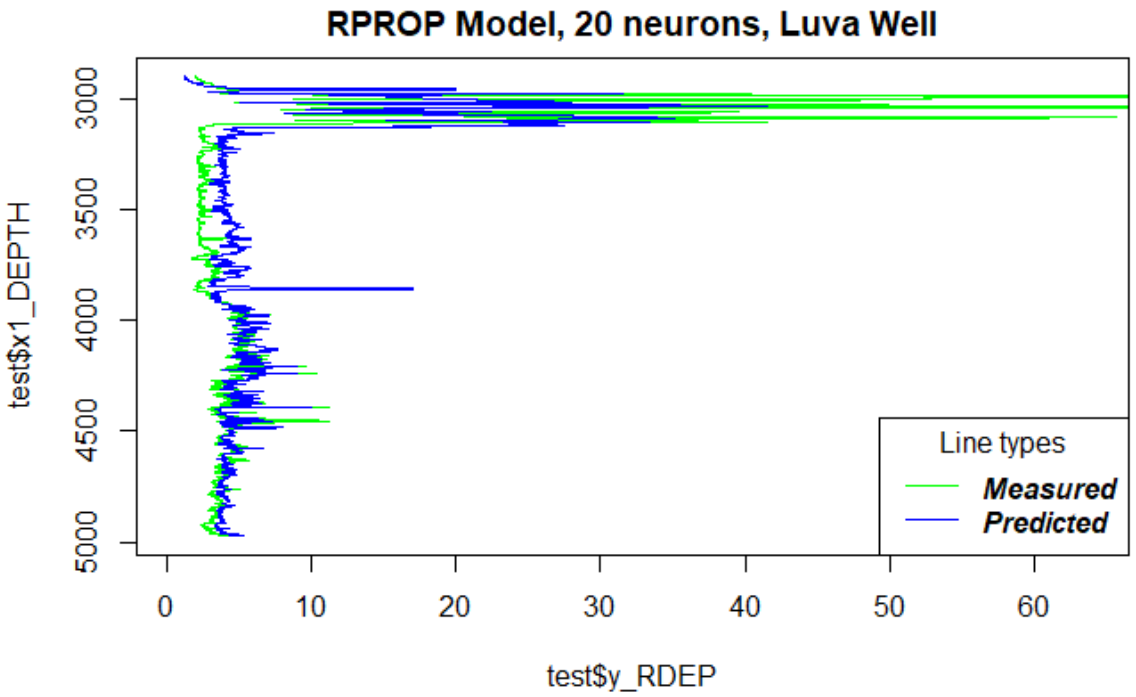
layer1	RMSE	Rsquared	MAE
3	0.482853	0.765594	0.318259
4	0.456678	0.791095	0.29825
5	0.449262	0.797447	0.290368
6	0.435177	0.810304	0.280554
7	0.423367	0.820061	0.274266
8	0.411404	0.829932	0.2669
9	0.409634	0.831528	0.262034

10	0.405902	0.833758	0.26407
11	0.398609	0.840188	0.256821
12	0.40253	0.836927	0.260215
13	0.403968	0.835995	0.258668
14	0.39784	0.840657	0.256607
15	0.400411	0.839025	0.255066
16	0.390212	0.846664	0.251358
17	0.394251	0.84421	0.252993
18	0.397937	0.840857	0.255679
19	0.396325	0.842005	0.253421
20	**0.387049**	**0.849415**	**0.248772**

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was layer1 = 20.

bestTune predictions:

	RMSE	Rsquared	MAE
luva	6.7554844	0.5210658	2.1741808
haklang	21.0693586	0.6301375	11.594043
vema	0.7764832	0.8382733	0.5655033



SCG method

Resampling results across tuning parameters:

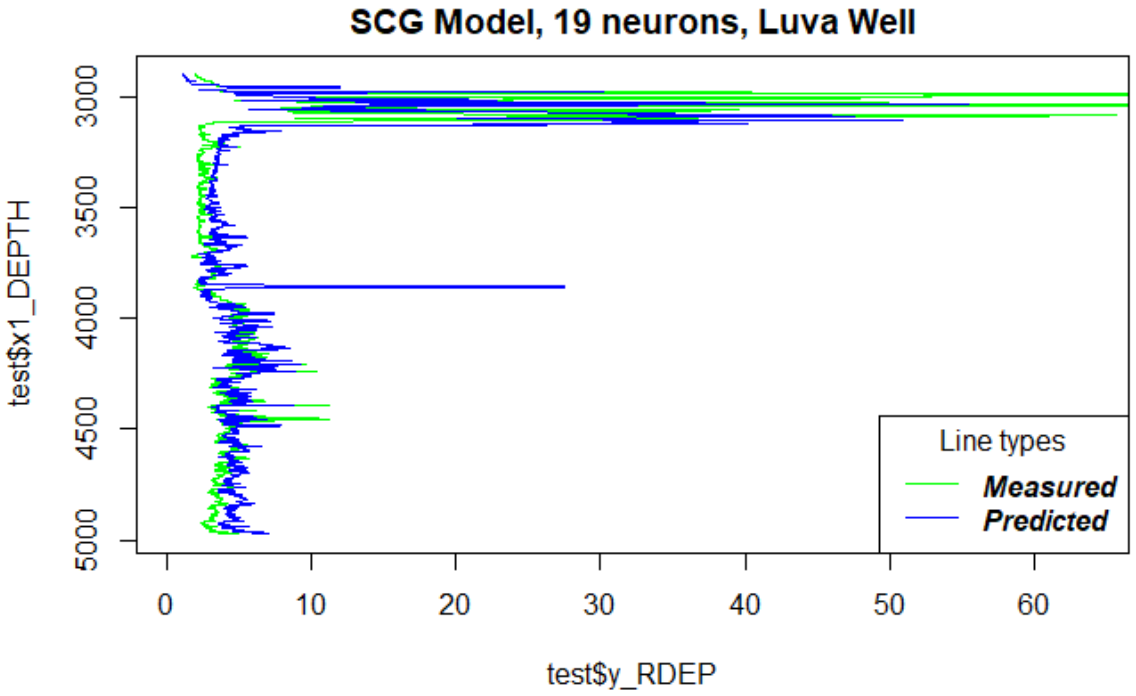
layer1	RMSE	Rsquared	MAE
3	0.4347583	0.8087393	0.2756449
4	0.4009394	0.8391296	0.2490681
5	0.3941327	0.8434172	0.2462572
6	0.3841873	0.8526534	0.2386997
7	0.3716077	0.8616564	0.227948
8	0.3675789	0.8638884	0.2245536
9	0.3712201	0.8609872	0.22643
10	0.3589929	0.8707017	0.2179408
11	0.3584107	0.8700407	0.2165278
12	0.3581387	0.8702195	0.216297
13	0.3719283	0.8580052	0.2268751
14	0.3507384	0.876483	0.2087093
15	0.3486167	0.877869	0.209409

16	0.354286	0.8735585	0.2125658
17	0.3590655	0.8673027	0.2157034
18	0.3487109	0.8782396	0.2070068
19	**0.3440567**	**0.8812607**	**0.2033639**
20	0.3521757	0.8737203	0.2101529

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was layer1 = 19.

bestTune predictions:

	RMSE	Rsquared	MAE
luva	8.0398078	0.2967753	2.4841986
haklang	16.7164851	0.7242923	9.0670853
vema	0.6999764	0.8685815	0.4486155



RBF method

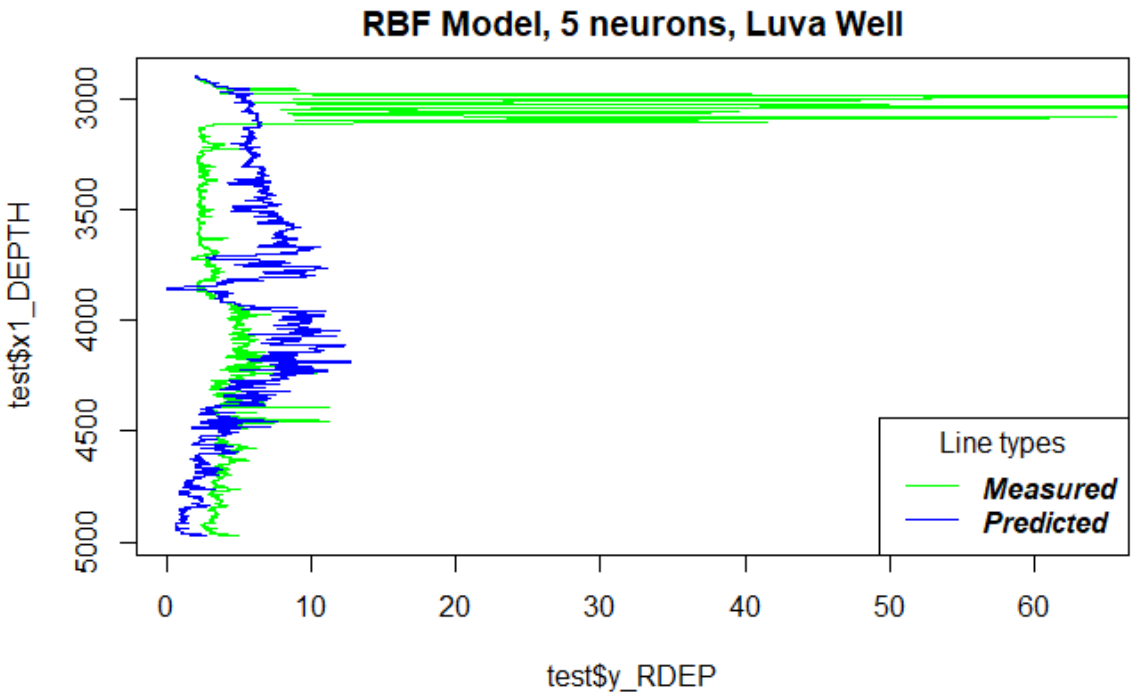
Resampling results across tuning parameters:

size	RMSE	Rsquared	MAE
3	0.864691	0.252157	0.475055
4	0.770022	0.410407	0.462917
5	**0.764172**	**0.419958**	**0.464998**
6	17.16214	0.361762	16.83608
7	2.672902	0.441588	2.33663
8	1.258047	0.443696	0.906179
9	3.852041	0.461635	3.483815
10	0.849561	0.525364	0.602251
11	24.52169	0.469156	24.01562
12	2.39019	0.520057	2.091085
13	23.21404	0.529794	22.51214
14	24.48174	0.540682	24.22317
15	0.858019	0.534428	0.58765
16	1.453262	0.538908	1.173941
17	2.691447	0.537173	2.278244
18	0.84525	0.532428	0.574785
19	8.113781	0.531613	7.767676
20	1.501288	0.532349	1.10317

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was size = 5.

bestTune predictions:

	RMSE	Rsquared	MAE
luva	9.740555472	0.002203734	4.362959174
haklang	31.4667409	0.1286855	17.5519486
vema	1.1219205	0.6951284	0.8222305


RBF Model, 5 neurons, Luva Well
test$x1_DEPTH
3000
3500
4000
4500
5000
0
10
20
30
40
50
60
test$y_RDEP
Line types
Measured
Predicted