TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

Department of Informatics

Eduard Netšajev

Motor Insurance Clients Risk Level Evaluation using Artificial Neural Networks and Deep Learning

Bachelor's Thesis

Supervisors: Eduard Petlenkov, PhD
Department of Computer Control at TUT

Aleksei Tepljakov, PhD
Department of Computer Control at TUT

Kristina Vassiljeva, PhD
Department of Computer Control at TUT

Tallinn 2016

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Informaatikainstituut

Eduard Netšajev

Liikluskindlustuse Klientide Riskitaseme Hindamine Tehisnärvivõrkude ja Sügava Õppimise abil

Bakalaureusetöö

Juhendajad: Eduard Petlenkov, PhD
Automaatikainstituut

Aleksei Tepljakov, PhD
Automaatikainstituut

Kristina Vassiljeva, PhD
Automaatikainstituut

Tallinn 2016

Author’s declaration of originality

Declaration: I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Eduard Netšajev

May 28, 2016

Abstract

Motor insurance clients risk level evaluation using Artificial Neural Networks and Deep Learning

Risk calculation is a fundamental part of the insurance industry. Previous research showed that the main problem of the currently used methods lies in ineffective use of available data.

The main objective of this work is to evaluate usage of artificial neural networks for estimating accident probability of motor insurance clients. Along with a grid search of the hyperparameters for the best performing neural network a comparison of different data preprocessing techniques is presented.

The results show that the use of artificial neural networks in the insurance industry is fully justified and should not be overlooked.

This thesis is written in English and is 46 pages long, including 5 chapters, 16 figures and 7 tables.

Annotatsioon

Liikluskindlustuse klientide riskitaseme hindamine Tehisnärvivõrkude ja Sügava Õppimise abil

Riski hindamine on põhiline osa kindlustussektorist. Varasemad uuringud on näidanud, et tänapäeval kasutuselevõetud meetodite peamiseks probleemiks on ebaefektiivne olemasolevate andmete kasutamine.

Käesoleva töö peamine eesmärk on hinnata tehisnärvivõrkude kasutamist liikluskindlustuse klientide liiklusõnnetuse tõenäosuse arvutamiseks. Kirjeldatud nii töö tähtsuse õigustus kui ka üldine tehisnärvivõrkude tööprintsiip ja erinevate koostisosade eksisteerimine. Töös on võrreldud erinevad tehisnärvivõrkude parameetrid ning andmete eeltöötluse meetodid. Eesmärgiks on saada ennustamissüsteem, mis kindlustuslepingu andmete sisestamisel väljastaks liiklusavarii tõenäosust selle lepingu jooksul. Lõplikud tulemused on testitud eraldiseisva andmete kogumi põhjal. Uuringute protsessis saadud tehisnärvivõrk on valmis teiste süsteemitega integreerimiseks.

Tulemused näitavad, et tehisnärvivõrkude kasutamine kindlustussektoris on igati õigustatud.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 46 leheküljel, 5 peatükki, 16 joonist, 7 tabelit.

List of abbreviations and Terms

ANN Artificial Neural Network

MLP Multilayer Perceptron

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement and Objectives

Risk calculation is at the heart of the insurance industry. Auto insurance companies set their rate depending on how likely a client is to file a claim in the future, and each use a number of different factors to evaluate that risk. Starting from the first time compulsory car insurance policy was introduced in 1930 [4] it became a costly obligation for millions of car drivers.

Using a constant premium to charge all clients from an insurance company's point of view is a good method to alienate their most important customer base — safe drivers who are less likely to cause an accident in the future. Such drivers will most likely choose some other company that recognizes their good history of driving by offering a smaller insurance cost. It is essential for an insurance company to have a good method of discrimination among different insurance buyers in order to be successful in the industry.

In the insurance business, the two most important factors are considered when analysing losses: probability of loss and severity of loss. Because severity of loss is not available in the source data acquired from the thesis supervisors, the focus of this work lays solely on the probability of loss.

The main objective of this work is to find an effective way of evaluating motor insurance clients risk level using Artificial Neural Networks (ANNs). Output of the final ANN should be easy to interpret for a non-expert user. Results of the work have to be evaluated in a trustworthy way, reproducible and ready for export in order to integrate in a more user-friendly system or for any other purposes.

In ideal (but impossible) scenario the insurance company could confidently predict whether an accident is going to happen with a 100% accuracy. If it is known that an accident is going to happen then it should refuse to underwrite such client. On the contrary, if it is known that no losses will be caused by this client, the insurance company should offer him the lowest price compared with offers from other insurance companies.

1.2 Current Solutions and Limitations

Conventional rating systems are primarily based on past realized losses and the past record of other drivers with similar characteristics. These include information about both vehicle (age, manufacturer, model, value) and driver (age, sex, marital status, driving record, place of residence, etc.). Although machine learning methods do not offer any groundbreaking methods in terms of gathering new data it has been studied that the main problem of current methods lies in ineffective use of available data [5] resulting in a very poor precision attributable to variables with stronger predictive power being completely overlooked. Artificial neural networks should be able to achieve better predicting performance thanks to their lack of bias and greater flexibility that allows them to use all the data available [6].

There are several limiting factors that hinder our ability to estimate probability of an insurance claim by any particular client. Firstly, uncertainty is a big factor in our everyday life that affects accidents' occurrence and makes them hard to foresee. Secondly, the source data available for this study is far from being perfect: less than a couple of dozens variables for each data point with questionable predictive power (measure of relevance to risk level). Finally, although the goal of this work is to calculate a risk level measure in real numbers — as the ideal scenario described above is just impossible — the source data contains information indicating only whether any accidents occurred as integer number. From a machine learning perspective it would be easier if instead of "0"/"1" value each data point had a human expert given risk estimation with a goal to train an automated system matching this expert.

Chapter 2

Theoretical Background

The field of Artificial Neural Networks and Machine Learning heavily relies on such fields as Linear Algebra, Multivariate Calculus and Probability Theory to name a few. Although in-depth explanation of underlying theory is out of scope of this work, in this chapter we shortly explain the basic theory behind ANNs with an aim for a reader not familiar with the field.

2.1 Brief Overview of Artificial Neural Networks

The human brain is a very powerful and complex system which is able to think, remember and solve different problems. No surprise that scientists see the path to general AI through simulation of its structure. Although it is indisputable that artificial neural networks are one of the most promising areas of artificial intelligence research, with average human brain consisting of approximately 86 billion neurons [7] current ANNs are still far away from reaching our brain's capabilities.

The inventor of the first neurocomputer, Dr. Robert Hecht-Nielsen, defines an artificial neural network as "...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs." [8]. In biology terms, a neuron is a cell that can transmit and process chemical or electrical signals. Figure 2.1 below illustrates the similarities between a biological neuron's structure and an artificial neuron's structure.

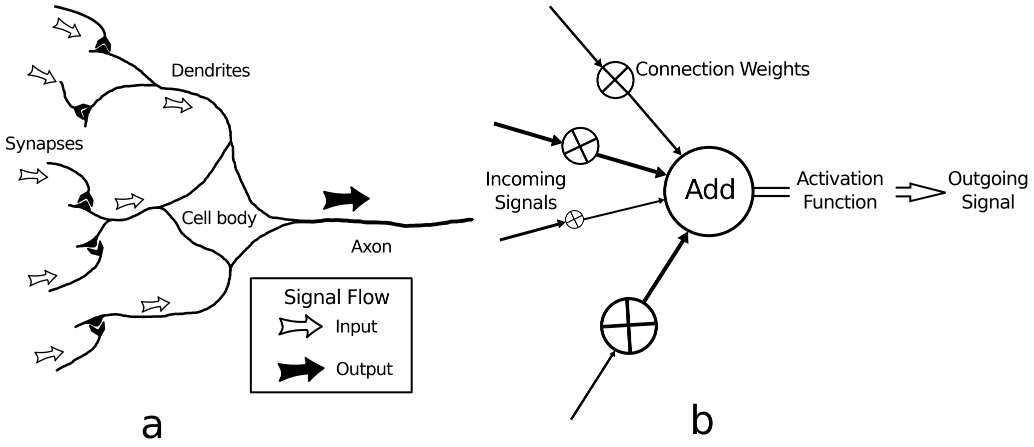


Figure 2.1: (a) A Biological Neuron, and (b) An Artificial Neuron, from [1]

To put it more formally, each artificial neuron has n inputs $x_1 ... x_n$ forming input vector X. Each input is multiplied by its corresponding synaptic weight value $w_1 ... w_n$ forming n products $x_1 w_1 ... x_n w_n$. Each neuron also has a randomly initialized bias parameter b which is added to the sum of these products. The bias value is usually viewed as a part of a weight matrix W with each layer (except for the output layer) having an additional constant unit input. Neuron's output y is calculated using this weighted sum as an input to an activation function $f : \Re \rightarrow \Re$ which is used to transform the activation level of a neuron into an output signal.

$$y = f\big(b + W \times X\big) = f\left(b + [w_1 ... w_n] \times \begin{bmatrix} x_1 \\ . \\ . \\ . \\ x_n \end{bmatrix}\right) = f\left(b + \sum_{i=1}^{n} w_i x_i\right). \quad (2.1)$$

2.2 Activation Functions

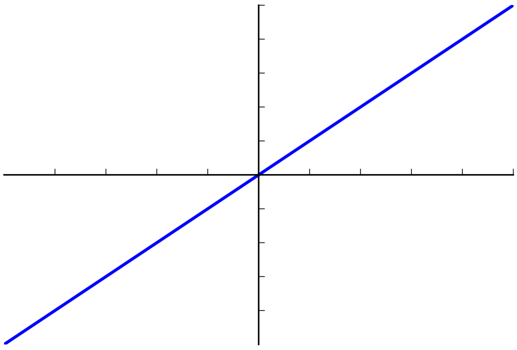
There are several common activation functions in use with artificial neural networks. Some desirable properties in an activation function include:

- Non-linear (meaning function cannot be expressed in form $y = a + bx$): neural network with a single hidden layer can be proven to be a universal function approximator (i.e. it can approximate any function given enough neurons in the hidden layer) if the activation function is non-linear [9].
- Continuously differentiable: necessary property for enabling gradient-based optimization methods [10] (learning method used in this work is gradient-based).

- Monotonic: with this condition the error surface associated with a Single Layer Perceptron model (i.e. no hidden layers) is guaranteed to be convex [11] (has no more than one minimum).

Selection of an activation function for a neural network or its part is an important task because different activation functions result in different training outcomes [12]. Listed below are the five main activation functions that were used during this work:

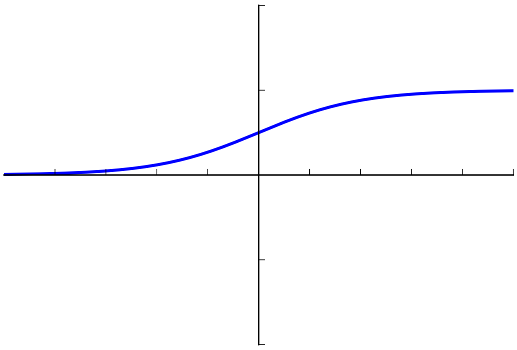
1) Identity function. Always returns the same value that was used as its argument.



$$f(x) = x. \tag{2.2}$$

Figure 2.2: Identity function

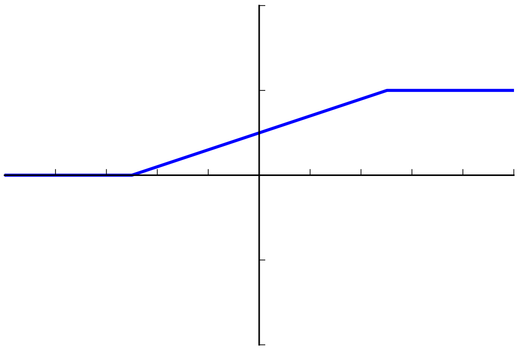
2) Standard logistic sigmoid function. Often called the sigmoid function. Historically, a common choice of activation function since it takes a real-valued input and squashes it to a range between 0 and 1. Thanks to this property it has a nice interpretation as the firing rate of a biological neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1).



$$f(x) = \frac{1}{1 + \exp(-x)}. \tag{2.3}$$

Figure 2.3: Standard logistic sigmoid function

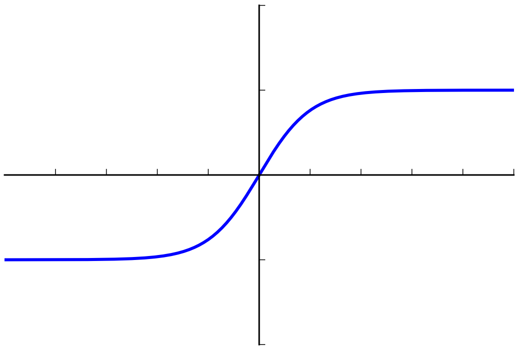
3) Hard sigmoid function. Approximated standard logistic sigmoid used for faster training by excluding computation of the exponential function which is computationally expensive.



$$f(x) = \max(0, \min(1, 0.2x + 0.5)). \quad (2.4)$$

Figure 2.4: Hard sigmoid function, per [2]

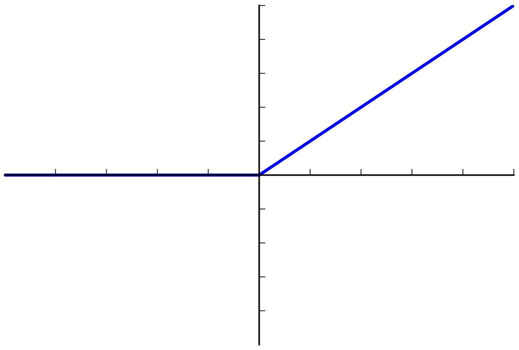
4) Hyperbolic tangent function. Although it is just a scaled and shifted version of the standard logistic sigmoid it has better properties for optimization (learning) purposes resulting in a faster training [13].



$$f(x) = \tanh(x) = \frac{2}{1 + \exp(-2x)} - 1. \quad (2.5)$$

Figure 2.5: Hyperbolic tangent function

5) Rectifier function. Also known as a ramp function or positive linear transfer function. Recent research has found that this activation function often works better in practice for deep neural networks greatly accelerating the learning process (e.g. by a factor of 6 in [14]).



$$f(x) = \max(0, x). \quad (2.6)$$

Figure 2.6: Rectifier function

We also use two more practical variations of the aforementioned activation functions:

1) Scaled hyperbolic tangent function.

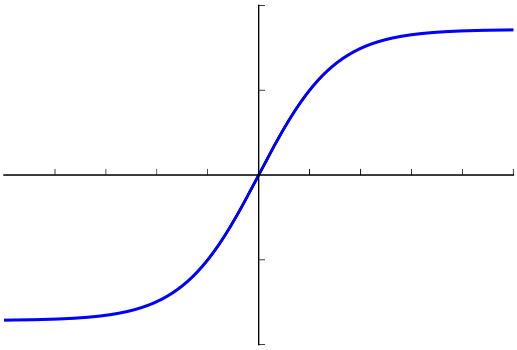


Figure 2.7: Scaled hyperbolic tangent function

$$f(x) = A\tanh(Sx), \tag{2.7}$$

where A is the amplitude and S determines function's slope at the origin. For our work we use recommended values $A = 1.7159$ and $S = \frac{2}{3}$ [15]. With these parameters equalities $f(1) = 1$ and $f(-1) = -1$ are satisfied which makes the segment between these two points almost linear thus theoretically accelerating the training process.

2) Leaky rectifier function. Allows a small, non-zero gradient which could facilitate ANN's learning.

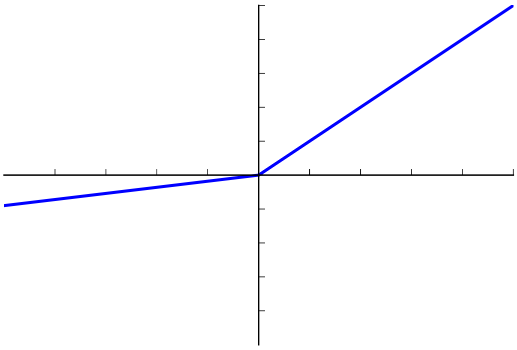


Figure 2.8: Leaky rectifier function

$$f(x) = \begin{cases} x & if\ x \geq 0 \\ ax & if\ x < 0 \end{cases}, \tag{2.8}$$

where a is a small constant. Some recent reports indicate that larger a yields better results so we experiment with $a = 0.18$ [16].

It can be observed that all activation functions mentioned in this section can be divided into two distinct groups based on whether they have a finite range of output.

2.3 Neural Network Architecture

A neural network is put together by hooking together many of neuron units so that the output of one neuron can be the input of another. Although there exist many different types of ANNs' architectures (meaning patterns of connectivity between neurons), only Multilayer Perceptron models (Figure 2.9) were studied in this work.

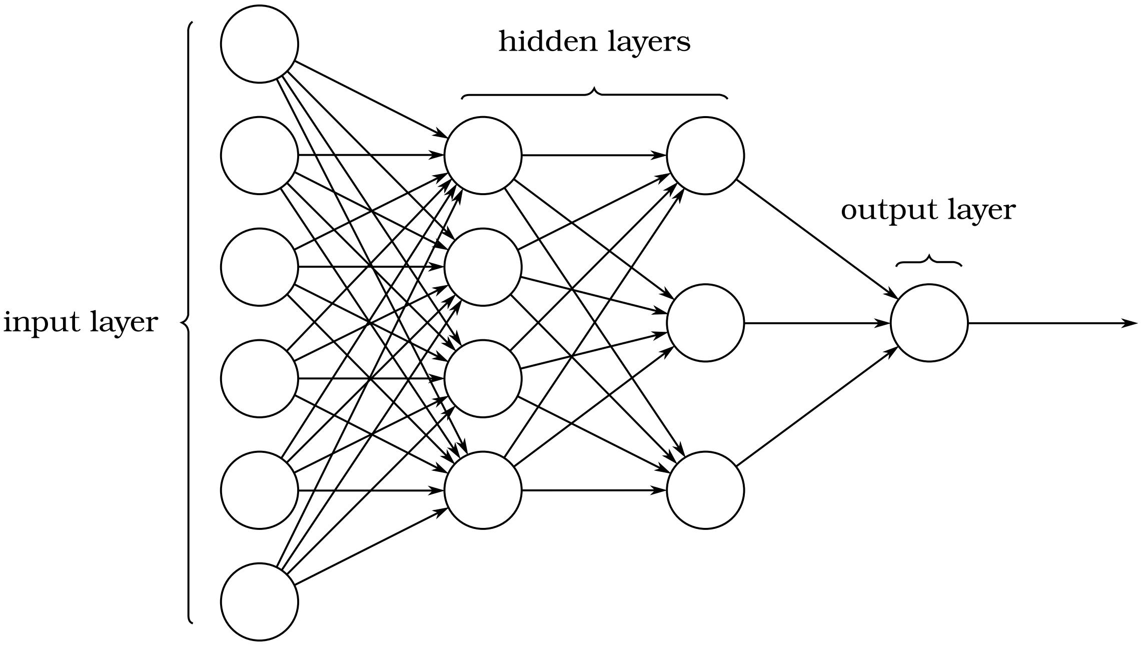


Figure 2.9: Structure of a multilayer perceptron (MLP)

The Multilayer Perceptron model consists of 3 consecutive parts: input layer, hidden layer(s) and output layer. All adjacent layers are fully connected between each other — every neuron from one layer has a connection with every neuron from the other layer, as shown in Figure 2.9. There is no connection between any non-adjacent layers. To calculate the output signal of the network we first need to calculate the output signal of the first hidden layer which is done by calculating output of each neuron in this layer using its weights vector (Equation 2.1). We then use this layer as an input to the next layer and repeat this process until we reach the final layer which outputs the target values.

We have observed that for a fixed MLP network the output signal depends entirely on ANN's weight matrix W and the input provided, i.e. $y = f(W, X)$. If we are not satisfied with its performance the only way of improving it is by adjusting the weights in the network as input is typically given from the outside and considered to be fixed. Given this we can see there are two things are still missing: a function to evaluate ANN's current performance (i.e. feedback) and a process to optimize ANN's performance with regard to that function.

2.4 Loss Functions

The task of evaluating ANN is accomplished by a loss function which is used to evaluate the performance of the neural network. Given a prediction $\hat{y}$ and a known correct value y, a loss function measures the discrepancy between the networks's prediction and the desired output. We can see the loss as the average amount of errors in our predictions which is the reason loss functions are often called error or cost functions.

When the target is to predict whether some event is going to occur it is a standard practice to use a cross-entropy loss function which is given by

$$CE = -\frac{1}{N} \sum_{n=1}^{N} \left[y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n) \right], \tag{2.9}$$

where N is the number of forecast-verification pairs. The cross entropy function is proven to accelerate the learning process and to provide good overall network performance with relatively short stagnation periods [17].

Although the cross-entropy error is preferred when we deal with classification problem (will an accident occur or not), the goal of this thesis is to evaluate risk level using a continuous scale, i.e. we are solving a probabilistic classification problem. The Brier Score was developed by Brier [18] as a means of assessing the relative accuracy of probabilistic forecasts. It is given form:

$$BS = \frac{1}{N} \sum_{n=1}^{N} (\hat{y}_n - y_n)^2. \tag{2.10}$$

The Brier Score values fall in range from 0 to 1 where lower value denotes better forecaster which means we can also use it as a loss function. It can be decomposed into 3 additive components: Reliability, Resolution, and Uncertainty [19]:

$$BS = REL - RES + UNC = \frac{1}{N} \sum_{k=1}^{K} n_k \big(\hat{y}_k - \bar{y_k}\big)^2 - \frac{1}{N} \sum_{k=1}^{K} n_k \big(\bar{y_k} - \bar{y}\big)^2 + \bar{y}\big(1 - \bar{y}\big), \tag{2.11}$$

where the data set is partitioned into K bins according to the forecast. $\bar{y_k}$ is the observed frequency of occurrence of the event in bin k and $\bar{y}$ is the base rate for the event to occur. The reliability term measures how close the forecast probabilities are to the true probabilities. The resolution term measures the ability of the forecast to distinguish situations with distinctly different frequencies of occurrence. The uncertainty term represents the variability of the observations which depends entirely on the data set. The last property is the reason why it is misleading to compare Brier scores on different data sets.

2.5 Learning Process

The purpose of learning is to find the optimal solution to a weight matrix W so that no solution has a lower loss, i.e. to minimize the loss function $L(W)$. The algorithm performed to calculate MLP's output given some input by pushing it through the network is called forward propagation. It leads to the generation of an output prediction which in our case is a single real number.

Backpropagation is the most widely used iterative algorithm to adjust neural network's parameters. Basically it takes the output of the neural network $\hat{y}$, compares it to the desired value y and calculates how much wrong the output neuron was. Using the calculated error it calculates the error associated with each neuron from the preceding layer. This process is then repeated until the input layer is reached.

Next we use a gradient descent optimization algorithm to minimize the loss function's value. Using the error measurements for each neuron the algorithm calculates the partial derivatives (also called gradients) — vectors pointing in the direction of the greatest increase rate of the loss function. To find a local minimum of a function using gradient descent the weight matrix is adjusted proportionally to the negative of the gradient of the function at the current point.

2.6 Overfitting and Underfitting

A major issue in choosing a neural network model for a given problem is selecting the level of its complexity which determines model fit. Understanding model fit is important for understanding the root cause for poor model accuracy. We can determine whether a neural network is underfitting or overfitting the training data by looking at the prediction error on the training data and the test data. Simple visual explanation of model fit is presented in Figure 2.10.

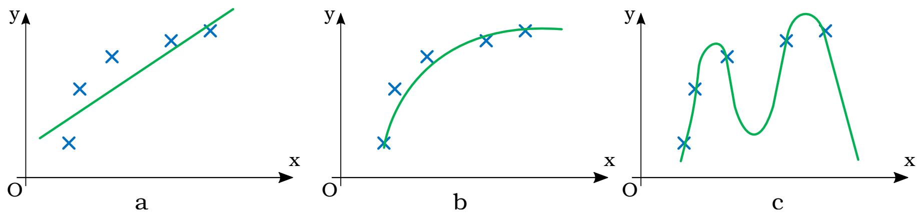


Figure 2.10: (a) Underfitting, (b) Good fit, and (c) Overfitting

The network is underfitting the training data when the model performs poorly on both the training data and the test data. This is because the model is unable to capture the relationship between the input X and the target Y. The model is overfitting the training data when the model performs well on the training data but does not perform well on the test data. This is because the model is memorizing the data it has seen and is unable to generalize to unseen examples.

Poor performance on the training data could be attributable to the neural network being too simple and its performance can be improved by increasing model's complexity and increasing the training time. If the model tends to overfit several good countermeasures exist as well: stopping the training early (before the model learned the training data too well), reducing model's complexity, keeping some neurons turned off during each training stage, and adding some form of regularization.

Regularization prevents any single weight from being given too positive or too negative value thus encouraging smoother network mappings. The strength of regularization can be finely tuned. More regularization means lower fit while less regularization means higher fit. If regularization is too strong the model will underfit and if the regularization is too weak the model is more likely to overfit.

2.7 Deep Learning

Deep Learning is a machine learning technique that tries to extract high-level abstract data representations through hierarchically combining simple features into more complex features layer by layer [20]. In context of Artificial Neural Networks deep learning refers to the usage of neural networks with more than 1 hidden layer. As opposed to shallow networks which have only one hidden layer deep neural networks perform far better on many problems, especially on problems such visual pattern recognition [21], [22].

The main problems experienced with deep architectures is that they are harder/longer to train and are more prone to stuck in local minima of a loss function. Although several different techniques exists to overcome this problem in this work we use the stacked autoencoders method which is a 2-phase process (Figure 2.11).

First we greedily train each layer to reconstruct its input with a restriction that the number of inputs to every layer is larger than the number of its neurons. This way each layer is intended to learn an increasingly abstract feature representation of the input. We also pretrain the output layer to estimate accident probability. In the second phase (fine-tuning) the whole network is trained to minimize the prediction error.

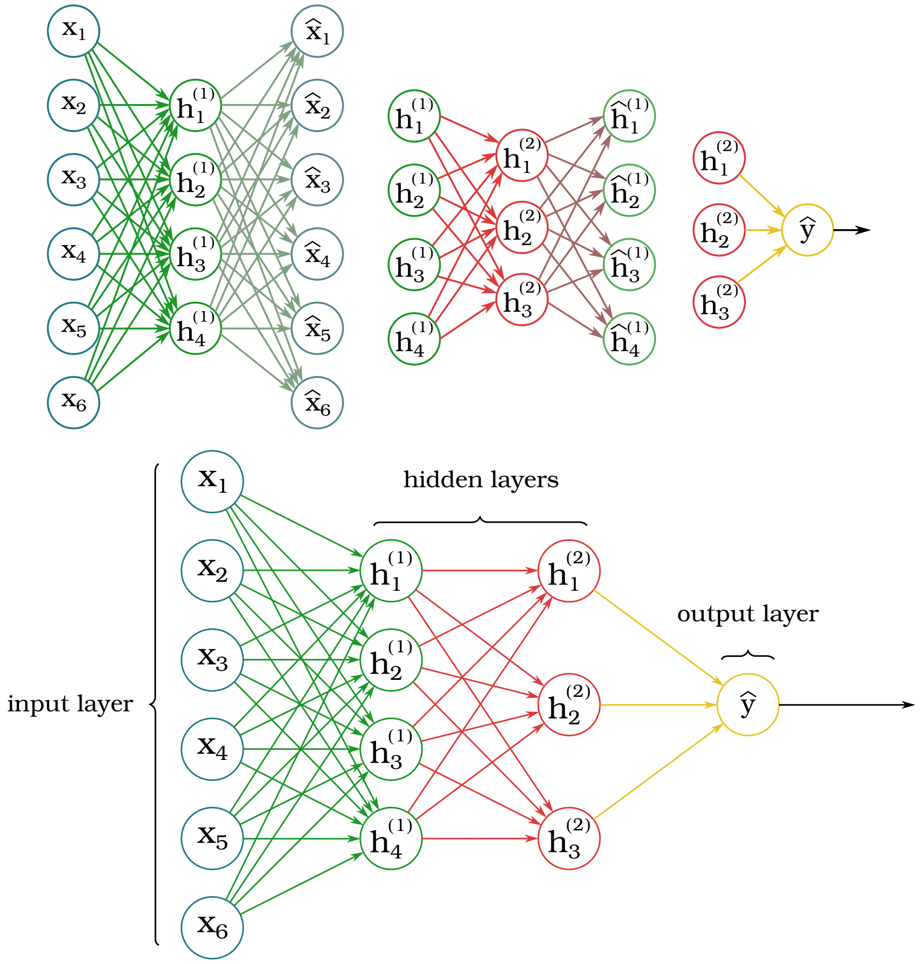

x_1 x_2 x_3 x_4 x_5 x_6
$h_1^{(1)}$ $h_2^{(1)}$ $h_3^{(1)}$ $h_4^{(1)}$
$\hat{x}_1$ $\hat{x}_2$ $\hat{x}_3$ $\hat{x}_4$ $\hat{x}_5$ $\hat{x}_6$
$h_1^{(2)}$ $h_2^{(2)}$ $h_3^{(2)}$
$\hat{h}_1^{(1)}$ $\hat{h}_2^{(1)}$ $\hat{h}_3^{(1)}$ $\hat{h}_4^{(1)}$
$\hat{y}$
hidden layers
output layer
input layer

Figure 2.11: Stacked Autoencoders technique. Pretraining and fine-tuning phases.

Initializing ANN's weights in this manner has been shown to decrease training times and start the training process much closer to good loss function minima than the usual random weight initialization.

Chapter 3

Tools

Focus of this chapter lies on the tools used to store/process data and conduct experiments with different artificial neural network models.

3.1 Programming Language

The two most popular languages in the field of data science and machine learning at the moment of writing this paper are R and Python. Python is a general-purpose, high-level programming language that emphasizes code readability and expressing concepts in fewer lines of code than is possible in languages such as C++ or Java. R is a programming language and software environment for statistical computing and data visualization, which Python also supports. Each of the two languages is stronger than the other in some aspect [23].

For this paper the only programming language used was Python. This decision was made mainly because of author's familiarity and previous experience with Python. It is worth mentioning that current job trends (Figure 3.1) indicate that both currently and in the future Python's presence in the field is stronger. Although popularity does not necessarily reflect quality, having a vibrant community that works on further advance of the language is a major key.

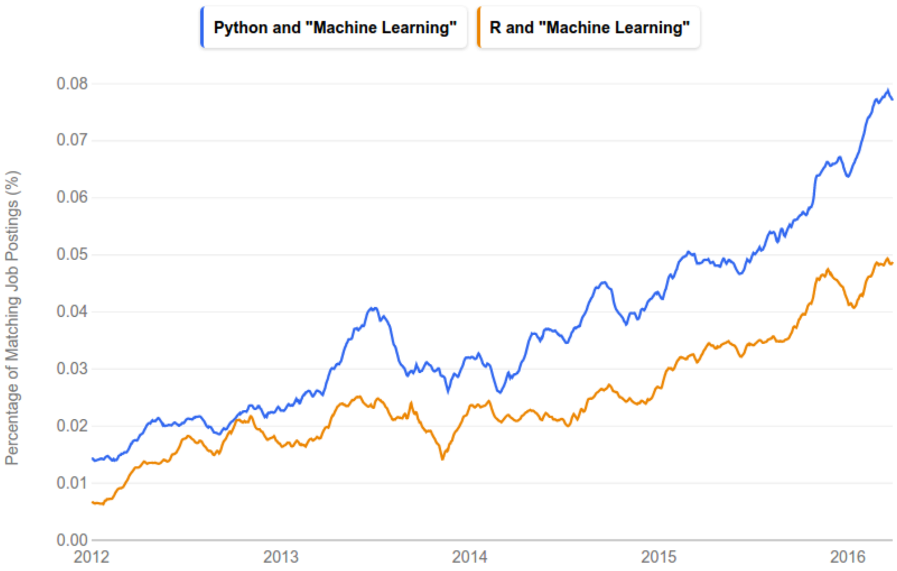


Figure 3.1: Job trends for R and Python according to [3]

3.2 Libraries

Owing to Python being a widely used general-purpose programming language [24] there is an enormous amount of different software packages available. As of May 2016, the Python Package Index, the official repository of third-party software for Python, contains almost 82,000 packages offering a wide range of functionality [25].

The fundamental open source package for numerical computation in Python is *NumPy*, which defines the multidimensional numerical array and matrix types and basic operations on them.

scikit-learn is a popular open source machine learning library for Python. The package contains various machine learning and data analysis algorithms and designed to interoperate with NumPy. This library was used to analyse performance of our neural network models.

For interactive experimenting we used *IPython*, a rich interactive interface for processing of data and quick testing of ideas. The IPython Notebook is a web-based interactive environment for creating IPython notebooks. An IPython notebook is a JSON document containing an ordered list of input/output cells which can contain code, text, mathematics, plots and rich media.

Matplotlib is probably the most popular open source 2D plotting library for Python. It can be used in both python scripts and IPython Notebook environment. We also used vector graphics editor *Inkscape* to create or modify all the graphic content presented in this work.

Keras is a young minimalist, highly modular neural networks library for Python which was developed with a focus on enabling fast experimentation. It also allows persisting of neural network models and their parameters in a simple format in order to use them on another machine without any training. Keras is built to work on top of one of the supported backend machine learning libraries. In this work we used *Theano* library backend. Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multidimensional arrays efficiently. One of the main features of Theano is its ability to perform efficient symbolic differentiation: the user composes mathematical expressions in a high-level description which allow Theano to provide symbolic differentiation.

Lastly, we introduce a library used in this work for saving, storing and loading of all data used for the purpose of training and testing ANN models. The *h5py* package is a Pythonic interface to the HDF5 binary data format [26]. HDF5 stands for Hierarchical Data Format and supports an unlimited variety of datatypes. It was designed for flexible and efficient I/O and for high volume and complex data. Originally developed at the National Center for Supercomputing Applications, it is supported by The HDF Group [27], a non-profit corporation working to ensure continued development of HDF5 technologies and the continued accessibility of data stored in HDF.

Chapter 4

Methods

There are several steps that need to be performed before the network is trained. They can be grouped into three categories: selection of data, data preprocessing, and choice of network type and architecture. We also need to pick several training parameters before we can start the process: loss function, number of maximum epochs to perform, optimization algorithm and its parameters. Finally, a method to evaluate the results of training an ANN is required in order to compare neural networks between each other and pick the best of them.

4.1 Data Selection

Neural networks represent a technology that is at the mercy of the data — the network will be only as good as the data that is used to train it. For a source data we have a MATLAB file that contains different information about ~540 000 insurance contracts from the last five years. Only the parameters listed next were deemed to be relevant and helpful in order to achieve this work's goals: duration of the contract, vehicle's engine power, age of the driver, age of the vehicle, loyalty rating given by the insurance company, vehicle's category, vehicle's type of usage, region of driver's residence, mass of the vehicle, manufacturer of the vehicle, model of the vehicle, fuel type used by the vehicle, whether it was an extension of a previous policy or a new client, bonus-malus rating of the driver, gender of the driver and whether any accident was caused by the driver during this insurance policy.

Information that was excluded from consideration include insurance contract version, policyholder's national identification number, vehicle registration plate, accident dates and similar irrelevant information. It is worth mentioning that risk class given by the insurance company was excluded as well because the main reason for this project was to create an independent risk evaluation system, not an aide to the existing one.

4.2 Data Preprocessing

The main purpose of the data preprocessing step is to assist network training. Data preprocessing consists of such steps as normalization, non-linear transformations, feature extraction, coding of discrete inputs/targets, handling of missing data, etc. The idea is to perform preliminary processing of the data to make it easier for the neural network to extract the relevant information. After collecting the data, we generally divide it into three sets: training, validation (selecting best model) and testing (evaluating final results). In this work the training set makes up 80% of the full data set, with validation and testing making up 10% each. Although it is very important that each of these sets represents the full data set — that the validation and test sets cover the same region of the input space as the training set — it is a rather difficult goal to achieve. As a simple good-enough method all data was randomly shuffled before splitting it into the sets. Additionally, each neural network model is trained four times using different portion of the training set: 20%, 50%, 80% and 100% of the training data.

Often the data on which a multilayer perceptron (MLP) is to be trained has inputs that are categorical rather than quantitative. Examples of categorical variables in our data are gender, model of the vehicle, usage type, etc. Categorical variables may be broken up into two types: ordinal and non-ordinal. Ordinal variable values may be ordered. A non-ordinal categorical variable will be a label of a category without ordering property as is true in the case of the examples listed above where no ordering can be placed. A quantitative variable on the other hand is one that assumes numerical values corresponding to the points on a real line, e.g. mass of the vehicle or age of the driver.

A MLP requires that all inputs are represented in numeric, continuously valued variables. One method of dealing with the problem of categorical inputs, which seems rather harmful, is to replace the categorical values by numeric ones and treat them as if they were continuously valued. This type of encoding creates an ordering of the values that does not exist, there is no meaningful relation between the continuous quantities generated this way and the original categorical values. Therefore, categorical variables should be distinguished from the continuous variables.

While it is worth mentioning that there exist rather promising but quite difficult methods for dealing with categorical variables [28] in this work we use simultaneously two quite simple methods of dealing with problem of categorical variables. The first technique is 1-of-C encoding: one input variable gets transformed into C variables where C is the number of feature's possible values. In 1-of-C encoding we set only one variable to value 1 and all other variables are set to 0. For example, transforming gender variable {male, female} we get two input variables where male can be represented by a pair (0, 1) and female by a pair (1, 0).

The second used technique is to encode categorical features using average response [29]. Explained shortly, we calculate the average accident rate for every given category (e.g. for every vehicle fuel type) and add this variable to our input vector. For the purposes of better performance the training set is assigned "all but one" average response, i.e. for each training data point the mean response is calculated excluding this particular data point itself. Furthermore, we also add a shifted version of this variable where we subtract the overall accident rate from the mean response in order to center data around zero point. It is worth to emphasize that all statistics (averages) were calculated using only the training data.

SPLIT	FUEL TYPE	ACCIDENT	ACCIDENT RATE
Training	Diesel	1	0.333
Training	Diesel	0	0.666
Training	Diesel	1	0.333
Training	Diesel	0	0.666
Validation	Diesel	—	0.500
Test	Diesel	—	0.500

Table 4.1: Illustrative example of the average response encoding

It is a standard practice to normalize the quantitative data before using it in an ANN. This way initializing the network weights to small random values ensures that the weight-input product will be small thus avoiding the saturation zone of an activation function. Saturation at the asymptotes of an activation function is a common problem with neural networks. Saturation occurs if the asymptote(s) of a function are parallel to the input axis, meaning that the first derivative is (or almost) 0. In this case the neuron can neither learn nor propagate the error any further. Additionally, when the input values are normalized, the magnitudes of the weights have a consistent meaning which further facilitates the training process.

There are two standard methods for normalization [30], both were used in this work. The first method called rescaling normalizes the data so that all feature's variables fall into a standard range — typically -1 to 1. This can be done with a simple transformation (4.1) where x_{min} and x_{max} are respectively the minimum and the maximum values of each element of a particular feature in the data set, and x_n is the resulting normalized input variable.

$$x_n = \frac{2 \cdot (x - x_{min})}{x_{max} - x_{min}} - 1. \tag{4.1}$$

An alternate normalization procedure called standardization is to adjust the data so that it has a specified mean and variance — typically 0 and 1. This can be done with another transformation (4.2) where x_{mean} is the average value of the feature in the data set, and x_{std} is the standard deviation of the feature.

$$x_n = \frac{x - x_{mean}}{x_{std}}. \tag{4.2}$$

Additionally, a third rather simple transformation (4.3) was used in order to rescale data so that it falls only in the range from 0 to 1. This can be done with a simple division by a constant value of a maximum value. This method's benefit compared with the previous ones is that it does not shift any values and as a result we avoid losing any information.

$$x_n = \frac{x}{x_{max}}. \tag{4.3}$$

The result of the methods presented above applied to the source data is a data set which contains 135 input features (not counting the target variable denoting whether an accident has occurred). Additionally, we have 2 simpler data sets in order to measure efficiency of the preprocessing techniques applied. In the first variation we omit the 1-of-C encoding which results in 100 features less in the input data (35 remaining).

The third data set is created using the most commonly used method which is done in two steps. First categorical data is simply transformed to consecutive integer numbers without any order. After that all features (both categorical and quantitative) are rescaled using transformation (4.4) which normalizes the data so that all variables fall into a range between 0 and 1. This data set does not contain any statistical information which is a big reason to favour such ANN which does not rely on externally provided statistics.

$$x_n = \frac{x - x_{min}}{x_{max} - x_{min}}. \tag{4.4}$$

4.3 Architecture Selection

The next crucial step in the ANN training process is the choice of a network architecture. The basic type of network architecture is determined by the kind of problem we are trying to solve. Once the basic architecture is chosen, we need to decide such specific details as how many outputs the network should have, what type of output we want to receive and how many neurons and layers we want the neural network to have.

As was mentioned previously (Section 2.3) the only type of ANN architecture used in this work was MLP. The first reason for that is purely practical one: this structure is one of the simplest to work with. Furthermore, it can be argued that in our case more complex architecture types will not bring any improvements to ANN's performance. This is explained by having inputs as independent values (contrary to visual data) without any particular ordering and lack of any connection among different data points (contrary to time series data). The purpose of our neural network is to represent a function approximating the mapping between a set of inputs and a corresponding real value representing the accident probability. For this and similar problems MLP is the standard neural network architecture.

The output of our ANN must be a single number ideally defining an estimated probability of an accident for a particular client. Since this value has to be in a standard range 0 to 1 (1 denoting 100% accident probability) we will use the standard logistic sigmoid function for a single output neuron. Other variants of output layer were taken into consideration as well, e.g. an output in range between -1 and 1 or even two outputs denoting probability of both outcomes "accident" and "no accident", but were dismissed due to their complex mapping to the objective value of an accident probability.

It is much more difficult to make a single choice in the remaining parts of ANN's architecture: number of hidden layers, number of neurons in each of the hidden layers and an activation function used in each neuron of the hidden layer. There is an infinite amount of possibilities for this choice and little to no indications of which is the best one. To shrink the choice space we will consider activation function choice not per individual neuron or layer but for all hidden layers of the network at once. Next we select 40 different structures representing the hidden layers using our own judgment of what a promising structure could be or at least is worth to try. The full list of the selected structures is presented in Appendix A. The resulting 200 hidden layers structures are achieved by combining these hand-picked structures with each of the 5 main activation functions that we have covered earlier (Section 2.2).

4.4 Evaluating Performance

After the training of a neural network is finished, we need to express its performance in a simple form in order to select the model performing the best. In our perception of "goodness" a good assessor is both accurate and bold in his predictions. The first quality reflects how close to each other are predicted and true probabilities. The second quality is expressed by how well are different cases discerned. It has been proven that both loss functions mentioned in Section 2.4 are applicable as a scoring rules to reflect a summary of these two properties using a single value [31]. Nevertheless, later it has been shown that using the Brier Score yields better results, especially when working with small probabilities and rare classes [32] which is the case in this work. For these reasons the Brier Score is used to rank the neural networks. It is also possible to quantify both of the properties mentioned above using Equation (2.11).

The properties described above can be visualized for easier understanding of ANN's performance. Reliability diagram is a graphical method for assessing reliability, resolution, and sharpness of a probability prediction. First we plot a calibration curve which maps the predicted probability to the observed accident rate, where the range of predicted probabilities is divided into K bins. Several straight lines are plotted as reference: the diagonal (represents perfect reliability), the no-resolution line (corresponds to the mean probability prediction as probability estimation), and the no-skill lines (predictions where the resolution and reliability components cancel each other out). Additionally, we plot prediction frequency histogram to show sharpness (the tendency to predict extreme values).

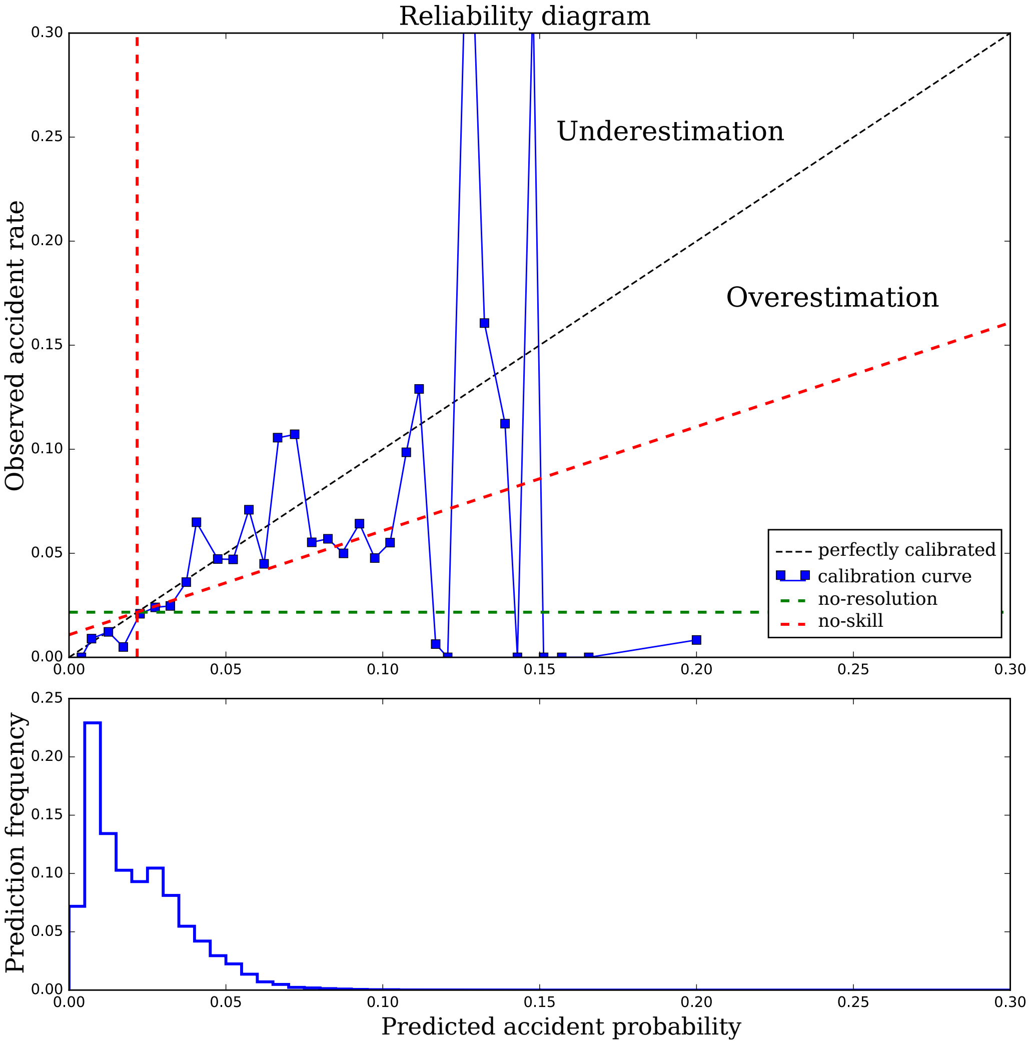


Figure 4.1: Example of a reliability diagram

Now we explain what does the reliability diagram tell us about ANN's performance. Reliability is indicated by the proximity of the plotted calibration curve to the diagonal. Points of the curve below the diagonal indicate overestimation (predicted probabilities are too high) and points of the curve above the diagonal indicate underestimation (predicted probabilities are too low). Points between the no-skill lines contribute positively to the Brier Score. The flatter the curve in the reliability diagram, the less resolution the neural network has. Sharpness is determined by how close our predictions are distributed to the end-points. A predicting system with good sharpness would output accident probabilities close to 0 and 1.

Reliability diagram is a great diagnostic tool to get an insight about performance of a prediction system. The biggest drawback is that for an accurate representation of the metrics described above it is required to provide a reasonably large data set to get useful and accurate results. Even then, if some bins contain only a few of the data points the visual representation gets heavily skewed and misleads the reader.

4.5 Training Parameters

There is a number of challenges connected to training ANN with the most basic types of gradient descent. For instance, the error gradient often applies too large of a change to the weight matrix and jumps over a possible optimum spot. Additionally, when the learning rate (i.e. training parameter that controls the size of weight changes) is too low the changes applied to the weight matrix can be too small resulting in an unnecessary long training process. Adadelta [33] is an algorithm for a gradient-based optimization that dynamically adapts the learning rate over time. The method requires no manual selection of a learning rate and appears robust to different model architecture choices and selection of training parameters, which is why it is the optimization algorithm used in this work. As recommended in Keras documentation [34] we leave the parameters of the optimizer as their default values.

Another important parameter when training ANN model is the number of training cycles i.e. epochs we perform. Epoch is a single pass through the entire training set, followed by testing on the validation set. As ANN learns the training data more and more it tends to overfit. The method used here to select the best parameters of the network is a variation of Early Stopping. The training process uses training data but after each epoch we calculate the loss function's value for the validation data set. Network parameters that result in the smallest validation error are selected for the evaluation process.

The number of epochs we need to perform before the network starts overfitting totally depends on the data used. For our data set it was found empirically that most of the models are starting to overfit after ~30 epochs (example in Figure 4.2). For this reason it was decided that each neural network would be trained for 30 epochs. Additionally, the training process was halted each time it was detected that the network is failing to discern the validation data for 5 epochs in a row which happens mostly with the deeper models when the network's weight matrix was disadvantageously initialized resulting in a strong underfit.

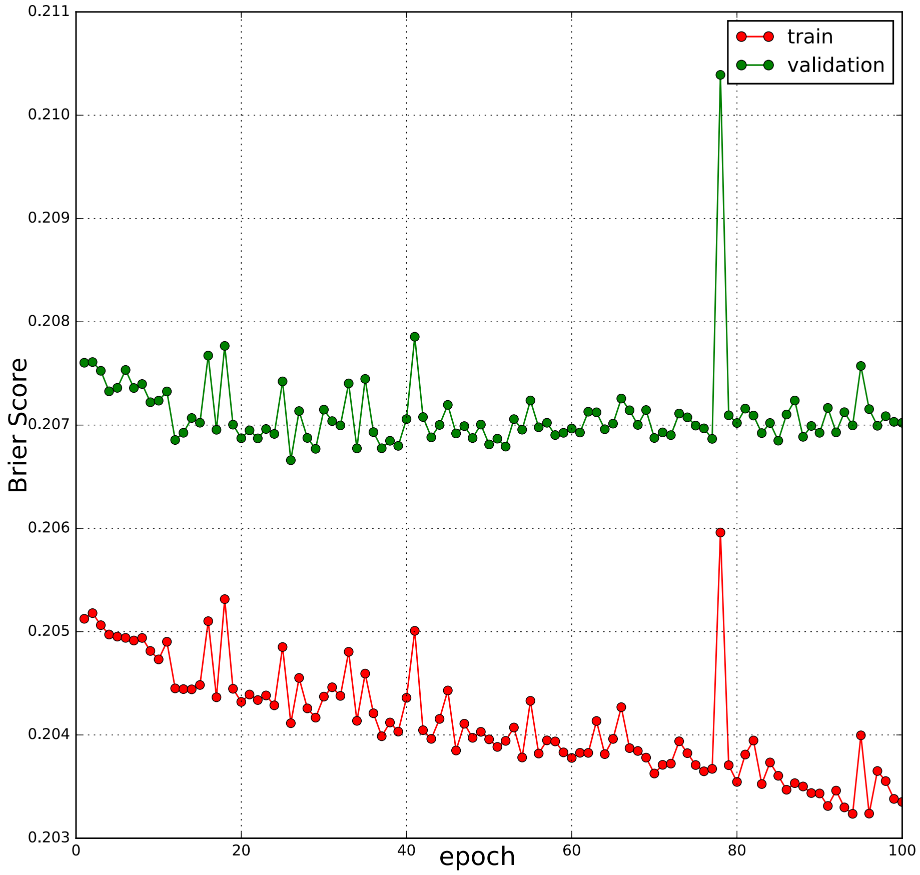


Figure 4.2: Example of ANN's training history, validation minimum at epoch #26

In Section 2.4 we have described two widely used loss functions. As it is not clear which one will yield better result we use both of them, i.e. train each network first with the Brier Score loss function and then using the cross-entropy error function.

4.6 Running Experiments

To summarize, we explicitly define all the variations of experiments that we have conducted in order to find the best artificial neural network to solve our problem. Here is the list of parameters (also called hyperparameters) that we combine with each other:

- 2 loss functions;
- 3 differently preprocessed data sets;
- 4 training set portions;
- 5 activation functions;

- 40 hidden layers structures.

In total using this systematic approach there were performed $40 \cdot 5 \cdot 4 \cdot 3 \cdot 2 = 4800$ different training sessions. This method is called a grid search algorithm and has been a traditional way of performing hyperparameter optimization. At the end of each training session we save neural network's performance on the validation data set. After all ANNs have been trained and validated we select the model with the best results on the validation data set as our final neural network which we evaluate using the testing data set.

Chapter 5

Analysis of Results

Considering the large amount of different components combined in order to train the 4800 different neural networks there is a lot of data to learn from. In this chapter we look at each training parameter variations on a one-by-one basis to find out which options outperform their alternatives and finally which combination resulted in the most accurate neural network. We use results on the validation data set to analyse performance of the training parameters keeping the test data for evaluation and more detailed analysis of the best performing model.

5.1 Comparison of Different Training Parameters

In order to avoid taking poorly performing models into calculation we use the 10 best models in each category to calculate the average performance. For example for comparison of the Brier score loss function with the cross-entropy error function we take the average of the 10 best models that use the former and the average of the 10 best models that use the latter. The complete table of results in a numerical form is provided in Appendix B.

Benefits of more complex preprocessing

From a standpoint of realization and later usage as a commercial product it is desirable that the data requires as less complex preprocessing steps as possible. In our work we used 3 different methods of preprocessing described in Section 4.2. The results of our tests undoubtedly confirm that in our case the most complex form of preprocessing yields better results while the simplest preprocessing method achieves the weakest results.

Training set size

In Section 4.2 we explained the reason to train each neural network with 4 different portions of the training set. Contrary to our concerns the results confirm a widely known notion that the more data is used, the better performance is achieved.

Loss functions

Despite the fact that we have selected the Brier Score loss function to rank the neural networks we still used cross-entropy error function as a variant of ANN training function. This decision was made mainly because of the theoretical boost of the training process provided with usage of the cross-entropy error function [35]. In line with the rationalization presented in Section 4.4 our tests showed cross-entropy error function's poorer performance.

Activation functions

In terms of activation functions we have found that hyperbolic tangent has the best properties for optimization out of the five main functions listed in Section 2.2. Currently the most popular rectifier activation function [36] comes second which is probably explained by the fact that the better performing neural networks we have trained were not deep enough to show its strength.

Rest of the activation functions are ordered by performance as follows: sigmoid, hard sigmoid, and identity function coming last as expected.

5.2 Performance of Deeper Neural Networks

Although we will not compare all the concrete variations we have tried as the structures of the hidden layers, it is still worth to take a look at the average performance of ANNs with different number of layers. Out of the ANNs we have trained the networks with two hidden layers have the most optimal structures. The second worst performance yield neural networks with one hidden layer which means that the layers were either too small or the function we are trying to learn is hard to approximate using only one hidden layer.

The worst performance is achieved by models with the maximum number of hidden layers we have tried which is six. The most probable reason is the problem of vanishing gradients [37]. Explained shortly, the error signal received at the output layer is getting more dilute with each next layer while propagating backwards. This results in the first hidden layers learning exponentially slower than the next layers.

To address this issue we conducted additional experiments that utilize the technique for training deeper neural networks described in Section 2.7. Using the observations laid out earlier in this chapter we shrank the additional search space to only 140 experiment configurations. These are achieved by combining 35 hidden layer structures listed in Appendix C with four activation functions: the two best performing activation functions (one from each group observed in Section 2.2) and their respective more practical variations. The rest of the parameters were set to their best performing options.

5.3 Final Results

The best performing neural network was achieved during the additional experimenting which made use of the pretraining technique. This network was trained on 100% portion of the data set created using the most complex preprocessing technique. The training algorithm used the Brier Score function as a loss function. There are 4 hidden layers in the structure of this network (120, 90, 60, 30 neurons respectively) which use the leaky rectifier activation function. The visual representation of the network's performance on the test data set (which was completely uninvolved in the process of training and model selection) is shown in Figure 5.1.

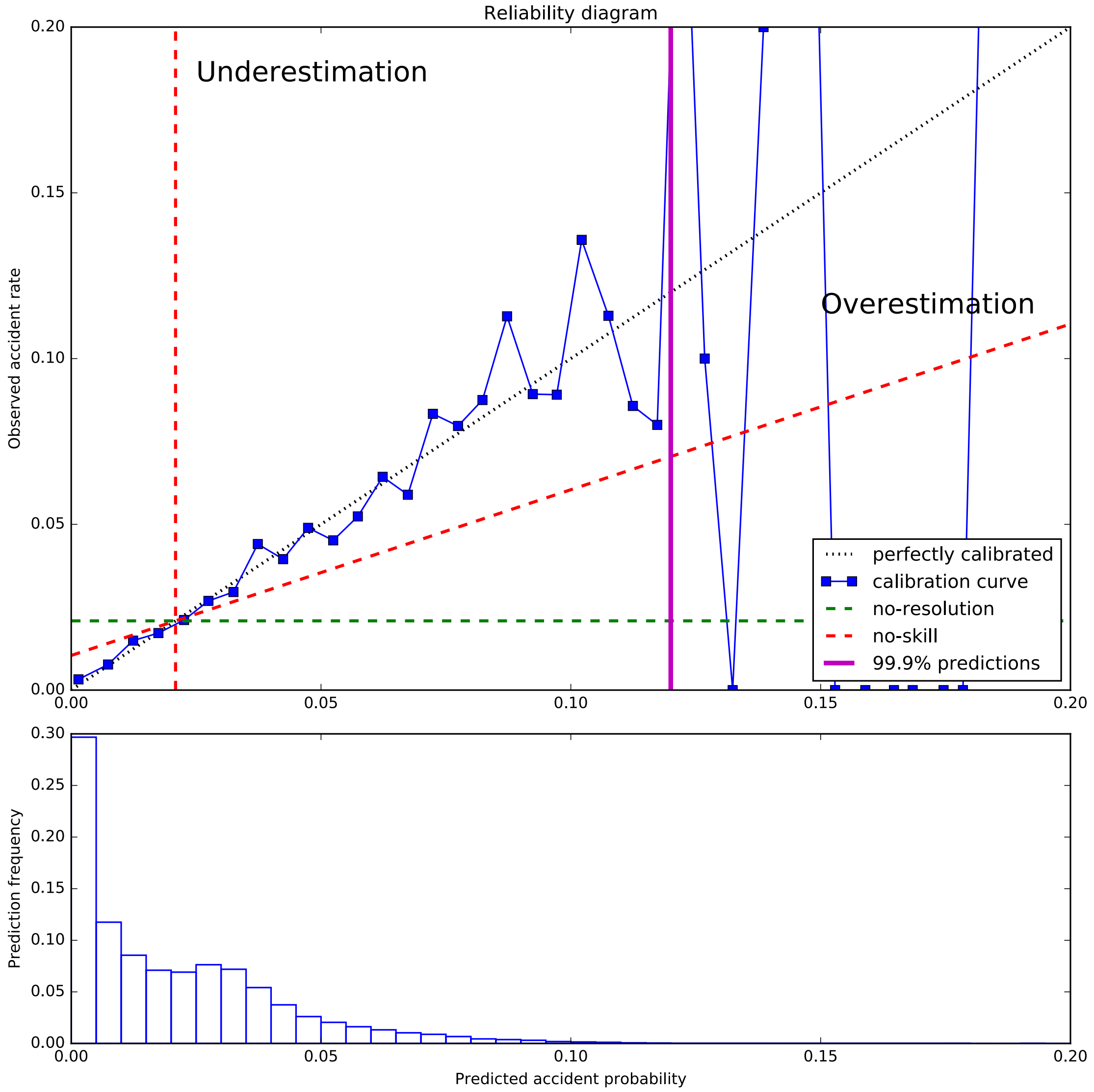


Figure 5.1: Performance on test data

As can be seen the network is making rather accurate predictions for the test set in the region from near 0 to ~12%. In Section 4.4 it was mentioned that the fewer data points are contained in the bin the less accurate the presented visual information is. The test data set has ~54 000 data points and the reliability plot shown before is created using bins of size 0.5% each. There is an exponential decline in the size of the bins from the lower risk bins to the higher risk bins as shown on the prediction frequency histogram. We have ~16 000 data points in the lowest risk bucket and approximately 700 data points in the bin representing range 6-6.5%. In the range from 10 to 10.5% there is already only 81 data points which is already quite a low number to adequately evaluate predictors accuracy.

It was calculated that 99.9% of the predictions lay in range from 0 to 12% which stresses once again how negligible are the errors past that point no matter how large they look.

Nonetheless, the bracket for the 12.5-13% range contains only 10 data points and is still quite on point with 10% observed accident rate. It is also worth mentioning that we are dealing with event probability evaluation which means that two adjacent peaks on opposite sides of diagonal in reality partially cancel each other out. For example, as illustrated below in Table 5.1 (which contains a part of the reliability diagram shown before) two adjacent bins with substantial errors to the opposite sides can cancel each other out really well when looked at as a single bin.

Bin range (%)	Bin size	Observed accidents	Observed rate of accidents (%)
6.5-7.0	560	33	5.9
7.0-7.5	480	40	8.3
6.5-7.5	1040	73	7.0

Table 5.1: Adjacent erroneous bins can correct each other

To portray a more general picture the resulting neural network's performance on the whole data set (training + validation + test) without any adjustments to the network is presented below in Figure 5.2.

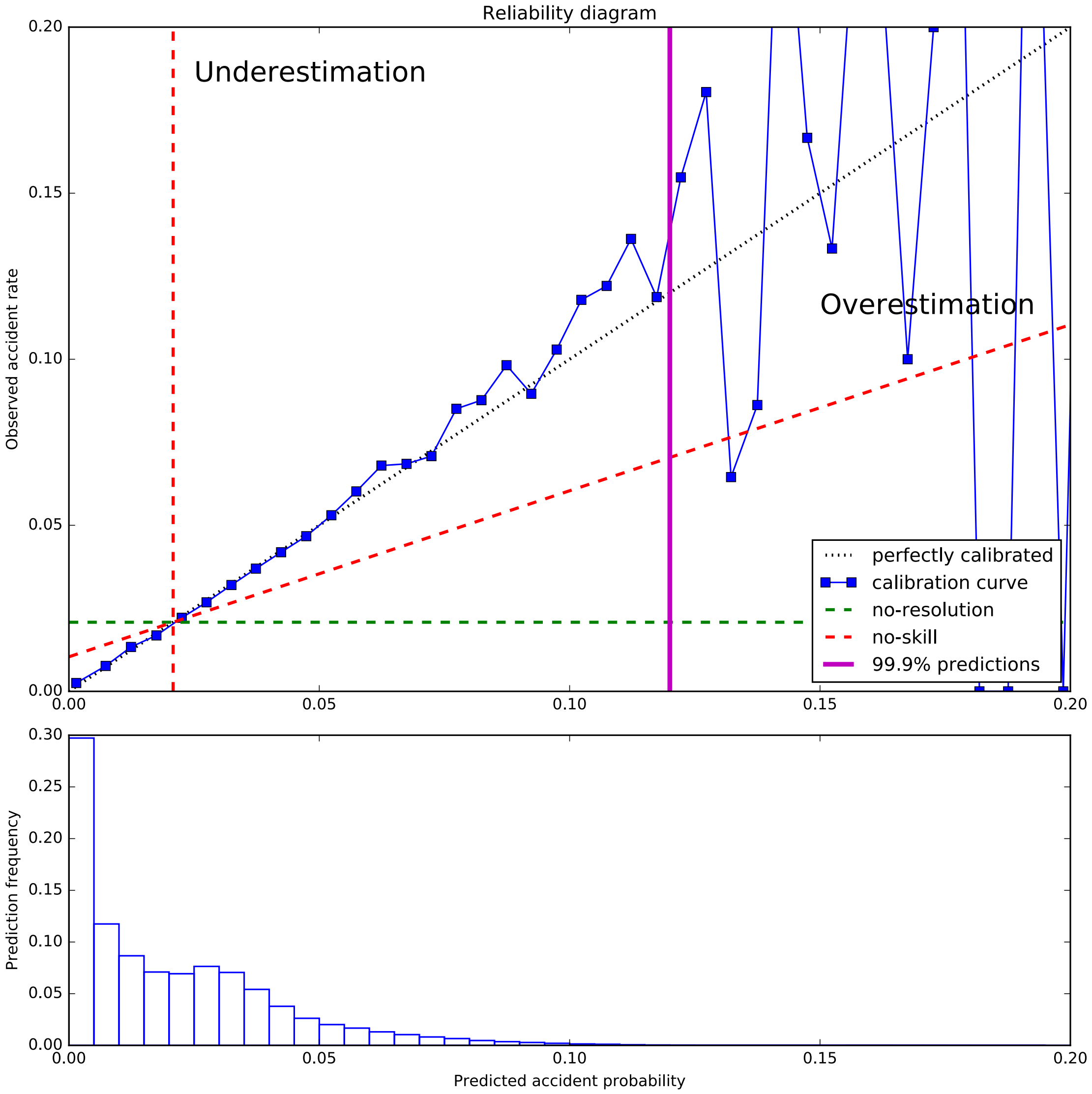


Figure 5.2: Performance on whole data set

In practice, calibration curve can also be used to improve the prediction model's performance. If it is noticed that the observed accident rate follows some linear or quadratic function of the predicted accident probability this knowledge can be used to modify the prediction system so that output is post-processed with that knowledge in mind. For example, our prediction system works really well on the whole data set until ~10% prediction probability. From that point on the neural network tends to underestimate the risk probability. One could use this knowledge to handle ANN's predictions past that point accordingly.

Conclusions

The goal of this thesis was to create a prediction system which would be able to evaluate motor insurance clients. More precisely, given data vector containing information about a person and his vehicle it would output the probability of an accident.

An extensive research was conducted in order to build such system using artificial neural networks. The final system outputs real values in range from 0 to 1 which are interpreted as a probability of an accident. The results were tested using a separate hold-out data set which was completely uninvolved in the process of this work up until the point of final evaluation. The produced neural network and its parameters can be easily saved and used as a motor insurance clients risk level evaluator on another machine.

All in all it can be concluded that the objectives of this work have been achieved. One of the possible directions for the future research is usage of more advanced algorithms to train deeper neural networks. It is also reasonable to try more advanced feedforward architectures of ANNs rather than the MLP architecture used in this work.

Bibliography

[1] K. Gurney, *An Introduction to Neural Networks.* Routledge, 2003-11-05.

[2] Theano open source github repository. Retrieved: 03.05.2016. [Online]. Available: https://github.com/Theano/Theano/blob/master/theano/tensor/nnet/sigm.py#L279

[3] Python and R job trends in machine learning. Indeed. Retrieved: 02.05.2016. [Online]. Available: http://www.indeed.com/jobtrends/q-Python-and-%22Machine-Learning%22-q-R-and-%22Machine-Learning%22.html

[4] *Road Traffic Act 1930*, Parliament of the United Kingdom, 1930.

[5] F. Kitchens, "Financial implications of neural networks in automobile insurance underwriting," in *International Conference on Fuzzy Sets and Soft Computing in Economics and Finance*, 2006.

[6] F. Kitchens, "Underwriting Automobile Insurance Using Artificial Neural Networks," in *Encyclopedia of Information Science and Technology*, 2nd ed. IGI Global, 2008, ch. 615, pp. 3865–3870.

[7] F. Azevedo, L. R. Carvalho, L. T. Gribergb, J. M. Farfel, R. E. Ferretti, R. E. Leite, W. J. Filho, R. Lent, and S. Herculano-Houzel, "Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain," *Journal of Comparative Neurology*, vol. 513, no. 5, pp. 532–541, April 2009.

[8] M. Caudill, "Neural nets primer, part VIII," *AI Expert*, vol. 4, no. 2, pp. 61 – 67, February 1989.

[9] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, December 1989.

[10] J. Snyman, *Practical Mathematical Optimization*, 1st ed. Springer US, 2005.

[11] H. Wu, "Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions," *Information Sciences: an International Journal*, vol. 179, no. 19, pp. 3432–3441, September 2009.

[12] P.Sibi, S. Jones, and P.Siddarth, "Analysis of different activation functions using back propagation neural networks," *Journal of Theoretical and Applied Information Technology*, vol. 47, no. 3, January 2013.

[13] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, *Neural Networks: Tricks of the Trade*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, ch. Efficient BackProp, pp. 9–50.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012.

[15] Y. LeCun, *Connectionism in perspective*. Elsevier, 1989, ch. Generalization and network design strategies.

[16] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *CoRR*, vol. abs/1505.00853, 2015.

[17] M. Joost and W. Schiffmann, "Speeding up backpropagation algorithms by using cross-entropy combined with pattern normalization," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 6, no. 2, pp. 117–126, April 1998.

[18] G. W. Brier, "Verification of forecasts expressed in terms of probability," *Monthly Weather Review*, vol. 78, no. 1, pp. 1–3, January 1950.

[19] A. H. Murphy, "A New Vector Partition of the Probability Score." *Journal of Applied Meteorology*, vol. 12, pp. 595–600, June 1973.

[20] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[21] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. D. Shet, “Multi-digit number recognition from street view imagery using deep convolutional neural networks,” *CoRR*, vol. abs/1312.6082, 2013.

[22] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *CoRR*, vol. abs/1409.4842, 2014.

[23] Choosing R or Python for data analysis? an infographic. The DataCamp Team. Retrieved: 02.05.2016. [Online]. Available: https://www.datacamp.com/community/tutorials/r-or-python-for-data-analysis

[24] TIOBE Programming Community Index Python. TIOBE Software Index. Retrieved: 02.05.2016. [Online]. Available: http://www.tiobe.com/tiobe_index?page=Python

[25] PyPI - the Python Package Index. Python Software Foundation. Retrieved: 02.05.2016. [Online]. Available: https://pypi.python.org/pypi

[26] A. Collette. HDF5 for Python. Retrieved: 02.05.2016. [Online]. Available: http://www.h5py.org/

[27] The HDF Group - Information, Support, and Software. The HDF Group. Retrieved: 02.05.2016. [Online]. Available: http://www.hdfgroup.org/

[28] R. K. Brouwer, “A feed-forward network for input that is both categorical and quantitative,” *Neural networks : the official journal of the International Neural Network Society*, vol. 15, no. 7, pp. 881–890, September 2002.

[29] O. Zhang. (2015, July) Tips for data science competitions. Presentation. DataRobot. Retrieved: 03.05.2016. [Online]. Available: http://www.slideshare.net/OwenZhang2/tips-for-data-science-competitions

[30] H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan, *Neural Network Design*, 2nd ed. USA: Martin Hagan, 2014.

[31] R. L. Winkler and A. H. Murphy, ““Good” Probability Assessors,” *Journal of Applied Meteorology and Climatology*, vol. 7, no. 5, pp. 751–758, October 1968.

[32] R. Selten, “Axiomatic characterization of the quadratic scoring rule,” *Experimental Economics*, vol. 1, no. 1, pp. 43–61, June 1998.

[33] M. D. Zeiler, “ADADELTA: an adaptive learning rate method,” *CoRR*, 2012.

[34] Keras optimizers. Retrieved: 21.05.2016. [Online]. Available: http://keras.io/optimizers/

[35] J. E. Bickel, “Some comparisons among quadratic, spherical, and logarithmic scoring rules,” *Decision Analysis*, vol. 4, no. 2, pp. 49–65, June 2007.

[36] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[37] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, March 1994.

Appendix A

Hidden Layers Structures List

Here we present a list of structures used to create the hidden layer(s) of the tested neural network models. To simplify their representation a simple notation is used: each layer is noted by the number of neurons it consists of delimited by an ’x’ mark. For example, 12x5 structure represents 2 hidden layers with the first layer consisting of 12 neurons and the second layer consisting of 5 neurons.

1 hidden layer: 5, 10, 25, 50, 100, 300

2 hidden layers: 5x5, 10x5, 10x10, 25x5, 25x25, 50x10, 50x50, 100x25, 100x100, 300x10

3 hidden layers: 5x5x5, 10x10x10, 25x25x25, 25x10x5, 50x25x10, 50x50x50, 100x50x10, 100x100x100

4 hidden layers: 5x5x5x5, 10x10x10x10, 25x25x25x25, 50x25x10x5, 50x50x50x50, 100x50x25x10

5 hidden layers: 5x5x5x5x5, 10x10x10x10x10, 25x25x25x25x25, 50x50x50x50x50, 100x50x25x10x5

6 hidden layers: 5x5x5x5x5x5, 10x10x10x10x10x10, 25x25x25x25x25x25, 50x50x50x50x50x50, 100x50x25x12x6x3

Appendix B

Tests Results Statistics

All tables are sorted ascending by the average Brier Score. As a reminder, lower value corresponds to better performance. Statistics below are composed using ANNs' performance on the validation data set. The best result achieved is a neural network described in Section 5.3 and the achieved Brier Score on the validation set is $2.066577 \cdot 10^{-2}$.

Preprocessing technique	Top 10 Average Brier Score (10^{-2})
Normalization + Mean Response + 1-of-C	2.067980
Normalization + Mean Response	2.075667
Rescaling Only	2.078897

Table B.1: Performance by preprocessing technique

Training set portion	Top 10 Average Brier Score (10^{-2})
100%	2.067992
80%	2.068690
50%	2.070298
20%	2.071387

Table B.2: Performance by training set portion

Loss function	Top 10 Average Brier Score (10^{-2})
Brier Score	2.067980
Cross-Entropy	2.068933

Table B.3: Performance by loss function

Activation function	Top 10 Average Brier Score (10^{-2})
TanH	2.068091
Rectifier	2.068508
Sigmoid	2.069189
Hard Sigmoid	2.069766
Identity	2.071281

Table B.4: Performance by activation function

Number of hidden layers	Top 10 Average Brier Score (10^{-2})
2	2.068376
3	2.068485
4	2.068764
5	2.068805
1	2.068973
6	2.069279

Table B.5: Performance by number of hidden layers

Appendix C

Additional Hidden Layer Structures

Here we present a list of structures used to create the hidden layers of the additionally tested neural network models. Same notation as in Appendix A is used.

3 hidden layers: 200x100x50, 100x75x50, 100x60x30, 100x50x25, 100x40x10, 75x50x25, 70x50x30, 75x40x20, 50x30x15, 50x25x10, 40x25x10, 25x10x5

4 hidden layers: 200x150x100x50, 200x100x50x25, 120x90x60x30, 100x75x50x25, 100x50x25x10, 80x60x40x20, 75x55x35x15, 75x50x25x10, 50x25x10x5, 40x30x20x10

5 hidden layers: 200x150x100x50x25, 200x100x50x25x12, 150x100x75x50x25, 100x85x70x50x35, 100x75x55x35x15, 100x75x50x25x10, 100x50x25x10x5, 75x50x25x10x5

6 hidden layers: 200x150x100x75x50x25, 100x85x70x55x40x20, 100x75x55x35x15x5, 100x75x50x25x15x5, 100x50x25x12x6x3