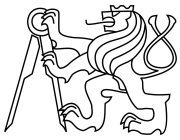
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF BACHELOR'S THESIS

Title:	Neural networks as anomaly detectors
Student:	Karel Rymeš
Supervisor:	Ing. Tomáš Borovička
Study Programme:	Informatics
Study Branch:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2017/18

Instructions

Artificial neural networks (ANN) are popular among many machine learning applications due to their versatility. However, different ANN architectures are suitable for different use cases. One of applications of ANN is anomaly detection. Autoencoders, Radial Basis Function Networks, Restricted Boltzmann Machines or Deep Belief Networks are ANN architectures that can be used as anomaly detectors.

Each of these architectures have hyper-parameters that specify the structure of an ANN or determine how an ANN will be trained.

The aim of this thesis is to review ANNs as anomaly detectors and analyse impact of hyper-parameters on its generalization capability, response, and performance.

1. Review and theoretically describe different architectures of ANNs suitable for anomaly detection.
2. For at least one architecture analyse impact of hyper-parameters on generalization capability, effectivity of the training and performance of an ANN.

References

Will be provided by the supervisor.

L.S.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague October 27, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Computer Science

Bachelor's thesis

Neural networks as anomaly detectors

Karel Rymeš

Supervisor: Ing. Tomáš Borovička

10th January 2017

Acknowledgements

I would like to thank Ing. Tomáš Borovička for extraordinarily patient guidance and for handing over his great engineering and science experience. I am equally grateful to the whole Data Science Laboratory at CTU FIT for offering me this thesis topic. Finally, I would like to thank all precious people close to me for both their support and advice.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 10th January 2017 .....

Czech Technical University in Prague
Faculty of Information Technology
© 2017 Karel Rymeš. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Rymeš, Karel. *Neural networks as anomaly detectors.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Tato práce teoreticky popisuje neuronové sítě a přináší obecné shrnutí přístupů používaných k detekci anomálií. Největší důraz je kladen na popsání aktuálního stavu technik detekce anomálií založených na umělých neuronových sítí. Ve velkém detailu jsou popsány sítě s radiální bázovou funkcí, autoenkoder, deep belief network a omezený Boltzmannův stroj. Tato práce též přináší experimentální doporučení ohledně výběru topologie a hyperparametrů autoenkoderu a stacked autoenkoderu. Experimentální část této práce je napsána v Pythonu.

Klíčová slova neuronové sítě, detekce anomálií, rešerše, optimalizace hyper parametrů, generalizační schopnosti, sítě s radiální bázovou funkcí, autoenkoder, Deep Belief Network, omezený Boltzmannův stroj

Abstract

This work theoretically describes artificial neural networks and overviews the anomaly detection branch. The biggest emphasis is then placed on the description of the current state of the anomaly detection techniques based on the neural networks. The radial basis function network, autoencoder, deep belief network and restricted Boltzmann machines are described in great detail. The thesis also provides experimental advice on hyperparameter and topology selection for the regular and stacked autoencoder. The experimental part of the work is written in Python.

Keywords Neural Networks, Anomaly Detection, Survey, hyper-parameter optimization, generalization capabilities, Radial Basis Function Network, Autoencoder, Deep Belief Network, Restricted Boltzmann Machine

Contents

List of Figures

List of Tables

Introduction

Motivation

In the current day and age, the data has inevitably entered everyone's life. Many devices and entities are responsible for this trend, particularly in the form of extensive data collection and data processing. Some of the data can be described as time series, a sequence of measurements collected successively in some intervals of time, e.g. a sensor measuring the temperature in one's smart home, while others take the form of either structured or unstructured data without any temporal connection, e.g. one's movie preferences recorded on Netflix [4]. This data is then bound to be harvested, cleansed, analyzed and possibly made predictions on.

The subjects involved in the first three stages of the mentioned data pipeline are not always faultless. The reasons for the flaws are different, e.g. device failure, change of surrounding environment, system security breach, and one cannot possibly and does not want to hand-pick the anomalies manually. Therefore, it is needed to automatically determine whether the obtained values match the general behavior and subsequently to raise alarms if they do not. For this task, one typically uses methods called anomaly detection which look for inconsistent records in studied data.

Another logical step then is to focus more on the artificial neural networks and decide which of their architectures and settings solve the problem the best under different conditions.

The Goals of the Thesis

The result of this work can be valuable to researchers and data scientist in practice as they often need to detect anomalies in their datasets. This work can serve them as an overview of the current methods based on artificial neural networks. The overview also contains information on when and if to use

distinct topologies of artificial neural networks in different problem settings. The resulting comparison and prototype are written in Python.

Organization of the Thesis

The work is organized into the following structure: First, the general machine learning notation and theory is introduced. Second, the anomaly detection problem is described in detail. It will be followed by a section on the basic notions of neural networks. Furthermore, I will describe how the different kinds of artificial neural networks are used for the anomaly detection tasks. In the last section ,the performed experiments that try to describe and deduce the generalization capabilities of different network hyperparameter selections, will be described. For this task, multiple common anomaly benchmark datasets are used.

Chapter 1

Machine Learning Overview

Supposedly, the term *machine learning* was coined by Arthur Samuel as "the field of study that gives computers the ability to learn without being explicitly programmed" as the articles [5, 6, 7, 8, 9] imply.

However, articles [5, 6] do not provide the exact source for the definition; they only place the definition into Samuel's mouth. Article [7] refers to [10] as a secondary source of information, in which such definition does not even occur. Also, the bibliography in [8] refers to Samuel's article [11] written in 1959 where such definition also does not occur! Wikipedia article [9] then uses the relevant quote from book [12] which points to yet another secondary online source [13] where Alex Holehouse only says that Samuel said it in 1959, Holehouse also talks about Samuel's chess artificial intelligence research. Coincidentally, the only paper by Samuel published in 1959 [11] is about chess but does not include the quote. It is, therefore, peculiar why the sources differ so much, and I was not able to find the original place where and if Arthur Samuel actually expressed the definition. Regardless of this inaccuracy, the definition is heavily used today as it can encompass all the different machine learning branches. The most notable fields where machine learning is used today inclyde speech recognition, self-driving cars and effective web search. It is even safe to say that by using a cell phone and a personal computer, one utilizes machine learning solutions a dozens times a day without knowing it.

As the whole chapter is very introductory and the information is well known, I only cite when I deem the information need a concrete founding. My sources for general knowledge are the following: [14, 15].

1.1 Introduction to Machine Learning

A machine learning algorithm in its most general form can be described as a function $f(\mathbf{x})$ which takes an input vector $\mathbf{x}$ and generates an output vector $\mathbf{y}$. Where the input vector $\mathbf{x}$ has a dimension of n and is considered to be the observation of the state, e.g. an image, one time step in a time series

measuring properties of a steel rod. Vector $\mathbf{y}$ then has a dimension of m which is independent of $\mathbf{x}$. Output $\mathbf{y}$ is considered to be the answer of the question $f(\mathbf{x})$ tries to solve. [14]

Data that occur in a machine learning algorithm can be understood as a set of L input vectors $\{\mathbf{x_1}, \ldots, \mathbf{x_L}\}$. The data is usually split into *learning* and *training* also called *testing* set. The requirement for sustaining the calculated class ratio of the whole dataset in both the training and learning portion is called *stratified sampling*. The requirement for equal class ratios in both learning and training sets is called a *stratified sampling*. When comparing performance of different machine learning models on the same data, a third split is performed, resulting in *validation* data. This set is used because one wants to compare the models' generalization capabilities on data that was not previously used during training.

The exact form of the function $f(\mathbf{x})$ is determined during the *training* phase, also known as the *learning* phase, on the basis of the training data. The ability to categorize correctly new examples that differ from those used for training is referred to as *generalization capability*.

1.1.1 Overtraining

In the course of the training phase, the error on the training data usually permanently decreases. The process of training, however, needs to be stopped at the right moment, otherwise the network might become *overtrained*. In this Thesis, the term *overfitted* is used interchangably. The system is overtrained when it is too adopted to the training dataset, loses the generalization capabilities, and can only perform the desired task correctly on the training data, in other words, fails to perform on the testing dataset. To detect the approximate step when to stop the training and therefore avoid overtraining, one can use the *cross-validation*. During cross-validation, one picks different parts of data in each epoch and keeps them aside. These set-aside vectors are then used to compare the errors of the network. When the error of the neural network on the cross-validation set begins to rise, it is better to end the training process.

However, the exact moment when to stop the training still needs to be chosen. One of the possibilities is to check the current error on the validation set after each epoch. In case the value has changed very little (little is subject to parametrization), then the learning is stopped, and the parameters before the halt of error improvement are announced as 'winners.'

1.1.2 Machine Learning Problems

Two main branches in machine learning are called *supervised learning* and *unsupervised learning*. Unsupervised learning tries to solve the task of inferring

a function which describes the hidden structure of *unlabeled* data. Naturally, labeled data have labels, whereas unlabeled data do not have labels.

An example from the medical field is used here. Different features of patient's lung scan are the input vector. In the case of the unlabeled data, that is all there is, whereas the labeled data would be accompanied by a tag that informs us about whether a patient has lung cancer. In supervised learning, the term *ground-truth* often occurs. It is used to denote the information contained in the data, which the model tries to learn.

The following machine learning methods are used in the rest of the thesis:

- In *Classification*, one tries to predict which category a new observation belongs to, e.g. predict one's vote in the 2016 US elections. Classification works with labeled data.
- In *Regression*, one tries to predict the values of a continuous function, e.g. the stock price of Alphabet Inc. after the 2016 US Presidential Elections. Regression naturally works with labeled data.
- In *Clustering*, one tries to group together a set of objects so that objects in the same group are more similar to each other than to those in other groups, e.g. clustering of US voters based on their previous election behavior. Clustering works with unlabeled data.

I also want to introduce an *optimization problem* which tries to solve the most general task of selecting the best element (with regard to some criterion) from some set of available alternatives. Optimization is usually categorized under Mathematics.

1.2 Algorithm Lifecycle

A scientist that wants to finish a machine learning or data mining project usually has to perform the following steps:

- Gather data for the project.
- Gain domain knowledge and insights into useful modeling assumptions. This step can be made easier by performing visualization of the data.
- Preprocess the data.
- Train a model which captures his assumptions about the problem and also successfully learns from the data present.
- Perform inference or make desired predictions.
- Evaluate results.

The steps of the lifecycle can be conducted multiple times to attain better results.

1.2.1 Data Preparation

When using data for machine learning models, there are a few procedures that should be used to prepare the data. It is also a common practice that one starts with a manual inspection of the dataset to get familiar with it, i.e. to discover first insights and have a good understanding of any possible data quality issues before applying any data preparation techniques. Data preparation is also about constructing a dataset from one or more data sources. Analyzing data that has not been prepared in the following ways can produce misleading results. That is why the Computer Science proverb "garbage-in-garbage-out" [16] is particularly relevant in the data mining projects

First, any missing data should be taken care of, i.e. either thrown away or heuristically filled in. Non-numeric data should be converted to numeric data, i.e. converting Likert scale to values 1 through 5. The data should be standardized and normalized to bring the features into proportion with each other. Last, the outlier detection methods might be performed, and the expert can then decide how to incorporate the outputs of the techniques into the project.

The *data normalization* is usually performed because the model can only accept data from a particular range; thus we normalize to the needed range.

$$x_{standardized} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{1.1}$$

Where x is a single training point, x_{max} and x_{min} the maximum respectively minimum value attained across all training samples.

With *data standardization*, the data is not bounded but rescaled to have a mean of 0 and variance of 1. Therefore, standardization typically measures how many standard deviations the value is from its mean.

$$x_{normalized} = \frac{x - \bar{x}}{\sigma} \tag{1.2}$$

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{1.3}$$

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{1.4}$$

Where N is the number of samples, $\bar{x}$ the mean and σ the standard deviation of the training samples.

1.2.2 Cost Functions

The *cost function* is sometimes also called *objective function*, *loss function* or *error function*. The usage depends on the field. In order to perform the training well, the machine learning models need to be evaluated. With supervised

learning, this comes naturally by comparing the prediction made by the model with the original labels. These are the most used cost functions:

- *Mean Error*

$$ME(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} \hat{y}_i - y_i \tag{1.5}$$

  The mean error has the property that negative and positive errors can balance each other out even though the particular errors can be high. The problem is solved by either an absolute value or a square.

- *Mean Squared Error*

$$MSE(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{1.6}$$

  A valuable property of squaring is that it penalizes big errors.

- *Root Mean Square Error*

$$RMSE(\hat{\mathbf{y}}, \mathbf{y}) = \sqrt{\frac{\sum_{i=1}^{n} (\hat{y}_i - y_i)^2}{n}} \tag{1.7}$$

  We can easily see that the following holds as square root is a monotonously increasing function: $RMSE(\hat{\mathbf{y}}, \mathbf{u}) < RMSE(\hat{\mathbf{y}}, \mathbf{v}) \Leftrightarrow MSE(\hat{\mathbf{y}}, \mathbf{u}) < MSE(\hat{\mathbf{y}}, \mathbf{v})$. Where $\hat{\mathbf{y}}$ is the ground truth, and $\mathbf{u}$ with $\mathbf{v}$ are different predictions made on the data. However, RMSE is more computationally expensive as it has the extra square root; thus the mean squared error is preferred in practice.

- *Mean Absolute Error*

$$MAE(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} |\hat{y}_i - y_i| \tag{1.8}$$

- *Mean Absolute Percentage Error*

$$MAPE(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{\hat{y}_i - y_i}{\hat{y}_i} \right| \tag{1.9}$$

Where n is the number of samples, $\hat{\mathbf{y}}$ is the ground truth, and $\mathbf{y}$ is the prediction made on the data.

The formulae where a square occurs can be multiplied by one half. The constant ensures computational ease during model training as we usually take a derivative. The half cancels out in the first derivative with the coefficient two that is produced from the square.

1.2.3 Gradient Descent

Gradient descent is an iterative general optimization method. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point. The gradient is the direction of the steepest descent (that is why the method is sometimes called the method of steepest descent) [14].

In machine learning, the cost function $f(\mathbf{x})$, where $\mathbf{x}$ is the model's parameters vector, might be minimized. The gradient descent method is also incorporated into the backpropagation algorithm 3.4.2. One iteration of the optimization method can, therefore, be written as:

$$\mathbf{x}^{i+1} = \mathbf{x}^i - \alpha \nabla_{\mathbf{x}^i} f(\mathbf{x}^i) \tag{1.10}$$

The learning rate α is a major component of the process as well. The learning rate multiplies the gradient that is used with the error adjusting. It represents how much the gradient in this iteration should influence the currently attained parameters. The greater the rate, the faster the method moves along the vector space, but the lower the rate, the more accurate the optimizing might be. Intuitively, with big learning rates, the algorithm can keep overshooting the minimum, with small learning rates the update is tiny. The learning rate can be adaptively changed throughout algorithm's lifetime, e.g. bigger learning rate in the earlier phases and smaller when the end is nearing.

Having a convex cost function guarantees that this function has only one minimum - the global minimum. If the cost function is not convex, gradient descent may find a sub-optimal solution by ending up in a local minimum [14].

1.2.4 Model Evaluation

A table serving visualization purposes of the binary classification algorithm's performance is called a *confusion matrix*. Each row of the matrix represents the predictions whereas the columns represent the actual classes (the vice versa representation is also possible). Table 1.1 visually defines the terms

Table 1.1: Terminology of a confusion matrix.

		Ground Truth	
		Positive	Negative
Classification Outcome	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

used in connection with the confusion matrix. In the actual confusion matrix, these cells would include the number of the specific instances that fall within the definition. Ideal model performance would place all the results either into true positives or true negatives.

The performance of a classifier can also be easily measured by the *receiving operator characteristic* (ROC). In a ROC curve, the *true positive rate* (TPR) is plotted as a function of the *false positive rate* (FPR) for different setting of model's parameters. The plotted variables abide by these equations:

$$TPR = \frac{TP}{TP + FN} \tag{1.11}$$

$$FPR = \frac{FP}{FP + TN} \tag{1.12}$$

ROC curves are usually used to represent the trade-off between the detection rate and the false alarm rate.

1.2.5 Regularization

Regularization is a technique employed in an attempt to solve the overfitting problem in machine learning models. There are two types of regularization used in this Bachelor Thesis - L1 and L2 regularization. Both of them try to keep the model parameters small so that the model learns simpler hypotheses and therefore might generalize better.

The equations for the regularization is as follows:

- *L1 Regularization*

$$\frac{\alpha}{n} \sum_{i=1}^{dim(\mathbf{w})} |w_i| \tag{1.13}$$

- *L2 Regularization*

$$\frac{\alpha}{2n} \sum_{i=1}^{dim(\mathbf{w})} w_i^2 \tag{1.14}$$

where $\mathbf{w}$ are the parameters of the model we try to find, n is the size of the training set and α is the *regularization parameter* which controls the importance of the regularization. The bigger the regularization parameter, the greater the importance of regularized parameters. It can be easily seen that in both L1 and L2 regularization, the larger the model parameter, the higher the cost penalty. The regularization parameter controls the trade-off between fitting the data well and keeping the magnitude of weights small.

1.2.6 Model's Hyperparameters

Machine learning models also present the need for *hyperparameters* which are higher-level model's properties set before the actual model's training begins. Setting the hyperparameters can be seen as particular model selection, i.e. choosing probability distribution to use from the hypothesized set of possibilities, how fast the model should train, or the regularization constants. They are often set by hand or selected by some hyperparameter search algorithm. One should note that rules-of-thumb and recommendations for hyperparameter choices should be taken with a grain of salt as they are not usually backed up with proofs but only with empirical data.

Chapter 2

Anomaly Detection

In this section, I discuss the basic terminology and the existing methods of anomaly detection. The anomalous patterns are often referred to as outliers, anomalies, discordant observations, exceptions, faults, defects, aberrations, noise, errors, damage, surprise, novelty, peculiarities or contaminants in different application domains [1]. Outlier as defined by Hawkins "is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism" [17]. The terms outlier and anomaly will be used interchangeably throughout this work. To determine an abnormal behavior, one collects data and then analyzes them with the appropriate methods. The methods, therefore, need to distinguish between normal and abnormal. They are called anomaly detection techniques.

Anomaly detection is used both in academia and in the industry today, e.g. detecting fraudulent bank transactions, system intrusions in corporate networks, tax frauds, human errors or machine failures in factories [18].

2.1 Terminology and Different Aspects of Outlier Detection Techniques

This section identifies and discusses the various aspects of outlier detection. Specific aspects of the problem are discussed, such as the input data, outlier types, the availability of resources as well as the outputs of the algorithms. The section promotes depth and breadth of the anomaly detection field.

2.1.1 Outlier Source

The source of the anomalies can be divided into the following [19]:

- Outlier indicates *an error*. The error might result from a failure of a sensor, data collection error, human error and many more. With these outliers present, the scientists might reach skewed conclusions.

Therefore, errors of this kind must be removed, or other appropriate measures must be taken (repair).

- Outlier indicates *a new event*. These values were created by a new mechanism of system operation. They are valuable as they might include interesting and useful information about rarely occurring events. They can be submitted to domain experts for hand classification and further investigation.

2.1.2 Outlier Types

The Thesis works with the following outlier classification [1]:

- *Type I Outlier* – This is the simplest type of anomalies and is the subject of the majority of existing outlier detection techniques. The data points are classified as outliers only due to the difference with values attained by normal instances. Systems that detect Type I outliers analyze the relation of an individual data point with respect to rest of the dataset.

- *Type II Outlier* – These data points are classified as outliers due to the occurrence of an individual data point in a particular context. Like Type I outliers, these outliers are also detected as individuals. The difference is that the Type II might not be an anomaly in different circumstances. The data context is induced by the structure in the data set. The underlying data set is usually either spatial or sequential which ensures that there exists an easily definable notion of context.

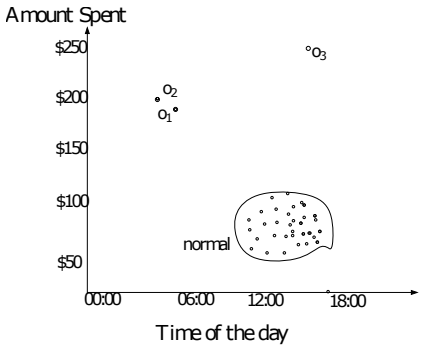
- *Type III Outlier* – These outliers are detected not as particular values but as an anomalous subset of the data in relation to the entire data set. Type III outliers are only present when the data has spatial or sequential nature. They are either anomalous subgraphs or subsequences occurring in the data.

It should be noted that Type I outliers may be detected in any kind of datasets. Type II and Type III outliers occur only with spatial or sequential structure in the data.

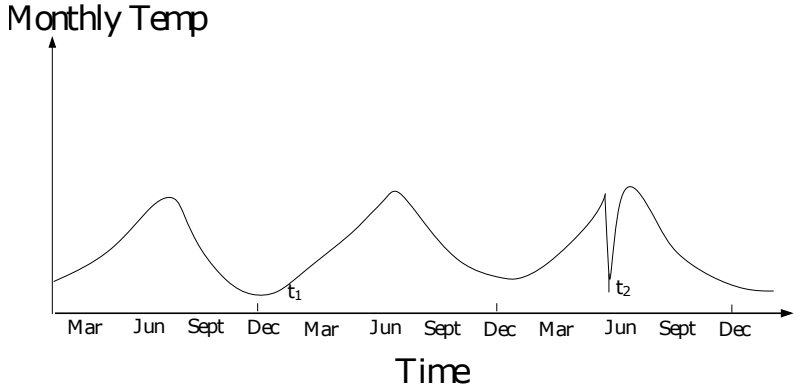
2.1.3 Algorithm Performance

The performance can be evaluated according to 1.2.4 where usually the positives will be the outliers and negatives will be the normal behavior. Anomaly detection techniques should aim to have a high detection rate while keeping the false alarm rate low [19].

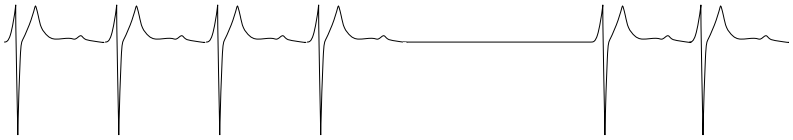
The efficiency of anomaly detection techniques is evaluated according to both time and space complexity. Efficient novelty detection methods should be scalable to large and high-dimensional data sets. Depending on the anomaly



(a) Type I outliers o_1, o_2, and o_3 in a credit card transaction data set. The normal transactions are typically placed during the day and range between \$10 to \$100. Outliers o_1 and o_2 are fraudulent transactions due to their strange time and amount. Outlier o_3 has only an unusually high amount spent.



(b) Type II outlier in a temperature time series. Note that the temperature at time t_2 is as low as at time t_1 but occurs in the summer and hence is considered an outlier.



(c) Type III outlier in a human electrocardiogram output. Note that the low value in the flatline also occurs in normal regions of the sequence.

Figure 2.1: Illustrative picture of different anomaly types. Source: [1].

detection task, the amount of memory required by a technique is typically considered an important performance evaluation metric [18].

2.1.4 Data Labeling

As explained in 1.1, the machine learning algorithms work with different kinds of data. I will now provide branch-specific information regarding anomaly detection. The following is the basic data classification [19]:

- *Supervised Learning* – Supervised outlier detection techniques learn the normal and outlier behaviors from labeled data explicitly. However, the problem is that accurately labeled training data might be hard to obtain because labeling is often done manually by a human specialist and hence requires a lot of effort to obtain.

- *Semisupervised Learning* – These methods tackle the inconvenience of acquiring labeled outlier data. Opposed to supervised learning, this approach uses labeled instances for only one class, i.e. the normal behavior. The typical approach of these techniques is to model only the available class and declare any instance which does not fit the pattern to belong to the other class, i.e. the outlier class.

- *Unsupervised Learning* – These methods can identify outliers without the presence of labeled data. For example, some techniques identify outliers based on a standard statistical distribution model, while others might classify outliers based on some similarity measures between data points. Compared to supervised and semi-supervised learning approaches, the unsupervised learning methods are more general because they do not need labeled data which might not be available in many of the applications.

The most often used methods are semisupervised and unsupervised. One of the reasons is that acquiring a labeled set of outlying data instances which cover all possible types of outlying behavior is difficult. Moreover, the outlying behavior is often dynamic in nature, e.g., new previously unknown types of outliers might arise. Lastly, outlying instances might correspond with very rare events that might be impossible to obtain and label in the training phase, e.g. airline accidents.

2.1.5 Algorithm Outcome

Another aspect of outlier detection methods is the manner in which the outliers are reported. Typically, outlier detection techniques fall into one of the following two categories [1]:

- *Labeling Techniques*

  These methods assign a normal or outlier label to each data point. Thus one might think of them as behaving like a classification algorithm.

- *Scoring Techniques*

  These techniques calculate an outlier score to each data point depending on the degree the pattern is considered an outlier. Consequently, the output of such techniques is a ranked list of outliers. The data scientist may choose to either manually analyze top few outliers or use a cut-off threshold to select the outliers. The disadvantage of a ranked list of outliers is the necessary choice of the threshold. Often times choosing the threshold is not simple and has to be arbitrarily fixed.

2.1.6 Types of Data Set

We describe several common kinds of data sets based on the attributes and characteristics of data [19]:

- *Simple Data Set* – Most of the existing outlier detection algorithms deal with data in which no structure and no complex semantics are assumed among the data instances. One refers to such data as point data.

- *Spatial Data Set* – The attributes of this data set are divided as non-spatial and spatial attributes. Spatial attributes contain location, directions, shape, and other geometric or topological information. The spatial characteristics of a data instance with respect to others is significant.

- *Sequential Data Set* – In the sequential data set, the data is naturally represented as a sequence of individual data points. The data instances have a natural ordering defined such that the data instances occur sequentially in the entire data set; with time-series being the most prominent example.

- *Streaming Data Set* – A data stream is data that is arriving continuously and fast in an ordered sequence. It is usually unlimited in size and occurs in many real-time applications, e.g. current stock prices.

2.2 Types of Anomaly Detection Techniques

The existing techniques choose to address the anomaly detection task by adopting concepts, principles and core ideas from different disciplines of machine learning, data science, and mathematics. Thus, the anomaly detection techniques can be divided into multiple categories. Most of the techniques described in the next sections deal primarily with Type I outliers, but some can be easily extended to handle Type II and Type III as well [1]. I would like to

point out that I do not go into much detail of the techniques. I only describe the most notorious examples of the different classes of anomaly detection or the models that need an explanation for later parts of the Thesis.

2.2.1 Classification based methods

Outlier detection methods based on classification operate in the same fashion as regular classification, using normal and outlier as the two classes - therefore it is a supervised outlier detection technique. However, there also exist the one-class classification techniques that try to characterize one class of entities and distinguish it from all other data points, thus learns a boundary around the readily available normal objects. A one-class classifier can thus be trained to reject the objects that do not fall into the category, therefore label it as an outlier. These techniques are the textbook examples of semi-supervised outlier detection techniques. With these techniques an instance is labeled as an outlier; however, there also exist classification methods that predict the confidence that an instance belongs to a class. Among the methods that are based on classification belong neural networks, Bayesian networks, support vector machines, decision trees and regression models. The neural networks will be described in great detail in the sections 3 and 4.

2.2.2 Clustering based methods

Cluster analysis or clustering is a technique that aims to group similar data instances into clusters. Cluster analysis is primarily an unsupervised technique though semi-supervised clustering also emerges.

The techniques discussed in this and 2.2.3 section require a distance computation between two data points. The techniques then assume that the feature set that defines the data discriminates the anomalies from normal points well enough. When such assumptions about the feature set cannot be made, classification based or statistical methods might be preferred, since they are more robust to the choice of feature set [19].

As stated, object similarity or proximity is fundamental for the two methods to work correctly. The metric chosen depends mainly on the types of features and their number. The similarity measure is often times based on object distances - Euclidean distance is the most popular choice. There are many more measures that can be used in different contexts.

Outlier detection and clustering are fundamentally very different from each other. While the former aims at detecting instances which are not similar to any other instances in the data, the latter aims at detecting groups with similar behavior. However, clustering based techniques take advantage of the fact that outliers should not belong to any cluster since there exist very few of them and are different from the normal instances.The assumed behavior of outliers is that they either do not belong to any cluster (form a cluster

on their own), or belong to small clusters, or are determined to belong to a cluster where they differ significantly from other members. Whereas, the normal instances belong to dense and large clusters. This observation makes it possible to separate the outliers from the rest of the data.

The clustering-based methods usually find the outliers as a by-product of their normal function; therefore they are not optimized to perform this task. The techniques might also introduce confidence score of the sample belonging to the given cluster . Thus, if no cluster is sure of the sample belonging, it must be pronounced as an anomaly.

The advantage of the cluster based approaches is that they are not supervised. Moreover, clustering based schemes can be used in an incremental mode, retraining the system easily with new points. One disadvantage of clustering based methods is that they are computationally expensive because of the computations of pairwise distances [1].

2.2.2.1 K-means

k-means is a popular clustering method. k-means aims to partition n data points into k clusters in which each observation belongs to the partition whose mean representative, also called a prototype, or centroid of the cluster, is the nearest. A notable disadvantage of the algorithm is that one needs to know the hyperparameter k in advance. As this algorithm will be used later in the Thesis, it is expressed in pseudocode in 2.1 [14].

Algorithm 2.1: The steps of the k-means algorithm.

1 Randomly select k prototypes from the training set.
2 **do**
    1. For each training data point:

       Assign the point to the cluster whose prototype has the least distance from it.

    2. For each cluster:

       Recalculate the prototype so that it is the center of all the points in the cluster.

    3. Calculate the error of the current clustering.
3 **while** *error does not meet stopping condition*

2.2.3 Nearest Neighbor Based Methods

Nearest neighbor techniques analyze data with respect to its nearest neighbors. This concept has been applied for different purposes such as classification,

clustering and also outlier detection. That is why the description might seem similar to the clustering based methods.

These techniques have an explicit notion of proximity, defined in the form of similarity measure or distance for any two data instances. While clustering based approaches take a global view of the data, nearest neighbor based methods analyze each data point with respect to its local neighborhood. The basic idea behind such methods is that an anomaly will have a surrounding where it will stand out, while a normal instance will have a neighborhood with similar neighbors. The strength of these techniques is that they can work in an unsupervised mode. A drawback of nearest neighbor based methods is the complexity required to compute the distances between all points as most of the techniques involve computing nearest neighbors for each point.

Nearest neighbor based outlier detection techniques can be further categorized into two types based on how the outliers are determined with respect to the nearest neighbors. The first category consists of techniques which measure the distance of an instance from its nearest neighbors and apply different tests to detect outliers. The second category computes the density of regions in the data and declares the instances in low dense areas as outliers. The most prominent example of such techniques is the Local Outlier Factor [1].

2.2.4 Statistical Based Methods

The statistical outlier detection techniques usually determine the generative probabilistic model and then test whether a data point is generated by that model or not. Like classification based approaches, these techniques typically operate in two phases - the training phase which comprises of estimating the statistical model and the testing phase where a test instance is compared to the derived model to determine if it is an outlier or not. An unsupervised technique determines a statistical model which fits the majority of the data points and any observation which resides in a low probability region of the model is declared as an outlier. The technique can also work in a semisupervised way by modeling only the normal instances, or outliers, depending on the availability of labels [1].

2.2.5 Information Theory Based Methods

Information Theory based methods analyze the information content in data using different information theoretic measures, e.g., entropy, relative entropy, etc. The concept behind these techniques is that outliers affect the information content of the data set because of their surprising nature. The basic approach is to measure the regularity of a data set with respect to every instance and classify the instance as an anomaly if it induces irregularity in the data. They typically operate in an unsupervised mode [18].

2.2.6 Spectral Decomposition Based Methods

Spectral decomposition methods work with the principle component vectors associated with a given data matrix. Thus in a way they try to detect the normal modes of behavior in the data. Spectral techniques can work in both unsupervised and semi-supervised settings [1].

Chapter 3

Neural Networks

In this chapter, I will provide a theoretical of the branch of artificial neural networks. In the first part, I will describe the model of a single neuron. Then, I will continue to explain how to build networks from these single neurons and also the various types of networks one we can create. Last, I will describe the common learning process of artificial neural networks. Same as in chapter 1, I would like to point out that most of the description is drawn from [20] and [14] and also that some of the information contained is considered common knowledge. Therefore, I will only cite where it is deemed helpful and necessary.

3.1 History of Neural Networks

Neural networks have been experiencing a renaissance in the last decade due to the advancement of computational power, the use of GPUs and the accessibility of data. In particular, neural networks proved their value in speech recognition and image processing [21].

3.1.1 Natural Inspiration of Neural Networks

The main inspiration of neural networks, as the name implies, comes from nature, specifically from the nervous system of animals.

First, I would like to explain the basic principles of the aforementioned biological model. The human brain consists of 10-100 billion neural cells which are called *neurons*. Each of these cells can be connected to 5000-10000 other cells.[1] *Dendrites* are the input transfer channels of the neuron whereas *axons* are the output transfer channels. Even though the end of axons is branched, the information at all its ends is the same. Impulses from other neurons are transmitted through dendrites to the body of the neuron. Subsequently, changing the neuron's inner state. If this state exceeds a specific threshold, i.e.

[1] The estimation for both numbers differ [22] vs. [20], thus the range.

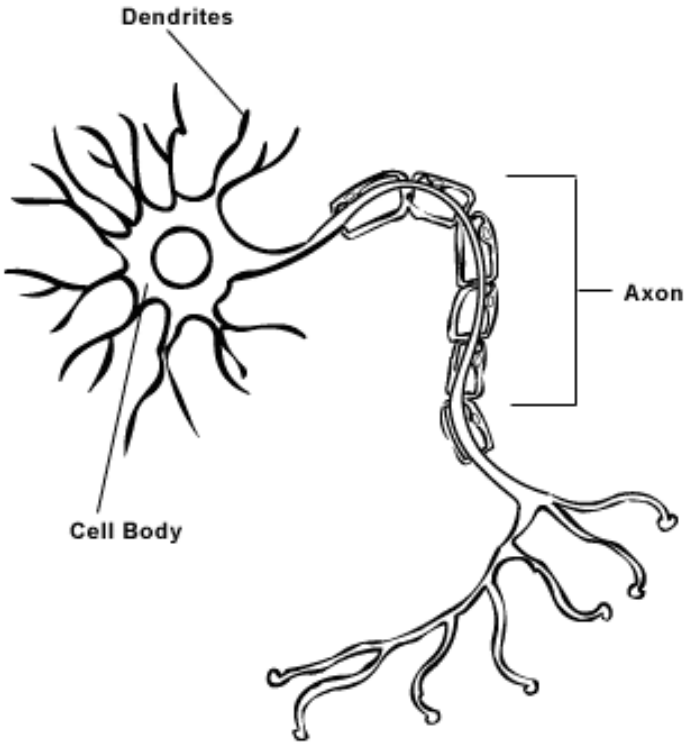


Figure 3.1: Figure depicting the basic structure of a neuron. Source: [2].

is activated, the neuron fires an electrostatic impulse which travels to other neurons through axons. The connection between ends of axons and dendrites is called a *synapse*. The synapses are the main storage of knowledge in the nervous system. The process of learning then consists of creating, destroying and changing the synaptic throughputs [20].

3.1.2 Short History

The first formal model of a neuron was introduced in 1943 by W. McCulloch and W. Pitts [23]. This model contained digital neurons; however, it did not have the learning capability. The network consisted of AND, OR and NOT elements. D. Hebb in his work [24] proposed that the functions of the neurons should permanently change, based on the system's adaptation to new things. In 1958, F. Rosenblatt proved that McCulloch-Pitts networks could be trained so that they are able to perceive and classify objects, thus the fitting name *Perceptron*. The main idea was to modify the neuron's settings if the outcome was not correct [25]. In 1969, M. L. Minsky and S. Papert in their book [26] showed that a single perceptron can only solve linearly separable problems. Two sets are *linearly separable* if there exists at least one hyperplane with all

points from first class on one side and all the points from the other class on the other side. The inseparability is most famously shown through the XOR function and is clearly depicted in figure 3.3. It is believed that this work caused the downfall of the Neural Networks for the next 20 years. They were brought back to the spotlight with the introduction of the training method called backpropagation that was able to learn multilayer perceptrons [27].

The movement striving for the exact computational model of the human brain is called Artificial Brain, the most notable example is the Blue Brain Project (BBP) by IBM [28]. The information about this project is sparse and constantly changing, however, one of the last reports was published in 2015 [29], BBP was able to simulate 31000 neurons and 37 million synapses, still lacking greatly behind the human brain.

3.1.3 Artificial Neuron

The simplest artificial model which is inspired by the biological neuron is called a *perceptron*. Perceptron, similarly as a neuron, processes n input signals and chooses exactly one appropriate output signal. The described basic model of an artificial neuron is shown in picture 3.2. The most general neuron operates

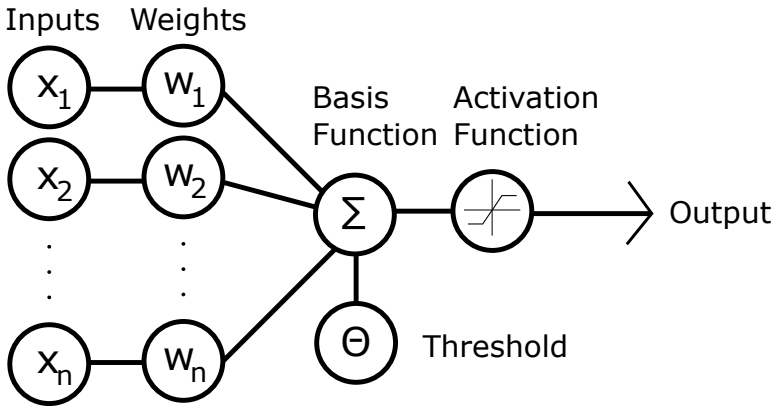


Figure 3.2: Figure depicting the original Rosenblatt's perceptron.

according to the following equation:

$$y = f(g(\mathbf{x}, b)) \tag{3.1}$$

where:

- y is the output value of the perceptron,
- x is the input vector,
- b is the bias which shifts the neuron's activation threshold and is a constant regardless of the input,

- f is the activation function of the neuron,
- g is the basis function of the neuron.

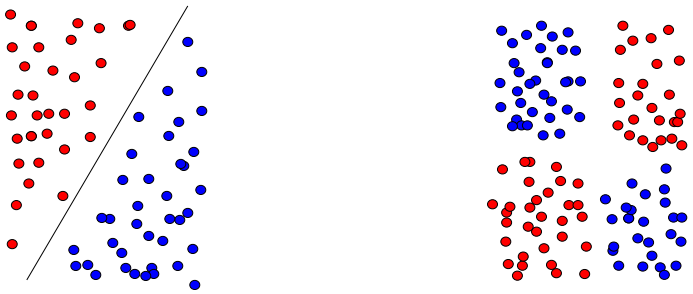
An intuitive mathematical definition of a perceptron: Having an input space of dimension n, the weight vector $\mathbf{w}$ then unambiguously determines the separating hyperplane in the input space, therefore splitting the data into two disjoint sets. The activation function applied on the inner potential then decides which halfspace the input vector falls in.

Rosenblatt's [25] original definition is more concrete as he uses only the Linear Basis Function.

There is usually a bias that resides inside a perceptron, regardless of the input values. The bias lets the engineer shift the activation function to the left or right. A sophisticated idea is used for the implementation of bias. An extra input whose output is permanently 1 is added to each neuron. This ensures that the biases can be taught the same way the weights are.

3.2 Terminology and Different Aspects of Neural Networks

In the rest of the chapter, unless stated otherwise, the terms artificial neural networks and neural networks are used interchangeably. Neural networks can solve both unsupervised and supervised learning problems. Neural networks can be among other things applications for classification, predictions, clustering, optimization and pattern recognition. A neural network implementation of a Turing machine was introduced by Franklin and Garzon in [30]. Thus, any algorithmic problem that is Turing solvable can be encoded as a problem solvable by a neural network. It can be, therefore, deduced that neural networks are at least as powerful as Turing machines.



(a) This figure shows a linearly separable dataset with the splitting hyperplane.

(b) This figure shows a linearly inseparable dataset - the XOR problem.

Figure 3.3: A figure showing two different kinds of datasets.

3.2.1 Neural Networks

The perceptron from the section 3.1 is only able to model linearly separable problems. Having multiple perceptrons parallelly in one layer is also not sufficient to be able to perform non-linear separation. If the neurons are connected serially to multiple layers, so that output of one layer is the input of the another layer, it is called a *multi-layer neural network*. The resulting structure is then able to solve linearly inseparable problems. The function of one layer with linear basis function can be subsequently written as

$$\mathbf{y} = f(W^T\mathbf{x} + \mathbf{b}) \tag{3.2}$$

Where $\mathbf{x}$ is the layer input vector, $\mathbf{W}$ is the matrix of weights, $\mathbf{b}$ is the vector of biases and f is the activation function. The whole network can then be seen as a chain of several such functions. The number of layers, the shape of their connections and then neurons in each layer determine the particular models of Neural Networks.

3.2.2 Basis Functions

The information at neuron's inputs is collected by means of basis function, producing the neuron's *inner state*. The inner state is then the input value of the activation function.

The following are the only basis functions used in the thesis:

- *Linear Basis Function (LBF)* is described as the weighted sum of the input vector with the following formula:

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \sum_{i=1}^{n} w_i x_i \tag{3.3}$$

- *Radial Basis Function (RBF)* is determined by a significant point called the center $\mathbf{w}$. It is described as the distance from the center. The norm is usually a Euclidean distance of vectors with the following equation:

$$g(\mathbf{x}) = \|\mathbf{x} - \mathbf{w}\| = \sqrt{\sum_{i=1}^{n}(x_i - w_i)^2} \tag{3.4}$$

Where $\mathbf{x}$ is the input vector and $\mathbf{w}$ are the weights of the neuron. For simplicity, when talking about Basis Functions, I will use the term weights even for the center that occurs in the RBF. With Linear Basis Function being the most often used one.

3.2.3 Activation Functions

The activation function determines the final output of the neuron. The activation function is also known as the transfer function. Its input is the inner potential. The activation functions usually satisfy the following conditions:

$$\lim_{x \to -\infty} f(x) \in \{-1, 0\} \tag{3.5}$$

$$\lim_{x \to \infty} f(x) = 1 \tag{3.6}$$

The activation functions used with the linear basis function are depicted in 3.4 and defined below [20]:

- *Identity*

$$f(x) = x \tag{3.7}$$

- *Sigmoid*

$$f(x) = \frac{1}{1 - e^{\alpha - x}} \tag{3.8}$$

  where α is the steepness parameter which is determined during the initialization of the neural network.

- *Hyperbolic tangent*

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}. \tag{3.9}$$

- *Rectified Linear Function (ReLU)*

$$f(x) = max(0, x) \tag{3.10}$$

- *Softplus*

$$f(x) = \ln(1 + e^{x}) \tag{3.11}$$

- *Piecewise Linear Activation Function*

$$f(x) = \begin{cases} a, & \text{for } x < c \\ b, & \text{for } x > d \\ a + \frac{(b-a)(x-c)}{d-c}, & \text{for } c \leq x \leq d \end{cases} \tag{3.12}$$

  Where a is the minimum value of the function, b the maximum, c the x value where the function begins to grow and d the x value where the function attains the maximum.

With x being the inner state.

The Rectified Linear Unit showed improvements in Restricted Boltzmann Machines [31]. This unit has no problems with vanishing gradients in large inputs. Computation of the function is also more efficient than other function. The function is not differentiable at 0; however, it can be replaced with softplus, which is analytic function approximating the rectifier.

Whereas the activation function most often used with RBF are the following:

- *Gaussian-like*

$$f(x) = e^{-(\frac{x}{\sigma})^2} \tag{3.13}$$

  , where σ determined during the initialization of the neural network and in the case of a RBF neuron is the neuron's center width (see: 4.4.1).

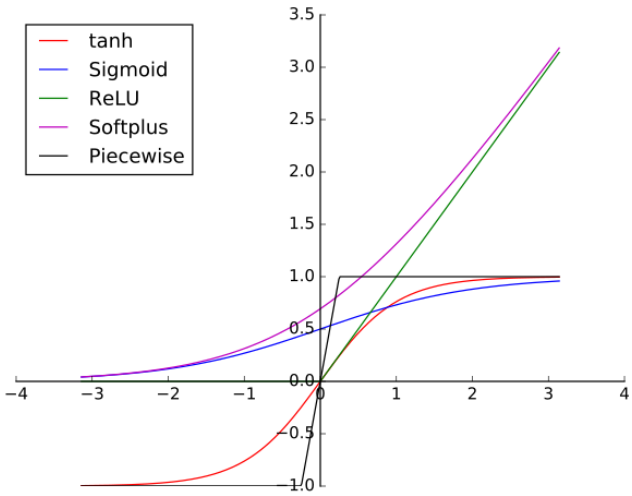


Figure 3.4: Comparison of different activation functions.

3.3 Topology of Neural Networks

The topology in the ANN context means the organization, ordering and quantities of neurons in different layers. Each topology usually has a problem domain where it performs the best. The topology belongs to the most important part of neural networks hyperparameters; others include the way and speed at which the network is trained.

Units from figure 3.5 will be used throughout the rest of the Thesis to explain the different topologies. Please note that over the years, some of the

improvements in the ANN models and topologies have drifted away from their natural inspiration, e.g. some of the neural nets are developed regardless of the natural inspiration. The human brain is used just for the fundamental idea.

The most basic classification is based on the number of layers – *one-layer* and *multilayer* neural networks (see 3.6a and 3.6b respectively).

As stated in previous sections, one-layer ANN can only perform linear and affine transformations, whereas multilayer networks have greater computational power and are also Turing complete. The multilayer networks contain *hidden layers*. They are called hidden because their inputs and outputs are not accessible from the outer world. The number of hidden layers and neurons in each layer depend on the problem and will be the subject of the research.

Two factors cause the improved generalization capabilities of multi-layer perceptrons. But equally important is the use of correct activation function. The activation function has to introduce some kind of non-linearity. Otherwise, the stacking of fully linear neurons would only produce a linear neural network. Hornik in [32] showed that a feed-forward neural network can approximate any continuous function to arbitrary precision, given the hidden layer is large enough and the activation function is continuous, bounded and nonconstant.

The multilayer networks can be classified based on the ways the signal propagates between neurons.

Acyclic networks or *non-recurrent networks* are the subset of multilayer networks. The Acyclic networks do not have connections between neurons in one layer. The connections can only exist between a neuron in layer i and layer j where $i < j$. Therefore, the graph can be classified as multipartite. The computational process in an acyclic network is much less computationally intensive compared to the cyclic networks.

The feed-forward network is a subset of the acyclic networks. It is the simplest architecture of acyclic neural networks. The connection in a feed-forward network can only occur between layer i and $i+1$. In the feed-forward network, the input signal is propagated through each layer into the connecting layer. The outputs of the first layer are the inputs of the first hidden layer, outputs of the first hidden layer are the inputs of the second hidden layer, etc.

The *cyclic networks*, or also called the *recurrent networks* 3.6c (as in they might return the signal to the input layer) propagate the signal without any restrictions being imposed on them, i.e. the connections can occur wherever. There are different kinds of recurrent networks intended for solving diverse problems; recently they are most notable for visual perception tasks. Noteworthy examples include Hopfield network, Elman network, long short-term memory, neural Turing machines and many others. The recurrent architecture resembles the brain more than the feed forward one.

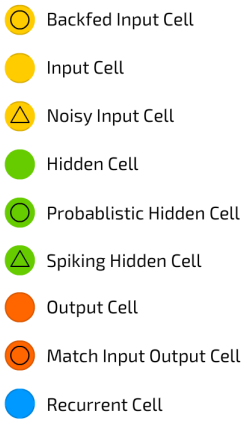
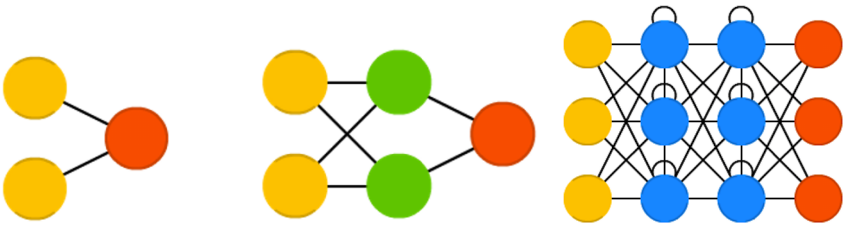


Figure 3.5: Legend for the topology figures in the rest of the thesis. Source: [3].



(a) One-layer neural network. (b) Multilayer neural network. (c) Recurrent neural network.

Figure 3.6: A figure showing different types of basic neural network topologies. Source: [3].

3.3.1 Softmax layer

The softmax layer is used in classification problems where an object can belong to only one of the classes. This layer computes a probability distribution over all classes. The neurons' activation function is an exponential. The sum over all output neurons activations must also be 1. This is achieved by normalizing the outputs.

$$y_i = \frac{\exp(z_i)}{\sum_{j=1}^{m} \exp(z_j)} \tag{3.14}$$

Where m is the total number of output neurons and z_i is the inner state of the i-th output neuron before the application of the activation function.

3.4 Training Neural Networks

Instead of following a set of rules, specified by a third person, the network teaches itself the necessary rules based on the data set. In the field of neural networks, the process is understood as choosing the right weights and network topologies so that it might perform the desired task efficiently.

The learning is performed by iteratively presenting the training data set, adjusting the network parameters and thus ideally minimizing the network's error. The goal of the learning process is to approximate a function which describes the desired behavior the best (provides the best outputs for given inputs). The whole training is performed in *iterations*.

A notable drawback of the ANN is their black box approach which causes the model to be very hardly interpretable and understood. The training phase can also be seen as very computationally heavy.

3.4.1 Initialization of the Training Procedure

Before the network starts learning anything, it needs to be initialized. The initialization consists of setting all the weights and biases. The key is to start with different weights as the neuron with same incoming and outgoing weights would attain the same gradient during training. The most basic form of initialization is setting the parameters to *random values*. One possibility is to initialize the weights with a random Gaussian variable with std. deviation 1 and mean value of 0. Bengio in [33], however, proposes to pick the initial weights proportionate to the square root of neuron fan-in and fan-out (indegree and outdegree of the node). As described in the section 1.2.5, it is better to keep the weights as small as possible.

The number of layers and neurons in each of these layers is important; however, there exist only general empiric observations and the parameters are usually the knowledge of a good neural network architect. In these simplifying terms, one can say that a small number of neurons in layers leads to insufficient training whereas large quantities of neurons lead to the loss of generalizing capabilities.

3.4.2 Backpropagation Algorithm

The change in one neuron activation influences many other activations in the network. These effects must be combined. Backpropagation tries to do that by working within a similar concept as the network's forward pass.

The forward pass takes an input, keeps computing activations of single neurons layer by layer until it reaches the last layer and calculates the desired output. Backpropagation computes gradients of weights for the last layer and propagates them back through the network using same weights as the forward pass. Then, for each node i in layer l, one would like to calculate an error

term $\delta_i^{(l)}$ that measures how much that node was responsible for any errors in our output.

The Backpropagation algorithm is derived using the derivation chain rule and the gradient descent technique from 1.2.3. The algorithm is in detail described in 3.1, where the core is taken over from [34].

One also has the following possibilities on how often to update the network's parameters [15]:

- *Online update* – Weights are updated after each training case.
- *Full-batch update* – Weights are updated only after every epoch.
- *Mini-batch update* – Weights are updated after a small sample of training cases.

A forward and backward pass where the neural network is fed the entire training set is called an epoch. A forward and backward pass where the neural network is fed the batch is known as a cycle or an iteration. The difference in the two terms can be seen in the different update policies, e.g. with the full-batch update, the iteration and epoch count are identical.

3.4.3 Dropout

Dropout is a recent invention in the field of neural networks [35]. It can be seen as a type of regularization, together with techniques like L1 and L2 regularization, and constraining the maximum value of weights. It tries to address the problem of overfitting.

The name dropout comes from the idea of dropping out some unit activations in a layer during training. Dropping out means setting them to zero. This can be seen as sampling a neural 'subnetwork' from the full neural network and only updating the parameters of the sampled network for the current train set.

3.4.4 Momentum

The adaptation of weights might not only be determined by the current training examples but also by the previous examples with less intensity; this is called *momentum*. Momentum is added to the equations to prevent getting stuck in a local minimum [33].

Algorithm 3.1: Learning of the neural network with the online back-propagation algorithm.

1 Choose topology of the neural network
2 Perform initialization of the network's parameters
3 **while** *stopping condition is not met* **do**

1. Perform a feedforward pass, computing the activations for all layers.
2. For each output unit i in layer n_l (the output layer), set

$$\delta_i^{n_l} = \frac{\partial}{\partial z_i^{n_l}} \frac{1}{2} \left\| y - h_{W,b}(x) \right\|^2 = -(y_i - a_i^{n_l}) \cdot f'(z_i^{n_l}) \tag{3.15}$$

   , where $\delta_i^{n_l}$ is the error term, $z_i^{n_l}$ is the value of the neuron i in layer n_l after the basis function is applied, y is the ground truth, $h_{W,b}(x)$ is the calculated network's output value, $a_i^{n_l}$ is the neuron's output value after the activation function has been applied, and $f(x)$ is the activation function. Therefore it is necessary that the activation function has a derivative.
3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$

   For each node i in layer l, set

$$\delta_i^l = (\sum_{j=1}^{s_{l+1}} W_{ji}^l \delta_j^{l+1}) \cdot f'(z_i^l) \tag{3.16}$$

   , where W_{ji}^l is the matrix of weights.
4. Compute the desired partial derivatives, which are given as:

$$\frac{\partial}{\partial W_{ij}^l} J(W, b; x, y) = a_j^l \delta_i^{l+1} \tag{3.17}$$

$$\frac{\partial}{\partial b_i^l} J(W, b; x, y) = \delta_i^{l+1} \tag{3.18}$$

   , where $J(W, b; x, y)$ is the mean squared error with respect to a single example.
5. Update the weights as follows:

$$W_{ij}^l = W_{ij}^l - \alpha \frac{\partial}{\partial W_{ij}^l} J(W, b; x, y) \tag{3.19}$$

$$b_i^l = b_i^l - \alpha \frac{\partial}{\partial b_i^l} J(W, b; x, y) \tag{3.20}$$

   , where α is the learning rate.

Chapter 4

Neural Network Based Anomaly Detection Methods

Anomaly detection can be done through many different techniques. The main purpose of this Thesis is to; however, research only a small subset of these possibilities, residing in the neural networks models. The rich field of neural networks has resulted in an equally vast number of neural network based anomaly detection methods. It is, therefore, natural that each of the numerous variations of the outlier detection problems can be addressed well by only certain types of neural network. Most neural network approaches work either in semi-supervised or supervised mode [1]. In this chapter, the architecture of only several types of neural networks is describeb. The chapter also includes examples of how the particular networks are used in novelty detection.

Unfortunately, there has been little survey and summarizing effort in the particular field of neural network based anomaly detection methods. Therefore most summarizing has been done in the general anomaly detection surveys [1, 19] in 2007 and [18] in 2014. However, the last and only specialized review of the neural network based techniques has been published in 2003 [36]. The important thing to note here is that the review has been conducted before the renaissance of neural networks in the last decade. Thus the general reviews may serve as an equally important source, if not better.

4.1 Restricted Boltzmann Machines

Restricted Boltzmann machines (RBM), as the name implies, are derived from the Boltzmann machines (BM). The name of this model stems from the statistical mechanics since the probability of particular states during the thermal equilibrium is equal to the Boltzmann distribution [20]. The learning process of BM is computationally demanding; that is why RBM are often used instead. RBM is also used as a building blog of deep belief networks 4.2. The

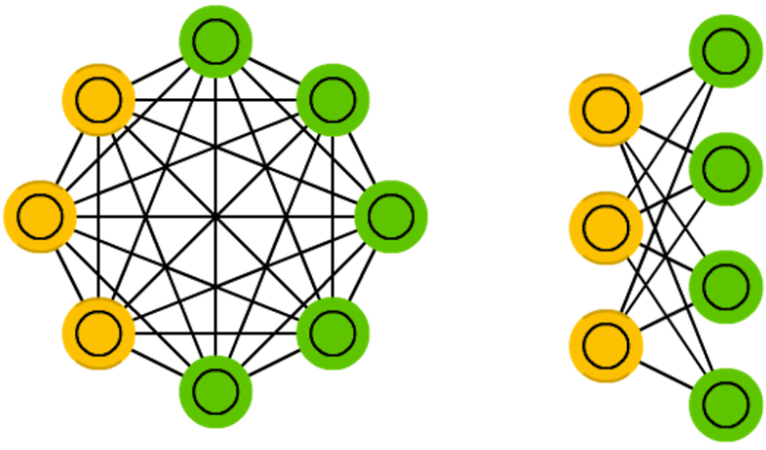
subsections 4.1.1 and 4.1.2 are both paraphrased from [37].

4.1.1 Topology and Operating Principles

The aforementioned restriction appears in the topology of the network. Boltzmann Machines are symmetrical networks of stochastically working units which can be interpreted as a neural network. BM is fully connected, and no restriction is imposed on the chromatic index of the network. RBM is arranged in two layers and must be a bipartite undirected graph (the difference in the topology might be seen in 4.1). The first layer has visible neurons and is the observational part of the network. The second layer contains the hidden neurons and models the relations between the particular features. Each neuron from both layers is connected with all the neurons from the other layer. The network is fully defined by a matrix of weights associated with the connections between hidden and visible units, as well as the biases for the visible and hidden units.

The standard type of RBM has binary-valued (boolean/Bernoulli) hidden and visible units; thus the name Bernoulli-Bernoulli restricted Boltzmann machine is sometimes used. There also exist the Gaussian-Bernoulli Restricted Boltzmann Machine which uses real-valued Gaussian units in the visible input layer. Naturally, the use of Gaussian-Bernoulli RBM is advisable when the input data is real-valued. The hidden layer is similar to the Bernoulli-Bernoulli RBM as it uses the binary stochastical neurons. For simplicity, the formulae described in this section assumes the Bernoulli-Bernoulli RBM. For more info on the GBRM, please, see [37].

RBM also belongs to energy-based models. Energy-based models associate



(a) Boltzmann Machine. (b) Restricted Boltzmann Machine.

Figure 4.1: Figures comparing classic Boltzmann machine and its restricted version. Source: [3].

energy to each configuration of the variables. Learning then modifies the energy function so that its shape has desirable properties. It is desirable that the plausible configurations have low energy. The plausibility is derived from the occurrence in the training data. Energy-based models define a probability distribution over hidden vector $\mathbf{h}$ and visible vector $\mathbf{v}$ (in this case, both boolean vectors) through an energy function [37]:

$$P(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})} \tag{4.1}$$

, where Z is a partition function defined as:

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{4.2}$$

, and $E(\mathbf{v}, \mathbf{h})$ the energy of a configuration defined as:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_i \sum_j v_i w_{i,j} h_j \tag{4.3}$$

Similarly, the marginal probability of visible boolean units over all possible hidden layer configurations is defined in the following manner:

$$P(\mathbf{v}) = \frac{1}{Z} \sum_h e^{-E(\mathbf{v},\mathbf{h})} \tag{4.4}$$

Because of the RBM's topology, the visible unit activations are mutually independent given the hidden unit activations and conversely the hidden unit activations are mutually independent given the visible unit activations. Which permits the following conditional probabilities formulations [37]:

$$P(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{m} P(v_i|\mathbf{h}) \tag{4.5}$$

, and

$$P(\mathbf{h}|\mathbf{v}) = \prod_{j=1}^{n} P(h_j|\mathbf{v}) \tag{4.6}$$

, where the individual activation probabilities are defined as:

$$P(h_j = 1|\mathbf{v}) = \sigma\left(b_j + \sum_{i=1}^{m} w_{i,j} v_i\right) \tag{4.7}$$

, and

$$P(v_i = 1|\mathbf{h}) = \sigma\left(a_i + \sum_{j=1}^{n} w_{i,j} h_j\right) \tag{4.8}$$

, where σ denotes the logistic function. The probability of turning on is determined by the weighted input from other units (plus a bias).

4.1.2 Training

RBM can be used to learn significant aspects of unknown probability distributions based on provided training samples. The learning tries to fix the parameters so that the probability distribution represented by the network corresponds to the training data and so that the arrangement expresses the relations between input features well. After successfully learning, the RBM provides a finite representation of the observation's distribution.

The training algorithm most often used is called the contrastive divergence (CD). The algorithm performs Gibbs sampling [2] that is combined with gradient descent optimization methods 1.2.3 to determine the weight updates. One also needs to decide how many Gibbs sampling iterations are performed on one training data point; the k in CD-k denotes this decision.

The CD-1 learning step for one sample can be summarized as follows [37]:

1. Initialize the visible units to a training sample $\mathbf{v}$

2. Compute the probabilities of the hidden units according to equation 4.6 and sample a hidden activation vector $\mathbf{h}$.

3. Calculate the outer product of $\mathbf{v}$ and $\mathbf{h}$ and call this the positive gradient.

4. Compute the probabilities of the visible units according to equation 4.5 and sample a reconstruction $\mathbf{v}'$ of the visible units.

5. Resample the hidden activations $\mathbf{h}'$ based on $\mathbf{v}'$. (Gibbs sampling step)

6. Calculate the outer product of $\mathbf{v}'$ and $\mathbf{h}'$ and call this the negative gradient.

7. Update the weight matrix W based on the positive and negative gradients:
$$W = W - \alpha(\mathbf{v}\mathbf{h}^\mathsf{T} - \mathbf{v}'\mathbf{h}'^\mathsf{T}) \tag{4.9}$$

8. Update the biases $\mathbf{a}$ and $\mathbf{b}$ analogously:
$$\mathbf{a} = \mathbf{a} - \alpha(\mathbf{v} - \mathbf{v}') \tag{4.10}$$
, and
$$\mathbf{b} = \mathbf{b} - \alpha(\mathbf{h} - \mathbf{h}') \tag{4.11}$$

[2] For the definition of Gibbs sampling, please, see: [14].

4.1.3 Applications in Anomaly Detection

RBM is described here because of its prominent role in the operation of the deep belief networks 4.2, which serve as a common anomaly detection technique.

In [38], the authors aimed at developing a self-learning anomaly detection model. They used the Discriminative Restricted Boltzmann Machine (DRBM) neural networks which combine strong generative modeling capabilities of the RBM with the classification in order to infer knowledge from the training data. The authors' hypothesis was that there exist deep similarities between normal cases that can be fully expressed using the DRBM so that anomalies can be spotted with high accuracy. Based on the results of experiments, the authors concluded that when the DBRM is tested on data that is "widely [sic] different" from the training data the performance suffers.

4.2 Deep Belief Networks

Deep belief network (DBN) is a type of multilayer generative stochastical neural network. DBN is a stacked architecture of mostly RBMs. The networks have been shown to be greedily trainable stack by stack, where each stack only has to learn to encode the previous one. The greedy method is then defined by [16] as "an algorithm that, with a certain goal in mind, will attempt at every stage to do whatever it can, whenever it can, to get nearer to that goal immediately. In other words the method surrenders a possible longer-term advantage in favour of an immediate move toward the objective."

4.2.1 Topology and Operating Principles

The RBMs are stacked so that the hidden layer of one RBM becomes the visible layer of the next RBM (see figure 4.2). The stacked architecture can then be trained to learn complex probabilistic models. In case the input data is real-valued, the first layer is then represented with a Gaussian-Bernoulli RBM and the rest of the layers with Bernoulli-Bernoulli RBMs.

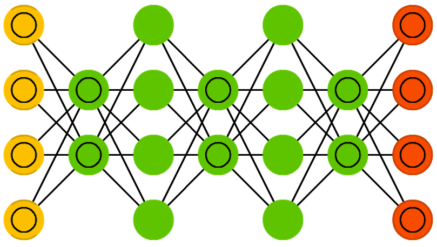


Figure 4.2: Figure depicting the topology of Deep Belief Network. Source: [3].

4.2.2 Training

First the DBN is trained layer by layer. Each RBM is trained individually with the contrastive divergence method from section 4.1.2 and its weights and biases are saved. Afterward, the hierarchy is treated as a feedforward neural network and is fine-tuned using the backpropagation algorithm from section 3.4.2 with the parameters initialized to the previously saved values.

4.2.3 Applications in Anomaly Detection

Authors in [39] show that DBN consisting of Bernoulli-Bernoulli RBM layers can be used effectively for semisupervised EEG anomaly detection. To ensure dimensionality reduction the DBN acts like an autoencoder, with progressively smaller encoding layers. Using the EEG data set, the DBN outperforms the state-of-the-art one-class Support Vector Machine.

The hypothesis in using a DBN for anomaly detection is that when the DBN does not see many anomalous training examples, it subsequently poorly reconstructs the outliers in the testing phase because it did not have the opportunity to learn to perform so correctly. Along those lines, the higher the reconstruction error, the more unusual the input sample. The reconstruction error, in this case, RMSE, for each sample, can be used with a threshold to create a classifier. Selecting an RMSE threshold k establishing the anomaly boundary can be viewed as a hyperparameter selection, which the authors search for along with DBN parameters using a validation set consisting of labeled anomalies. The authors also explore how the hierarchical feature layers of a DBN can be used to extract the anomalous features of individual samples; therefore it can shape the understanding of the very anomalies it detects.

4.3 Autoencoders

Autoencoder sometimes referred to as an autoassociator, is a multilayer ANN where the number of neurons in the output layer is the same as the number of input features. The goal of the network is first to encode the data (reduce dimensionality, compress), hence the name, and then reconstruct the data (generalization capabilities). It trains its parameters so that the values in the input layer are as close to the values of the output layer as possible.

4.3.1 Topology and Operating Principles

The network's shape resembles an hourglass; as one can see in 4.3. The autoencoder's input layer represents the input values. It is followed by one or more hidden layers, which represent the transformation of the features and usually contain a smaller number of neurons than the input layer. The output

layer then has the same number of neurons as the input layer and should correspond with the input values because of the reconstruction.

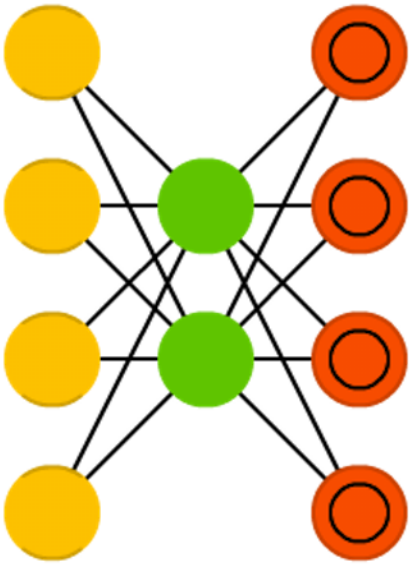


Figure 4.3: Figure depicting the Autoencoder. Source: [3].

Autoencoders are usually symmetrical around the middle layer. The middle layer is called the code and is usually the smallest; thus it is the place where the information is most compressed. The part up to the middle encodes, while the part after the middle decodes.

In an ideal case, the following applies to the autoencoder's operation after its training process:

$$\begin{aligned} \mathbf{code} &= encode(\mathbf{x}) \\ \mathbf{x} &\approx decode(\mathbf{code}) \end{aligned} \tag{4.12}$$

, where $\mathbf{x}$ is the input vector.

The autoencoder is forced to compress the input data into lower dimensional subspace because the hidden layers have a smaller number of neurons compared to the first layer. It does so by saving the information about feature relations in the weights of the hidden layers. The autoencoders can perform nonlinear dimensionality reduction without any a priori knowledge or assumptions. The suitability of such technique manifests itself when the input variables are nonlinearly correlated or temporally dependent, which would not be tractable for methods of linear dimensionality reduction, e.g. principal components analysis.

The following improvements to the basic autoencoder architecture were also introduced in the recent years:

- One can add a costraint on the hidden layers which penalizes high values of total neurons' activations. This constraint is called a *sparsity regularizer*. In effect, it manifests itself by forcing most neurons' activations very near to zero. It is believed that this regularizer improves autoencoder's generalization capabilities. [34]

- Denoising autoencoder tries to learn the identity from a corrupted input. Thus, the autoencoder is trained to undo the effect of the corruption process. The corruption can be caused either by a Gaussian or salt-and-pepper-noise. One can also cause the corruption by randomly forcing some of the input features to 0. This form of corruption is called a *dropout*. Similarly to the sparsity regularizer, it is believed that the denoising autoencoder forces the hidden layers to discover more robust features. [40]

- Bengio in [40] introduces a stacked denoising autoencoder. He builds the network by successively stacking layers of regular denoising autoencoders. The representation, code, of one autoencoder then becomes the input of the succeeding layer. The layer-by-layer training is performed similarly to a DBN in section 4.2.2. Once the separate layers are trained, the fine-tuning is performed with a backpropagation algorithm from section 3.4.2. The stacking is believed to improve the autoencoder's learning speed. This is counterintuitive as the training process of the stacked autoencoder includes the extra pretraining compared to the regular autoencoder. However, the speedup is achieved by much earlier fulfillment of the stopping condition in backpropagation algorithm 3.1. The stacked autoencoder can also increase the network's performance by suitably initializing the weights and thus bypassing the problem of deteriorating weights in backpropagation.

4.3.2 Training

Even though the method is not presented with labeled data, the network is trained in a supervised-like manner as the method actually possesses the desired output of the network in the form of original input vectors. These types of networks are called self-supervised. Therefore the network can be trained by the backpropagation method from section 3.4.2.

In the learning phase, the algorithm tries to find such weights so that the equation 4.12 holds true. The method attempts to minimize the reconstruction error, i.e. the difference between the data obtained after an autoencoder pass and the original data. The recalling part of the network works the same way as a regular forward pass.

4.3.3 Applications in Anomaly Detection

A sample is considered anomalous when the model recreates the input data with substantial reconstruction error. However, to choose the correct anomalies, one also needs to set a threshold for the reconstruction error.

Japkowicz et al. in [41] present a threshold setting algorithm. The authors use labeled data and utilize the labels for the threshold selection. They propose two algorithms: one for the noiseless data, which requires only one type of data points and one for the noisy data, which requires both normal and outlier data points. They believe that in the noisy data "the separation between [anomaly] and [outlier] data is not clear in that although the majority of [normal] instances have low reconstruction errors and the majority of [outlier] instances have high reconstruction errors, some [normal] examples have a high reconstruction error and some [outlier] examples have a low reconstruction error."

Surace and Worden in [42] used autoencoder to detect cracked beams. The authors proposed to add noise to the normal data set so as to force better generalization capabilities from the network. They polluted each data point independently by adding Gaussian noise with mean of 0 and standard deviation equal to 1% of the average feature value.

The authors also compared calculating the reconstruction error by means of Euclidean and Mahalanobis distance. The Mahalanobis distance is defined as:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \Sigma^{-1} (\mathbf{x} - \mathbf{y})} \tag{4.13}$$

, where Σ is the covariance matrix of the training data defined as:

$$\Sigma_{ij} = \mathrm{E}\big[(X_i - \mathrm{E}(X_i))(X_j - \mathrm{E}(X_i))\big] \tag{4.14}$$

To establish which of these two distances is more sensitive to the indication of anomaly, they formulated a Novelty Index Efficiency (NIE) indicator.

$$NIE = \frac{\bar{v}_{ANOMALY} - \bar{v}_{NORMAL}}{\bar{v}_{NORMAL}} \tag{4.15}$$

, where $\bar{v}$ is the average reconstruction error of the normal or anomaly class. Based on the experiments, they concluded that the Mahalanobis distance is more suited to make the distinction between normal and anomalous data.

4.4 Radial Basis Function Networks

As the name implies, the radial basis function network (RBFN) uses the radial basis function (RBF) from section 3.2.2. The structure of the network is shown in the picture 4.4. Its topology is similar to a feed-forward network except the basis and activation functions. The RBF neuron localizes the data points into a hypersphere whereas the perceptron globally separates the input space into two hyperplanes.

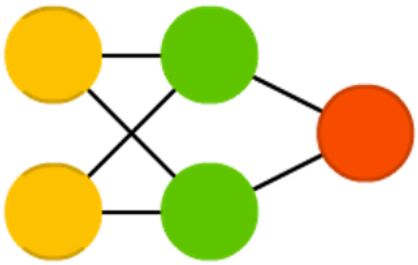


Figure 4.4: Figure depicting the radial basis function network. Notice the similarity to 3.6b as they only differ in the activation and basis functions. Source: [3].

4.4.1 Topology and Operating Principles

The RBF networks can be utilized in places where the usual perceptron networks experience difficulties. The RBFN usually consists of three layers. The first layer contains the input neurons; the second, hidden, layer is composed of the RBF neurons, and the third, output, layer contains linear basis function neurons with identity function as activation. Similarly to a feed-forward network, the radial basis function neuron defines weights $\mathbf{c}$ and one extra coefficient for each neuron, which in the case of radial basis neuron is not the bias but the width b. The weights of the RBF neuron, however, determine the center for the RBF. In general, it calculates the Euclidean distance between the input vector and the middle of the hypersphere. [20]

The neuron's inner potential can therefore be calculated as [20]:

$$g(\mathbf{x}) = \frac{\|\mathbf{x} - \mathbf{c}\|}{b} \tag{4.16}$$

Combining with the Gaussian activation function yields the following formula:

$$f(x) = e^{-\frac{\|\mathbf{x}-\mathbf{c}\|}{b}} \tag{4.17}$$

The final activation of the i^{th} output neuron of the described network can be formulated as:

$$y_j = W_{0,j} + \sum_{i=1}^{n_h} W_{i,j} e^{-\frac{\|\mathbf{x}-\mathbf{c_i}\|}{b_i}} \tag{4.18}$$

4.4.2 Training

The learning is described based on the explanation from [20]. The learning process of the RBFN differs from the FFN. The algorithm needs to determine the coordinates of the hyperspheres c_i and its radii b_i. Each of such hyperspheres belongs to one neuron of the RBFN. The algorithm also needs to set the weights associated with the output neurons.

In the first stage, all the hypercubes' parameters are chosen. Only in the second stage, can the algorithm modify the weights of the output layer. It is crucial to select an algorithm that finds the centers of the hyperspheres. One such candidate is the k-means algorithm 2.2.2.1.

The number k of desired clusters matches the number of RBF neurons. The number of neurons is either decided by an expert designer or by algorithmic means. The inputs of the k-means algorithm are then all the training samples. The output of the k-means algorithm is information about each cluster.

After finding the center of the clusters with the k-means algorithm, one needs to calculate the radius of the hypersphere, which is the standard deviation of all the points belonging to one cluster. Afterward, one searches for the optimal weights of the output layer. The output layer can be trained, among other methods, with the described backpropagation algorithm from section 3.4.2.

4.4.3 Applications in Anomaly Detection

The earliest observation was that anomalies presented to the RBFN result in low values at each of the network's output nodes. A notable disadvantage of such techniques is that the right threshold for anomaly alert has to be chosen, thus creating another problem to tackle and optimize. [36]

By adding reverse connections from the output layer to the central layer Albrecht in [43] shows how a generalized radial basis function can self-organize to form a Bayesian classifier that is also capable of novelty detection.

Jakubek and Strasser in [44] propose multi-step neural-network based anomaly detection which in the neural network topology uses ellipsoidal basis function. The advantage of these functions is that they can be fitted to the data with more accuracy than RBFs. Results from experiments showed that the proposed network uses fewer basis functions than an RBF network of equivalent accuracy.

4.5 Other Neural Network Models

Among other neural network based methods belong [18]:

- *Multilayer Perceptron* might map the input features to a smaller number of output nodes. Thus they achieve theoretically dimensionality reduction as well as clustering of the data into fixed number of partitions. The most common way to determine if a test point is an anomaly or not is by thresholding the activation of the output nodes. If none of the output nodes are activated above a certain level, the input test data point is declared to be an outlier. One disadvantage with multilayer perceptron is that they tend to generalize and assign even a novel instance to one of the learned classes, thus sometimes leading to missed outliers.

- *Self-organizing Maps* (SOM), also called Kohonen maps are unsupervised neural networks which cluster the input data into a fixed number of partitions. The SOM is a neural network with a grid-like architecture that is primarily used for identifying clusters in a dataset and dimensionality reduction. When normal data are used to train a SOM, it creates a kernel-based representation of the normal behavior which can be used for novelty detection.
- *Hopfield Networks* uses binary threshold nodes with recurrent connections between them. Similarly to Boltzmann machines, it is an energy based system. The value of the energy function is then lower for normal points and higher for novel points.

Chapter 5

Implementation

5.1 Choice of Language and Technologies

The choice of language mainly depends on the purpose and requirements of the application. Having in mind that this work is not necessarily focused on production-ready implementation but more on prototyping and confirmation or disproval of different ideas, I chose Python [45].

Python is an open-source dynamically typed object-oriented scripting language. It is multiplatform and contains a significant number of free library packages. Python is considered to contain the most open-source machine learning packages. The packages are often build hierarchically, i.e. the more abstract functionalities depend on well written and highly optimised lower-level functions from other packages (which are often written partly in C), e.g. Numpy and Pandas. Their integration works out of the box and thus makes the programmer's life easier. Among other advantages belong the language's simplicity and productivity. The source of common inconveniences is the fact that Python 2 and 3 are not compatible in all parts of the language.

5.1.1 Used Technologies

Python scientific, machine learning and visualization libraries that are utilized in the prototypes:

- **Numpy** [46] is a Python library used for scientific computing. Numpy among other algorithms and functions contains efficient N-dimensional array implementation, linear algebra algorithms, Fourier transform and many others.
- **Pandas** [47] contains a higher level abstraction of matrix functions compared to Numpy.
- **matplotlib** [48] is a comprehensive 2D plotting library.

- **SciPy** [49] is a collection of libraries which include matplotlib, pandas, numpy and others.

Python frameworks and environments that were used:

- **H_2O Deep Learning** [50] is a scalable framework that can advantage of large clusters of computing nodes for efficient and correct utilization of the current research into deep neural networks. Currently, it can be used with Java, Python or R. The user can decide in what operating mode a dataset will be propagated – every node can work with the entire dataset, each node can have a different part of the dataset, or the framework can operate in a strict single node mode.

  One can set regularization techniques (L1 and L2), dropout, momentum, cross-validation, grid search and also different activation functions as the framework's parameters. The framework also has a useful checkpointing functionality, where the user can stop and continue the training of the neural networks as he pleases. It also contains a basic implementation of autoencoder which will be used heavily in the experimental phase.

- **Jupyter Notebook** [51] is an online development environment which facilitates fast prototyping in multiple languages (mainly in Python).

Non-python environments and tools that were used:

- **Docker** [52] is a popular software containerization environment.

- **Apache Spark**[3] [53] is an open-source cluster computing framework. Spark provides an interface for dealing with entire clusters and guarantees out-of-the-box data parallelism and fault-tolerance.

- **Gitlab** [54] is a web-based open-source Git repository manager.

I also followed the established guidelines that DataMole[4] produced.

5.1.2 Computing Cluster

The computing cluster that was used for the experiments is called *Lely Cluster*. It contains two Intel Xeon E5-2670 v3 CPUs. Each CPU then consists of 12 physical and 24 logical cores with a frequency of 2.30GHz. The operation system is Ubuntu 16.04.1 and the system has 251 GB of main memory.

[3] Although users refer to the framework mostly as Spark. The full name serves a distinguishing purpose here as there is a great number of sparks in the industry (Java framework, iOS application, programming language). (See: `https://en.wikipedia.org/wiki/Spark`).
[4] See the company's website on: `https://www.datamole.cz/`

5.2 Experiments

The second, smaller, goal of the Thesis is to analyze the impact of hyperparameters' selection on generalization capabilities, effectivity of the training and performance of at least one ANN architecture. The architectures chosen are autoencoder and also the stacked autoencoder. The experiments were performed in the semi-supervised setting, i.e. the autoencoder was trained only on the normal classes and later tested on a combination of both the anomaly and normal classes. The performance of different model's was evaluated by the area under the receiver operating characteristic when the threshold for anomaly determination is varied. The particular goals set out for the experimental part of the thesis are the following:

- Evaluate the following hyperparameters' choice impact on model's performance in different datasets:
  - number of neurons and number of hidden layers,
  - sparsity,
  - input corruption, i.e. denoising autoencoder.
- Evaluate the learning speed and performance of stacked autoencoder compared to a regular autoencoder.
- Decide if Mahalanobis distance proposed for thresholding the outliers in paper [42] achieves higher accuracy compared to the Euclidean distance.

5.2.1 Related Work

The biggest obstacle at the beginning was to correctly define the methodologies, tests and data sets for such act. The search for papers which attempted to attain the same goal was futile as at that moment there were none. Only during the last weeks of the time assigned, did I find a fitting work which was created in the fall of 2016. Its name is Hyperparameter Selection for Anomaly Detection with Stacked Autoencoders – a Deep Learning Application [55]. However, I managed only to obtain the poster of this Bachelor Thesis. I contacted the people at Technische Hochschule in Brandenburg by email asking for the text of the Thesis. Unfortunately, my request was denied as the work was contractual with the Zeiss company. Also, the law in Germany differs as they are not obliged to make the Bachelor thesis publicly available [56].

5.2.2 Datasets

There is a webpage created by Shebuti Rayana [57] which assembles datasets that can be used for testing different anomaly detection techniques. The author uses not only pure outlier datasets but also datasets that have high

class imbalances. He utilizes such datasets by merging different classes into outlier or normal class and thus creates a binary dataset that is suitable for anomaly detection techniques.

The datasets were chosen so that the its dimensionalities and anomaly ratios are diverse. The following datasets were used for the experiments:

- *Forest cover* is a multiclass classification dataset. It is used in predicting forest cover type from cartographic variables only.
- *Http and smtp* datasets contain data that should be used to predict network intrusions. The original dataset is divided into two parts based on the network protocols the recorded requests use.
- *musk* contains information about different musk and non-musk classes. Shebuti Rayana used musk classes 213 and 211 as outliers.
- *satellite* is a multiclass classification dataset of satellite images. The author combined the smallest three classes to form the outlier class.
- *shuttle* is a multi-class classification dataset that contains information about space shuttle measurements. The smallest five classes are combined to form the outlier class.

The dimensionality information and class balances are recorded in the table 5.1.

5.2.3 Training Speed

The training speed of different models was measured in all parts of the experiment. It is understandable that the biggest impact on the training speed is caused by the number of data points and the number of their features.

The learning time was not affected by neither sparsity nor noise corruption. The number of neurons in the hidden layers did not, however, impact the learning speed as much as the number of layers, although it still had bigger impact than sparsity or noise corruption.

Lastly, the learning speed of stacked autoencoder and regular autoencoder were compared. As explained in section 4.3.1, the assumption was that the

Table 5.1: Information about the datasets that were used in the experiments.

Dataset	**#points**	**#dimensions**	**#outliers (%)**
Musk	3062	166	97 (3.2%)
Shuttle	49097	9	3511 (7%)
Satellite	6435	36	2036 (32%)
Http	567479	3	2211 (0.4%)
Smtp	95156	3	30 (0.03%)

stacked autoencoder would train faster. This was disproved as in almost all cases, the stacked autoencoder learned slower as can be seen in the table 5.3.

5.2.3.1 Comparison of Stacked Autoencoder and Regular Autoencoder

As explained in section 4.3.1, the layer-per-layer pretraining might lead to a higher performance of the stacked autoencoder. The results on all the datasets are very similar, in that the stacked autoencoder does not systematically achieve better performance.

5.2.3.2 Comparison of the Thresholding Capabilities of the Mahalanobis and Euclidean Distance

Surace in [42] achieved better results with the Mahalanobis distance used for reconstruction error and subsequently for the thresholding of anomaly determination. I was able to replicate the results only on the shuttle dataset and completely disprove on the smtp_kdd dataset. Such sharp decisions were reached because the area under ROC that used the Mahalanobis distance was always greater, or lower than the one that used the Euclidean distance. The results in the rest of the datasets varied greatly between runs and other parameter selections.

It is therefore advisable to evaluate the possibility of choosing the Mahalanobis distance for the reconstruction error calculation. However, one should not take this advice with grain of salt and inspect both options.

5.2.3.3 Hyperparameters Search

The hyperparameter search was performed with *grid search*. Grid search is an exhaustive search through a specified subset of the hyperparameter space. The real-valued parameters must be discretized before the application of grid search. Disadvantage of the search is that it is exponential with the number of explored features.

Table 5.2: The table shows the development of the learning speed of the regular autoencoder based on the number of hidden layers. The data for the table is acquired from the musk dataset runs.

Hidden Layer Count	Learning Time
1	73.426231
2	105.808226
3	200.344532

The results are inconclusive as the performance of the different hyperparameter choices varied greatly. The biggest impact on the performance was the selection of the number of hidden layers, followed by the number of neurons. The sparsity and input corruption's variance was so great that it is impossible to reach any conclusions regarding these hyperparameters.

5.2.4 Reproducibility

Unfortunately, the exact reproducibility of the experimental results is troublesome as the H2O framework uses deliberate race conditions in the form of Hogwild! stochastic gradient descent. However, the chance of repeatability is increased by seeding model's initialization process with the same value for all runs. This value was arbitraly chosen to be 578902.

Table 5.3: Comparison of stacked autoencoder training time and regular autoencoder training time on the shuttle dataset in semi-supervised mode.

Grid Search Iteration Number	Stacked Autoencoder Training Time	Regular Autoencoder Training Time
0	66.620332	52.940831
1	70.404195	46.103253
2	67.797991	51.434634
3	68.385448	59.164770
4	67.899863	56.315683
5	69.670865	49.988830
6	58.064238	58.371360
7	54.976591	48.128357
8	59.359946	47.074302
9	61.977139	55.740820
10	64.128959	53.600283
11	63.080530	49.108493
12	55.449193	44.620188
13	65.151406	48.654592
14	57.763715	51.857390
15	62.979031	50.982999
16	58.783786	51.800387
17	55.349647	45.291698
18	56.718359	54.203642
19	57.085396	47.282607
20	56.335313	43.394289
21	66.084140	45.591142

Conclusion

This work describes in great detail artificial neural networks used for the anomaly detection task. The radial basis function network, autoencoder, deep belief network and restricted Boltzmann machines are described in great detail. The current state of the survey work is of introductory and broader general overview character. Therefore, the next natural step would be to delve more into detail of the different neural network anomaly detection methods so that a review as complex as [36] might be created with more up-to-date methods.

The thesis also provides experimental advice on hyperparameter and topology selection for the regular and stacked autoencoder. The experimental part of the work is written in Python. However, some of the experimental conclusions are inconclusive and would deserve a closer examination. Also, the results achieved in the case of anomaly thresholding based on different distances and the stacked autoencoder are surprising as they do not match the results of other authors. The practical part of the Thesis deserves to be broadened and extended on the side of tests, methodologies and also methods. The goals mentioned might be fulfilled in future Master's thesis

Bibliography

[1] Chandola, V.; Banerjee, A.; Kumar, V. Outlier detection: A survey. *ACM Computing Surveys*, 2007.

[2] Woodford, C. *neuron-axon-dendrites.gif*. [Online; accessed 10-Jan-2017]. Available from: `http://cdn4.explainthatstuff.com/neuron-axon-dendrites.gif`

[3] Veen, F. v. *The Neural Network Zoo*. Sept. 2016, [Online; accessed 10-Jan-2017]. Available from: `http://www.asimovinstitute.org/neural-network-zoo/`

[4] Neflix developers. *Netflix Prize: Home*. [Online; accessed 10-Jan-2017]. Available from: `http://www.netflixprize.com/`

[5] Munoz, A. Machine Learning and Optimization. *URL: https://www. cims. nyu. edu/ munoz/files/ml_optimization. pdf*, 2014.

[6] Umratkar, S.; Karnewar, J. K-Means Algorithm for Selective Filtration of Malicious Android Mobile Applications. 2014.

[7] McClendon, L.; Meghanathan, N. Using Machine Learning Algorithms to Analyze Crime Data. *Machine Learning and Applications: An International Journal (MLAIJ) vol*, volume 2.

[8] Vineyard, C. M.; Verzi, S. J.; James, C. D.; et al. Repeated play of the svm game as a means of adaptive classification. In *2015 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2015, pp. 1–8.

[9] contributors, W. *Machine learning*. Dec. 2016, [Online; accessed 10-Jan-2017]. Available from: `https://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=754624898`

[10] McCarthy, J.; Feigenbaum, E. A. In memoriam: Arthur samuel: Pioneer in machine learning. *AI Magazine*, volume 11, no. 3, 1990: p. 10.

[11] Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, volume 3, no. 3, 1959: pp. 210–229.

[12] Simon, P. *Too Big to Ignore: The Business Case for Big Data*. John Wiley & Sons, Mar. 2013, ISBN 978-1-118-63817-0.

[13] Holehouse, A. *01_02_Introduction_regression_analysis_and_gradient_descent*. [Online; accessed 10-Jan-2017]. Available from: `http://www.holehouse.org/mlclass/01_02_Introduction_regression_analysis_and_gr.html`

[14] Bishop, C. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2007, ISBN 0-387-31073-8.

[15] Russell, S. J.; Norvig, P.; Canny, J. F.; et al. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.

[16] Butterfield, A.; Ngondi, G. *A Dictionary of Computer Science*. Oxford Paperback Reference, Oxford University Press, 2016, ISBN 9780199688975. Available from: `https://books.google.cz/books?id=GDgICwAAQBAJ`

[17] Hawkins, D. M. *Identification of outliers*, volume 11. Springer, 1980.

[18] Pimentel, M. A.; Clifton, D. A.; Clifton, L.; et al. A review of novelty detection. *Signal Processing*, volume 99, 2014: pp. 215–249.

[19] Zhang, Y.; Meratnia, N.; Havinga, P. A taxonomy framework for unsupervised outlier detection techniques for multi-type data sets. 2007.

[20] Šíma, J.; Neruda, R. *Teoretické otázky neuronových sítí*. Matfyzpress, 1996, ISBN 978-80-85863-18-5.

[21] Stanford Vision Lab. *ImageNet*. [Online; accessed 10-Jan-2017]. Available from: `http://image-net.org/about-overview`

[22] Mastin, L. *Neurons & Synapses - Memory & the Brain - The Human Memory*. [Online; accessed 10-Jan-2017]. Available from: `http://www.human-memory.net/brain_neurons.html`

[23] McCulloch, W. S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, volume 5, no. 4, 1943: pp. 115–133.

[24] Hebb, D. O. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[25] Rosenblatt, F. The Perceptron: A Probabilistic Model For Information Storage And Organization in the Brain. *ResearchGate*, volume 65, no. 6, Dec. 1958: pp. 386–408, ISSN 0033-295X, doi: 10.1037/h0042519. Available from: https://www.researchgate.net/publication/9964510_The_Perceptron_A_Probabilistic_Model_For_Information_Storage_And_Organization_in_the_Brain

[26] Papert, M. M. S.; Minsky, M. Perceptrons: an introduction to computational geometry. *Expanded Edition*, 1969.

[27] Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. *Learning internal representations by error propagation*. Technical report, DTIC Document, 1985.

[28] EPFL. *Bluebrain | EPFL*. [Online; accessed 10-Jan-2017]. Available from: http://bluebrain.epfl.ch/

[29] Markram, H.; Muller, E.; Ramaswamy, S.; et al. Reconstruction and Simulation of Neocortical Microcircuitry. *Cell*, volume 163, no. 2, Oct. 2015: pp. 456–492, ISSN 0092-8674, 1097-4172, doi: 10.1016/j.cell.2015.09.029. Available from: http://www.cell.com/cell/abstract/S0092-8674(15)01191-5

[30] Franklin, S.; Garzon, M. Neural computability. *Progress in neural networks*, volume 1, no. 128,144, 1990.

[31] Nair, V.; Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.

[32] Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural networks*, volume 4, no. 2, 1991: pp. 251–257.

[33] Bengio, Y. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, Springer, 2012, pp. 437–478.

[34] Ng, A. *CS294A/CS294W - Unsupervised Deep Learning*. [Online; accessed 10-Jan-2017]. Available from: https://web.stanford.edu/class/cs294a/handouts.html

[35] Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; et al. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[36] Markou, M.; Singh, S. Novelty detection: a review—part 2:: neural network based approaches. *Signal processing*, volume 83, no. 12, 2003: pp. 2499–2521.

[37] Hinton, G. A practical guide to training restricted Boltzmann machines. *Momentum*, volume 9, no. 1, 2010: p. 926.

[38] Fiore, U.; Palmieri, F.; Castiglione, A.; et al. Network anomaly detection with the restricted Boltzmann machine. *Neurocomputing*, volume 122, 2013: pp. 13–23.

[39] Wulsin, D.; Blanco, J.; Mani, R.; et al. Semi-supervised anomaly detection for EEG waveforms using deep belief nets. In *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*, IEEE, 2010, pp. 436–441.

[40] Vincent, P.; Larochelle, H.; Lajoie, I.; et al. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, volume 11, no. Dec, 2010: pp. 3371–3408.

[41] Japkowicz, N.; Myers, C.; Gluck, M.; et al. A novelty detection approach to classification. In *IJCAI*, volume 1, 1995, pp. 518–523.

[42] Surace, C.; Worden, K.; Tomlinson, G. A novelty detection approach to diagnose damage in a cracked beam. In *PROCEEDINGS-SPIE THE INTERNATIONAL SOCIETY FOR OPTICAL ENGINEERING*, Citeseer, 1997, pp. 947–953.

[43] Albrecht, S.; Busch, J.; Kloppenburg, M.; et al. Generalized radial basis function networks for classification and novelty detection: self-organization of optimal Bayesian decision. *Neural Networks*, volume 13, no. 10, 2000: pp. 1075–1093.

[44] Jakubek, S.; Strasser, T. Fault-diagnosis using neural networks with ellipsoidal basis functions. May 2002, pp. 3846–3851 vol.5, doi:10.1109/ACC.2002.1024528.

[45] Python developers. *Welcome to Python.org*. [Online; accessed 10-Jan-2017]. Available from: `https://www.python.org/`

[46] NumPy developers. *NumPy — NumPy*. [Online; accessed 10-Jan-2017]. Available from: `http://www.numpy.org/`

[47] pandas development team. *Python Data Analysis Library — pandas: Python Data Analysis Library*. [Online; accessed 10-Jan-2017]. Available from: `http://pandas.pydata.org/`

[48] matplotlib development team. *matplotlib: python plotting — Matplotlib 1.5.3 documentation*. [Online; accessed 10-Jan-2017]. Available from: `http://matplotlib.org/`

[49] SciPy developers. *SciPy.org — SciPy.org*. [Online; accessed 10-Jan-2017]. Available from: https://www.scipy.org/

[50] Candel, A.; LeDell, E.; Parmar, V.; et al. *Deep Learning with H2O*. H2O.ai, Inc, fifth edition. Available from: http://docs.h2o.ai/h2o/latest-stable/h2o-docs/booklets/DeepLearningBooklet.pdf

[51] Jupyter developers. *Project Jupyter*. [Online; accessed 10-Jan-2017]. Available from: http://www.jupyter.org

[52] Docker contributor. *Docker*. [Online; accessed 10-Jan-2017]. Available from: https://www.docker.com/

[53] The Apache Software Foundation. *Apache Spark™- Lightning-Fast Cluster Computing*. [Online; accessed 10-Jan-2017]. Available from: http://spark.apache.org/

[54] Gitlab contributors. *GitLab - Official Website*. [Online; accessed 10-Jan-2017]. Available from: https://about.gitlab.com/

[55] Meyer, T. *Hyperparameter Selection for Anomaly Detection with Stacked Autoencoders – a Deep Learning Application*. [Online; accessed 10-Jan-2017]. Available from: http://ots.fh-brandenburg.de/downloads/abschlussarbeiten/2016-10-14%20pl_tobias_meyer.pdf

[56] *Studienabschlussarbeit*. Mar. 2016, [Online; accessed 10-Jan-2017]. Available from: https://de.wikipedia.org/w/index.php?title=Studienabschlussarbeit&oldid=152710864

[57] Rayana, S. *ODDS – Outlier Detection DataSets*. [Online; accessed 10-Jan-2017]. Available from: http://odds.cs.stonybrook.edu/

Appendix A

Acronyms

ANN Artificial Neural Network

BBP Blue Brain Project

BM Boltzmann Machine

CD Contrastive Divergence

DBN Deep Belief Network

FFN Feedforward Network

FN False Negative

FP False Positive

FPR False Positive Rate

GPU Graphics Processing Unit

LBF Linear Basis Function

MAE Mean Absolute Error

MAPE Mean Absolute Percentage Error

ME Mean Error

MSE Mean Squared Error

NN Neural Network

RBF Radial Basis Function

RBFN Radial Basis Function Network

RBM Restricted Boltzmann Machine

RMSE Root Mean Square Error

ROC Receiving Operator Characteristic

TN True Negative

TP True Positive

TPR True Positive Rate

SOM Self-organizing Map

Appendix B

Contents of enclosed CD

`readme.txt` ..... file with CD contents description
- `datasets` ..... directory containing the datasets that were used for experiments
- `reports` .. directory containing the html reports from the experiments
- `src` ..... directory containing source codes
- `latex_source` .... directory containing latex source code of the Thesis
- `BP_Rymes_Karel_2017.pdf` ..... thesis text in PDF format