Juho Ojala

Machine learning in automating bank statement postings

Helsinki Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master's Thesis

9 November 2018

Helsinki Metropolia University of Applied Sciences

PREFACE

Seems like it's finally done. Though I was always confident about building the technical solution, I was not confident in my ability to transfer the technical implementation into words for this thesis paper. However, after finishing this project I feel like I have become more comfortable with academic writing and that planning a good structure is the most important step of writing.

The thesis topic certainly entertained me with several challenges, starting from the business problem to handling of the data, figuring out an effective way to do iterations and evaluating the results.

I learned a lot about challenges that come when there is a lot of data. I also became more familiar with the tools I was working with and noticed what was lacking in the process. There was a lot to be still improved, but the line where to call it good enough had to be drawn somewhere and of course the employer had strong say in that as well.

I would like to thank all my instructors at Metropolia for providing great classes during the master's studies, my employer and especially thesis supervisor (employer side) Juha Havu for helping me arrange the time to implement the thesis project.

Espoo, 9.11.18
Juho Ojala

Author(s) Title Number of Pages Date	Juho Ojala Machine learning in bank statement ledger posting 44 pages 9 November 2018
Degree	Master of Engineering
Degree Programme	Information Technology
Specialisation option	Networking and Services
Instructor(s)	Juha Havu, Head of Software Engineering, MSc Antti Koivumäki, Principal Lecturer

The thesis studies the possibility to use machine learning in bank statement ledger posting. It's a feature that could potentially save a lot of manual work from accountants. The scope was restricted to determining ledger account instead of the full ledger posting. The goal was to build a proof of concept, rather than a production ready software. A secondary objective was to study potential tech choices for future use in the company for machine learning projects.

Creating a functional set of ledger posting rules for thousands of different companies with the ability to change over time is something that is very difficult and arduous to do using traditional programming techniques and provided a great opportunity to test machine learning concepts.

A proof of concept was built using a combination of machine learning techniques and some traditional programming fine-tuning it. This thesis covers the applied theory, materials and methods that were used, the detailed description of the solution that was built, an evaluation of the software performance and conclusions that can be drawn from those.

The results of the thesis indicate that some companies are more applicable for this kind of automatic processing than others and for production ready usage it is important to do better categorization of the companies. Overall the problem is more complex than initially expected and requires further development for production ready solution. Additionally, tech choices used were good and scaling computing power can be considered a requirement for effective machine learning development with larger amounts of data.

Keywords	Machine learning, Accounting, Bank statement, TensorFlow, Neural networks

Table of Contents

List of Abbreviations

API	Application programming interface
BIC	Bank identifier code
POC	Proof of concept
PRH	Patentti- ja rekisterihallitus
ReLu	Rectified linear units
VAT	Value Added Tax

1 Introduction

The company for which the thesis is conducted for is Accountor Finago Oy. Accountor Finago is a pioneer in digitalized financial management and it's softwares are being used already by over 100000 companies in the Nordics. One of main products for Accountor Finago is Finago Procountor, a financial management software that brings accountants to the same platform with their customers in cloud. Client companies can manage and pay their invoices, see their bank statements, see different kinds of reports and lots of other features related to financial management through the software. The accountants then can perform accounting directly related to those invoices and payments through the same software. In the usual case the client company handles all their invoices and other payments through Procountor and accountants can do accounting based on these and very little work is required. However, in the case where the client company is doing payment-based accounting, the accountant is working mostly based on the bank statements. To ease the pain of doing manually the accounting there is a feature for manually making ledger posting rules. The problem with this approach is that it's tedious and unfeasible to manually create all the rules for the client company. In addition, the rules are only applicable for the future bank statements and not for the ones already present. Furthermore, they don't adjust to changes over time automatically. They also have certain limitations and are not adaptable for all situations.

Accounting rules for the ledger posting are dependent on large number of factors depending on the company and the information about the bank statement event. For this reason, a traditional programming cannot provide a solution to this problem. Machine learning, also known as inductive learning however, can potentially provide a solution to the problem by learning all the changing factors and by being able to react to changes over time.

The research focuses on tackling this problem by building a machine learning proof of concept, which provides a ledger account for the bank statement events. Determining the ledger account is a crucial part for the full ledger posting process. Each company has a defined chart of accounts, where the ledger account numbers usually vary between 1000-9999, however there are exceptions to this chart as well, and there are sometimes big differences how the transactions are marked into the accounts. The outcome proof

of concept aims to answer the questions: is it feasible to use machine learning for this business problem.

The outcome is a Machine learning algorithm, which can determine ledger account for bank statement transaction and an evaluation in form of this thesis. The other values of postings are not included in this thesis, as they would push the scope too big.

The workflow strongly follows the ordering of the sections. The following Figure 1: Research plan, provides some insight to the process. The study started by conducting interviews with an accountant who is experienced with Procountor and handling the payment-based accounting. After the problem was defined, some preliminary research was done on possible technologies as well as the availability of existing data. The next step was creating data pipelines for preprocessing the data and reading it with the learning algorithm. After the machine learning model was implemented. The machine learning model as well as the data preprocessing steps are iterated repeatedly based on the evaluation results.

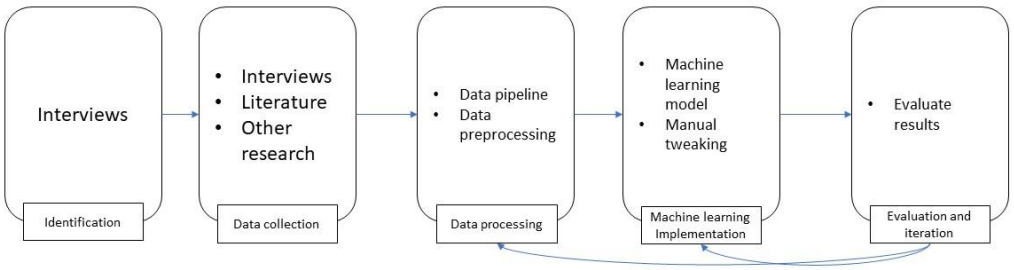

Figure 1: Research plan

This thesis is divided into 6 sections. The first section, “Introduction”, introduces the problem, scope of the solution and the outcome. The second section, “Theory”, provides

background knowledge about the topic, which is needed for understanding the work done in this thesis. Third chapter, “Materials and methods”, goes through materials and methods including data, evaluation methods and oversights. Fourth section, “Solution building”, is all about the machine learning solution that solves the ledger account. Fifth chapter, “Solution evaluation”, analyzes the results analytically. Sixth section, “Conclusions and discussion”, provides conclusions from the results and discusses potential future improvements.

2 Theory

This chapter focuses on going through the fundamentals for understanding the thesis. It provides enough information to understand the upcoming parts, without going too deep into the details. The chapter also discusses tools that were chosen one-by-one and why that specific tool was selected for the job.

2.1 Machine learning

Machine learning, as the name suggests has the program learning from data. This is similar to how humans learn from experiences [1]. Typically, programmers have been creating fine-tuned algorithms to solve the problems at hand, but machine learning brings a new approach where the program can learn on its own. Saying that the approach is completely new would be lie, as one of the first papers written on the topic was already published 1956 by Ray Solomonoff, titled "An Inductive Inference Machine". The reason why it has become a recent phenomenon is due mainly because of two things: The increase in computing power and most importantly: the increase in the data volumes [2]. During this age of digitalization, organizations have a lot of data available in digital format and thus enabling machine learning in a completely new scale.

Machine learning is but a single branch of artificial intelligence and should not be confused with the general term AI, which includes other subsections as well. Machine learning itself can be split into three different categories: supervised learning, unsupervised learning and reinforcement learning [3]. Unsupervised learning is all about modeling data that has no labeled training data [4], meaning that we don’t know the answers for our sample data beforehand. With the problem at hand, we have labels available and hence

it's not a suitable unsupervised learning problem. Reinforcement learning is all about exploration and exploitation. In this learning model the environment is a teacher and the algorithm is constantly balancing between exploring new options or exploiting earlier discovered rewarding behavior [5]. In the business problem at hand, exploring is not beneficial as the desired way of functioning is about finding the same solutions in same situations repeatedly based on the labeled data. That leaves one more category: supervised learning. In supervised learning, the program will learn from laid out examples [3]. The data needs to be labeled, meaning that the answer to the examples must be known [3]. This data can be used to learn the patterns that lead to the result and can then be reapplied in real-life scenarios to predict the answers. This thesis focuses mostly on supervised learning. There are a lot of different methods for supervised learning, including decision trees, decision forests and many others, but one of the most popular ones is neural networks, which will be the focus in this thesis. The reason for choosing neural networks is due to general popularity as well as many studies pointing at its effectivity in multi-class classification, one such study was presented in Computer Science and Engineering International Conference 2017 [6].

2.1.1 Neural networks

Artificial Neural networks, or often within the context just neural networks, are named after the biological neural network due to some similarity in functionality. Neural networks are built from layers, which consist of neurons. The neurons of a layer are connecting with the neurons of the next layer, transferring data forward. Input layer receives the original data and passes it onwards to a hidden layer. Hidden layer will perform calculations on the data and pass it forward, either to another hidden layer or to an output layer. Output layer will produce the final output of the network. There may be n-number of hidden layers in the neural network model. A model with a lot of hidden layers is often considered a deep neural network, due to the depth of the layers. In each layer, there are n-number of neurons [2]. See Figure 2 below for a visualization of a simple, fully connected neural network.

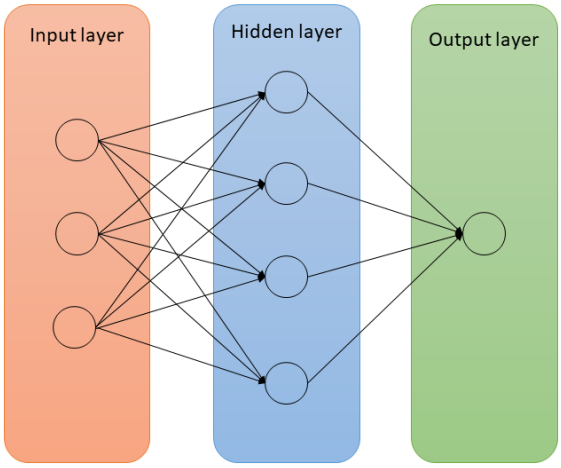

Figure 2: Simple fully connected neural network

Some of the most common neural network types are fully connected neural networks (dense), convolutional neural networks (CNN) and recurrent neural networks (RNN). Despite the names indicating that each one of them is a single network, a single neural network can contain elements from all three.

Generally, in fully connected layer the operation performed by a hidden neuron is $\sum_{i=1}^{n} x_i w_i + b$ [2]. This function will allow us to figure out linear calculations and to introduce non-linearity to it, an activation function is applied [2]. An example of an activation function is ReLu (rectified linear unit), in which $f(x) = max(0, x)$. So, the final output from a neuron could be for example $f(\sum_{i=1}^{n} x_i w_i + b)$, where f is ReLu. The fully connected network gets its name from the fact that all the neurons of layer n – 1 are connected to all neurons of layer n. Figure 3 below demonstrates what happens in each cell during forward pass.

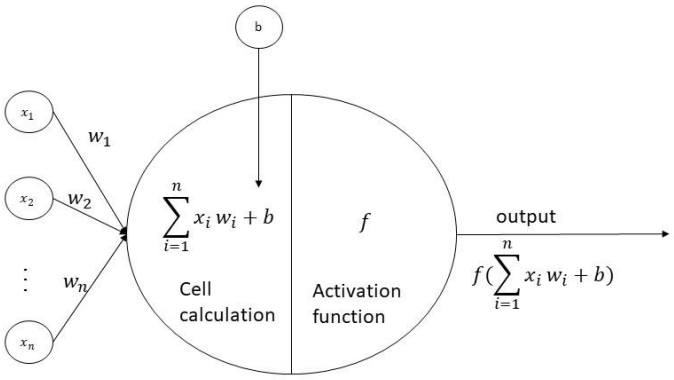

Figure 3: Cell activation

The main idea of convolutional neural networks is that a local understanding is enough. In many cases the number of parameters that get input to the network is too large, and learning becomes inefficient [7]. A common example would be image recognition. For example, an 1024x1024 image with RGB values per pixel would generate $1024 * 1024 * 3 = 3145728$ parameters, and that is the input layer only. Convolutional neural networks introduce two core concepts: filters and pooling. Filters can detect useful local features, such as edges in an image [8]. Pooling is technique to reduce the size of an output, thus reducing dimensionality. For example, a max pooling is a technique to pick the maximum value out of a defined window, replacing all values in that window by a single value [9]. Example visualization of 2x2 max pooling can be seen in figure 4 below.

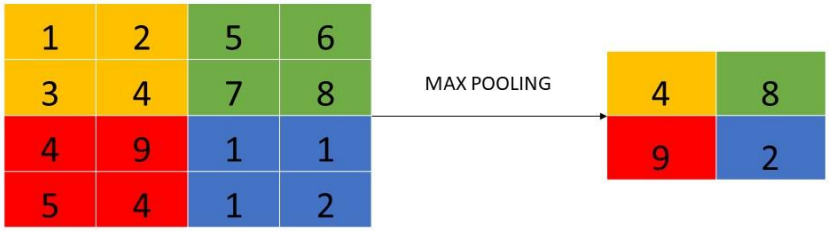

Figure 4: Max pooling

Recurrent neural networks are sequential models where inputs have relationship. It's used when the context is important to the input. A good example would be natural language processing, where a word placed in a sentence with different context can largely influence the meaning [10].

2.1.2 Training a neural network

With neural network training it's important to understand the general flow. This chapter is written mainly with supervised learning in mind and is slightly simplified. After a neural network structure has been created and one starts to train the data first phase is so called "Forward Propagation". In forward propagation the data goes through the cells and the calculations are applied according to the existing weights and biases in the network. Finally, the network outputs a value and with supervised learning, the labelled (the correct answers for classification are commonly called labels) training data exists, and one knows if the answer is correct or not. The final layer from the neural networks is commonly called a logits layer. With logits and labels the loss can be calculated. Loss is often called cost as well. Loss can be a custom calculation (though there are some general ways for calculating it) representing, how good the output was [2]. Both when calculating loss and making predictions for multiclass classification problem, a function called Softmax is utilized in this thesis. Softmax is a simple function that squashes each number

into 0…1 range and the sum of numbers adds up to one [11]. This can be used as an effective way to assess neural network “confidence” if each cell is the correct answer.

After the loss is defined we can do so called “Backward Propagation” using optimization algorithm [2]. The optimization algorithm is basically an algorithm that readjusts the weights and biases based on the loss. This way the network keeps evolving and getting more accurate as more and more training data gets fed in. One of the most common optimization algorithms is gradient descent. Without dwelling into too deep mathematics, the gradient descent is all about finding the gradient of a function to find out which direction values should be adjusted to find a minimum point from a convex (to reduce the loss) [2]. A learning rate is a multiplier that is used together with the derivative to adjust the weights in a controlled manner [2]. A learning rate too big causes the algorithm to easily pass the local minimum and can have hard time finding the exact point. A learning rate too small will take a very long time to travel to the local minimum. Adam is another optimization algorithm that has been increasingly popular in replacing the classical gradient descent. Adam optimizer is clearly inspired by gradient descent. It uses gradient descent with momentum that will keep some of the momentum from previous and combines it with elements from RMSProp which attempts to reduce the movement in unnecessary directions. In practice Adam algorithm has been proven to work well in many different machine learning problems and is widely used. Terminology used in this thesis also includes “batch” and “minibatch”. A batch is effectively a batch of training data to be processed. A minibatch is a smaller piece of the actual batch. The reason to break the batches into minibatches is to increase the efficiency by updating the weight and bias variables based on the result of the minibatch. Using minibatches greatly reduces the training time [11].

2.1.3 Overfitting & regularization

Overfitting is a symptom when the neural network is too detailed in comparison to the complexity of the data. Let’s examine an example case: If one has a neural network trying to recognize if a photo is representation of a cat or not and the network is huge, but there are only two different images of cats for training. The network is going to learn that exactly those two images are cats but is unable to create any general rule sets that could be applied to any other pictures. The example with only two images is quite extreme but illustrates the point well. To reduce overfitting one of the most popular techniques is regularization. One very powerful regularization technique is called “dropout”. If dropout is applied to a neural network layer, each neuron in that layer has a defined

change to be dropped out. What this causes in practice is that during training the neural network can't always use the same input to arrive at the correct conclusion, but rather, must generalize the weights in a way that allows multiple routes to lead to the same outcome [11]. Dropout is generally used only in the training phase. In the figure below there is the familiar neural network photo, but this time two neurons in hidden layer (marked with red), are being dropped out.

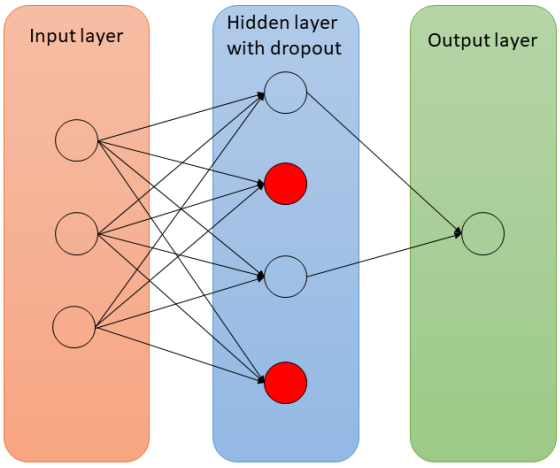

Figure 5: Dropout

2.1.4 Preprocessing & Feature Engineering

This section explains necessary terminology and theory related to preprocessing and feature engineering.

One of the key aspects to machine learning is preprocessing. Neural networks are performing series of calculations and that means that the inputs must be numerical as well. Preprocessing is the step we take to make sure our data is numerical and can be input to the neural network.

With feature engineering one transforms the preprocessed data further. One common application is normalization of the numerical data. The reason one wants to normalize data is because imbalances in numerical scales of the features causes problems in performance [11]. For this reason, linear numerical data should always be normalized. One

such value could be for example sum of transaction, where you might be able to make decisions based on whether sum is larger than certain amount or not. Not all data is linear, for example with business line of a company you can't draw conclusions with statements such as "business line is greater than 1000", because each business line is category of its own. This kind of data is categorical data and a popular way to apply this to a neural network is by doing a one-hot encoding. One-hot encoding transforms a categorical data with n-possibilities to an n-sized array with zeros in all indexes, except a one in the index representing that specific category. This way only the relevant index data weights in to the outcome. The following figure visualizes how numbers 0…3 get transformed into a one-hot encoding.

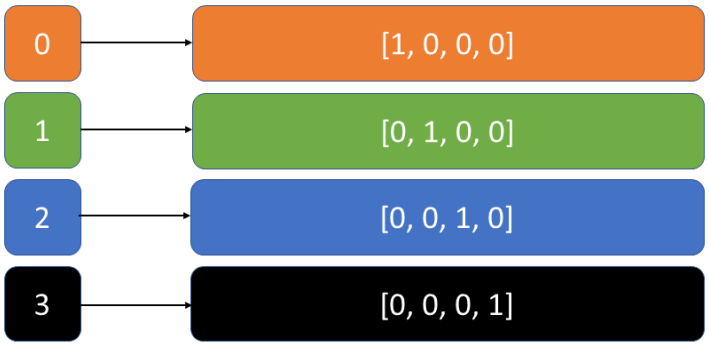

Figure 6: One-hot encoding

One-hot encoding is not the only solution available when it comes to categorical data. There is another, very crucial technique called embedding. Embedding becomes an extremely useful technique in a situation, where there are lots of options available. Imagine building one-hot encoding into a category with millions of possible outcomes. One ends up with an array with millions of cells of zeros and a single "one". This introduces two problems. One of them is sparsity of the data: with millions of parameters added for this single class, training the models becomes time consuming and data hungry as it takes a lot of time and data to find out the significance of each parameter. Another problem that is potentially even bigger is the memory consumption. For example, a training set of 100000 lines where each line has dozen parameters and then one of those parameters

must be converted into an array with the size of millions of numbers is likely something that most computers can't handle. Handling this kind of data is very inconvenient. These two are some of the top reasons to use embeddings instead. Embeddings are multidimensional vectors representing the values in that space. Embedding is a complex topic and in the context of this thesis, it's enough to understand that they are multi-dimensional float vector representations of the associated values making it possible to process them in the neural network. The value of each class embedding is adjusted during the neural network training depending how it affects the outcome. In the following figure, there is an example visualization of embeddings where numbers from zero to one hundred have been embedded into three-dimensional float vector.

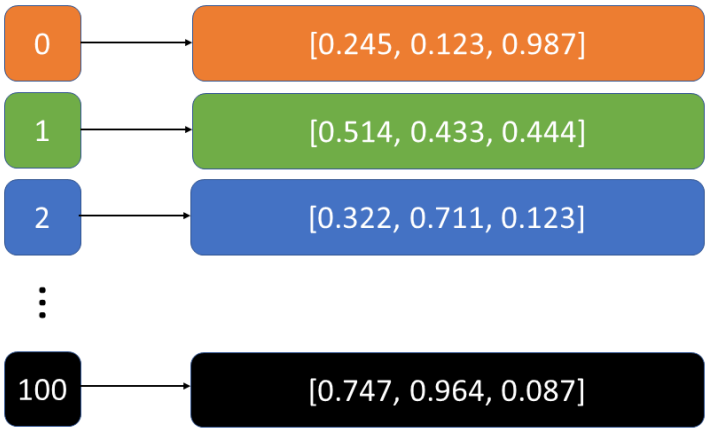

Figure 7: Embeddings

2.2 Tools

This chapter goes through the tools chosen for the job and why these specific tools were chosen.

2.2.1 Python

Python was chosen as the programming language based on several reasons. The number one reason was the fact that Python is becoming more and more the standard development language for machine learning solutions. Most machine learning libraries come with a fully featured Python API. Another relevant choice would have been C++,

but as it is considered a more difficult language, finding additional resources to the project in future, would have been more challenging. Also there are plenty of discussion and examples revolving around the machine learning frameworks in Python, so it was a natural choice.

2.2.2 TensorFlow

TensorFlow is an open source machine learning framework for building deep-learning solutions. It is the primary framework used in the project covered by this thesis. TensorFlow was chosen due to great documentation, general popularity, suitability for the task (as it is one of the most popular ones for creating neural networks) and trendiness. In the “Figure 1: Machine learning trends over last two years” below, one can see TensorFlow compared to some of the other most popular machine learning frameworks and the increase in trend is speaking for itself. Picture taken from Google Trends using the search conditions visible in the figure.

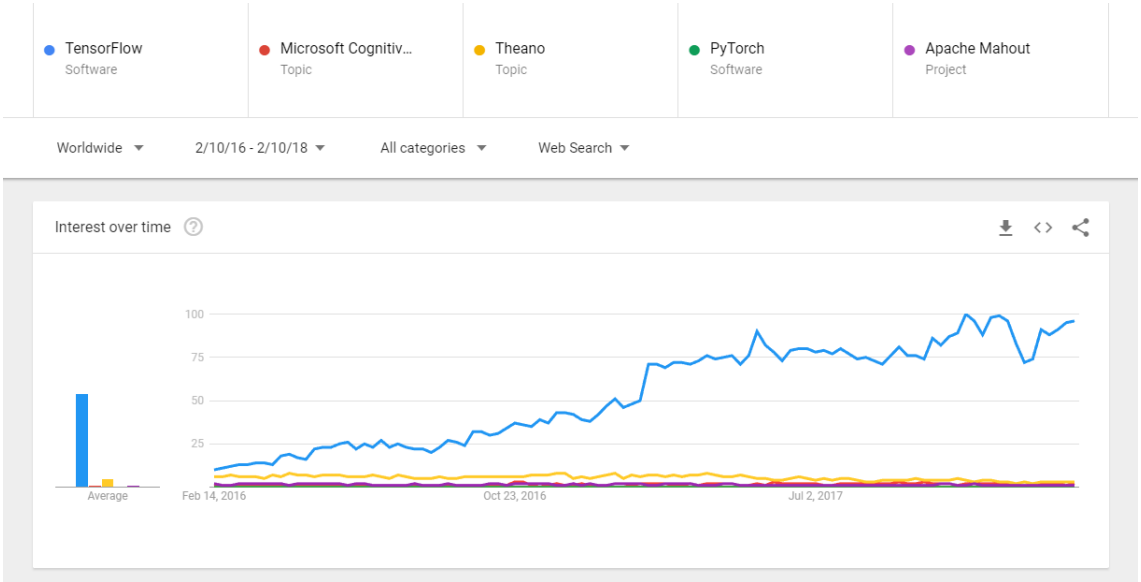

Figure 8: Machine learning trends over last two years

For the sake of understanding this thesis, it’s important to know the basics of how TensorFlow operates. When using TensorFlow with Python, one must first define a graph. A graph is kind of a description of what the structure of the network is like and what kind of variables, constants and parameters it includes. A TensorFlow session is a session where actual calculations within the graph get executed [12]. In this thesis TensorFlow

estimators are being used. Estimators are a high-level API that make defining layers quite a bit easier. Estimators build graphs and sessions on the background without coder having to worry about them and provide many convenience functions such as checkpoints, variable initialization and easy summary saving [13].

2.2.3 TensorBoard

TensorBoard is a tool built for visualizing what is going on with TensorFlow programs. TensorBoard allows one to better understand what is going under the hood, allowing easier debugging and optimization. It's up to the programmer to decide what kind of data to export from TensorFlow and then TensorBoard helps to visualize it. Visualization of the data is critical in many machine learning applications as training of a model can take as long as several weeks. Visualization of what goes on during training can provide information already before the training has finished. TensorBoard can also visualize TensorFlow graphs, which can be a nice way to visualize the machine learning model structure.

2.2.4 Pandas

Pandas is used for data structure handling and was chosen because it's one of the most popular libraries for data analysis and structure handling. Additionally, many machine learning frameworks including TensorFlow, directly support the Pandas data structures.

2.2.5 NumPy

NumPy is a scientific computing package for Python. It provides extremely optimized mathematical functions and effective n-dimensional array manipulation. It's extremely popular when working with statistic, data-analysis and machine learning. It's also being used by most of the Python machine learning frameworks, including TensorFlow.

2.2.6 Other tools

For computing and development, a single personal computer is being used. The operating system is Windows. Data is stored in MySQL database.

3 Materials and methods

This chapter introduces most influential information sources for the thesis, the data that is used, evaluation methods and key terminology for evaluation as well as some intentional oversights that were made to keep the scope of the thesis in check.

3.1 Preliminary information

Currently the ledger account is determined manually by an accountant and considering we want to produce similar results with the machine, the most relevant source of information about the process is someone with accounting experience. Qualitative information gathering was done in form of interview with a Process Manager from Accountor Holding Oy. Additional points of considerations came up in discussions with Accountor Finago Operations Manager and Development Manager. A lot of machine learning knowledge was already familiar to the author from Andrew Ng's deeplearning.ai program. This knowledge was used and further backed up by referencing other existing sources, such as kindle book "Machine Learning with TensorFlow" by Nishant Shukla.

3.2 Data

The data used was randomly picked bank statement data from real database. The amount of data used for the proof of concept included roughly 2.5 million bank statement events. The data used was not only payment-based accounting companies, but rather all kinds of companies. There main reason was: the difference in the accounting result should not be that different between the normal case and payment-based cases, rather the difference is mainly in the process, which is irrelevant for training purposes. A second reason was that the amount of data from companies using payment-based accounting would be very limited as these companies are a minority in the database and usually very small companies, that don't have that much data.

3.3 Intentional oversights

The thesis does not consider companies with extra-long ledger account numbers. Most companies follow ledger account chart where the numbering ends at 9999 and this is used as a limitation for the thesis. Also, as mentioned in previous subchapter, the data used was overall bank statement data, not only data from the target problem group.

One option was to use the comment field of bank statement event as it can sometimes provide valuable information. However, it's a freeform text and could be written in any language or contain any notes and processing it would've complicated implementation significantly. For this reason, it was left out.

3.4 Potential

In this thesis the word "potential" is used often when describing a machine learning models maximum ability to learn in ideal conditions, provided that no regulation is applied, and all the data is at dispose for the model. So, in context of this thesis the word "potential" will mean accuracy of the model given the above circumstances. Please note that this is not an official term, but the author of the thesis deemed it a term, that describes the situation well.

3.5 Evaluation methods

The original evaluation method planned for the machine learning model was splitting the data into training, development and test datasets. The training data set would be used for training, development data set would be used for verifying the model performs well on not seen data and the never seen before testing set would be used for evaluation. The reason for separate test and development data sets is that when iterating the solution over the results of development dataset, one will create unintentional bias and thus, the never seen testing data set is needed. This is a common way of verifying the functionality of a machine learning model, but unlike intended, it did not fit this problem well. The reasons are discussed in a later chapter.

The author came up with a method more suitable to the situation: A simulation emulating a real-life situation. After discussing with thesis supervisor Juha Havu, it was decided that a final simulation would be done by removing several companies completely out of data and testing how the models reactively learn by retraining them after a month of data. So, the final evaluation is done for five companies, from the same business line whose data has been completely removed from the training material. In the evaluation it's most important that the network doesn't give wrong answers. It is okay to leave some unanswered. In the evaluation the main two things looked at are answering percentage and accuracy of those answers.

4 Solution building

This chapter will present in detail how the solution was built and explain why the decisions were made.

4.1 Feature selection, Preprocessing and Feature building

This chapter goes through the selected features, why they were selected, what kind of preprocessing is done and how the feature vector is built. The full listing of fields is summarized in the following table. Each field is covered in detail in the following subchapters.

Fields	Processing 1	Processing 2	Feature build
Receiver	String->int mapping	-	Embedding
Sum	Normalization	-	Linear
Event type	-	-	One-hot
BIC	String->int mapping	-	One-hot
Code	int -> incremental int	-	One-hot
Code explanation	String->int mapping	-	Embedding
Pay day	Day of month	Normalized	Linear
Month	Months from January 2016	Normalized	Linear
Company id	-	-	Built-in
Business line	business id -> business line	String->int mapping	Built-in

4.1.1 Company id

The company id is a unique identifier that exists for each company. This kind of unique identifier is crucial for the requirement of doing company specific decisions. Such identifier already existed within the data and no conversions were needed. Furthermore, the identifier is built-in to the neural network architecture itself (as is discussed in detail in the chapter 4.2) and thus needs no preprocessing.

4.1.2 Business line

The business line should be a unique identifier to differentiate each business lines from each other. During initial data gathering it was discussed that within a business line companies are likely to share similar conventions, and this is a way to identify the business line. This information did not exist within the data and some data processing had to be done to get it. The solution was to use the business id, which existed in the data, to fetch further data about the company from PRH API, which is a public API for querying information about Finnish registered companies. These business lines were transformed to lower case to avoid any duplication of data that might or might not exist within PRH system and then inserted into training database to receive an auto-incrementable integer key. This integer key then could be used to uniquely identify the business line. It is noteworthy that the PRH API lacked information from quite a few companies for unknown reasons. These unknowns are considered “0” in training data. Much like company id, business line was also built-in to the neural network architecture and is discussed in more detail in chapter 4.2.

4.1.3 Receiver

Receiver is the receiver of a payment. In initial information gathering this was considered as most important field as it could provide a lot of information as most likely many companies paying for the same receiver do same kind of payments and hence, use similar ledger accounts, at least to an extent. Receiver is a string value so there is a need for preprocessing to make it a numeric value that can be used by a neural network. The first

thing is lower casing all the letters as there are differences on how people write the receiver name within lower and upper cases. After the value has been lower cased it's inserted into training database as a unique value and provided an auto-incremented id. It is worth noting that in production usage these id maps need to be updated regularly because there are constantly new receiver names appearing. Unknown receiver names are handled as id "0". If the number of receivers would go out of control, it's possible to only track for example one million of the most commonly appearing names. Just numerizing the receiver name is not enough as this is categorical data. There is no meaningful connection between the size of the numbers and it's impossible to write meaningful rules to compare with the size of the number. Instead each name brings its own unique touch to the result. Since there are lots of receivers and receiver is categorical data, it's built as an embedding feature.

Alternative option was also considered: the target bank account. Author ran tests with setups of both receiver and bank account, as well as only bank accounts but the only difference it made in training was the increase in training time due to more complex parameter. It had no effect on training accuracy at least within the scope of the training data used in the thesis.

4.1.4 Sum

Another field that was also mentioned as important one in initial information gathering interviews was the sum. The amount of the payment could quite likely hint at what kind of payment event was this. The sum is already a numerical value, so it simply is normalized, and it's built as linear feature.

4.1.5 Event type

Event type includes information such as if the event was withdrawal, deposit or correction. The reason for including this information is simplification of the process. When the neural network does not need to solve the problem of what kind of event it was (which is solvable from other values) it should increase the performance. Event type is already a numerical value and it is a categorical value so to it is built as a one-hot encoding feature.

4.1.6 Code

Code is the code that the bank uses to identify what kind of event it is. There are dozens of possible values including values such as “700”, which is described as “transaction processed in some payment service” or “721” as in “card payment”. There are some bank specific differences to usages of these codes and the previous example codes are used by Nordea. The reason to use the code is simply the idea that it could be crucial information whether the payment is made by card or as a bank transfer. The codes are already integers, but they are arbitrary integers and don’t follow strict ordering and thus they are easier to handle after mapping them to incremental integers starting from 1. The number 0 is always left as an option for unknown values. There is a restricted amount of different codes that the banks use and thus the feature is built using one-hot encoding.

4.1.7 Code explanation

Code explanation is a string value that banks use to provide further information related to the code. It is used as a feature with the hope that these details further help to specify what kind of payment it was, which could be used to determine the correct ledger account. As it is string, it is first converted to integer by lower casing and adding it as unique value to training database and using auto-increment to provide integer values. There are far more code explanations than codes and thus it is built an embedding feature.

4.1.8 BIC

BIC is the bank identifier code, which is a unique string identifier each bank has. The BIC mentioned here is the one related to company whose bank statement is being studied, rather than the target bank accounts. The reason for including this is because there are bank specific differences in how they use event codes, so without this information it’s impossible to differentiate between the bank specific differences of code usage. As with previous string fields this field is also preprocessed into integer by using lower casing and database. There are limited amount of different BIC codes appearing in the training data so one-hot encoding was used for feature building.

4.1.9 Pay day

Pay day is the day of the month that the payment was done, meaning that it's a value varying from 0 to 30 (programmatically zero is the first index instead of one). The reason to include this was the intuition that certain events might occur always on the same day of the month, for example salary paying. The data is stored as dates so the first thing to do for preprocessing is to only extract the day of the month from that value and store it to training data instead. Day of the month is clearly a linear numerical value, so it is also normalized and built as linear feature.

4.1.10 Month

The training data is fetched starting from January 2016, meaning that data earlier than that does not exist for the neural network training. Any new data will naturally be newer than that, because such is the essence of time. For the neural networks it's listed as one of the requirements to be able to adjust to changes over time. These changes don't occur every day, but they could be a result of a change in accountant firm or a legislative change that will affect the ledger accounts. The month feature is introduced to keep track of time in linear manner. It is the number of months that has passed since January 2016, creating a monthly changing value for tracking time in linear fashion, but keeping it infrequent enough so that the values will never escape a reasonable range and cause too much sparsity. So, in the training data this means that month is the number of months from January 2016 to the date when the accounting was done. This value is then normalized and built as a linear feature.

4.2 Neural network technical design

In this chapter the goal is to briefly go through the different technical methods that were tried and then focus on the final implementation that was chosen.

Technical design of the neural networks and their retraining is an important part of the process as the original requirements of network include the ability to adapt changes over time. Therefore, the network must be trained over time on a frequent basis. Consequently, the time it takes to train the network should be manageable. The logic should handle differences between companies as the results are likely to vary a lot between them. As a result, individualizing the company in some way is a mandatory requirement for the network. Based on the initial information gathering there is also a strong suspicion

that companies from same business lines share many similarities in their accounting conventions. The simplest solution would be to make a huge neural network where a unique company identifier is one of the parameters. Testing this approach revealed quickly few problems: due to lots of different companies and varying amount of data the network had to be huge to account for all the different parameters and sparsity became a problem. The large network took absurd amount of time to train and was clearly out of question. One option was also to use convolutional neural networks to reduce the number of variables to calculate, but it lost some information on the way.

The solution to the problem, is training an individual network for each company. That way the outcome is very individualized for each company and training time is just a small fraction of what it would be on all comprehensive networks. Company level networks have another problem however: quite often there is not enough data for the company to make a reasonable guess. For example, when starting a new company, there is no pre-existing data and it would take a while to build enough relevant data. Furthermore, it's more than likely that there are similarities between the companies and that the data of other companies could be utilized.

The solution author came up with is a 3-layer hand tuned algorithm of neural networks, visualization can be seen in the following figure. The algorithm is based on the Softmax functionality where the distribution of values can also be observed as a confidence level. Starting by running the values through the company level neural network and seeing if any answer receives 95% confidence of higher. If the confidence level of company level neural network is not satisfied, then a business line specific neural network will be consulted. The confidence expectation for business line neural network is 90%. If this condition also remains unsatisfied then the third and final layer is the general neural network. The general neural network will provide a prediction, should the confidence of that prediction be equal or higher than 85%. And finally, if this neural network also fails to provide a confident answer, then the neural network structure simply won’t predict. The reason for this is to reduce the number of guesses that are likely to be wrong. If all three networks are failing to provide a decent confidence level, there is a very high chance that all three would predict incorrectly and this is something to be avoided. The reason for diminishing confidence levels between networks is that as we combine more companies to the same dataset without distinguishing between the different companies during the training time, we end up with something like a “most common answer”. For example, within a business line many companies have similar accounting conventions, but not all. This means that

the network will find the most common answer to similar situation and the confidence drop from 100% will be the differentiation that companies who do it differently provide. Consequently, within a company level neural network the confidence should be high, business level neural network should be lower but still relatively high and general level neural network should be lowest confidence of the three. The target confidence percentages for each specific neural network were result of manual fine-tuning and testing. All three networks are very similar implementation wise but end up performing very differently in practice. Feature wise all of them use the features mentioned in chapter 4.1, except that company id is only built-in to the company network and business line is built-in to the business line network by default.

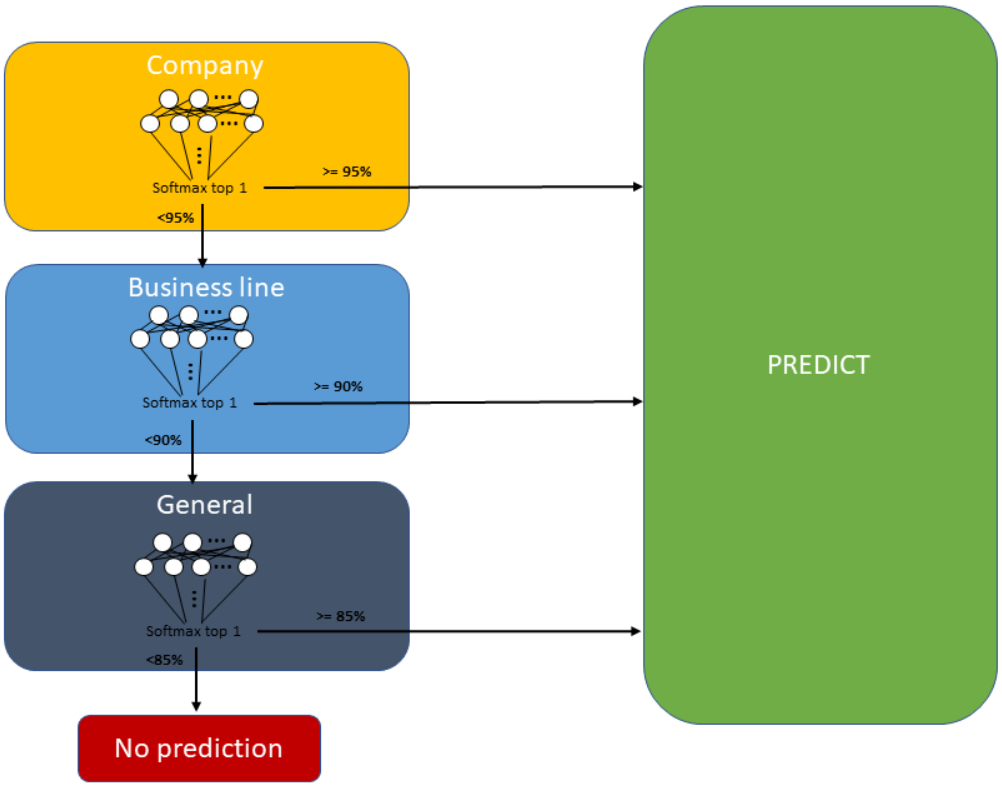

Figure 9: Neural network structure

4.2.1 Company network

Company level neural network is the first network that gets asked for a prediction. Its main responsibility is to learn company specific accounting traits. The company itself doesn't need any specific id to be provided to the network as the fact that it's trained only from the company data already individualizes the network. This means that each company will have a neural network of their own.

The company level neural network is built as a dense, fully-connected network. The size was manually fine-tuned by picking biggest company (data wise) from the training data and checking that the depth and width were enough to satisfy the complexity. It can include a lot of neurons because even with the biggest companies the data amounts don't stretch the training time too much and in fact to capture all the special cases a rather specific network is required. The input feature vectors and feature builds are as described before. First hidden layer is a 3000 neurons wide dense layer. The first layer goes through ReLu activation function. Second hidden layer has 2500 neurons that are densely connected and is also activated by the ReLu activation function. The third hidden layer is 2000 neuron fully connected layer going through the ReLu activation. After the third final layer, should the network be ongoing through training a dropout is being applied. A dropout of 25% is introduced to reduce overfitting. In the prediction and evaluation modes the dropout is skipped, as it is only relevant for preventing overfitting in training and would only hurt the performance of other two. Finally, a dense layer of 10000 units is introduced, which works as the logits layer. Each unit represents a ledger account. For predictions a Softmax function is applied. For training and evaluation modes Softmax is used to calculate the loss. Accuracy, that is used for monitoring purposes is calculated by comparing the index of the largest value in the logits layer to the correct ledger account in the labels. Finally, if the mode is training, the Adam optimization method is being used to adjust the weights and biases to improve the network performance. The values used for Adam optimizer were learning rate 0.001, beta 1 as 0.9, beta 2 as 0.999 and epsilon as 1e-08. These values are commonly, yet unofficially considered a good starting point before further optimization and worked fine in this case. The visualization of the company level network as TensorFlow graph in TensorBoard can be seen in the following figure.

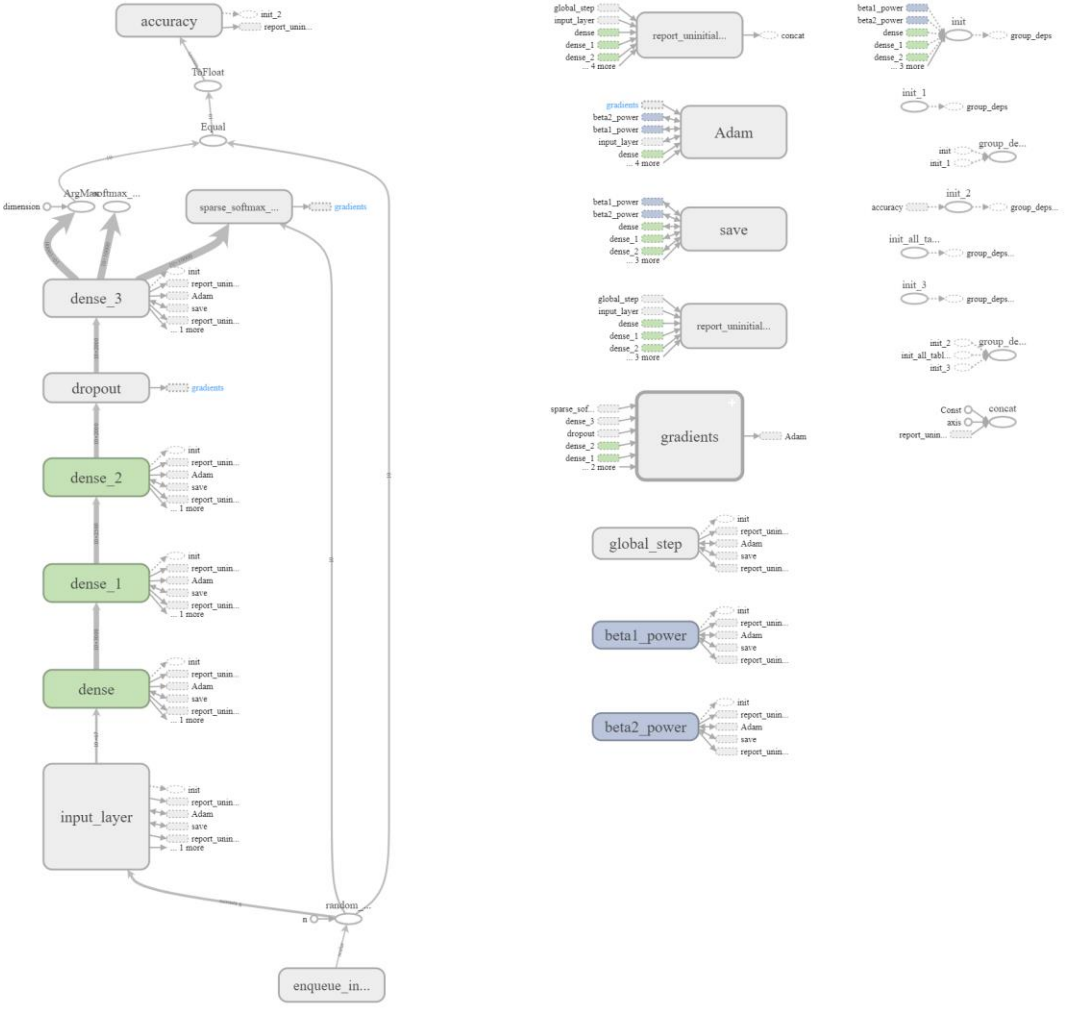

Figure 10: Company neural network graph

4.2.2 Business line network

Business line network is built based on data of certain business line. It's the second network in the line as it provides more detailed data than general neural network, but less detailed than the company level neural network. Each business line gets a network of their own. The reason for building a business line network is due to the original data gathering where it was hinted that companies from same business line might be sharing similar conventions in accounting. Because business line neural network has a lot of data already and no need for as specific pattern finding as the company level neural network, the technical implementation of the neural network is less complex. Using previously mentioned features as inputs and building features as also mentioned for them.

The first hidden layer is a 1500 neuron dense layer going through the ReLu activation function. The second hidden layer is 1000 neuron dense layer also going through ReLu activation. The third hidden layer has 800 neurons, also fully connected and passing through ReLu activation. If the mode is training, then a dropout of 10% is performed after the third layer. Finally, as always, a logits layer of 10000 neurons, one for each ledger account, is added as a dense layer. Accuracy, loss and predictions are all calculated the same as within the company level neural network using Softmax and argmax. The optimizer is also Adam, using the same values as the company level network. TensorFlow graph of the business line neural network visualized in Tensorboard can be seen in the following figure.

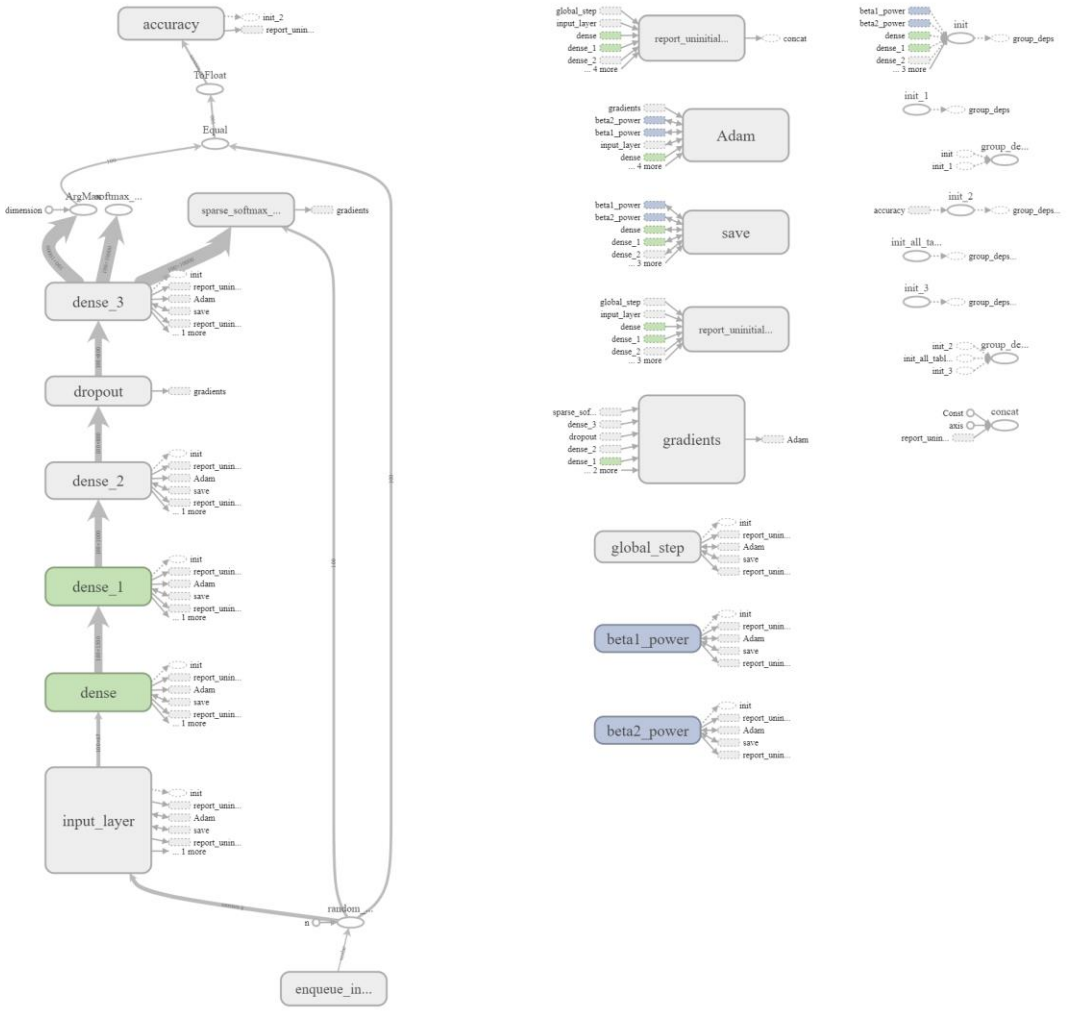

Figure 11: Business line neural network graph

4.2.3 General network

The general network is a neural network built on all the available data. There is no differentiation between companies and business lines. Therefore, the prediction will always be the most common answer and the confidence is likely never going to be 100% and this is intentional. The whole idea is to provide the most common answer from all the data.

The technical implementation of the general neural network follows closely the business line one, with only one exception. The exception is the removal of the dropout layer. The dropout layer is not necessary due to massive amount of data compared to the network size. The amount of data regularizes itself automatically in this network. Optimization, accuracy, loss calculation et cetera is all done exactly same with the other two neural networks. In the following figure is the TensorBoard visualization of the general neural network TensorFlow graph.

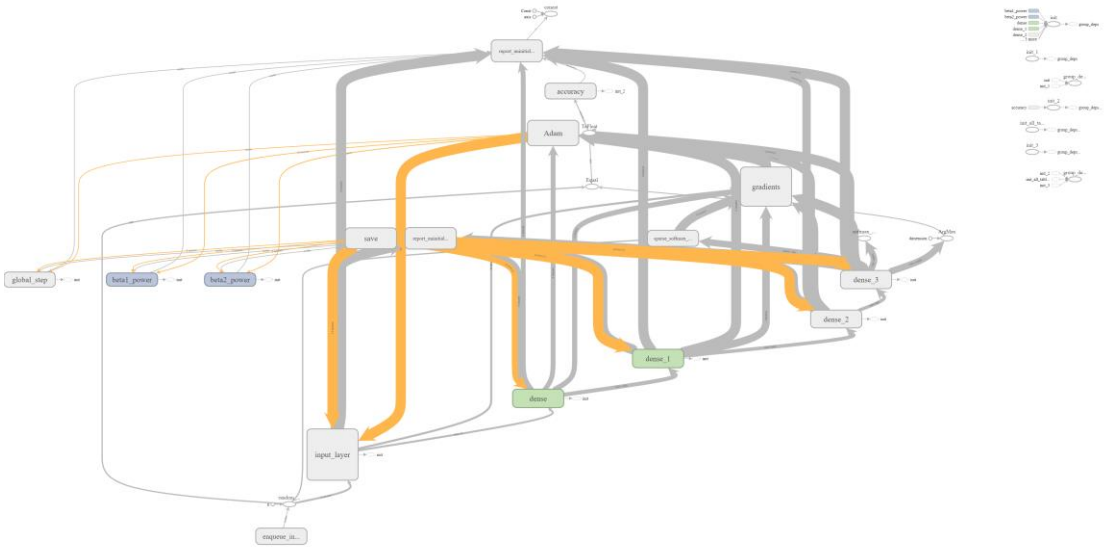

Figure 12: General neural network graph

4.3 Training

The training data was stored in MySQL database and the training was done in batches. It had to be done in batches due to volume of the data. Consequently, a separate normalizer tool had to be built that separately stored min and max values for fields to enable normalization as not all the data would ever be available at a single time. Originally the data was split to training data, dev data and test data, but seeing as the repetitive nature of data caused the evaluation to always return the same results with the actual training

already during early implementations of the model, all the data was handled as training data.

To load the training data during training, a MySQL cursor was created and used to select all the data wanted in the training of that specific network, meaning no conditions in general network training, business line as a condition when training a business line specific neural network or a company as a condition when training a company specific neural network. The full result was then ordered by random but using seed to make more comparable results during early iterations. This query is very slow to execute, but it's not a problem since this is done only in the beginning and not for each training batch. Cursor was then used to fetch the data in batches of 10000 and mini batch size for training was 100. In the case that the number or results fetched was less than 10000, which could happen in case of some business lines and many companies the batch size would be one if there were 100 or less results and ten if there were more than 100 but less than 1000 results. The cursor traversing is done as many times as there are defined epochs (5 epochs in the final solution) and the mini batches are gone through until each cursor batch is completely gone through.

The data is loaded into Pandas dataframe and this frame is passed to the custom-built normalizer that makes sure that values are normalized in a manner where they are within a range of 1 from each other.

Finally, the according TensorFlow estimator is loaded from specified path (including the id of business line or company in their case storing each in unique folder) and training is resumed from the previous checkpoint (which could be non-existent causing a new one to start). This means that training can be either continued or restarted from scratch based on intentions. Keeping in mind the point that the models should be retrainable overtime this is an interesting possibility. On one side, continuing the training from an existing checkpoint could save a lot of time, but on the other hand it creates bias towards the more recent data. The bias toward the recent data could be fine though, possibly even a desired feature. In the scope of this thesis the detailed analyzation of checkpoint loading versus training from scratch was not analysed in detail and training for simulation was done by training from scratch. Checkpoint loading was used during the training however, to store a working estimator after each epoch. The following figure visualizes the training cycle of a single model.

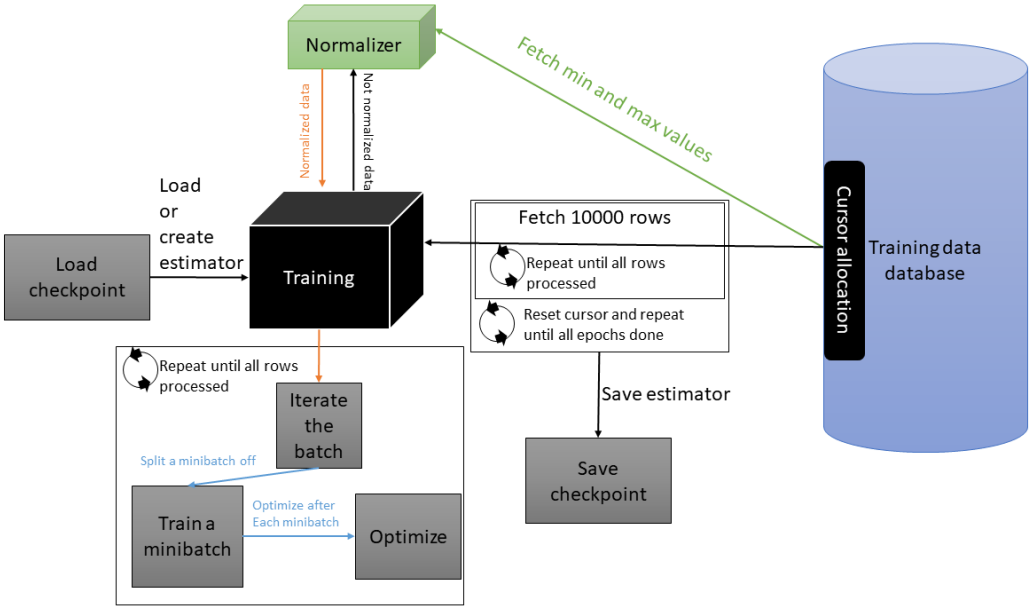

Figure 13: Visualization of training a single model

4.4 Simulation building

As discussed in chapter “Evaluation methods” the best way to evaluate the performance of this system in a life-like situation is by building a simulation where certain companies are removed from the data completely and then introduced as new companies. So first the business line with most data was chosen for this simulation. From that business line five companies are randomly picked (though data-wise smallest companies are filtered out from random selection). Those companies are stored in separate csv files and removed completely from the training data.

After that the business line neural network is trained for the target business line and general neural network is trained as well. The data of each company is read in month-by-month. All inputs are run through the three-model architecture starting with company model, backing up to business line model if confidence was not high enough and finally to general model if business line model was also not confident enough. As a result, there will either be a prediction or no prediction. The aim is to study both accuracy of the done predictions and the percentage of provided answers. After a month of data has been handled, that data is inserted to the training database and the models are retrained from scratch to repeat the process again and again, until finally all the data is handled. The prediction results from the three-model structure are hard to visualize in a single TensorBoard so they were saved as csv and Excel was used to generate graphs out of them.

5 Solution evaluation

This chapter will go through the results without speculating too much why they look the way they do. The discussion regarding the results is done in the next chapter.

5.1 Simulation evaluation

The very first thing we should look at is the potential of each model. For all the upcoming figures of potential, the solid line has been smoothed for easier viewing, the half-transparent line are the exact values for each 100 training steps. The Y-axis is accuracy percentage of the model and the X-axis is training time in hours. In the figure below, we can see the potential of the general model. The training time for the general model is a lot longer than other models as it trains over all the data. This is okay, as the general model has no need to be trained often. After three and half hours of training it looks like the smoothed accuracy is settling for roughly 87%. Because in this model we don't individualize the companies or business lines, and the network size could not possibly overfit the problem, it means that 87% of ledger account data follows similar rules cross-companies and business lines.

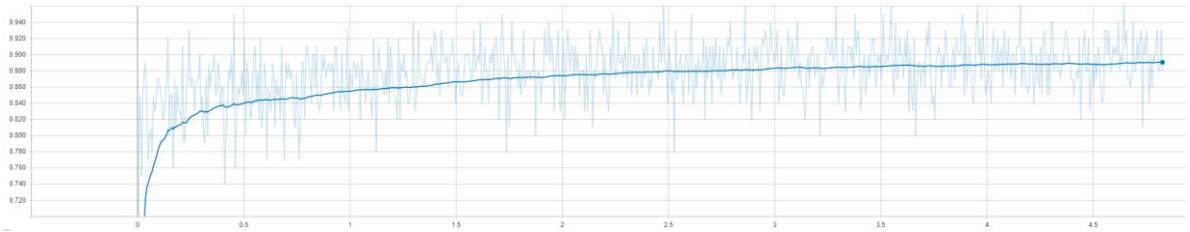

Figure 14: General model potential

In the next figure we see the potential for business line model used in the simulation. The training of this model is already considerably faster, as the data for a single business lines only consists of a fragment of the full data volume. After roughly 42 minutes of training, the smoothed curve settles for about 92% accuracy. So, the difference that it can bring compared to the general model, is roughly five percentage points.

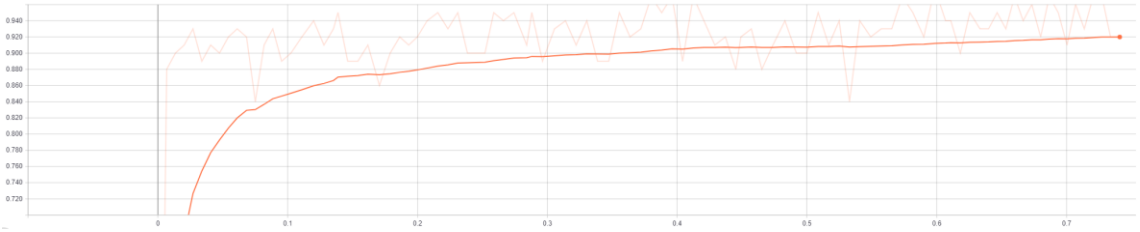

Figure 15: Business line model

The next three figures represent the potential for companies that could be called “happy cases”. Within few minutes of training all three of them have settled for 100% accuracy. This means that for these companies using this kind of model provides a theoretical chance to get 100% accuracy in simulation.

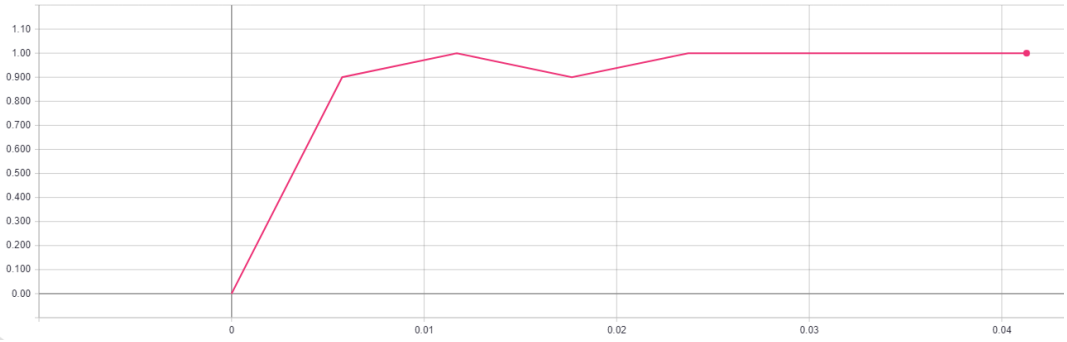

Figure 16: Company 1 potential

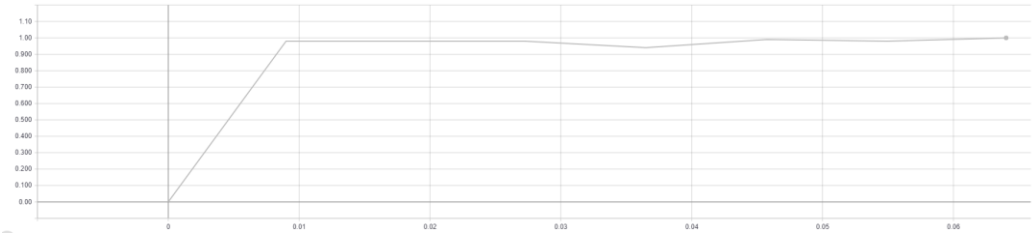

Figure 17: Company 2 potential

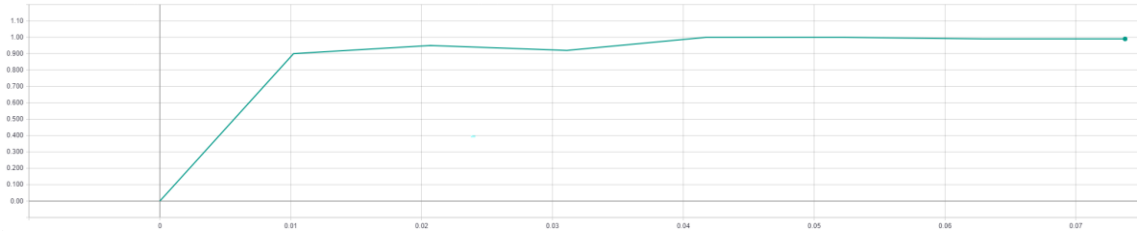

Figure 18: Company 3 potential

The next two potential figures are for companies that can’t reach steady 100% and they have occasional batches of 100 training steps that drop down from there. They do reach high accuracy percentages regardless.

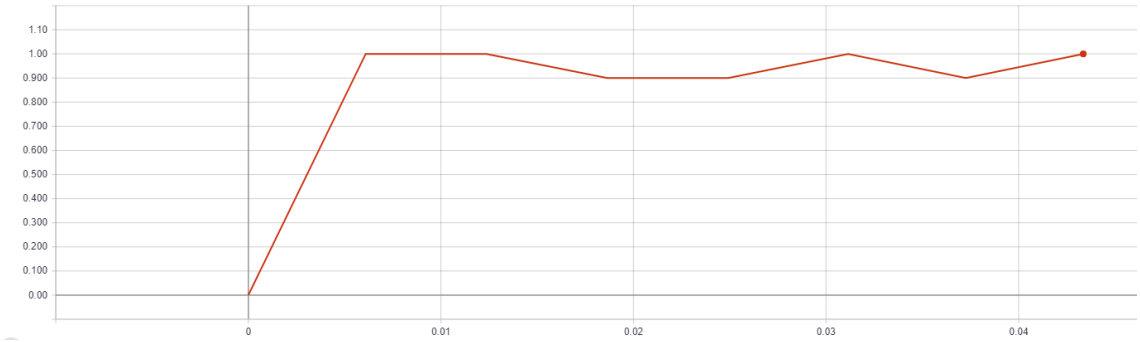

Figure 19: Company 4 potential

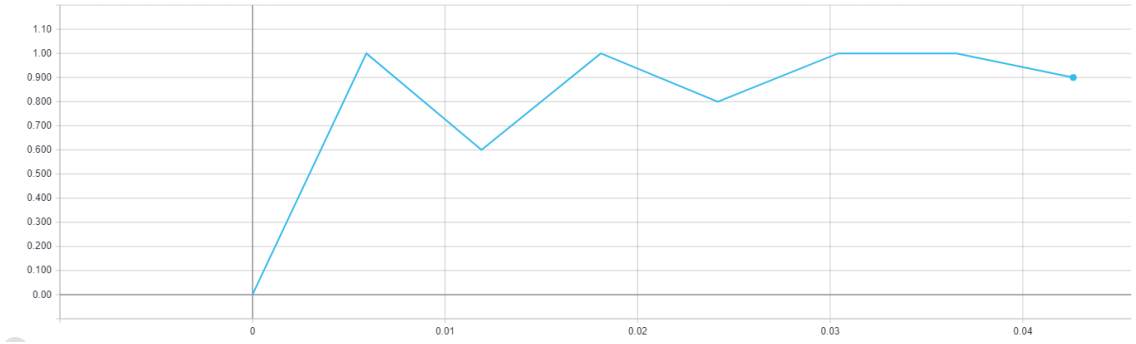

Figure 20: Company 5 potential

For the following figures about the simulation results in case companies the Y-axis is the percentage of correctness or the percentage of answering (depending on graph) and X-axis is months. It is important to notice that the scale of Y-axis is not same in all the graphs to better visualize the differences within that graph. The following two figures are the simulation results for case company one. The answering percentage started near 80% but increased to 100% quickly. Accuracy had huge variance in the early months, varying between 40% and 100%, later stabilizing to 100% and for the last few months being at around 90%.

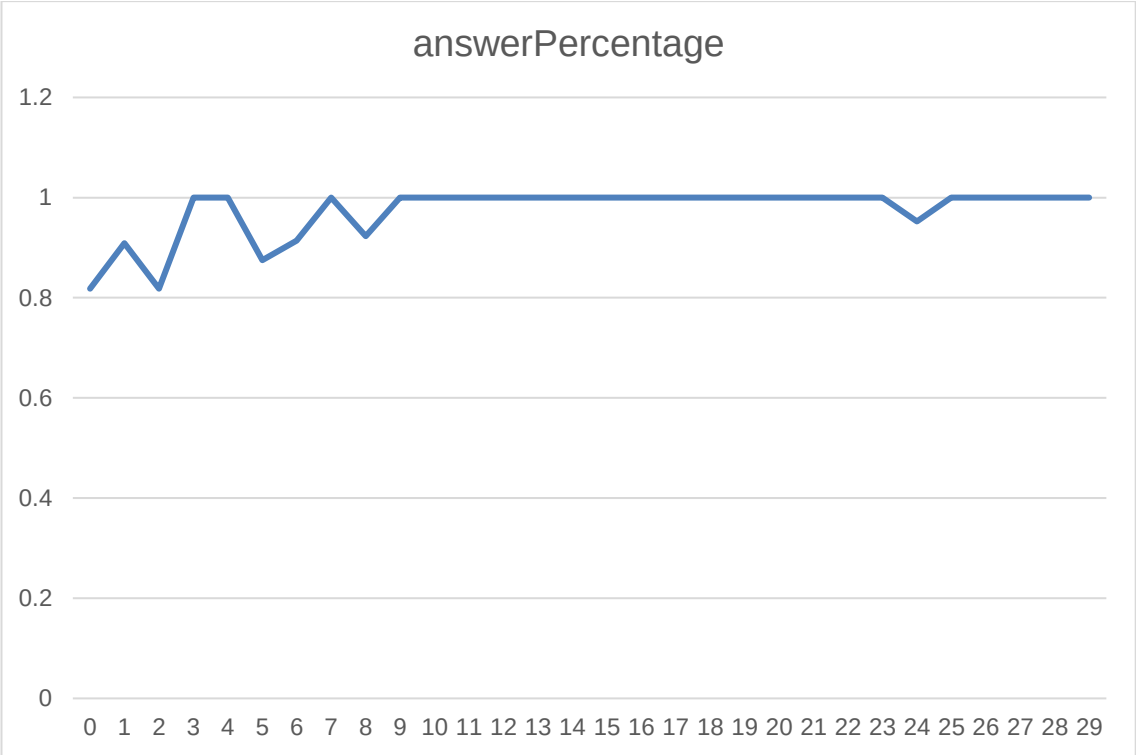

Figure 21: Company 1 simulation answer percentage

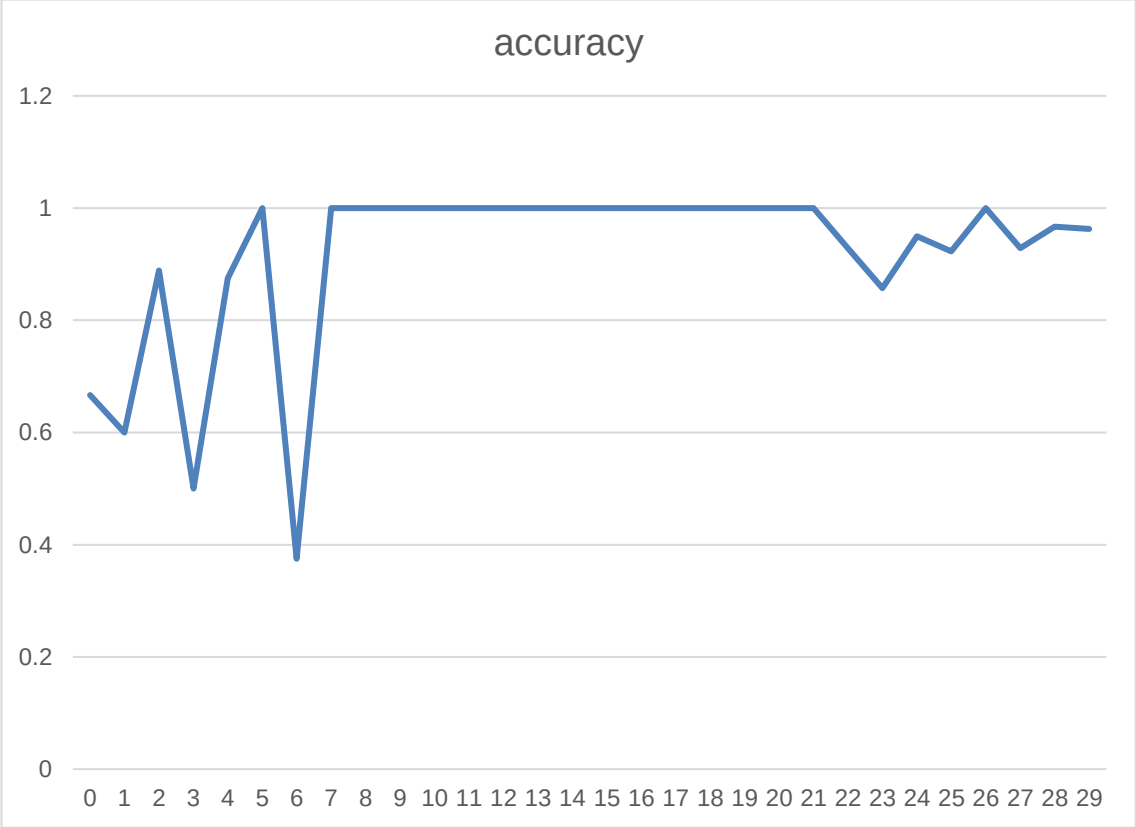

Figure 22: Company 1 simulation accuracy

Case company two's results are in the next two figures. Answering percentage was always quite high, moving between 100% and 95% during the 29-month period. Accuracy was also relatively high, moving between 98% and 100% for most of the time, but suddenly dropping down to ~93% in that 29th month.

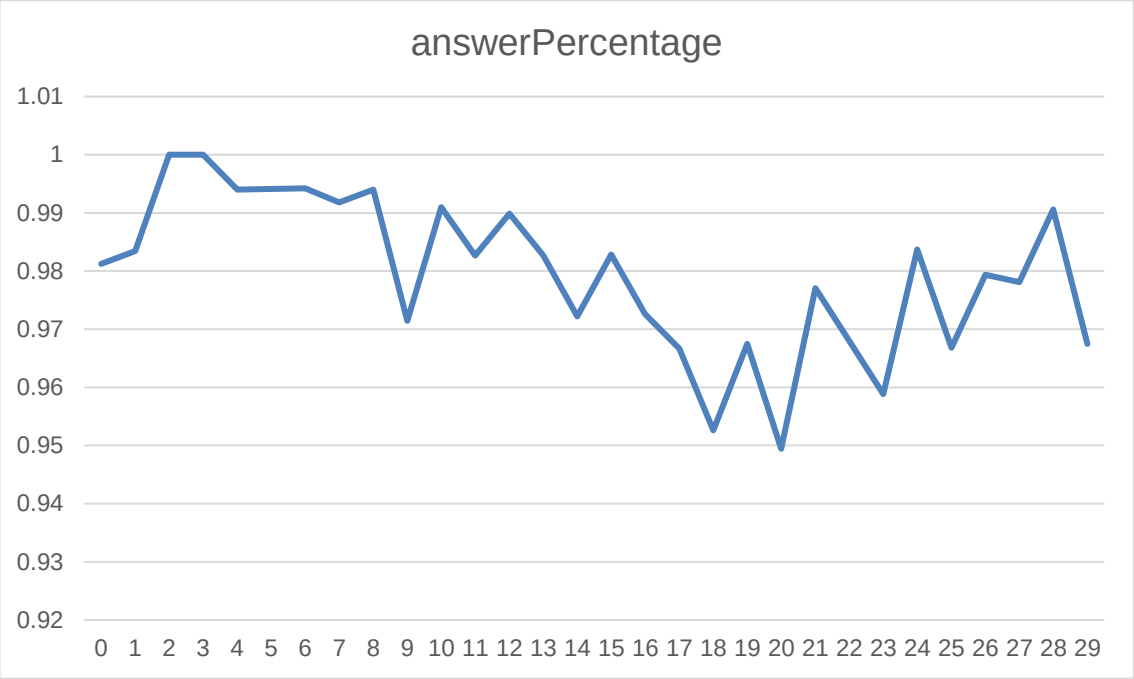

Figure 23: Company 2 simulation answer percentage

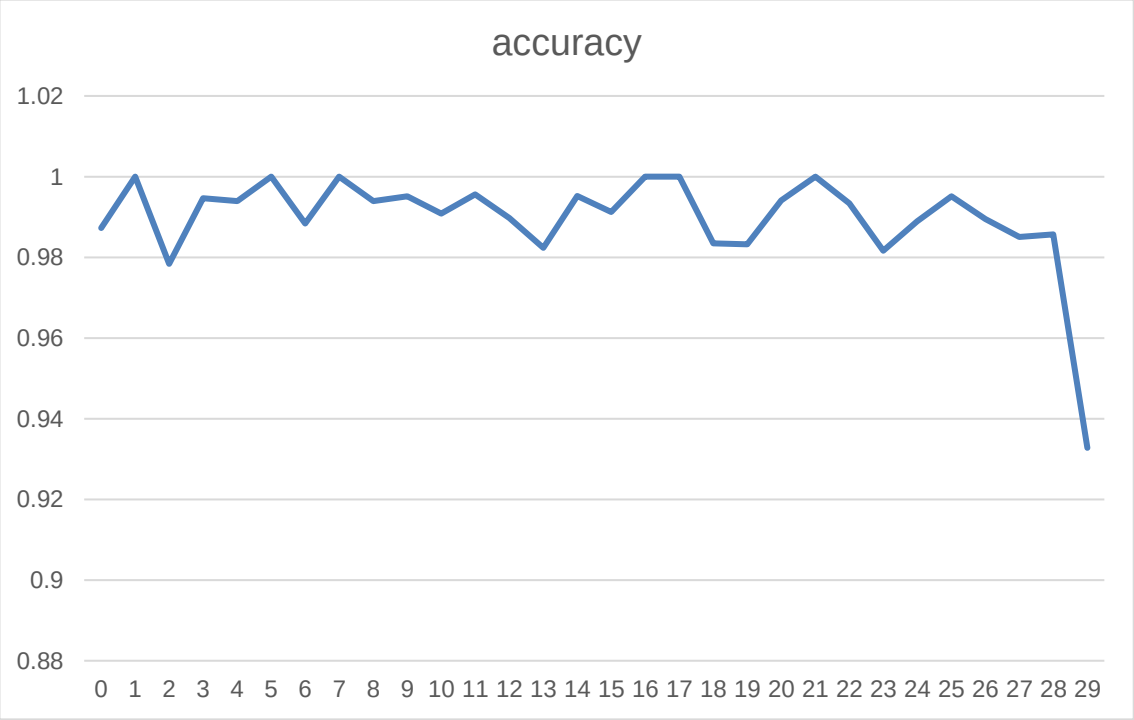

Figure 24: Company 2 simulation accuracy

Moving to the simulation results of case company three, there was a clear improvement in the answer percentage after beginning, starting out at 96% and then being around ~99% - 100% after the few first months. Accuracy had big variance in case company three. At times hitting 100%, but dropping as low as 70% during month 16.

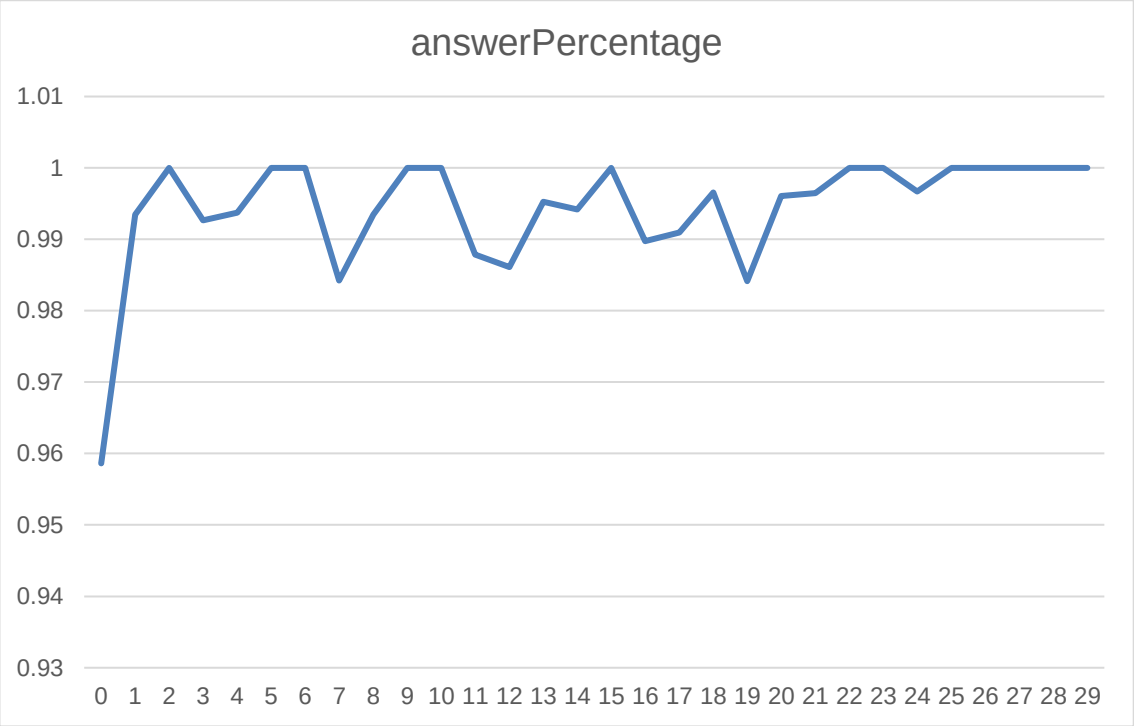

Figure 25: Company 3 simulation answer percentage

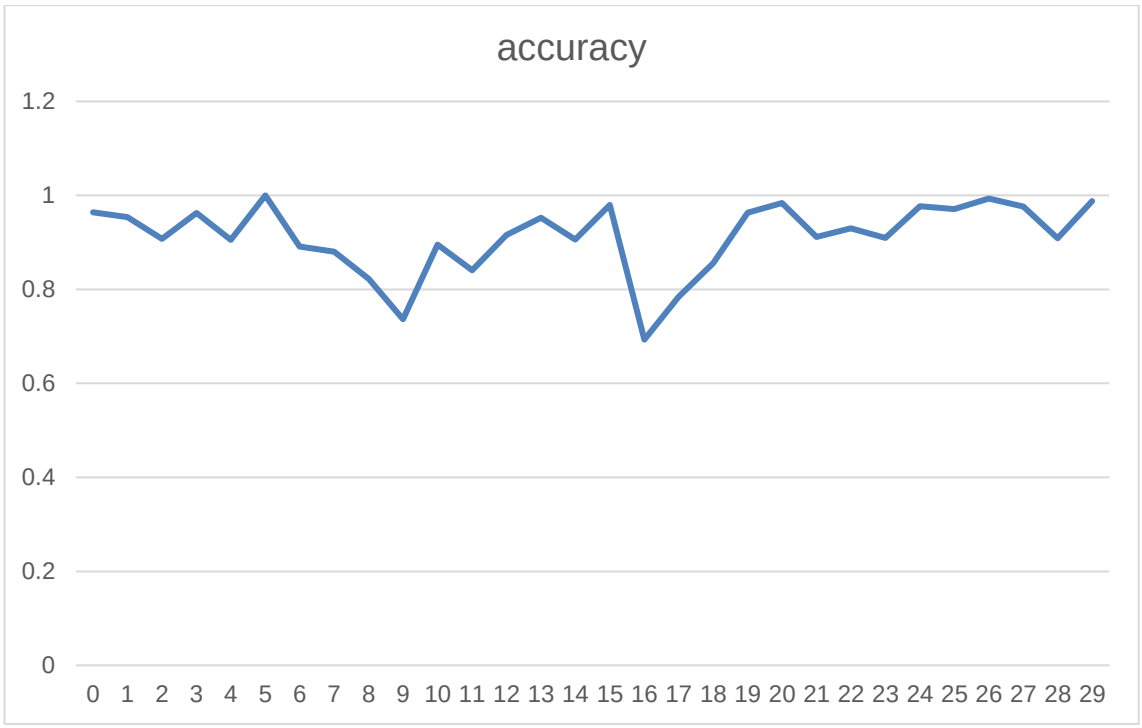

Figure 26: Company 3 simulation accuracy

In the case of company four the answer percentage was sailing between 80 and 100 most of the time. For the last six months, the answer percentage was 100. Accuracy displayed some peculiarities as it was 100% most of the time but dropped to around 90% on three occasions.

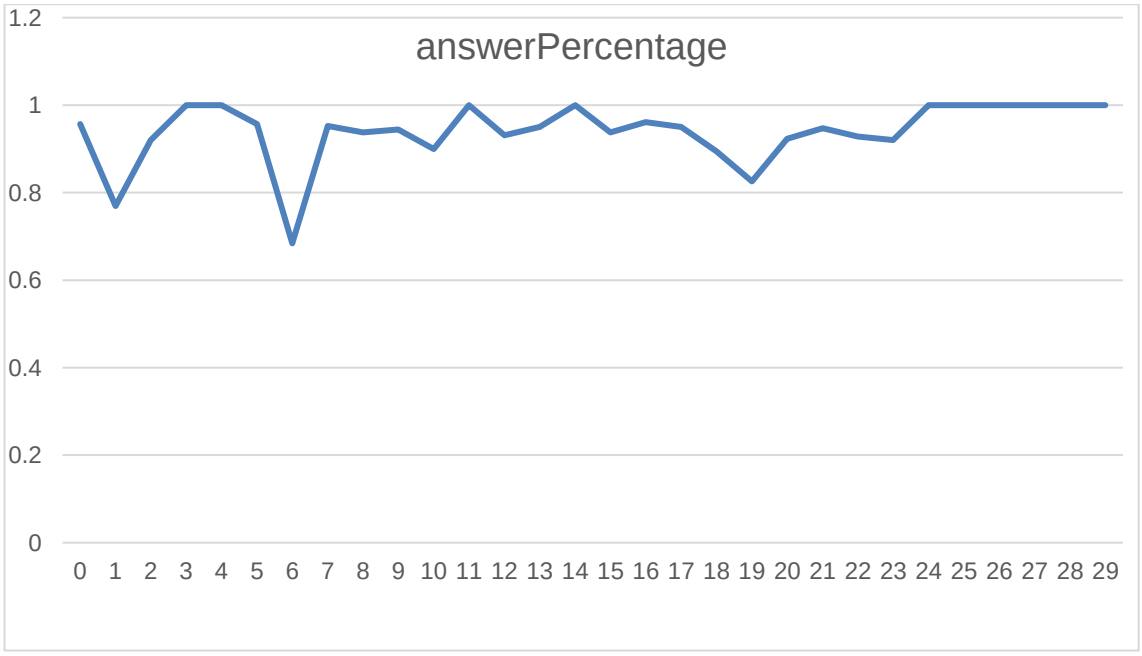

Figure 27: Company 4 simulation answer percentage

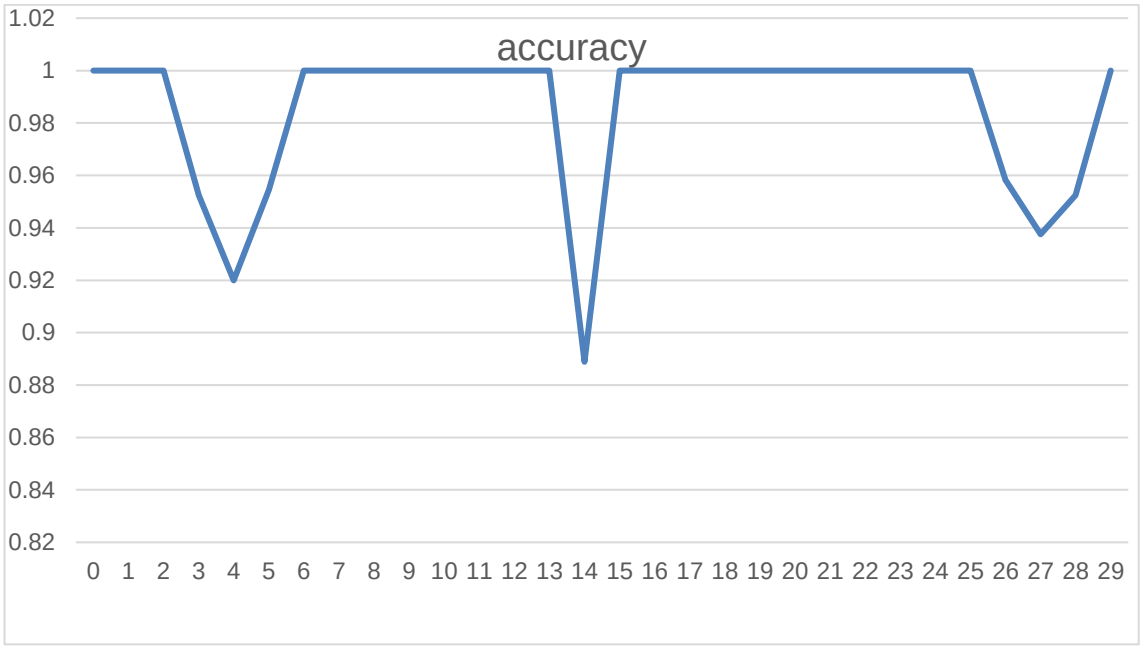

Figure 28: Company 4 simulation accuracy

As for the company five, the results show some clear instability. While both accuracy and answer percentage are between 80% and 100% most of the time, there seems to be constant movement in the graphs and no settling in sight.

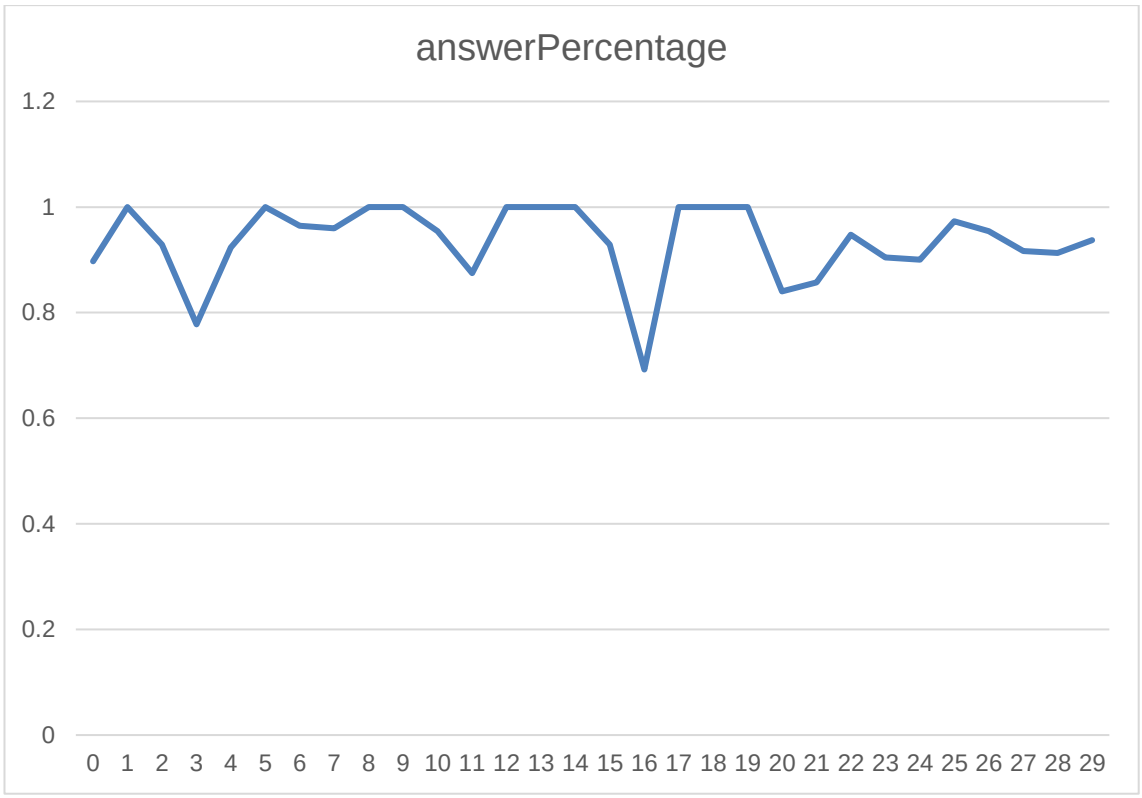

Figure 29: Company 5 simulation answer percentage

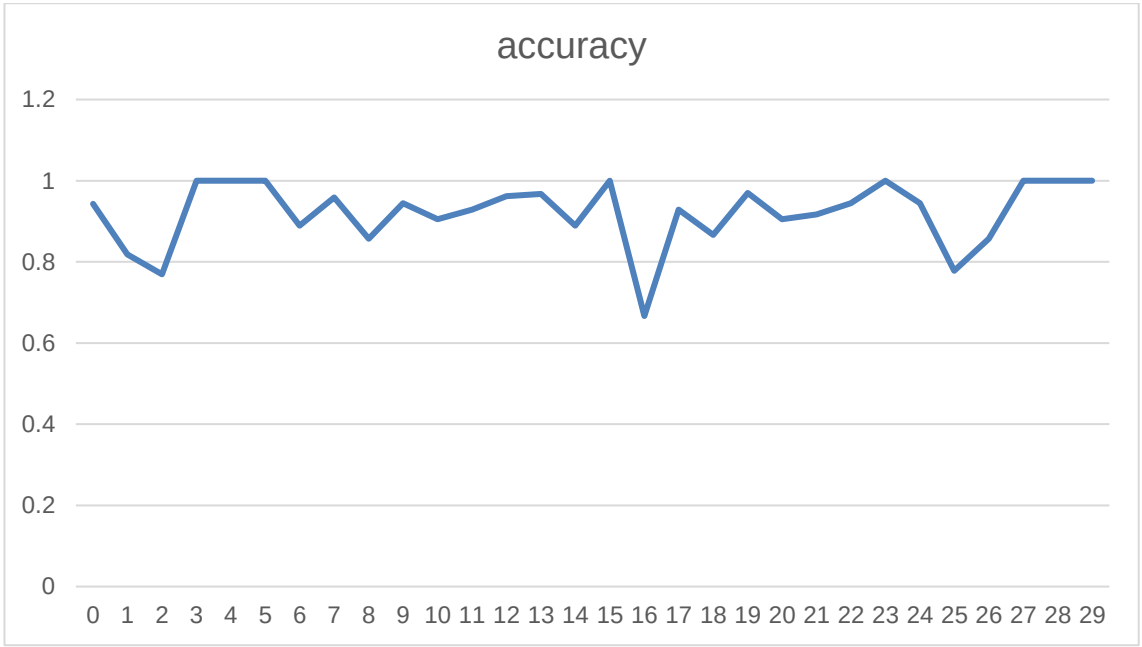

Figure 30: Company 5 simulation accuracy

Finally, to make it easier to see overall trends, below there are two figures summarizing answer percentage and accuracy respectively.

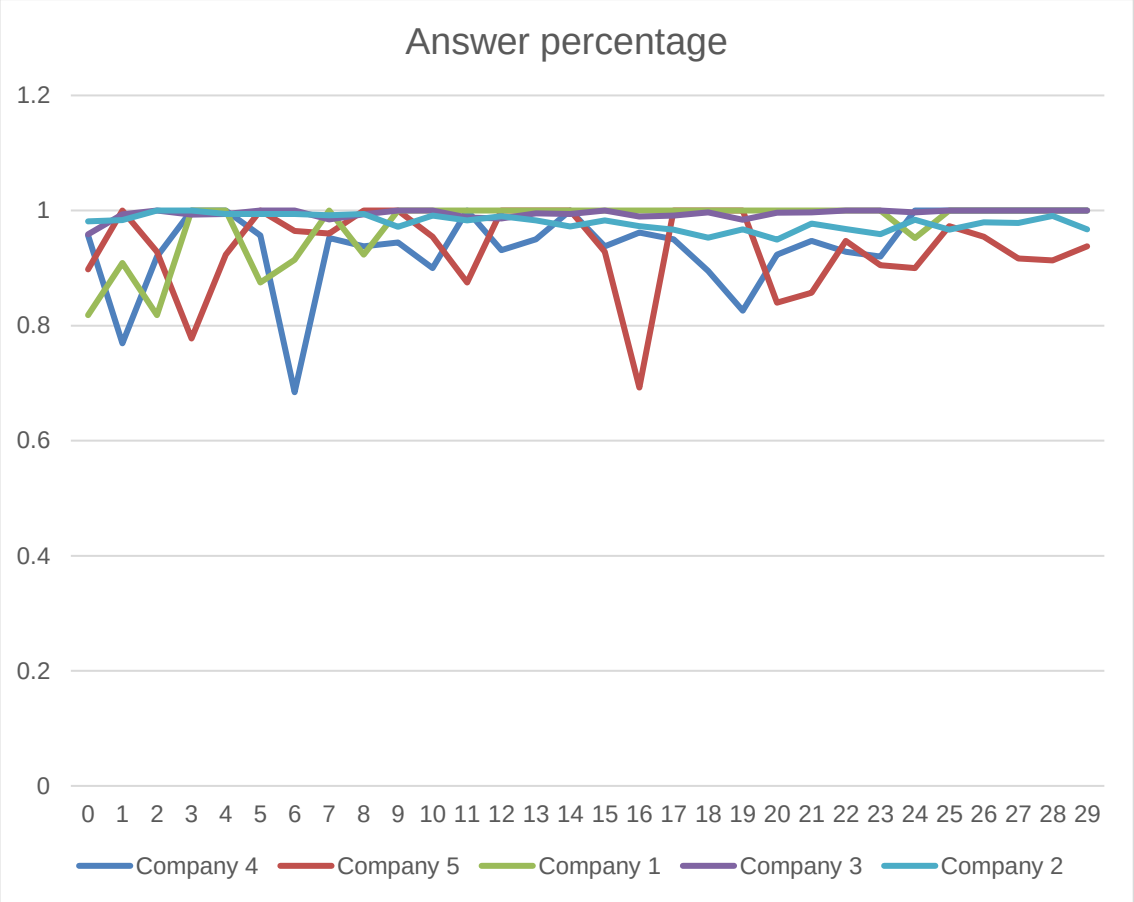

Figure 31: Summary of answer percentage

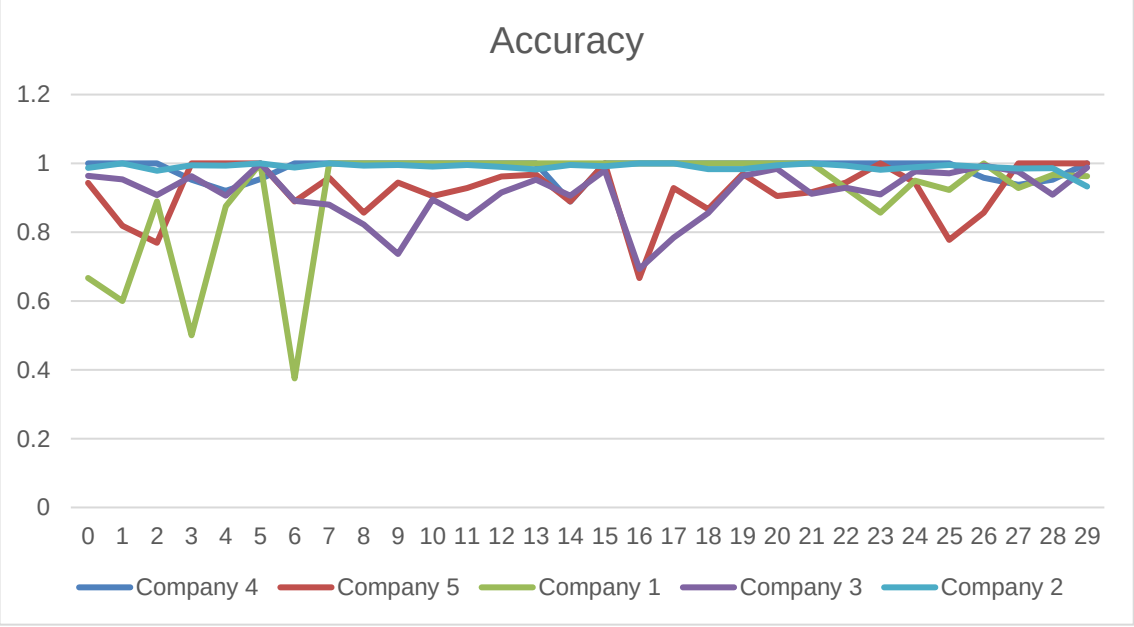

Figure 32: Summary of accuracy

5.2 Evaluating tech choices

This chapter will evaluate how the author felt that the selected tools fit the project after the technical implementation had been done. Infrastructure was not the main focus of this thesis, but due to clear issues that came up with infrastructure, that topic will be covered in this chapter as well.

Python was definitely a good choice for the project. All the tools needed were available and there were a lot of answers available online to any questions related to working with the frameworks in Python language.

TensorFlow also felt like a good choice, there was nothing missing from the framework and development was fluid and observing the results was easy with TensorBoard. Both TensorFlow and TensorBoard felt like good choices.

Pandas was used very minimally. It contained a lot of helpful functions that could not be utilized since all data could not be read into memory at once. Therefore, many operations that Pandas library offered had to be recreated by the author to fit large dataset purposes. In hindsight, Pandas was not really a mandatory addition.

NumPy was very useful, and mandatory because Tensorflow uses it as well. Most likely, NumPy will be needed for any machine learning project in Python.

The biggest hurdles on tech side were more infrastructure related. This thesis was run by using a single PC, computing everything locally. Especially during the early phases were the networks were in very early stages, they had to be iterated again and again and the iterations were painfully slow on a single machine. Without doubt for future development scaling computing solution must be available. Whether it is private cloud locally managed or public cloud it does not matter, but there must be an easy way to distribute computing and try several experiments simultaneously. Biggest chunk of time went to waiting for iteration cycles to complete. Additionally, when mass training models for multiple companies, it would be very beneficial to be able to split the task over several machines. Database used in this thesis was MySQL and while there were no major issues, exploring the programming interfaces of the frameworks suggested that perhaps some kind of filesystem solutions are more common practice with the machine learning environments.

6 Conclusions and discussion

6.1 Unintended oversights

A big oversight was assuming that all bank statement data could be used. It was not noticed until evaluating results, that while companies using payment-based accounting and ones using double entry bookkeeping are both receiving and sending payments from similar parties, they are marked down very differently for the bank statements as the double entries have a more similar marking at the invoice level and then a simpler one on bank statements. Hence the problem the author was studying was unintendedly more complex as it included both kinds of data as well as not evaluating the requested results. This affected the general and business-line specific models but should not affect the company level networks.

When studying the results, it also came to light that there are some rare cases where a single bank statement event can be split into multiple ledger accounts. The solution built in the thesis is completely based on the assumption that the data could be mapped 1:1 with ledger accounts.

Lastly, the originally planned evaluation metrics by using dev-set and test-set did not work at all for the problem. It took a while to understand this and come up with a new evaluation strategy, and due to this the time left for iteration after the final evaluation was very limited, practically non-existent.

6.2 Why potential is not perfect

Why the potential can’t reach 100%? In the case of general model, this is to be expected. As different companies might have differences in their accounting and the general model doesn’t study differentiate individual companies, it learns to pick the most likely choice and as we can see, this leads to a roughly 87% accuracy, which speaks of quite a big

similarity in the data between different companies. Meanwhile business line specific accuracy was around 92%, which proves that using business line specific models can benefit in comparison to general model. When it comes to individual company models some were able to reach 100% accuracy in potential, but for others it was not possible. So, the question is why not? The answer is simple: from the used values it is not possible to differentiate some answers from each other. In the other words similar data entries are producing different kinds of results. As mentioned one reason was the oversight on multiple possible ledger accounts. On other occasions there must be extra information that has been used to determine the result, which was not used in this thesis. First things that come to mind are the message text that was left out due to increased complexity and information available in possible attachments e.g. an image of a receipt.

6.3 Unsatisfied potential

Why is the simulation result worse than the potential? Why was the potential not satisfied? Simply put there are some new values coming in every now and then and predicting them based on the previous values is not possible. General and business line models handle these cases quite well on average, but all case companies seem to have some data that differentiates from the general predictions.

6.4 Conclusions from the results

There was instability in all simulation cases, regardless of the potential. In the case of company one there was a clear improvement in the accuracy as the company specific network was learning from previous events, for the other cases, there seemed to be no clear improvement from the training of the company network. Looking back at the potentials, some of the company specific potentials can reach 100% accuracy, while business line potential hovers at 92%. In case of small companies, overfitting was possible, which was not the case for general networks with lots of data in comparison to network size. The original overfitting tests were done with the general networks and there was not enough time to sufficiently test good regularization amounts for the company level networks. In addition, had the company level networks been dynamically adjusted to the data amounts, the training would be a lot faster and the overfitting would’ve been easier

to control. The size of the network was chosen by originally running tests with the largest company and dynamical adjustments were left as a future improvement.

6.5 Improving the POC

To really answer the question can it help with payment-based accounting ledger posting the very first thing would be to rerun the tests using only payment-based data. Due to misunderstanding of the data, as mentioned in chapter 6.1 the thesis does not really answer the originally deployed example question. Also, as mentioned dynamically scaling the company level network sizes, possibly also business line networks sizes and optimizing regularization could greatly improve the performance. The ledger account was picked as an example value, but the long-term goal would be to provide other values automatically as well. It occurred to the author that there might be more connections between the different values than first thought. For example, to determine ledger account correctly under the hood the VAT type must be determined. Currently this happens under the hood in the network, but it could be easily determined by a second network. Then it could be fed to the ledger account network as an already determined value, reducing the complexity of the ledger account network. VAT type is a clear example, but it's very likely there are connections between other values as well. For example, if the VAT percentage could be determined from an attachment image, that could provide some hint as well. The connection between the values should be studied and it could be used to reduce complexity and provide accurate answers. The correct order of determining the values is important for the result, if aiming for being able to provide all ledger posting values. In the original info gathering interviews it was discussed that most likely fields that can be used to determine the values are the receiver name and the sum. At least with the added complexity of not knowing whether it's payment-based company or not, these values were not enough to determine it reliably enough for production use.

This brings us to another potential improvement suggestion. It was determined that categorizing business-lines improves the accuracy clearly. It occurred to author, that it might be highly beneficial to do either clustering or categorizing to divide the companies into several different categories, not restricted to business-lines only. Currently in the database there is no knowledge what kind of accounting the company uses, that is between the company and accounting firm and does not require different behavior from the pro-

gram, this could be something determinable by clustering or categorization. So, it's possible that using clustering or machine learning categorizing we could end up with far more effective ways of categorizing than the business-lines and that could be the real key to very accurate percentages.

In the hind-sight the metrics still need improvements. Instead of checking the potential of a random business lines against the general one, it would be much better to run every single business line potential and then get metrics such as average and median of those. The same thing should be done for the business lines, or in the different categorization cases, similarly for those categories. Also, the potential overall was studied as running set of batches of varying sizes, which gave a good idea, but made it impossible to pinpoint the exact accuracy. Maybe running the whole test data as an evaluation data to get the exact percentage in the whole data would give a more exact percent at a given point of training.

Finally, the current solution is built on static context. The production solution needs regular retraining, especially on company level networks. This was taken into account during the thesis and the models are quick to retrain from scratch and it's also possible to continue the training from a checkpoint (making it even faster), though continuing from check point would create bias towards the new results (which might or might not be a problem).

7 References

[1] N. Shukla, ”Machine Learning with TensorFlow MEAP v10,” Manning Publications Co. Kindle Edition., p. Kindle Locations 76.

[2] A. Ng, ”www.coursera.org // deeplearning.ai,” [Online]. Available: https://www.coursera.org/learn/neural-networks-deep-learning.

[3] N. Shukla, ”Machine Learning with TensorFlow MEAP v10,” Manning Publications Co. Kindle Edition., pp. Kindle Locations 284 - 324.

[4] N. Shukla, ”Machine Learning with TensorFlow MEAP v10,” Manning Publications Co. Kindle Edition., pp. Kindle Locations 326-327.

[5] N. Shukla, ”Machine Learning with TensorFlow MEAP v10,” Manning Publications Co. Kindle Edition. , pp. Kindle Location 340-360.

[6] E. A. Aji Mubalaike Mubarek, ”Multilayer perceptron neural network technique for fraud detection,” tekijä: *2017 International Conference on Computer Science and Engineering (UBMK)*, Antalya, 2017.

[7] N. Shukla, ”Machine Learning with TensorFlow MEAP v10,” Manning Publications Co. Kindle Edition, pp. Kindle Location 5168 - 5188.

[8] N. Shukla, ”Machine Learning with TensorFlow MEAP v10,” Manning Publications Co. Kindle Edition, pp. Kindle Location 5372-5425.

[9] N. Shukla, ”Machine Learning with TensorFlow MEAP v10,” Manning Publications Co. Kindle Edition, pp. Kindle Location 5536 - 5570.

[10] N. Shukla, ”Machine Learning with TensorFlow MEAP v10,” Manning Publications Co. Kindle Edition, pp. Kindle Location 5897 - 5922.

[11] A. Ng, ”www.coursera.org // deeplearning.ai,” [Online]. Available: https://www.coursera.org/learn/deep-neural-network.

[12] ”TensorFlow graphs and sessions,” [Online]. Available: https://www.tensorflow.org/guide/graphs.

[13] ”TensorFlow estimators,” [Online]. Available: https://www.tensorflow.org/guide/estimators.

[14] N. Shukla, ”Machine Learning with TensorFlow MEAP v10,” Manning Publications Co. Kindle Edition, pp. Kindle Location 4140 - 4192.