VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
DEPARTMENT OF COMPUTER SCIENCE II

Master's thesis

Text analytics for crypto-currency price predictions

Done by:
Pranas Kieras signature

Supervisor:
dr. Linas Bukauskas

Vilnius
2018

Contents

Glossary

ANN - artificial neural network
CNN - convolutional neural network
RNN - recurrent neural network
LSTM - long short term memory used as memory cells in deep recurrent neural networks.
GAN - generative adversarial network
REST - representational state transfer
GPU - graphic processing unit
CUDA- Compute unified computer architecture
ReLu - rectified linear unit
Crypto-currency - decentralized trust-less currencies usually built on a block chain
MCMC - Markov Chain Monte Carlo, probabilistic machine learning method.
TAN - Tree Augmented Naive Bayes algorithm.

Abstract

This masters thesis is aimed at predicting crypto-currency price movements using social media data. The main motivations to do this research is the recent developments deep learning and the vibrant crypto-currency market. Furthermore, the is a lack tools for fundamental analysis of the currency prices. The experimental part of the work has been split up into 3 parts. The first part is data collection. For this a REST client application,that draws the tweets related to crypto-currencies and saves them to a csv file, was created. Price data is also collected for the experiment using a REST client . The second part of the work is data cleaning an preprocessing. The twitter feed is cleaned of duplicates that are not relevant to the experiment. Also, the tweets themselves are cleaned and pruned removing tweets that are too long and too short, promotional tweets and other noise in the data. The price data is also cleaned and gaps in the data are filled. In the third part the 4 different deep learning models are created. The first one is a multilayer recurrent neural network with LSTM cells and embedded inputs. The second model is a convolutional neural network with parallel convolutional layers that also uses embedding for inputs. The third model is a stacked convolutional neural network with 2 hidden layers and an embedding layer. The final model is a recurrent neural network that has no embedding layer and operates solely on the price movement data. The experiments are run on a toy dataset to validate that the model is working properly and then on the real dataset. The results of the experiments indicates that the social media data is not enough to predict price movements with any degree of certainty. The main problem is the quality of the data. The results could potentially be improved if other data sources would be used, that specialize on investment and crypto currencies.

Santrauka

Kripto valiutų kainų prognozė naudojant teksto analitiką

Pagrindinis šio darbo tikslas - prognozuoti kripto valiutų kainas, naudojant socialinių tinklų informaciją. Darbe pasitelkiami giliojo mokymosi metodai - rekurentiniai ir konkvoliuciniai neuroniniai tinklai. Experimentinėje dalyje sukuriamas duomenų surinkimo skriptas bei įgyvendinami keturi skirtingi giliojo mokymosi modeliai. Tik vienas iš modelių sugebėjo užfiksuoti šiek tiek geresnius negu atsitiktinius resultatus su testiniu duomenų rinkiniu. Pagrindinė prastų resultatų priežastis - nekokybiška tekstinė informacija. Juose daug dublikuotos informacijos ir triukšmo, kurį sunku išvalyti automatizuotais skriptais. Kita prastų rezultatų priežastis - nenuspėjamas kainos judėjimas, dažnai nesusijęs su tuo kas skelbiama socialiniuose tinkluose. Darbo rezultatai galėtų būti pagerinti naudojant tikslensius informacijos šaltinius, kaip specializuotos platformos, kombinuojant juos su kainos informacija, kaip įvestį į modelį.

Introduction

The motivation for this masters thesis comes from the recent developments in deep learning. Systems based on algorithms like Inception and Google translate are showing great results in computer vision and machine translation. They are outperforming humans in many tasks.
The other source of motivation is the developments in the crypto-currency space. There has been a lot of money moving into digital currencies and a lot of it was based on subjective factors that cannot be identified with fundamental analysis.
The speculative nature of the price movements indicates that there is a network effect influencing the price. This network effect can be tracked through social medial like twitter, facebook or reddit.
The purpose of this thesis is to predicting crypto-currency prices using social media information.
The goals to achieve this purpose are:

1. Create a system to effectively gather and process textual social media data on crypto-currencies.
2. Identify and fix problems in the textual and price data - like missing values, noisiness and duplication.
3. Find the best deep learning algorithm for price predictions.
4. Find the link between social media data and price movements, if it exists.

There are a few main problems in creating a good price predicition model. Firstly, finding the right data source is difficult, because there are a lot of options with different characteristics. In this thesis multiple data sources were considered. The pros and cons of all of them are evaluated by 4 different criteria.
Secondly, most of the main textual data sources are either expensive or do not allow to download data for the required period. This problem is solved by creating a client applications that draws live twitter data through their API. This application also handles errors and gathers a continuous dataset with as few gaps as possible. Furthermore, a client that gathers historical minute by minute data is created.
Secondly, there are data quality issues that are addressed in this work. The data has gaps and missing price values and a lot of duplicates. The repeating items are removed and noisiness in the data is cleaned as much as possible without corrupting the dataset.
Finally, potential deep learning model architectures are prototyped. Then the models are validated on a toy dataset to check if they can generalize on the given problem. The same models are tuned by searching and identifying the best hyper-parameters.
The masters thesis is organized into 2 parts. The first part is just a gentle background introduction into the field of deep learning and crypto-currencies. Some practical and theoretical details are discussed justifying the decisions made in selecting certain cryopto-currencies, resources or techniques.
The second part of the thesis focuses on the end to end implementation of different models. There is detailed information on how the data sources are selected and the information is collected. Most of the challenges that occurred during this stage are specified along with the solutions that helped to overcome them.
Furthermore, there is a detailed description of the deep learning models that are used. Some justification for the overall architecture and certain design decisions are laid out. There is also a comparison of the implemented solutions which highlights the pros and cons of each model and

ways on which they could be improved.
All in all, after running training the 4 models there no direct link was established between the twitter data and price movement. The final validation and test accuracy on all 4 model types were too low, to prove a strong relationship. However, the simple convolution model with 2 stacked layers has shown the best results out of all the models that were tested.

1 Related Work

Sentiment analysis is quickly becoming an important topic in text mining and text analytics. It has wide applications in both academia and business. In recent years there has been a growing interest in analyzing sentiment for predicting various real world events like sports matches, election results and stock prices.
In the recent past a few surveys have been conducted to overview the work done in text analytics for predicting stock market behavior. A recent study by Kumar and Ravi [5] highlight the following 8 types of scientific techniques used in sentiment analysis research:

1. Support Vector Machines (SVM's)
2. Naive Bayes (NB)
3. k-nearest neighborhood (KNN)
4. Decision tree (DT)
5. Multilayer perceptron
6. Linear Regression (LR)
7. Classification and regression trees (CART)
8. Group method of data handling (GMDH).

The researchers point out these machine learning techniques are well know to deliver good results for a broad range of Natural Language Processing problems. However, this survey does not mention some of the state of the art techniques like Deep Learning and Recurrent Neural Networks with Long Short Term Memory [4], which are becoming more popular in real work applications. In another survey done by Cavalcante, Brasileiro, Souza and Nobrega [1] the biggest emphasis is put on the same techniques as study [5]. They focus more on applying text mining in the financial domain. The authors suggest that a major part of successfully predicting stock prices lies in feature selection and extraction. What is more, they provide evidence that hybrid techniques, like ensemble models, outperform the traditional machine learning algorithms. Once again, the newest trends in the field, like RNN's, are not considered. However, Cavalcante, Brasileiro, Souza and Nobrega mention that the applications of Deep Learning in financial forecasting are unexplored [5]. From both surveys done quite recently it is clear that up until now traditional modeling techniques were more popular for text analytics and financial market predictions. In part, this may be because it is easier to explain how basic machine learning models work with clear rule sets of how the input maps to the output. The intrinsic problem with Deep Learning is that currently researchers only have an intuitive understanding of why and how such models work and there is no clear mapping from input to output.
Furthermore, by looking at similar studies it is clear that most of them focus on well know and established machine learning techniques. Song, Kim, Lee, Kim and Youn [18] propose a novel approach of using Naive Bayes in mining Twitter sentiment. The study focuses on a way to use a running positive and negative word count and eliminate the use of insignificant words. However, the authors admit this approach suffers from a hyper-parameter selection problem. Chu, Wong, Chen and Lee [3] go further and propose to use text mining to extract sentiment from both numeric and textual data. What is more, they introduce the idea of sentiment-of-topic (SoT). The authors

manage to get good results, by mining a wide range of textual and numeric sources like news feeds and company revenue reports. On the other hand, the model is quite complex and requires a lot of fine tuning and expertise in the field of finance. Chousa, Cabarcos and Pico [11] use a focused approach and collect data only from financial micro-blogging platforms like StockTwits. They target bloggers who are well established in the investment and trading community. This allows them to filter out a lot of noise that would come from sources like Twitter. The research [11] utilizes the Stanford CoreNLP Sentiment Treebank to classify the twits into 3 categories. Similar to Margoudakis and Serpanos [5] the researchers apply Bayesian statistics to extract sentiment from textual data. However, in their model the authors employ a Markov chain Monte Carlo method (MCMC) to estimate the conditional probability distribution of network structures that are obtained by a Tree-Augmented Naive Bayes (TAN) algorithm [11]. The main drawback of this method is that the training data needs to be manually annotated. Furthermore, MCMC inference is time consuming at each TAN construction phase.

Yang, Mo, Liu and Kirilenko [20] propose a completely different approach of genetic programming optimization for the hyper-parameter search. Similar to [3] multiple sources of data are used (various news-feeds along with Twitter). They use network metrics like meta-data in their model. This enhances the basic sentiment analysis model. Li, Xie, Chen, Wang and Deng [7] use a generic stock price prediction framework and plug-in six different models. They use the Harvard psychological dictionary as the base reference for sentiment analysis. The researchers conclude that even though their model is better than a simple bag of words approach sentiment analysis is not sufficient to accurately predict stock price movement. Schumaker, Jarmoszko and Labedz [16] use Twitter to mine the public sentiment for Premier league wins spread predictions. The authors use OpinionFinder to categorize tweets into 8 categories: Model 1 — Subjective Negative tweets, Model 2 — Objective Negative, Model 3 — Subjective Positive, Model 4 — Objective Positive,Model 5 — All Subjective, Model 6 — All Objective, Model 7 — All Negative, and Model 8 — All Positive. The study produces good results in matching sentiment to match outcomes. However this is a limited use case that is difficult to implement for the financial domain. This is because the matches are one time events and stock prices are continuous. It also indicates that many more factors need to be taken into consideration, like temporal data and periodicity of the measurement.

The research by Pagolu, Challa, Panda, Majhi emphasize the importance of the effective market hypothesis [10] in stock price predictions. They use word2vec and N-grams to model tweet sentiment. The researchers find a strong correlation between the rise and fall of stock prices and public sentiment. The article by [8] Nguyen, Shirai and Velcin use Support Vector Machines to evaluate 6 different methods of data collection: 1) Price Only, 2) Human Sentiment, 3) Sentiment classification, 4) LDA-based method, 5) JST based method, 6) Aspect-based sentiment. In this experiment the best results were shown by the Aspect Based Sentiment model. The research focuses most on data collection and labeling. The authors put less emphasis on the modeling part presuming that SVM's are a good enough tool to do the job. Olivera, Cortez and Areal [9] find that micro-blogging negative sentiment can be a good predictor for price drops. They also find that positive sentiment may predict price increases but the relationship is much weaker than with negative sentiment. Checkley, Higón and Alles [2] discuss the importance of time horizons in stock stock price predictions. They conclude that news can have an immediate impact on prices changes so picking a very short time horizon is important. Their biggest contribution is using very granular temporal intervals for sentiment analysis. The authors predict at a 2 minute time horizon. They provide evidence that in modern stock price forecasting even such short intervals are not enough as the markets change almost instantly at any news about a particular stock. Researchers in another

study also uses StockTwits as the primary source of data [19]. The authors focus on the statistical properties of twits rather then sentiment. Furthermore, they target micro-blogging sites as their only source of data. Sun, Lachanski and Fabozzi sets their study apart by develop a model based on sparse matrix factorization to predict stocks prices.

The related literature review provides a few important insights. Firstly, most of the studies stress the importance of data collection and preparation and the modeling is usually quite basic. However there are a few studies like Chu, Wong, Chen and Lee [3] where the models are quite complex and need a lot of manual tuning. There are only a few articles that use more advanced modeling techniques and none that use Recurrent Neural Networks with LSTM's.

Secondly, most studies use micro-blogging services as the data source. This is because such sites provide real time, limited length semi-structured data. Some studies pull information from Twitter filtering by hash-tag. Other also use hash-tags, but follow only the well established experts. Another data source that was mentioned in literature often is StockTwits mentioned in [2] and [11]. It is a good starting point for data collect.

Finally, most studies use sentiment analysis as their primary decision tool. In this master's thesis a few models will be compare. Sentiment analysis is used with more basic techniques to provide a benchmark for further modeling. More advanced tools such as sequence to sequence mapping with RNN's are the main focus of this research.

2 Background information

2.1 Crypto-currencies

Crypto-currencies are a relatively new technology, starting in 2009 with Bitcoin and expanding rapidly during the past 3 years. Bitcoin is still the main crypto-currency with a market cap of 250 billion dollars and a market dominance of 48%. However, many alternative coins are gaining ground. Protocols like Ehtereum, Ripple, Litecoin have proposed solutions to a lot of the issues that Bitcoin is facing.
At the moment of writing there are 1373 crypto currencies listed on Coinmarketcap [13]. Most of them are bellow 1 million dollars in market cap and do not have a big following on social media. Bitcoin is the most prominent and well established crypto-currency and is the focus of research in this masters thesis.
There are multiple exchanges that have different prices for the same currency fluctuating 10-20% between them at any given moment. Binance is one of the most popular exchanges, that is why it was selected as the source of price information.

2.2 Artificial neural networks

In its simplest form a artificial neural network is a sum of its inputs multiplied by weights in the synapses connecting those inputs to the neuron in the next layer with a bias value added to it 1.The output unit returns the result of some activation function $f(h)$, where h is the input to the output unit see 2.1.

$$h = \sum w_i x_i + b \tag{2.1}$$

At its heart ANN's are just another machine learning model that enables the mapping from a set of inputs to a set of outputs. ANN's are valued for their ability to approximate any linear and non linear functions.
During training artificial neural networks minimize the error between the expected output and the

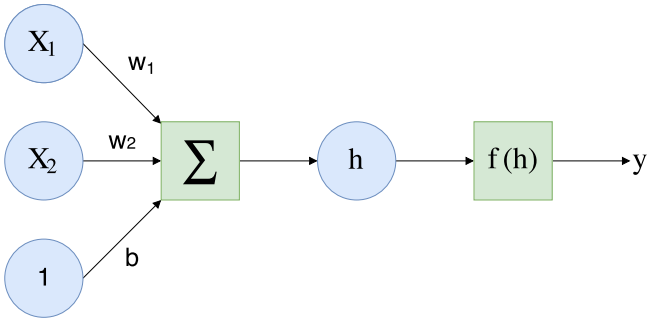

Figure 1. The simplest architecture of an Artificial Neural Network with 2 input nodes, and one neuron.

actual prediction, given an input values. The most widely used error metric in ANN's is the mean squared errors or MSE 2.2. This is the metric that is going to be utilized in section 3 to calculate calculate the cost of a training run.

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(Y_j - \hat{Y}_j)^2 \tag{2.2}$$

The main strength MSE is that the result is always positive and large errors are penalized more than small ones, which is crucial when working with textual data embedded as integers.
In this master thesis batching is used to facilitate the learning process by not calculating the error for each training example. Then the error for the batch is calculated by averaging the errors for each individual training example in the batch. This gives a smoother descent down the loss function.
Gradient descent makes small steps towards a lower value of the cost function. To find which way the step needs to be taken, the tangent of the loss function is found. It points in the direction where the loss function is smaller (see figure 2). In this example x would need to decrease by Δx to get to a lower value of the loss function. Backpropogation is the driving force behind all of the deep

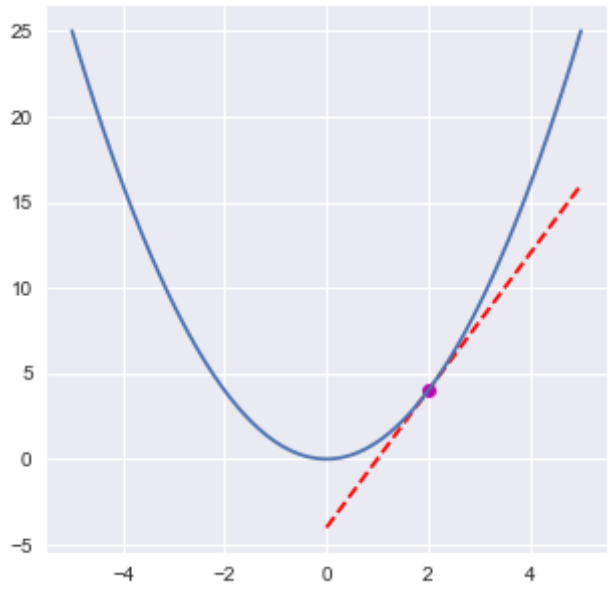

Figure 2. the tangent of function $f(x) = x^2$ at point $x = 2$

learning algorithms. It allows the use of gradient descent throughout the whole deep network by applying the chain rule [4]. The chain rule is a method that calculates the partial derivative of the error with respect to a weight at any layer from the error in the previous layer and the current values of the weights.
The output of a each layer is determined by the weights between the layers. The error from the neurons is scaled by the weights going forward through the network. Since the error at the output can be calculated, it is possible to work backwards to hidden layers using the weights. If the error for the output layer is δ_k^o for each output unit k, then the error for a previous layer j is the output errors, scaled by the weights between the layers.
The algorithm computes a gradient vector that shows how each weight in the network needs to be changed in order for the error to decrease.

2.3 Introduction to deep learning

Deep learning is a technique used by many new AI technologies. Simply put DNN is just an ANN with more than 1 hidden layer. In practice the number of layers can be as many as 1001 [17]. There are many different architectures to deep learning systems, but they all share a common trait. In their article in Nature [6] LeCun, Bengio and Hinton point out that deep learning allows computational models that are composed of multiple processing layers to learn representations

of data with multiple layers of abstraction. Deep learning improves the the benchmark results in speech recognition, visual recognition, natural language processing and many other fields. Given the right architecture enough data and enough computational power deep learning networks can outperform any traditional machine learning algorithm. The main strength of DL is that it can uncover hidden structures in data, because of the capacity for approximating even the most complex functions.
The main drawback of deep learning is that it requires a lot of data and powerful hardware to be effective. Furthermore, due to their large representation capacity DL networks tends to over-fit the data and not generalize well with new examples, if the training is carried out incorrectly. However, there are existing solutions to avoid over-fitting and some of them are used in this master thesis. The training data also needs to be labeled.
Finally, deep learning has become a go to technique for modern AI applications. Many real world problems rely on rules that for humans seem like pure intuition and can not be pre-programmed. For example, classifying pictures of dogs and cats it would be hard to tell a computer each and every possible difference that the two classes might have. Deep learning solves problems like these by finding subtle abstract representations for important features like shape of nose and eye placement while at the same time ignoring huge features, like the size of the body and the scale of the picture. The same representation capacity is useful in natural language processing. Other techniques concentrate on small scale features like n-grams and their placement in the sentence. By having multiple layers of abstraction a deep network can find features that signal abstract relations like sarcasm, passive aggressive sentiment and similar. The language case is less intuitive than finding objects in pictures, because language in itself is an abstract representation of the world. Subjectivity is also a major hindrance since a statement might be perceived as sarcasm by some and taken for its literal meaning by others.

2.4 Recurrent Neural Networks

Section 2.3 touches upon a variety of deep learning neural network architectures. There are convolutional neural networks(CNN's), generative adversarial networks(GAN's) and many more. The main focus of this thesis is on a specific variety called recurrent neural networks (RNN's) [4]. It is a type of architecture that is well suited for learning sequential data, where the next item in the sequence is determined by the previous items. RNN's process input sequential one element at a time [6]. The size of the element can be chosen. Some RNN's operate on a character by character basis, other process sequences word by word.
The main problem with recurrent networks is that at each time-step the gradients of the network grow or shrink. Since the network feeds back on itself after many iterations the gradients tend to either vanish or explode. Usually RNN's employ some kind of a memory model, that saves the hidden state of the network. For example if a network is encoding a sentence it can save that sentence as a representation in memory and then use it when initializing the next sentence. Figure 3 shows a simplified version of a recurrent neural network.
All in all, RNN's can be seen as very deep neural networks where each layer represents a step in time. At each time step it gives a prediction for what the next element in the sequence could be. This makes the learning process of back propagation very similar to any other ANN. There are also multiple hyper-parameters that can be used to tune this network, like sequence length and number of iterations. In their book Goodfellow, Bengio and Courville [4] present a few different memory models. The most widely used is the long short term memory. LSTM is the model that is going

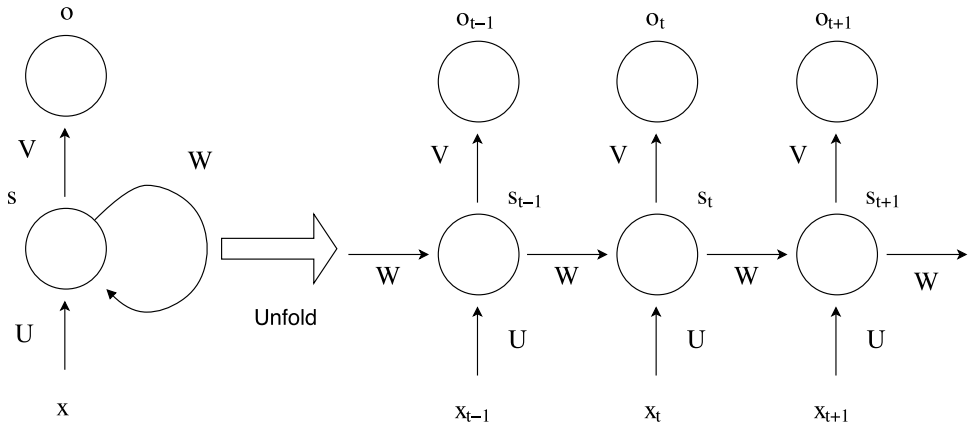

Figure 3. A recurrent neural network unfolded in time

to be used in this master thesis. It transfers cell state via a combination of remember and forget gates, which selectively keep and discard the relevant information for predicting the next item in the sequence.

2.5 ANN regularization techniques

Regularization is a list of techniques to decrease the test error. In [4] regularization is described as any modification to a learning algorithm that is intended to reduce its generalization error but not its training error. This is usually done by regularizing estimators or in other words trading increased bias for decreased variance in such a way that variance decreases significantly while the bias increases slightly.

There are many regularization methods. This chapter mentions a few that are used in this masters thesis. They have been chosen over others, because they tend yield better results when combined with recurrent neural network models and LSTM's. Some of these techniques are just common practices well know to deliver good results for increased generalization. Early stopping is a technique described in [4]. Most ANN hyper-parameter sets have a U shaped validation and test set performance curve. This means that at some point the model stops generalizing and begins overfitting. Early stopping allows to find hyper-parameters that stops the training just in time. It also controls the capacity of the model.

The main drawback of the early stop approach is that the validation set evaluation needs to be run in parallel with training. This is usually done on separate CPU or GPU. Furthermore, when using early stopping a copy of the best parameters needs to be kept. This regularization approach can be used either on its own or with other regularization techniques that are discussed in sections **??** and **??**. Dropout is another very effective, computationally inexpensive regularization technique [4]. At first glance it may look similar to ensembling as it involves training a large ensemble of deep neural networks. It done by training the on a subsets of a smaller neural networks constructed from the original network as displayed in figure 4. The figure shows the most basic deep network with two input units x_1 and x_2, two hidden units h_1 and h_2 and one output unit y. The number of all possible combinations of sub-networks by dropping non output units is 16. It is clear that some of the network architectures are unusable as there is no input unit or there is no path from the input to the output. However, in a larger deep network architecture this problem becomes insignificant, because this type of arrangement become very unlikely as the network becomes wider and deeper. The main difference between dropout and other ensemble methods is that the sub-networks

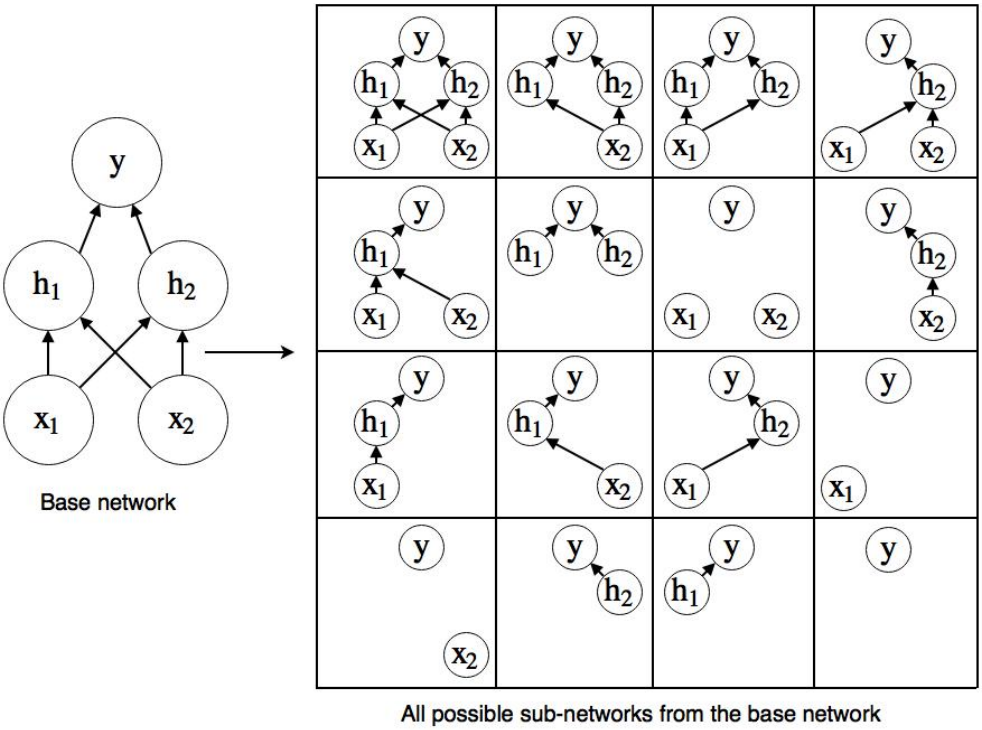

Figure 4. The list of possible subsets of a neural network that has 5 nodes, where some of the variations are invalid.

in dropout are not independent and are not trained separately, therefore the computational costs are insignificant when compared to an ensemble model where there are more than a few deep networks being trained. The main idea of dropout is to disable some of the non output units in the network during the training process on different iterations. This way the training is impeded but the overall generalization capability of the network increases [4].

2.6 ANN optimization techniques

Firstly, a distinction needs to be made between learning and optimization. Learning or training in the context of artificial neural networks is a process by which the network finds the best weight set to approximate a predictor function from the data that it has been given. The Deep Learning Book [4] introduces several model optimization techniques. Deep learning algorithms are highly influenced by the initial parameter selection. Since they are iterative in nature the final state of the network is deeply linked to the initial state of the network. Initialization selection can mean the difference between weather the algorithm converges or not. Even if the training algorithm does converge bad initialization can lead to an infective training procedure where it takes much longer that it should to reach good results.

It is not well understood how initialization influences training. An even bigger problem is that sometimes the initial states of the network that help training are detrimental to generalization. The only thing know with some degree of certainty is that initial network parameters need to "break symmetry" between different units [4]. For example if one input unit connects to a few different hidden units then these hidden units need to have different initialization parameters, otherwise they will just be complimenting one another and not doing real learning. This leads to a logical conclusion, that it is best to initialize the network parameters with non-zero random values. As a rule the weights are initialized by random sampling from the Gaussian distribution. Larger values

for weights tend to have better "symmetry breaking" effects. However, if the initial values of the weights are too large they might "explode" during forward or backward propagation through the network. This will cause the neurons to always fire no mater how small the inputs are. The opposite is true for values that are too small. In this case the weights might vanish, so parts of the network might become "dead" and never fire no matter what the input are.
Another popular optimization algorithm is Adam [4]. It stands for "adaptive moments". It is an adaptive learning rate optimization algorithm. This is the optimization method of choice in this master thesis as it is used widely in solving a lot of deep learning problems and has shown to work well with textual data and RNN's.

3 Algorithm implementation

3.1 High level description of the project

First the data resources needs to be established. All of the on-line resources are considered. This includes Twitter, on-line news portals, financial news portals, specialized blogging platforms and aggregated news API's.
After selecting the resource an automated data collection script is setup to draw the information. This script works through REST. It does some initial filtering and cleaning of the data. Processed on-line data is stored in a csv file. This is done for both the textual and price data.
After collecting all the necessary information the data is preprocessed and explored. It is cleaned of any duplicates and gaps are filled or removed. Additional variables are created if necessary. The processed information is then stored on a separate file csv file.
Four types of deep learning models are created:

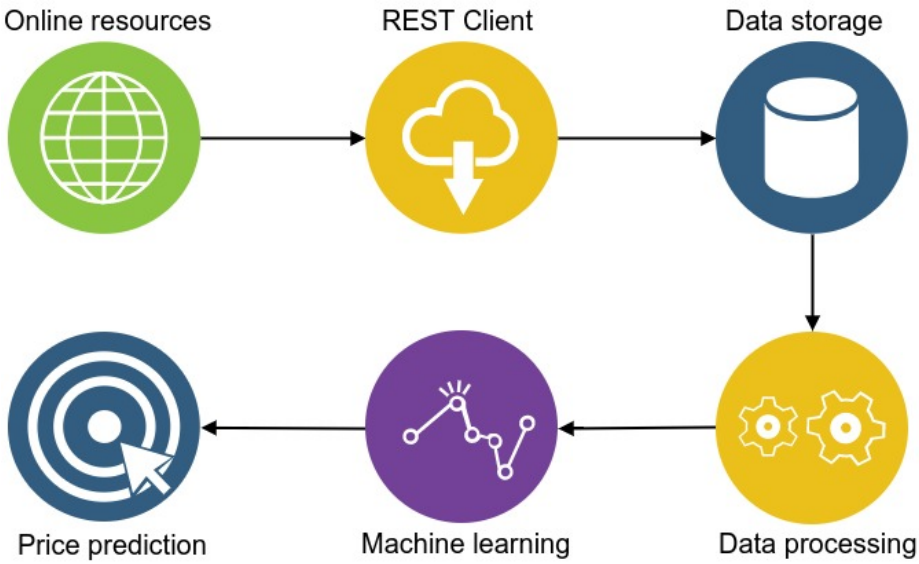

Figure 5. A flowchart of the project structure and relationship between different parts.

- The first model is a text based deep recurrent neural network with sequential LSTM cells and word embedding.
- The second model is a parallel convolutional neural network with 3 parallel layers and softmax.

- The third is a deep neural network with 2 stacked convolulitional layers and softmax.
- The fourth is a price based deep recurrent neural network with LSTM cells.

After creating the models they are trained and tested with various hyper-parameters to establish the best configuration for each model.
Finally the best configurations of each network are compared on validation and test accuracy to establish the best overall performer. If necessary additional models are also created.

3.2 Initial assessment of tools and techniques

The client for data collection was written in python. It uses libraries like "requests" and "twitter". Data processing is done on Jupyter Notebook. Numpy is used for numeric data and tables are handled with the "pandas" library. Inbuild python IO libraries are used to work with csv files. Matplotlib, pyplot and seaborn are used for visualizing data exploration exploration steps.
In the modeling part is done with the python library - Tensorflow [14]. It has low level API's for combining different deep learning elements like LSTM, convolution, emedding and softmax. It allows quick prototyping to test different architectures finding the best ones. It also provides a tool for visualizing graphs and training progress in Tensorboard. The cost and accuracy graphs as well as weights and predictions distributions were plotted using this tool.

3.3 Online resource selection

On-line textual resources can be assigned to a few different categories. Firstly, there are the big news portals like Blomberg, New York Times, Financial Times and others.
The second category is news aggregators like newsApi.org and marketwatch.com. They collect data from various sources a give an aggregated news stream.
Finally there are also micro-blogging sites like Twitter and StockTwitts. Potentially they are all good sources, so in this project 4 criteria were established to select the best one:

Reliability how reliable is the source. This means that the content is generated by experts in their field, the sources are reviewed, fact checked and edited before publishing. Official news portals score high in this category. However, credible data sources are sparse.

Abundance the sources need to be abundant enough to run a deep learning algorithm. This means being able to collect at least 10000 separate instances relating to a topic.

Ease of use the API to retrieve the data has to be easily available, preferably through a REST service. The data needs to be search-able by keyword and date or in case of a stream it should have a filter.

Data quality the data needs to be standardized, each item in the dataset needs to have the same basic atributes like text, time-stamp and id.

A primary evaluation of the data-sources shows that StockTwits and Twitter are the most suitable source for the purposes of this thesis. These results are reinforced by the 2 articles that used

Table 1. Information sources compared by their reliability, abundance, ease of use and data quality

	Reliability	Abundance	Ease of Use	Data quality	Total
NY Times	9	4	3	3	19
Bloomberg	7	5	8	7	30
NewsApi	9	6	3	8	26
MarketWatch	9	4	6	5	22
Twitter	3	10	9	8	30
StockTwits	6	6	9	9	30

StockTwits and the 3 articles that use Twitter as their main source of information in Section 1. However, Twitter was chosen over StockTwits as the main data-source, because it is able to produce much more information than StockTwits. The focus of this work is to use deep learning methods to make price predictions and a more abundant but less reliable source is favored. Twitter also has a much better API that allows streaming the incoming messages, whereas StockTwits has a service which needs to be called constantly and only returns the last 30 items posted on the topic.
The minute by minute price data is retrieved from a blockchain info website [15]. It is the only API that allows minute time steps. However the data has quality issues like missing values. Hourly price data for 6 months is retrieved from coinmarketcap [12] where the data quality is much better but the minimum time interval is 1 hour.

3.4 API access and data collection

The first thing necessary to stream data from twitter is an account and an API key. They can be easily obtained from their website. For this thesis the "twitter" python library was used. It has streaming capabilities, which allow to create a python generator to draw tweets as they are posted and save them to a file. It allows to use keywords and hash tags to filter the steam. The REST response contains many fields that are not used in this masters thesis so the script only collects the following fields:

- **id** is a unique identifying marker for each tweet.
- **text** is the textual content of the message.
- **created_at** is the date and time at which the message was created in a yyyy-mm-dd hh-MM-ss format.
- **hashtags** are the hash-tags that are used in the tweet from the ones used in the filter.
- **user** is the unique identifyer of the creator of each tweet.

Twitter does not allow retrieving historical data so only new tweets can be retrieved. The tweet data collection script is run for 22 days from 2017-11-25 to 2017-12-15.
The minute by minute price data is collected from blockchain.info. Their service returns minute by minute data for 6 hours. The script iteratively collects the price data for the same period as the tweets collection script. The REST call returns a unix timestamp, an opening price, closing price, high price and low price. Since only the closing price is relevant the script saves 2 columns:

- **timestamp** the date and time in a yyyy-mm-dd hh-MM-ss format.

- **price** the closing price for the period.

Their API only supports the Bitcoin price, therefore only Bitcoin data is collected.
The hourly price data is collected from coindesk. They allow the possibility to download csv files with historic data. The fields in the dataset are: currency_name, price, timestamp and volume.

3.5 Data Processing

During the 22 day data collection stage a total of 757.9 MB of raw twitter data relating to Bitcoin was collected with a total of 3828270 tweets. The script got stuck at some stages of the collection process leaving chronological gaps. Some of them were for just a few second long and some lasted for a few hours. This was then taken into account when processing the twitter feed. The minute by minute price data also has gaps which were filled during data processing.

3.5.1 Removing duplicates and noise

The firs cleanup step is to remove duplicated tweets from the dataset. There are two types of duplication to consider. Firstly, there are the hard-duplicates. These tweets are duplicated in "tweet_id", meaning that they are the same tweet that appears in the dataset multiple times. This might occur due to a glitch in twitter api or in the data collection script itself. All of the hard-duplicates are removed removed. After this 3781342 tweets are left in the dataset.
A much greater proportion of all the tweets are the soft-duplicates. These are the tweets that differ in tweet_id but have the same "tweet" field.
The easiest approach would be to remove all duplicates. But that destroys some information about the dataset, as duplicates are usually the tweets that got re-tweeted by other people without additional comments. However, this could be an argument for their significance being larger as more people found them important enough to spread around. This is especially true for crypto-currencies where most of the price changes comes not from hard numbers, but the associated hype and network effects.
This is why the soft-duplicates should be handled with a little more consideration, . For example out of all 3828270 tweets on bitcoin only 1289377 are non soft-duplicates (33.6%). However, most of the duplicates are just a few items. In figure 6 the most common duplicated tweets are shown.
The chart 6 shows that most of the soft-duplicated tweets come from a few sources. There are 196440 of the first most duplicated tweet it makes up 5,1% of the total dataset. The text reads: "RT @TravWeav: Bought 1,500 bitcoin in 2011 for $2.87 each. I will pick 5 random people who retweet this and give one to each of y…'. The wording reads that this is some kind of a bitcoin givaway, so basically spam. In the top 20 tweets there are about 18 that could be classified as spam. They all either have the words "Bitcoin givaway", "random people who re-tweet this and give one to each" or just "giving away" and "bitcoin" somewhere in the content. It seems that they skew the dataset and the distribution towards tweets with spam content. So all of the duplicated tweets have the before words "bitcoin" and "giveaway" or "give-away" or "give away" are removed.
After removing the non-relevant soft duplicates there are 1378461 tweets left in the dataset. This cuts out about 64% of the tweets from the dataset.

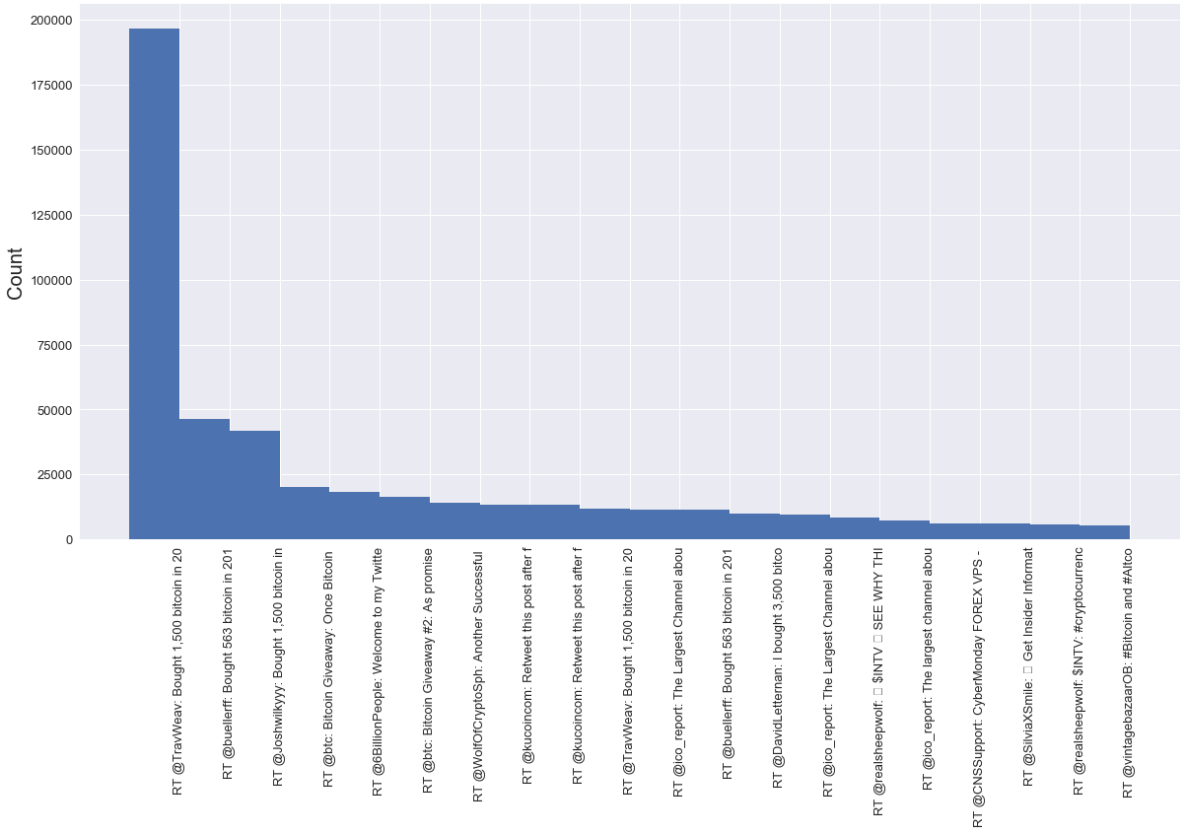

Figure 6. Histogram of the top 20 most duplicated tweets by sorted by number of occurrences

3.5.2 Exploring the dataset

The next step in cleaning the data is cleaning the text. The tweets contain a lot of noise. There are a lot of duplicates not covered by section 3.5.1. Some tweets are the same apart form a link or a cyphered text and could also be removed as duplicates.

Firstly, the punctuations are removed from the tweets. The list of punctuations are used from the python string library. In language translation or text generations punctuations would be left, but for price predictions they are just noise. All letters are converted to lowercase sot that words like Bitcoin and bitcoin would mean the same thing.

Secondly, the words that contain the letters 'http' in a sequence and the words that are more than 25 letters long are removed. This lowers the amount of unique words in the dataset as most of the links and cyphered words are there only once, this will be useful when it comes to embedding.

Thirdly, after removing punctuations there are still some undesirable characters or character combinations left. All of the new line characters like characters and various combinations that represent pictograms are removed.

After removing the punctuations and other noise there are 454477 unique words in the dataset. On one hand this is a lot considering that the whole English language has about 1000000 words. However, there are a lot of words that are misspelled. Also many words are not part of the Oxford English dictionary and are used only on the Internet.

Figure 7 shows the distribution of words sorted by the number of times they are used in the cleaned tweets dataset. Bitcoin is clearly the most dominant word in use, mentioned 880421 times which can be attributed to it being used as a hash-tag. After that there generic English words like the, to, it and similar. Out of the top 50 words the ethereum, cryptocurency, blockchain and futures

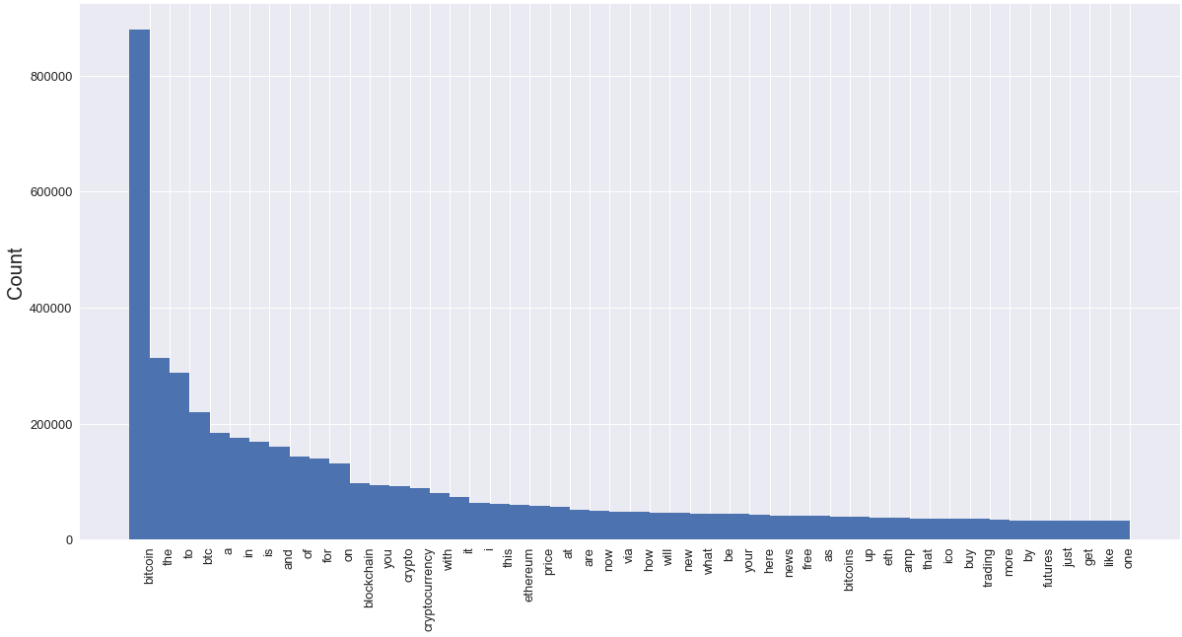

Figure 7. Top 50 words sorted by the number of times they are used in the dataset

prevalence is specific to this domain. There are 310112 words in the dataset, that are only used

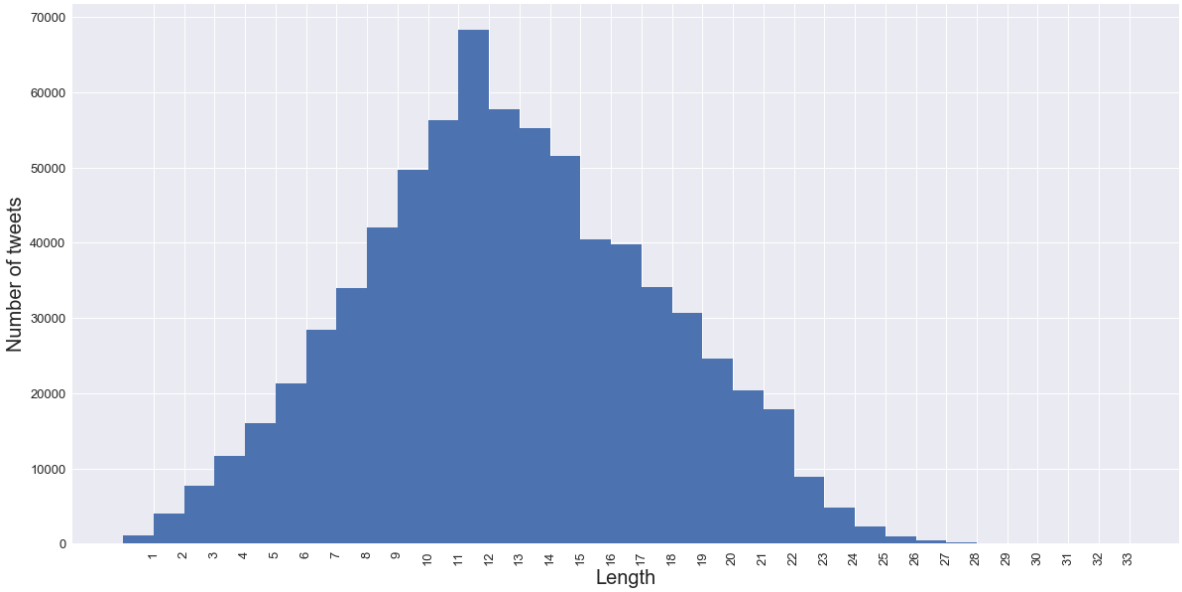

Figure 8. Distribution of tweets by their length in the bitcoin dataset

once. This makes up 68.2% of all the unique words. Most of them are either misspelling, numbers or mangled words. It would be a good idea to remove them , but among them are words that might be significant and a deep neural network should be able to distinguish between the ones that are relevant unique words and ones that are not.

The tweets differ in length. In figure 8 looking at the tweets the distribution is quasi normal with no deep outliers. The tweets length varies from 1 to 33. The shortest tweets up to 3 words are removed as they do not useful information. The mean value is 13.1 and the mode is 12. The shortest tweets of length 1 and 2 are removed as they are not long enough to carry any meaningful content.

3.5.3 Cleaning price data

The minute by minute price data has a lot of discrepancies. The main problem is missing values. There are up to 1 hour periods where the data is missing. Since blockchain.info is the only site where such data is available it is not possible to augment it from other sources. To fix the gaps simple averaging is used.

In figure 2 an excerpt of the data has a missing value in time step 2017-12-02 02:02:00. The

Table 2. Missing time step values in the price dataset

Time	Price
2017-12-02 02:00:00	10840.86
2017-12-02 02:01:00	10850.9
2017-12-02 02:03:00	10837.39

missing value is calculated by averaging the price for the previous end next time step. The fixed value is shown in figure 3.

In case the values are missing for more than 1 consecutive time step the period the middle value

Table 3. Fixed missing time step value in the price dataset

Time	Price
2017-12-02 02:00:00	10840.86
2017-12-02 02:01:00	10850.9
2017-12-02 02:02:00	10844.15
2017-12-02 02:03:00	10837.39

in the period is filled with the average from the available values in the previous and next period. This process is repeated iteratively until all the values are filled.

Another problem in the price data are duplications. For some reason the API returns multiple values for some steps. If the price value for the duplicates are the same then one of them is removed, else the last duplicated value is left.

In table 4 an excerpt of the price data is shown. There are missing and duplicated values. This

Table 4. Duplicated price data example

Time	Price
2017-11-09 18:41:00	7173.26
2017-11-09 18:42:00	7176.68
2017-11-09 18:42:00	7176.68
2017-11-09 18:43:00	7160
2017-11-09 18:44:00	7168.51

is fixed by first removing the duplicated values and then iterating through the data and filling the

missing values with averages of the previous and further time period. This is not ideal, but for a recurrent neural network to work it needs to be continuous. The fixed data is shown in table 5.

Table 5. Fixed duplicated price data

Time	Price
2017-11-09 18:41:00	7173.26
2017-11-09 18:42:00	7176.68
2017-11-09 18:43:00	7160
2017-11-09 18:44:00	7168.51

3.5.4 Augmenting price data

Once the text is preprocessed the the textual data is assigned to price data. This is done by composing a map. One price point maps to multiple text sources for the period previous to the price evaluation point. The goal is to have multiple text sources mapping to one price value.
Also latency is considered. The messages do not influence the price at the moment that they are posted - some time needs to pass. A hyper-parameter for preprocessing data is used called latency. The default value for it is 1 minute, but various latency intervals are tried in the course of training. The simplest type of problem is classification. The network is trying to decide weather the price of the crypto-currency will rise and fall in the next period given the textual input. Additional variables need to be created. If the price rose compared to the previous time step the value of the column is 1 if the price fell the value is 0.

$$C(p_i, p_{i-1}) = \begin{cases} 1 & \text{for } p_i - p_{i-1} \geqslant 0 \\ 0 & \text{for } p_i - p_{i-1} < 0 \end{cases} \tag{3.1}$$

In 3.1 the method of assigning the change value is shown. Here C is the function that represents the change from the previous time step, p_i is the price at the current time step and p_{i-1} is the price at a previous time step. If the prices are equal 1 is assigned as only decreasing price values are assigned 0.
The other type of problem is regression. The network is trying to predict the both the direction and the magnitude of the change.

$$C(p_i, p_{i-1}) = (1 - \frac{p_i}{p_{i-1}}) \times 100 \tag{3.2}$$

The price data is augmented with the column the value for each row is calculated in 3.2.

3.6 Augmenting text data

3.6.1 Converting text to integers

In order to feed the textual data into the recurrent neural network it cannot be in textual form. For this it needs to be converted into numbers. There are multiple ways on how to do it. It is

possible to one-hot-encode it, but in this case the vector length for each word would be the same as the dictionary length so it is not very efficient. Another way is to use embedding. This means converting each word into into an integer and creating a dense representation of a tweet.
The first step is to create a list of all the word in the textual corpus. So the tweets are split at spaces in the text. Some tweets have tabs and multiple spaces, for this the created array needs to be cleaned of excessive spaces.
The second step is creating a dictionary from the word list. This is done by iterating through the word set and assigning the iteration step value starting from 1. This way each word will have a unique integer representation that is can be processed by the network.
The third step is converting the textual tweets into an integer array. All the need to be of the same size. This is done by selecting a sequence length. In this case it is the length of the longest tweet word count. A zero value array of sequence length is initialized and the tweet integer array is projected on the end of the 0 array. In figure 9 the full process from raw tweet text to a fixed size

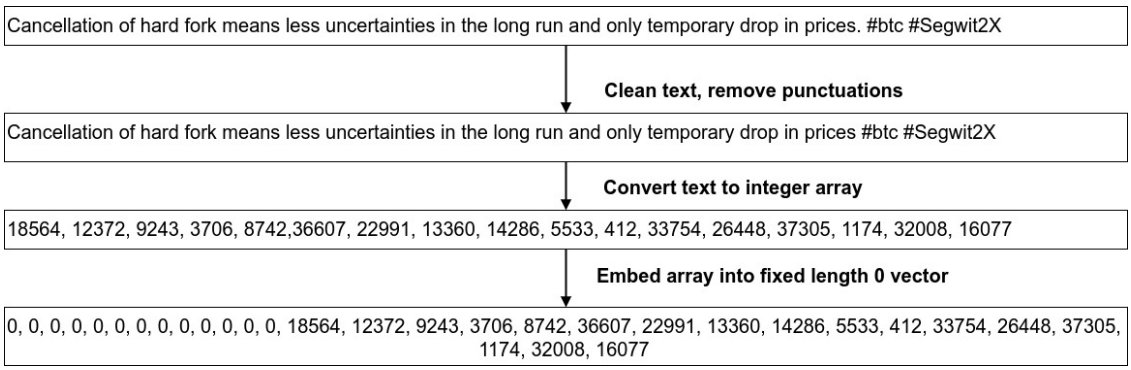

Figure 9. An sample full sentence converted to an integer array with padding.

embedding is shown. This is an actual tweet from the dataset embedded using a dictionary created from the dataset. The beginning of the input is padded with 0, which are a neutral number, which the neural network ignores as it is found in most tweets. The network iterates over the zeros and gets to the actual tweet words represented by integers.

3.6.2 Mapping textual data to price data

In order for the network to learn tweets need to be assigned a label which the recurrent neural network will use in training. The first thing to consider is the realistic time scale at which a tweet relates to a price. It cannot be instantaneous and realistically a tweet has the strongest relation to a price in the future.
The twitter dataset has time values in the in second precision. On the other hand the price data comes in every minute. To make things easier the twitter data is rounded up to the next minute so that an example time of 2017-11-09 18:41:12 is rounded to 2017-11-09 18:41:00.
A price map is created to map the price change to the twitter text. The mapping algorithm takes a latency parameter in minutes. This allows to create datasets with different latency.

3.6.3 Text embedding

The tweets that are converted to integers need to be embedded to save space on the model. Tensorflow has an embedding API. It takes the length of the dictionary and the desired length of embedding and converts it into a sequence. This API finds words that are used in similar context

and places them close to each other in the multidimensional plane. So instead of using one hot encoding and having very sparse and memory consuming one-hot vectors a dense representation is used where the input words is encoded in a n dimensional space where $n < wordcount$. A toy example in figure 10 shows how this is done. Words with similar meaning will be embedded into vectors that have a similar length and direction.

In the graph bi-grams are encoded. Combinations like "two days" and "few days" will be close to

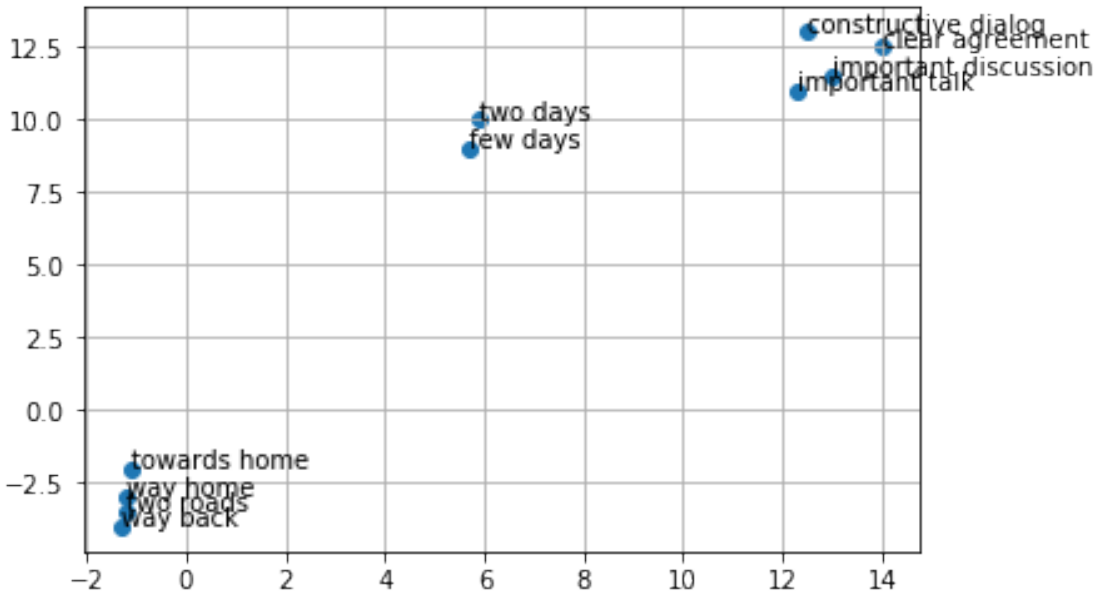

Figure 10. A plot of a bi-gram embedding in a 2-dimensional plain

each other and far from combinations like "toward home" and "way home". Having encoded the words into their coordinates in this multidimensional plain allows for this information to be passed into a ANN.

3.7 Machine learning model setup

Machine learning can be formulated as a classification or regression problem. The first is easier as it only requires to guess which way the price is moving. The success of this type of model can be measured by both cost and accuracy. Cost is calculated by a mean squared error 2.2. Accuracy is just the measure of how many classifications the model got right in comparison to the whole dataset. The regression tries to predict an exact amount of change. The cost function is calculated the same way, however the accuracy cannot be calculated as it will almost always be 0.

Since there are multiple tweets in a single time step the data mapping for can be either one-to-one or many-to-one. For the most basic model a one-to-one mapping is tried. For the rest of the models, the many-to-one mapping is used. Here the time the tweets are aggregated at each time step into a single text that is mapped to a price point. This is a valid approach for both the classification and regression type models. It does take into consideration the order of the tweets. However more information is bound to get lost and the number of tweets would not play an important role in the model. Another major drawback of this approach is that it would shrink the dataset considerably as multiple tweets would fall under the same price point and the number of total data mappings would fall by the average of the number of tweets at each time step. However, this model has a strong dependency on the order of the tweets, as they all fall chronologically at each time step.

Each model has multiple hyper-parameters. The one that is unique to this project is latency. As information posted on a social media network usually does not have instantaneous results the tweet mapping to a price point can be delayed. This can be done in multiple ways. The first one is a simple latency parameters in minutes. Another approach is to aggregate the price change of a period, because the minute by minute changes might be too stochastic and will not have a direct relation to the tweets. Furthermore, some news might have an impact the next minute where as other tweets have a more delayed effect.
Two types of deep learning models are used. Recurrent neural networks have the following hyper parameters:

- **LSTM size** how many LSTM cells are coupled together for the model.
- **Embedding size** indicates how big the embedding of the sequence should be.
- **Number of layers** - how many hidden layers there are in the network.
- **Learning rate** how big are the steps that the training algorithm takes at each training iteration.
- **Batch size** indicates how big the batches are for training
- **Keep probability** indicates the rate of dropout being used.
- **Number of epochs** indicates how many times the training algorithm iterate through the data.
- **Latency variable** decides how many time-steps the tweets and the price relationship is lagging.

Convolutional neural networks have these additional parameters:

- **Padding** can either be "same" or "valid".
- **Strides** how big a stride does a convolutional filter make.
- **Filter size** how big the filter is.

These are all tuning knobs for the model and a wide range of them needs to be tried out to find the best performing predictor. Every model is tried out with multiple hyper-parameter sets and the best performing set is selected.

3.8 Experiment setup

Three different datasets are used in this project. One is a toy dataset of tweet sentiment that is used to check if the network architecture is valid. If the model train and validation error is going decreasing the model is trained on the second dataset.
The second dataset is the tweets collected over a 22 days period. They are mapped to a data-time value which is used to map them to the 3rd dataset.
The third dataset is a price list for the 22 days in mentioned before. This is a price to date-time mapping which is used to create labels for the 2nd dataset.
The fourth dataset is a 6 months hourly price change collection. It is used to run the price prediction on just the price movement. The data is split into training, test and validation sets with a 8:1:1 split. The validation dataset is used during training as a reference for the training algorithm. As training progresses and the training loss decreases it is expected that the validation error would decrease as well. When the model is fully trained and the best hyper parameter set is found it is tested on the test dataset.
In the experiment GPU enabled Tensorflow is used. The graphics card is a 4 GB Nvidia GeForce 960M with 680 CUDA cores.

3.9 First model: deep recurrent network with text embedding

3.9.1 Testing on a toy dataset

The first model is a multilayer recurrent neural network. The textual inputs are first embedded so that each word integer would be represented not by a one-hot encoding, but by a dense n-dimensional representation where n is less than the total word count in the dataset. The overall architecture is show in figure 11. After the input data has been embedded it is passed to an "rnn" layer, which is an abstraction for a multilayer multi-lstm cell with dropout applied to the output of each layer.

Finally, the "rnn" output goes into a "fully connected" layer. The outputs of this layer are used

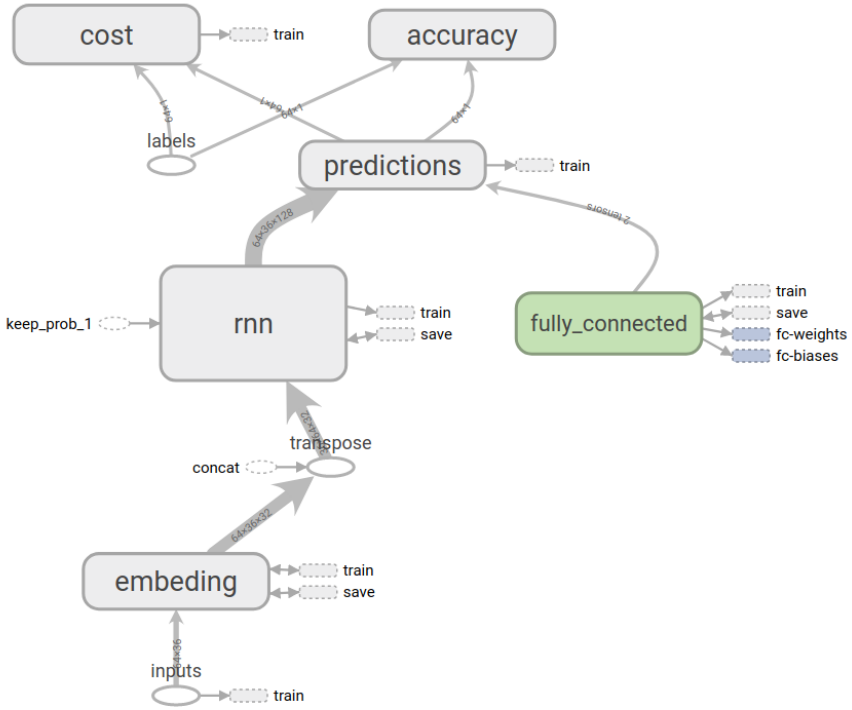

Figure 11. First deep learning model - a RNN with an embedding layer and LSTM cells.

to calculate the mean squared error of the batch. Also sigmoid activation is applied to the same output and it is used to calculate the "accuracy".

The model uses Adam optimizer to train the network. This training algorithm adapts the weights depending on how the big the training step needs to be [4].

This network is tested by giving it a know problem to classify the sentiment of tweets for a labeled airline review dataset. The hyper parameters for the this are:

number of epochs is 20
batch size is set to 64
embeding size is set to 32
sequence length is 36
number of lstm cells per layer is 128
number lstm layers is 1
learning rate is 0.001
keep probability for dropout is 0.7

The input vector at each iteration has a shape of (batch size, sequence length) and the output shape

is (batch size, 1). The initial setup is basic, but suffficient to check if the network is reducing the training and validation cost.

The training stage runs for 106 seconds and the last validation accuracy is 0.77. Figure 12 shows

Figure 12. Training and validation results of the first deep learning model on a toy dataset.

the accuracy and cost of the train and validation data sets. The model is training and generalizing because the training and the validation accuracy is going up and the cost is going down. However, the model begins to over-fit at iteration 1500. This would be a good place stop training on a real dataset.

On the test run the prediction accuracy is at 0.81 so over 81% of the test data was classified correctly. Looking at the predictions distribution during the training run in 13 the algorithm starts of at random guessing, as the distribution is spread evenly between 1 and 0.The last iteration most of the predictions cluster around 1 and 0, meaning that the algorithm is more and more sure of what the prediction needs to be.

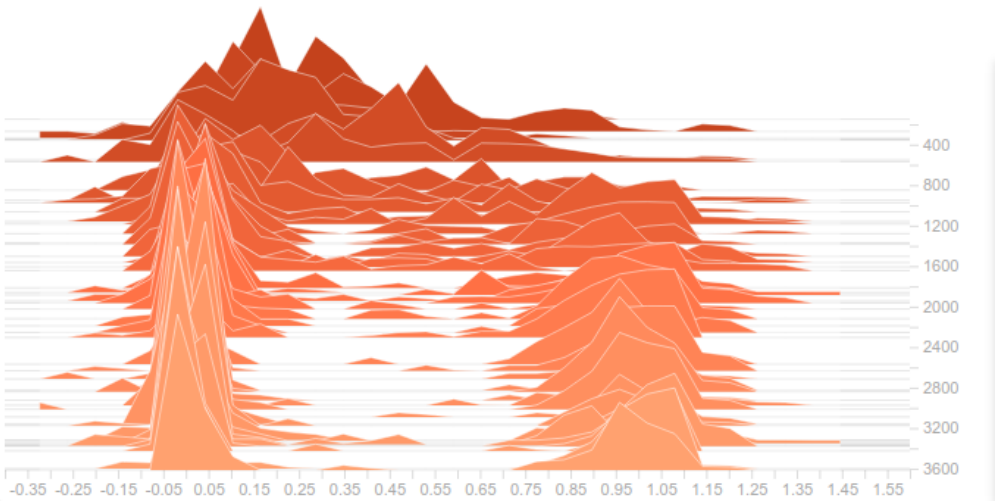

Figure 13. Prediction distribution of the first model on the toy dataset.

3.9.2 Bitcoin tweets dataset

The second step is to try out the working model on the real dataset of tweets collected over 3 weeks and the price changes that happened after those tweets were posted. All of the hyper-parameters are the same as in the previous run, only the batch size is set to 1000, because the training dataset consists of 1378461 tweets and price mappings. The sequence length is set to 33 which is the number of words in the longest tweet.
Also an additional hyper-parameter for latency is used and it is set 1 minute, meaning that the prediction is made for the price movement 1 minute after that tweet was posted. Otherwise the architecture is the same with 1 LSTM layer and 128 LSTM cells per layer.
The training for the second experiment runs for 2804 seconds and yields worse results than the

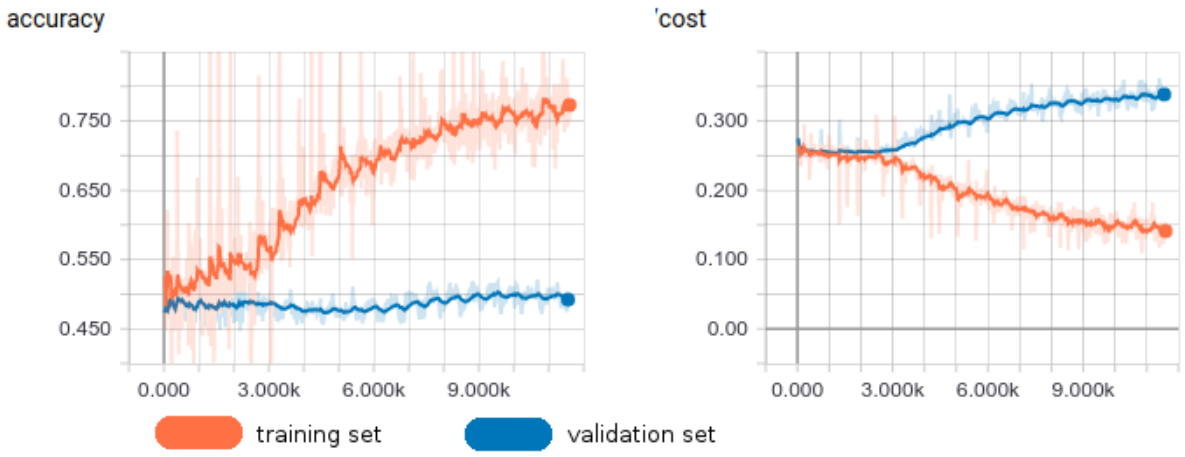

Figure 14. Training and validation results of the first deep learning model on a bitcoin dataset.

toy dataset The cost and accuracy in figure 14 show that both runs improved on the training data, but only the first run improved on the validation data. It is model for Bitcoin tweets is over-fitting. The training cost and accuracy are improving but the validation accuracy stays the same and the cost is increasing. This indicates that the model is not generalizing and is memorizing the training dataset.
Looking at the predictions distribution in figure 15 the model does not seem to be as certain of which one of the 2 values it should choose. It starts of by mostly yielding predictions around 0.5 and slowly spreading it out with peaks forming at 0.5, 0.0 and 1. However the predictions are not as clear cut as with the toy dataset.
To minimize over-fitting and get better generalization it is possible to impede the training algorithm optimization by tuning the dropout parameter so that the dropout layer would disable some of the connections going into the next layer. In the next run the dropout value is set to 0.5, also to improve the model capacity the number of layers is increased to 3. This allows for the model to find more complex hierarchical structures in the data which a 1 layer network might not do.
For the next training run the other hyper parameters stay the same, except for the number of epochs which is lowered to 10. If there is no improvement with the changes applied to the network it might indicate that there is something wrong with the training data structure.
The third run of the training algorithm takes 3673 seconds and yields a test result of 0.502, which is better than run 2, but the improvement is clearly just by chance as parameters like validation accuracy and cost show a decline in the results.
The results for run 3 are shown in figure 16. The cost and accuracy are smoothed for the trend to be

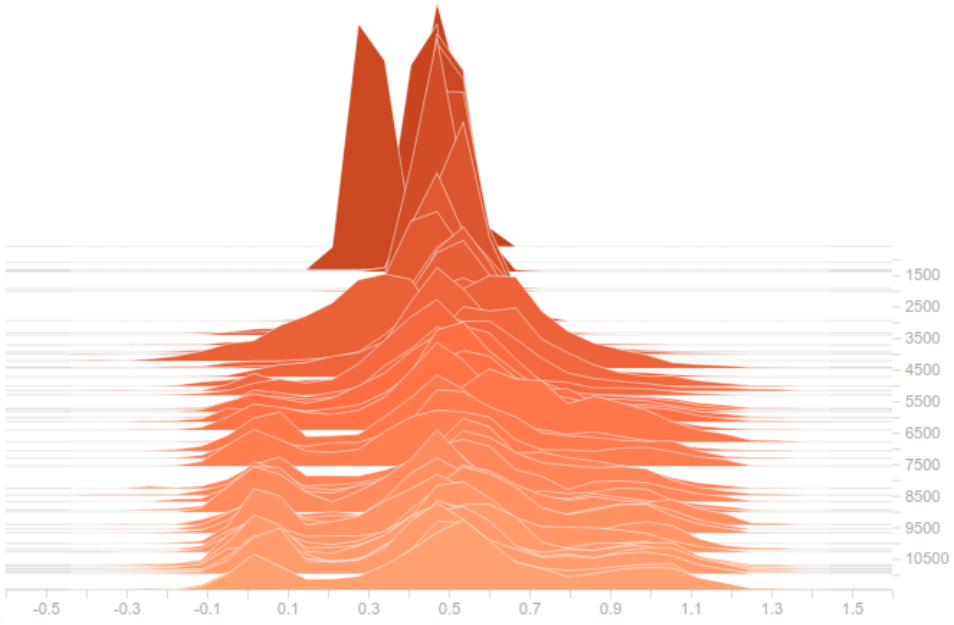

Figure 15. Prediction distribution of the first model on the bitcoin dataset.

more visible. The fact that the training run cost and accuracy are jumping around so much might indicate that the learning rate is too high and the changes in the weights by the current learning rate overshoot the the target on the cost function. However, it is clear that training cost is decreasing and accuracy is increasing, but the opposite is true for the validation set.

The predictions distribution as shown in figure 17 indicates that the training is even worse at

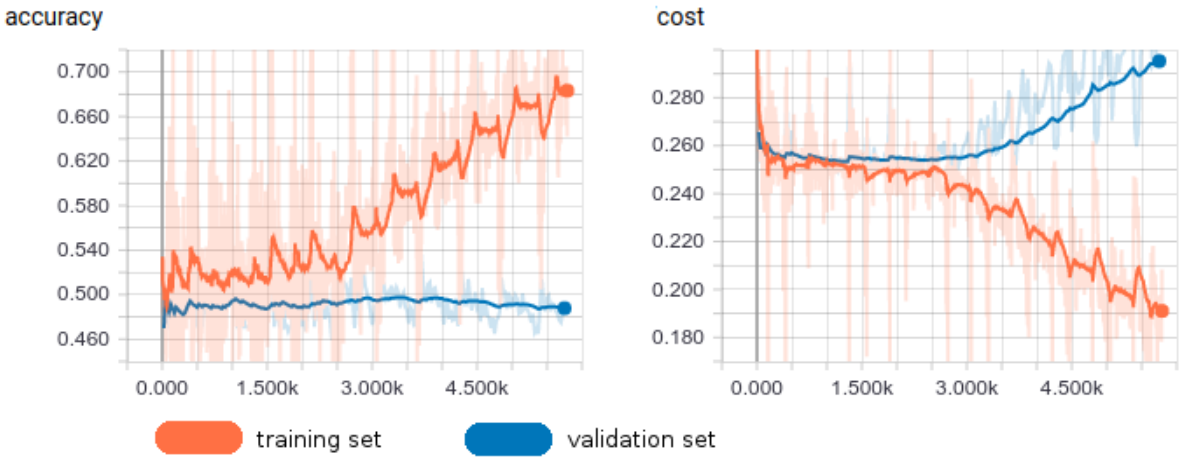

Figure 16. Training and validation results of the first deep learning model with 3 hidden layers on a bitcoin dataset.

predicting the results as most of the predictions cluster around 0.5. This means that the training process cannot find a way towards a lower point on the cost function so it over-fits the data by minimizing the cost by assigning predictions with values between the 2 possible targets 0 and 1.

Figure 18 shows that the training process in the two cases finds a different weight matrix in the fully connected layer. This is in part due to the fact that the 3rd run was done with 3 hidden layers. The weights in the 3rd run cluster around -0.10 and 0.10 which, this indicates that the learning algorithm failed to capture a complex approximation function and in part explains the different

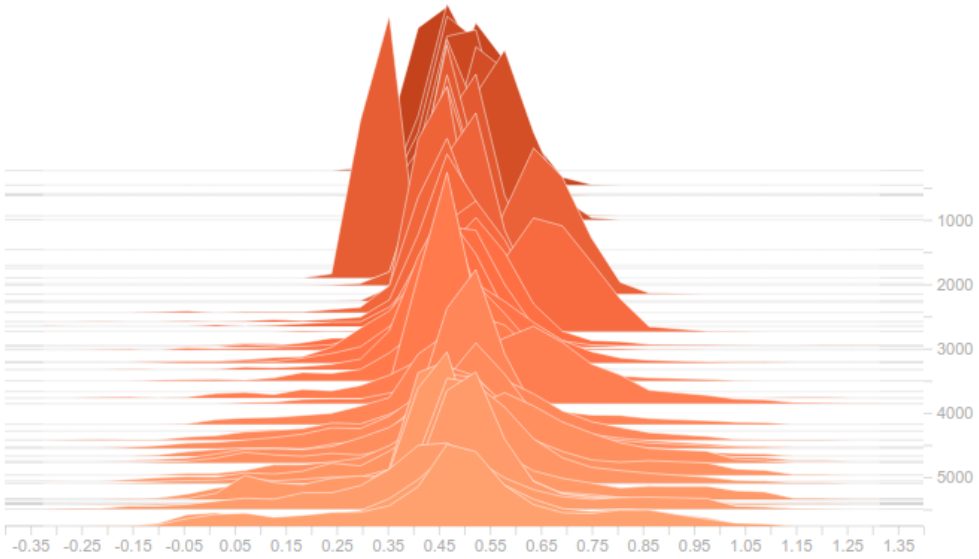

Figure 17. Prediction distribution of the first model on the bitcoin dataset.

prediction distributions on the 2 runs.

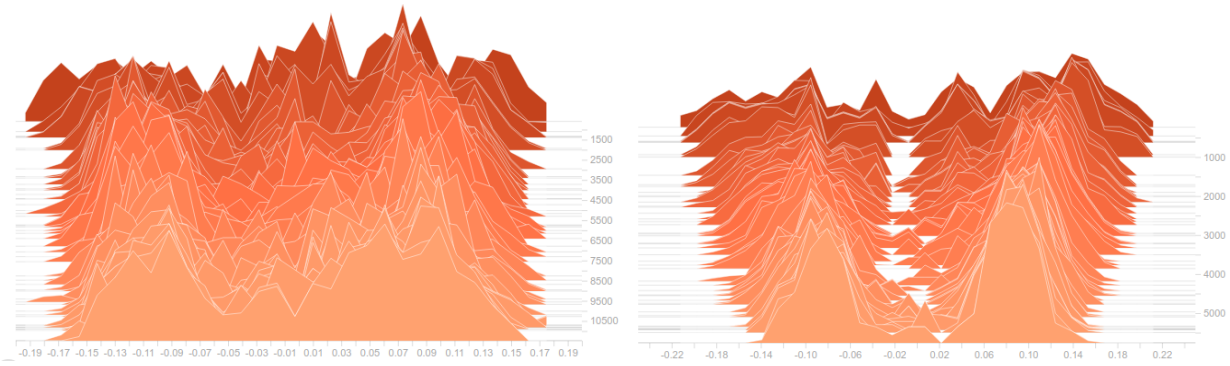

Figure 18. Weights distribution comparison of the first model on the bitcoin dataset with 1 and 3 hidden layers.

3.9.3 Merged tweets dataset

The tweet by tweet mapping to a price change shows improvements for the training cost but if fails to generalize and the validation error and accuracy stay the same or decline during training. One possible reason for that is that when mapping each tweet to a price change value it might introduce an imbalance to the dataset. There are time steps where there are as many as 70 tweets in one minute and on other time steps there are as low as 18 in one minute. Overall the dataset is quite well balanced with 52:48 proportion of positive and negative price changes. However, with tweet by tweet mapping the balance shifts to 57:43.

This problem can be partially solved by merging the tweets at each time-step into a single text. This will decrease the number of data points in the dataset, but it will remove the imbalance and even out the significance of each time step in the training.

The biggest problem that occurs with merged tweets is that the word count in a single data-point increases. Figure 19 shows the diffference in word word count distributions for merged and non

merged datasets. The longest text in a non-merged dataset in 33 words. The longest text in a merged dataset is 2053. However, only a 237 texts are longer than 400. To keep the model smaller the texts that are shortened to dropping the beginning of the text and leaving the last 400 words.
The model is run with these hyperparameters:

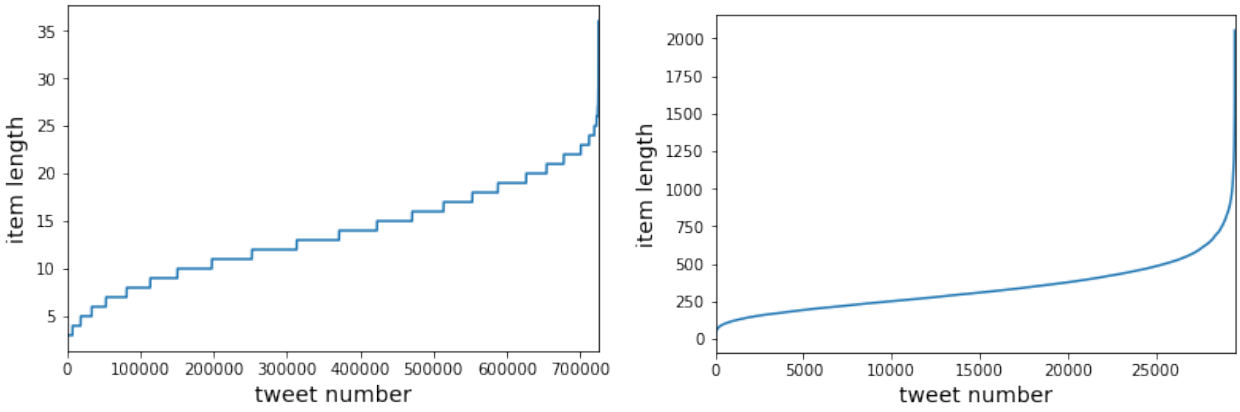

Figure 19. Tweet length distribution comparison between merged and non-merged datasets

number of epochs is 10
batch size is decreased to 250
embedding size is the same at 32
sequence length is raised to to fit the input vectors 400
number of lstm cells increased to 512
number of rnn layers is set to 3
learning rate is 0.001
keep probability for dropout is 0.7

Merging the tweets does not yield lower costs or higher accuracy on the validation set. The algo-

Figure 20. Training and validation results of the first deep learning model with 3 hidden layers on a merged bitcoin dataset.

rithm is predicting with with around 0.5 accuracy which is the same as random guessing. The cost function for training and validation sets diverges at iterration 250 so the training algorithm starts to over-fit the data again. Changing the training data structure and the hyper-parameters does not improve the generalization of the model.

The prediction distribution in figure 21 moves from clustering around 0.5 at the beginning of training, to evening out and forming small peaks around 0.0 and 1.0 at the end of training. However, this does not result in improved accuracy and cost on the validation set.

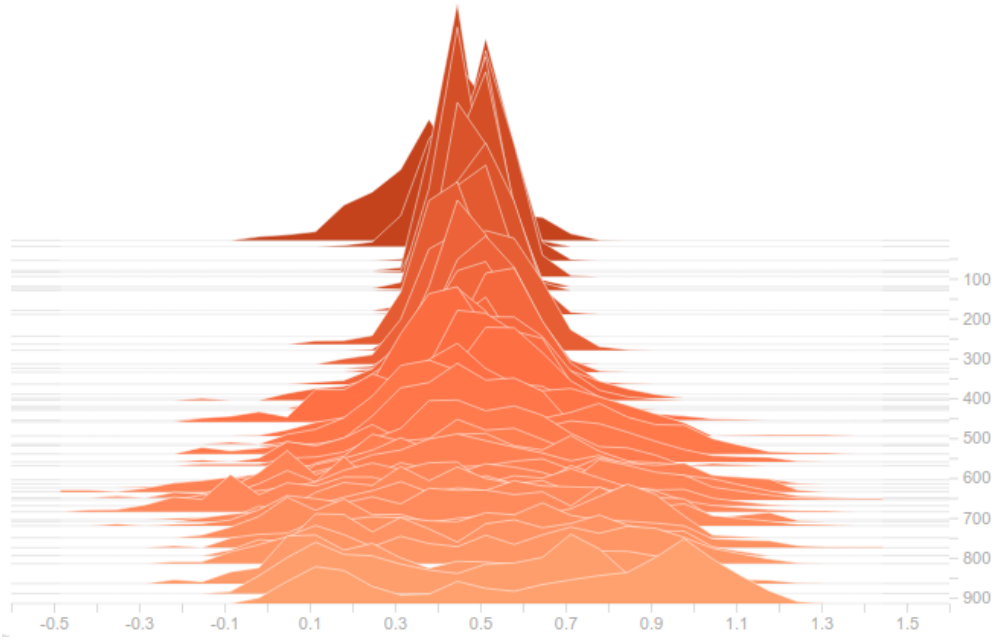

Figure 21. Prediction distribution of the first model on the merged bitcoin dataset.

3.10 Second model: parallel convolutional neural network with embedding

The second model is a convolutional neural network with 3 parallel convolutional layers. Similar to the first model the input is embedded. Then the dimensions are expanded from (batch size, sequence length, embedding size) to (batch size, sequence length, embedding size, 1) in order to fit the convolutional model. The expanded input is fed into each of the paralel models seperately.
The convolutional layers haven 8, 16 and 32 filters and kernel sizes of $(5, 5)$, $(4, 4)$, $(3, 3)$ respectively. The output at each layer is fed into separate max pooling layers with a pool size of (2,2) and a strides of 2 each. The outputs of the max pooling layer are concatenated along the zero axis, creating a tensor of (batch size, sequence length, embedding size, total number of filters). Dropout is then applied and the tensor is reshaped to (batch size, sequence length * embedding size * total number of filters). The reshaped tensor is fed into the fully connected layer which applied a sigmoid and outputs a tensor of (batch size, 1) as a prediction vector for each of the items in the batch. The architecture of the model is shown in figure 22. The cost is used as input for Adam Optimizer training operation.

3.10.1 Testing on a toy dataset

First the model is tested with the toy dataset to check if it can learn and produce similar generalization to the rnn model. The hyper-parameters for the toy dataset:

number of epochs is 20
batch size is 64
embedding size is 32
sequence length is 36

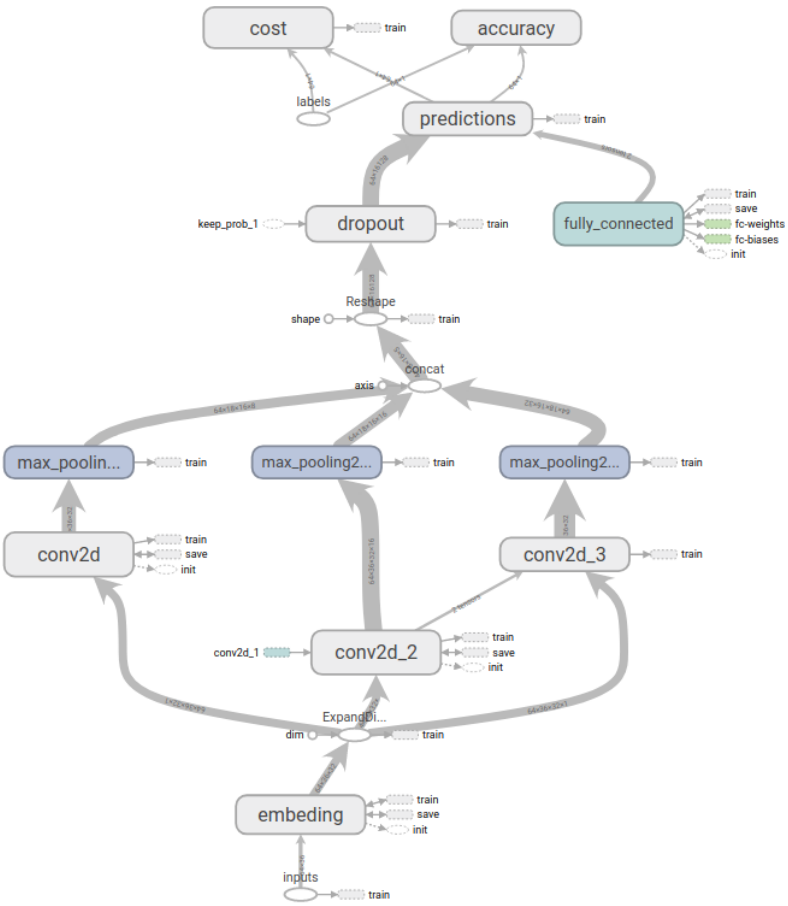

Figure 22. Convolutional neural network with 3 paralel convolutional layers

learning rate is 0.0005

keep probability for dropout is 0.6

The model trains for 93 seconds on the toy dataset and gives a test accuracy of 0.776. This is slightly worse than the first model but the difference is not significant and could be changed with some hyperparameter tunning. The cost and prediction accuracy during training are similar to the

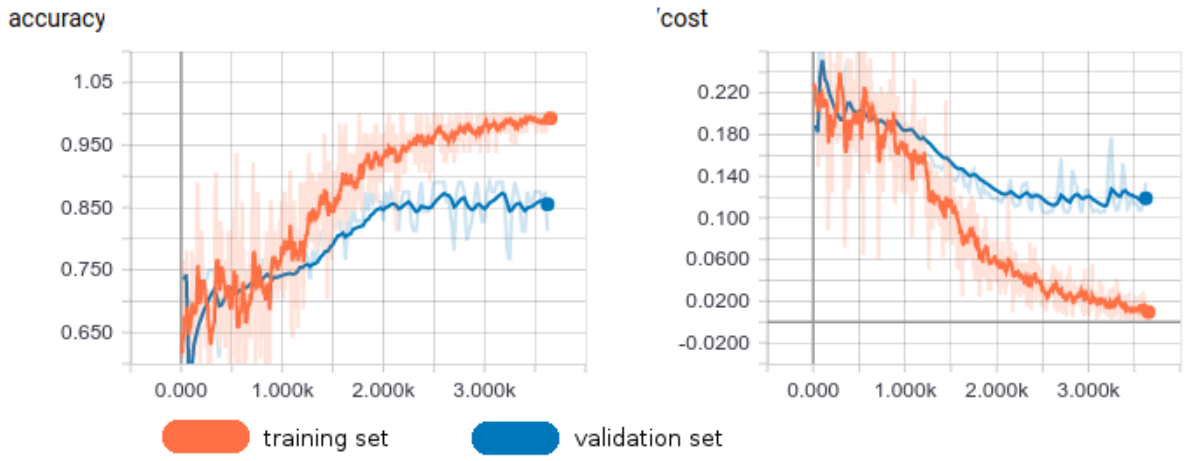

Figure 23. Training and validation results of the second deep learning model on a toy dataset.

first model. However, in figure 23 the validation cost evens out at iteration 2000 and does not start

to grow indicating that the model generalizes better and does not start to over-fit the data as much. This is a desirable result as the first model suffered from over-fitting with the bitcoin tweets dataset. In figure 24 the weights distribution change is shown for the training run. The weights are initialized at around 0 and as the training progresses the distribution flattens around 0, maintaining a quasi-normal distribution curve. This is the desired result for training with the real dataset.

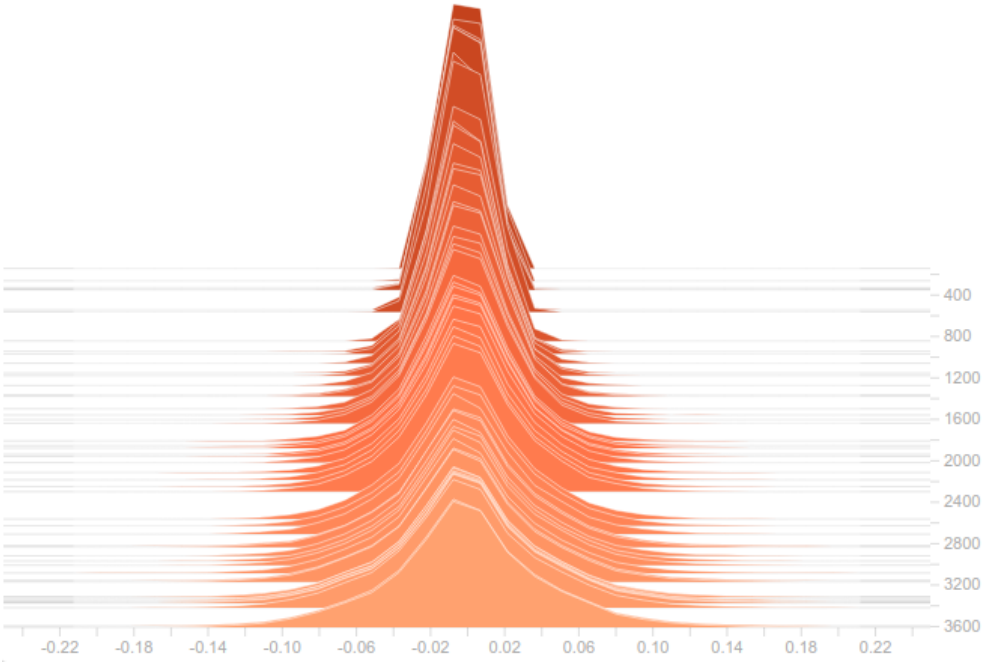

Figure 24. Weights distribution of parallel convolutional network on toy dataset

3.10.2 Merged tweets dataset

The model is then run on the merged tweets dataset with parameters:

number of epochs is 20
batch size is 100
embedding size is 32
sequence length is 400
learning rate is 0.001
keep probability for dropout is 0.7

It takes 1655 seconds for the model to train and it gives a test accuracy of 0.498. Again the results are the same as random guessing. The cost for the training dataset decreases until it reaches about 0.1 and then it hovers around that value, however the validation cost after dropping to about 0.3 start increasing from the 600th iteration (see fig. 25). The validation accuracy follows the training accuracy, but then flattens and stops generalizing. The training accuracy increases to about 0.8 where it evens off and the validation accuracy plateaus at 0.45, which is even worse than random. This might be caused to the class imbalance.

The predictions distribution for this run are shown in figure 26. During the course of the training run the predictions spread out across a range between -0.5 and 2.3. This result shows that the network does not optimize even on the training run as the values would be expected to cluster around 0.0 and 1.0 for the 2 possible classes in the dataset. Running the training for more epochs could

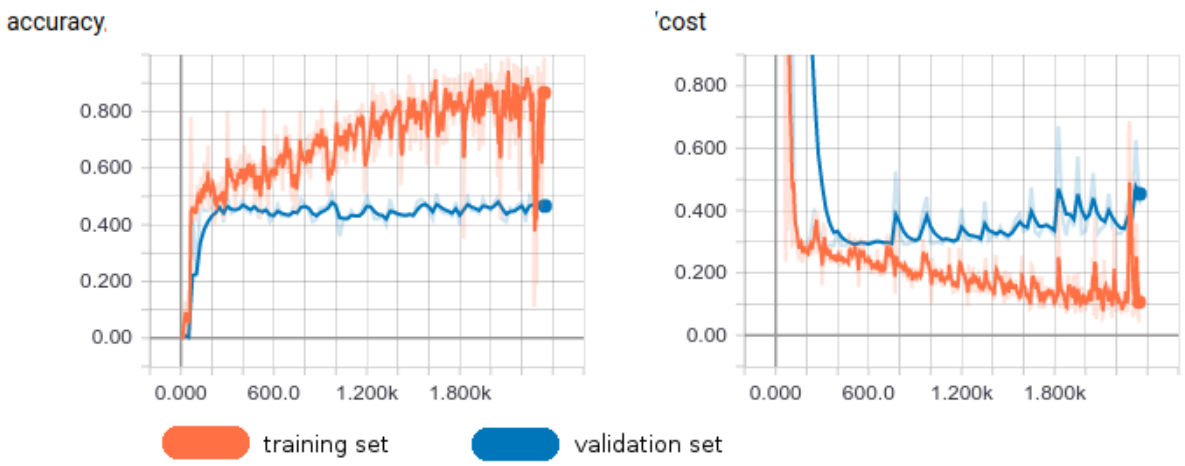

Figure 25. Training and validation results of the second deep learning model on a merged bitcoin dataset.

have improved the training accuracy and cost, but the validation cost stopped decreasing so any gains with the training set would have come at a cost of over-fitting.

The weights distribution (see fig. 27) for this training run is almost a bell curve which would be

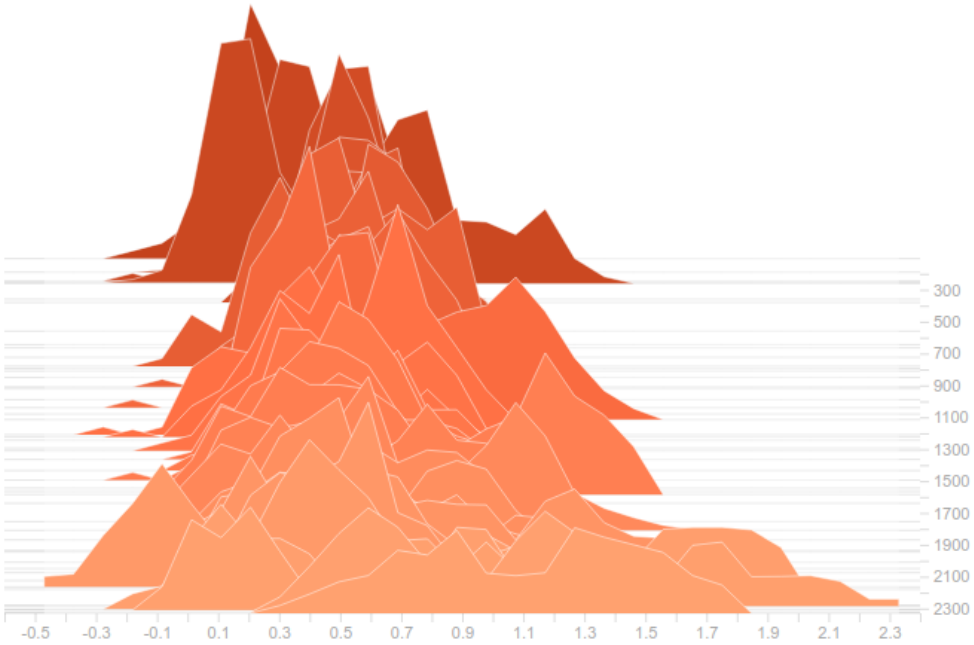

Figure 26. Predictions distribution of parallel convolutional network on toy dataset

expected, enabling some of the connections at the expense of others, creating paths in the network for the relevant data to flow. However, this does not result in good performance on either the training nor the validation sets.

All in all, the second type of the deep learning model does not show any improvement in generalizing and correctly predicting the validation dataset over the first recurrent network model. Usually deep learning models are good at finding hierarchical abstractions in textual data. This was shown with the toy dataset where both models with no parameter tuning and low network depth could make predictions with about 0.8 accuracy.

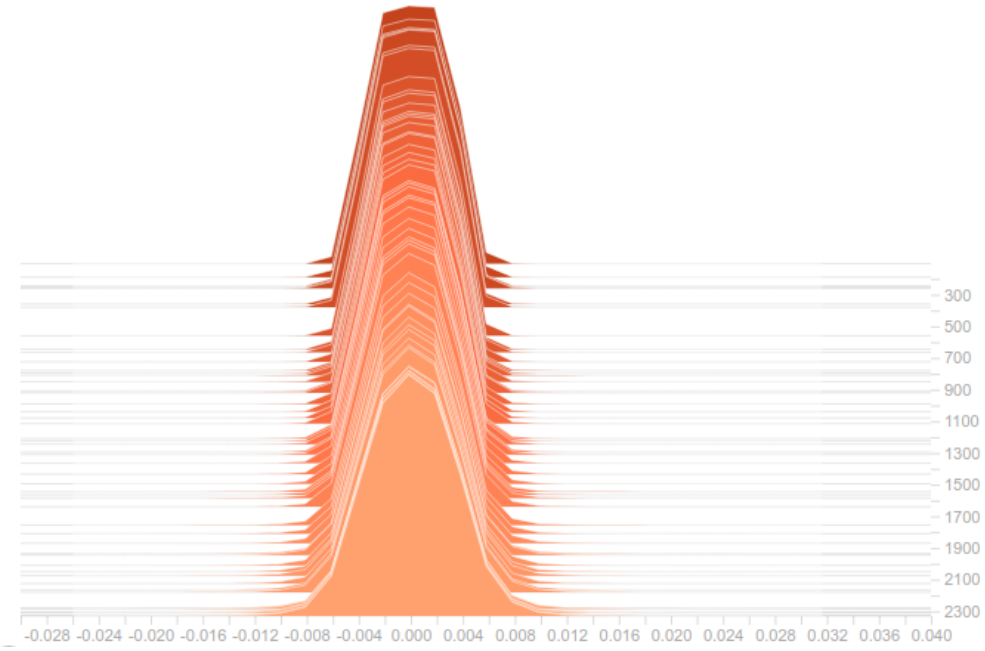

Figure 27. Weights distribution of parallel convolutional network on toy dataset

3.11 Third model: deep convolutional neural network with embedding

The third model is also a convolutional neural network. However instead of having multiple parallel convolutional layers this model stacks them sequentialy and the output of the first layer becomes the input of the next layer.
The high-level architecture of the model is shown in figure 28. The first layer is the input. This is a 2 dimensional vector of batch size and sequence length. The input is fed into the embeding layer. Each word in the sequence is then embedded into an integer array of a selected length. This makes 3 dimentional tensor of batch size, sequence length and embedding size.
The dimensions of the outputs are then expanded to 4 dimensions to fit the first convolutional layer. It has 32 filters and a kernel size of (5,5) with "same" padding and a relu activation function.The outputs of the convolutional layer are fed into the max pooling layer, which has a pool size of (2,2) and a strides of 2.
The tensor is further fed into the second convolutional layer with 64 filters, kernel size of (5,5), "same" padding and relu activation. The outputs are again fed into a max pooling layer with the same configuration as the previous one.

The outputs of the last layer are then reshaped to 2 dimensions (batch size, 1) and dropout is applied. The results are fed into a fully connected layer with no activation function. The results have 1 prediction for each item in the batch. Mean squared error is calculated for Adam optimizer to do gradient descent on. The predictions themselves are used to calculate the accuracy of the batch.

3.11.1 Testing on a toy dataset

First this model is tested on the toy dataset with the following parameters:

number of epochs is 10
batch size is 100

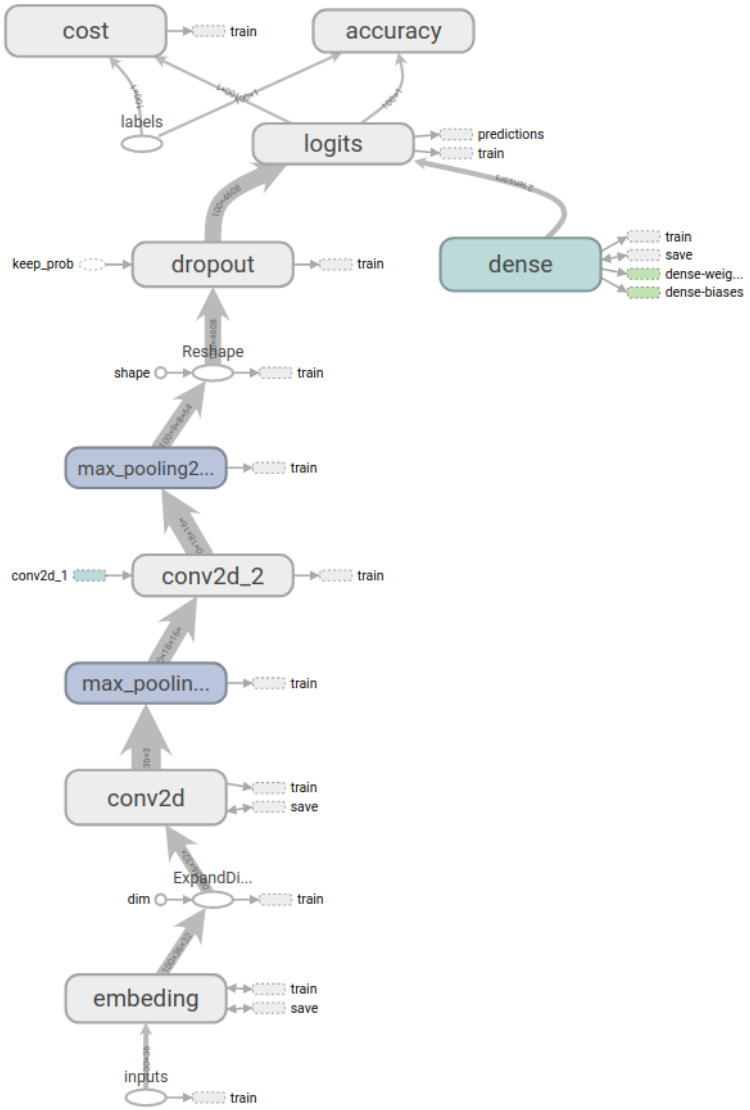

Figure 28. Convolutional neural network with 2 hidden layers

embedding size is 32
sequence length is 36
learning rate is 0.001
keep probability for dropout is 0.7

The training runs for 63.24 seconds and gives a final test accuracy of 0.83. This is the highest testing accuracy for any of the models. However, this improvement might have just occurred randomly.

The cost function indicates that the training might have been stopped prematurely as it was still on the decline. Since this is a toy dataset no tunning was done. Even without finding a minimum value of the cost function the predictions for the validation set reached around 0.7. Figure 29 shows that the validation accuracy did not start increasing until iteration 800, the same is applicable for the cost.

Finally, the the predictions also indicate that the model was able to generalize quite well as the predictions (as shown in fig. 30) started at random guessing and in the final iteration of training were clustered around 0.0 and 1.0. However, it is clear from the cost function that the peaks around the desired locations are still not prominent indicating that the model is not certain of the choices

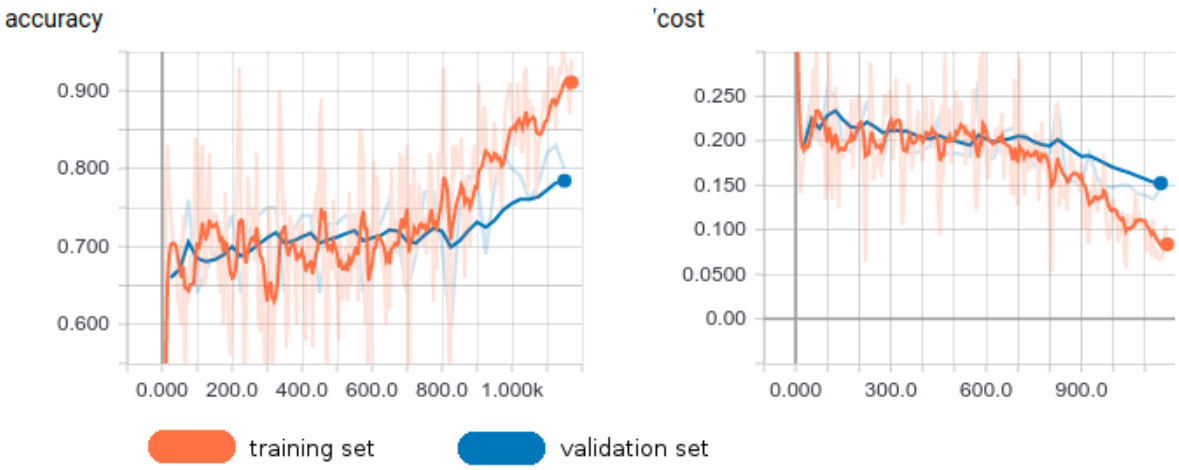

Figure 29. Training and validation results of the third deep learning model on a toy dataset.

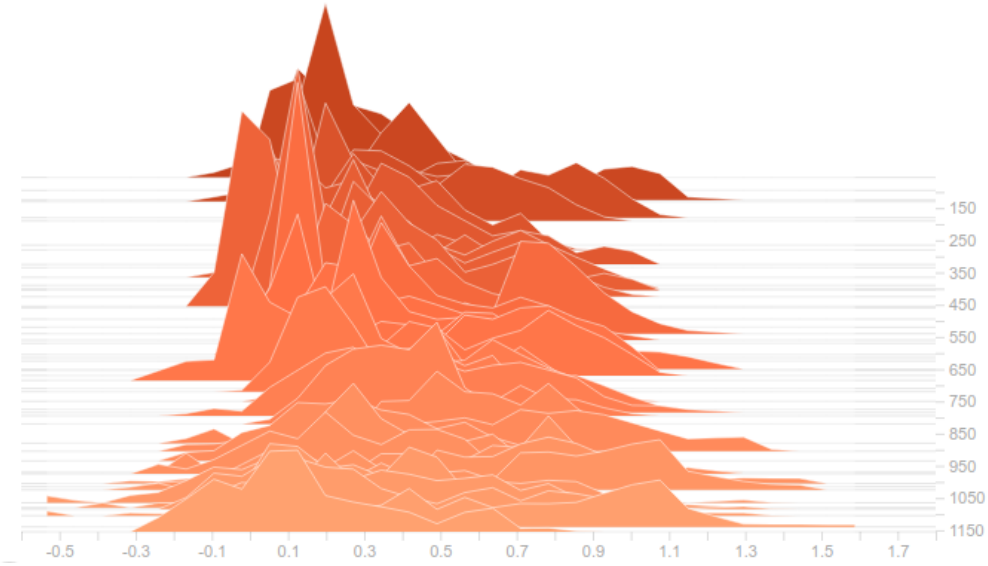

Figure 30. Weights distribution of deep convolutional network on toy dataset

it is making. This might have changed with more training iterations.

3.11.2 Merged tweets dataset

This model is then tested on the merged tweets dataset. The model was tested with many hyper parameter sets and the following one was found to give the best results on the training an validation sets:

number of epochs is 10
batch size is 100
embedding size is 32
sequence length is 400
learning rate is 0.001
keep probability for dropout is 0.7

The experiment runs for 906 seconds and gives a final test accuracy of 56.7 percent. This is higher than any previous architecture. The result is not conclusive and is not high enough to indicate

that there is definitely is a link. However, looking at all previous runs which yielded test scores of around 0.5, this is a clear improvement.

Furthermore, the prediction accuracy for the validation set does not seem to improve much and

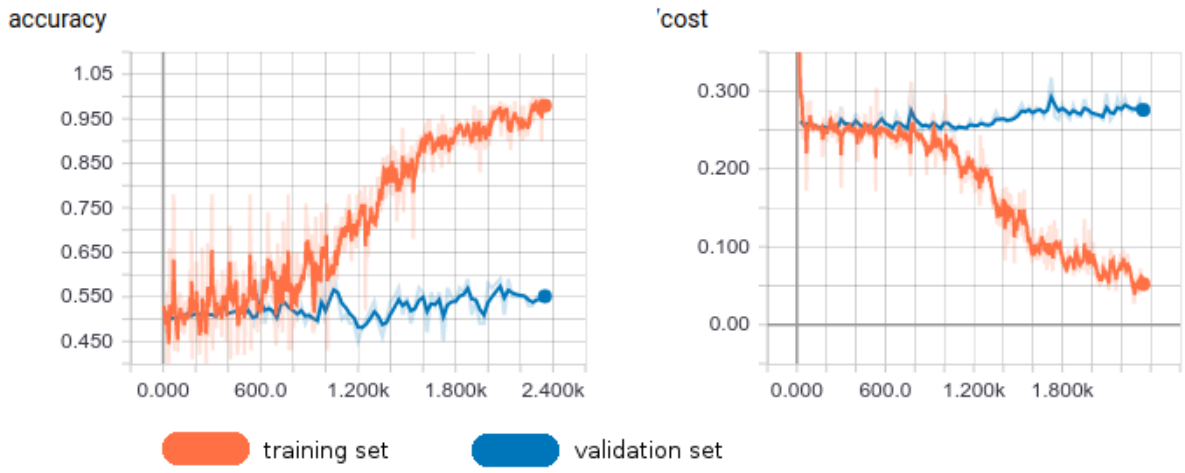

Figure 31. Training and validation results of the third deep learning model on a merged bitcoin dataset.

jumps around between 0.45 and 0.6 durinf the whole test run. Comparing figure 31 to other training runs the cost and accuracy functions do not jump around as much and are more consistent. The training accuracy increases steadily to 0.95 and the train cost decreases to around 0.05. The algorithm does not generalize very well, but it does seem to find some element of correlation between the textual data from twitter and the price movement.

Figure 32 shows the prediction distribution for the training run. The first few epochs in the train-

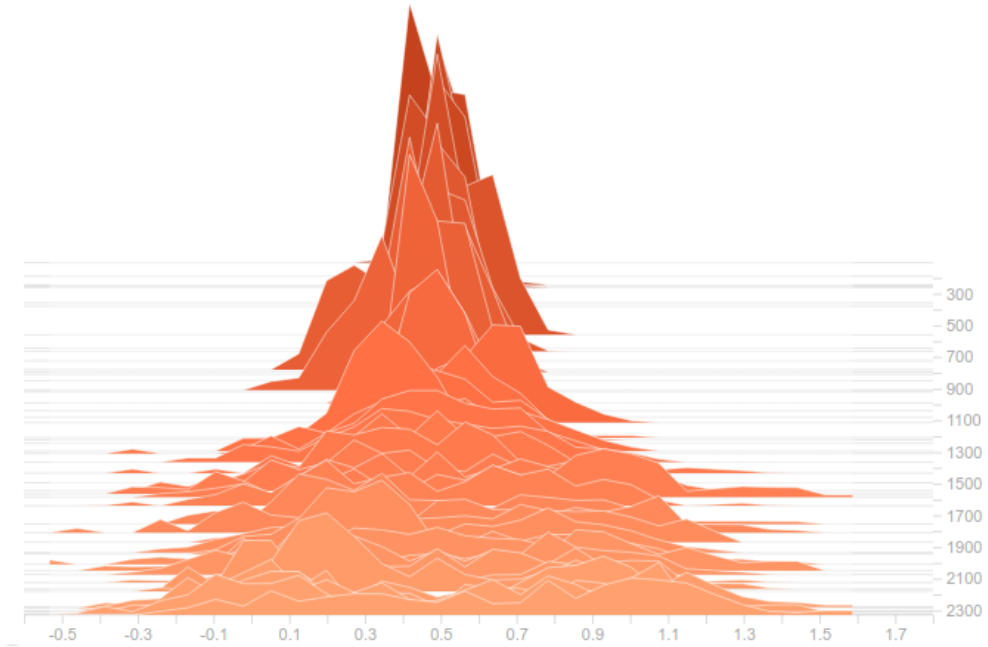

Figure 32. Predictions distribution of deep convolutional network on bitcoin dataset

ing start out at guessing around 0.5 and from about iteration 1000 the predictions spread out and cluster around different values during training. In the final few hundred iterations the algorithm

forms small peaks around 1.0 and 0.8 and spreads in between all values from -0.5 to 1.5. Of course the label values are either 0.0 or 1.0 so not having an activation function that squashes the outputs between 0.0 and 1.0 gives more room for error, however, after just a few iterations the algorithm is able to compensate for the big outputs by adjusting the weights.

The weights distribution for the training run is shown in figure 33. It starts of around 0.0 and dur-

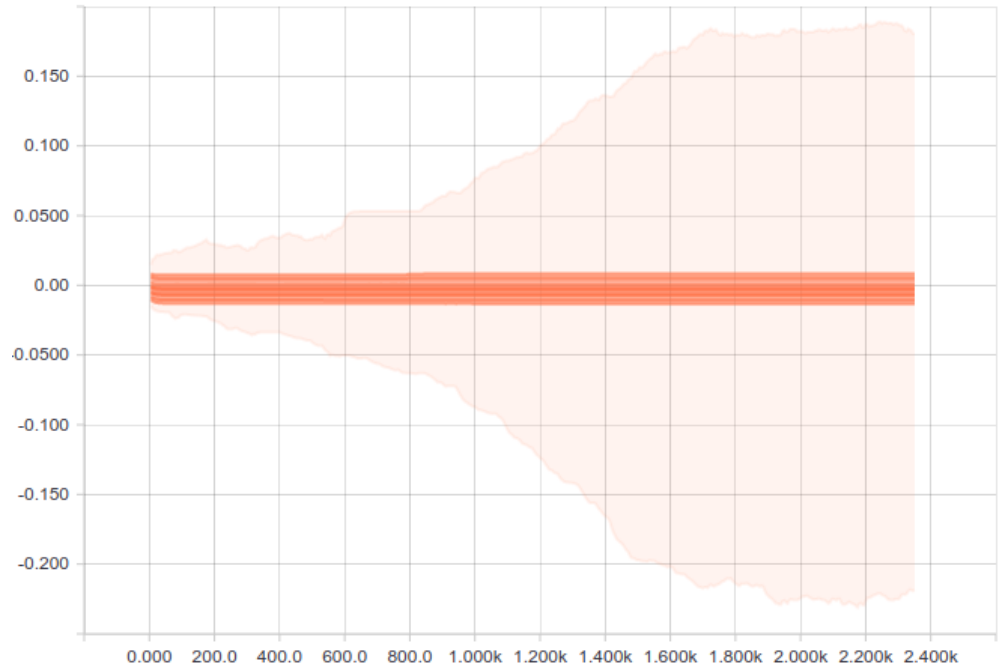

Figure 33. Weights distribution of deep convolutional network on bitcoin dataset

ing the training run spreads out to a more normal distribution that spans from -0.25 to 0.2 during the course of training, although most most of the weights still cluster around 0.0.

From all the models the deep convolutional network yields the best results with both the validation set and the test set. Further research needs to be made, because the results are not good enough to conclude the dependence between the tweet text and price movement.

The biggest problem with the twitter dataset is that there is a lot of noise which the algorithm is not able to cut through. Also there is the question weather twitter in general is a good data source for such experiments. After doing data exploration most of the tweets turned out to be duplicates and the rest are often not substantial enough to find any links to the price. Thirdly, the nature of the crypto-currency market is very stochastic and driven by irrational factors rather than solid information. Finally, the tweets often follow the price changes and not vice versa.

3.12 Fourth model:price based deep recurrent network

The final model is a recurrent neural network. However, it does not operate on tweets but just on the price. The aim is to predict the price given only the price in the previous time-steps. It operates on a minute by minute data same as the tweets dataset. The overall architecture for the model is shown in figure 34. The main difference from the rnn constructed in section 3.9 is that it does not have an embedding layer. It is replaced by just stacking the prices as scalar values and feeding that as input into the first layer of the rnn.

The inputs are a tensor of shape (batch size, sequence length, 1). This is then transposed to make the row vector into a column vector. The rnn itself is a stack of 3 lstm layers with the size of 256

lstm cells each. The inputs are of size 32 steps each step containing a single price point. The batch size is set to 500, the learning rate is 0.001 and the dropout keep probability is 0.7. Also the other difference between to the other rnn's in this thesis is that it does not calculate accuracy as it predicts the price movement in percent and not as a binary 0 if the price decreased and 1 if the price stayed the same or increased.

Furthermore, the price data is not totally clean so there are spikes between minute time intervals

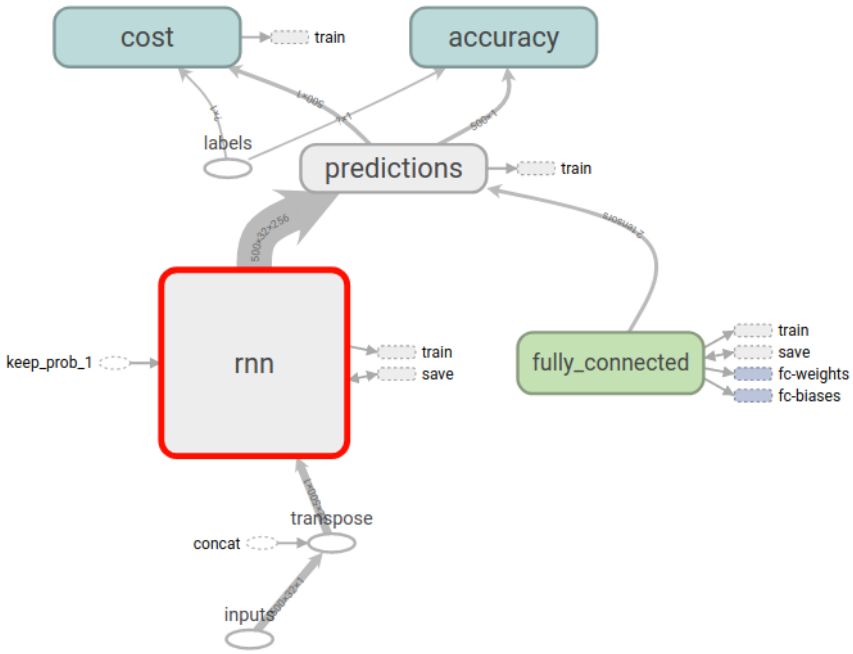

Figure 34. Deep recurrent neural network with no embedding layer.

of 10 percent and more. The data is normalized so that all the price changes that are above 2 percent in either direction are rounded down to -2 or 2 percent respectively. Figure 35 shows how the distribution looks like after the outliers values have been squashed.

The training results are shown in figure 36. The cost function goes down sharply in the first 500

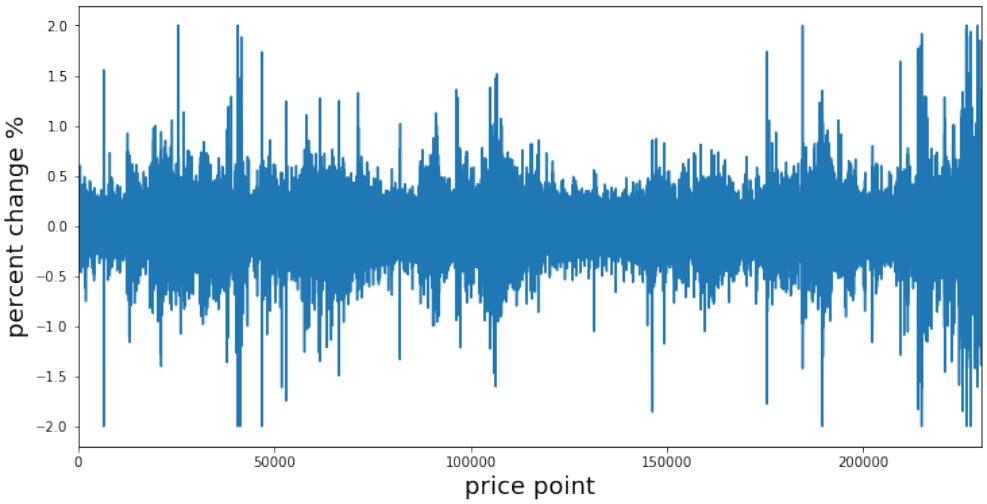

Figure 35. Bitcoin minute by minute price changes for the whole training period, with big values rounded to -2.0 and 2.0 percent.

iterations for both training and validation sets. After reaching a minimum value of around 0.001 it

goes up sharply then again gradually goes down. This is not expected because the spike happens at the 4th epoch meaning that the data was already iterated through 4 times.

The weights distribution change indicates that the network goes through a lot of different weight

Figure 36. Training and validation results of the fourth deep learning model on a merged bitcoin dataset.

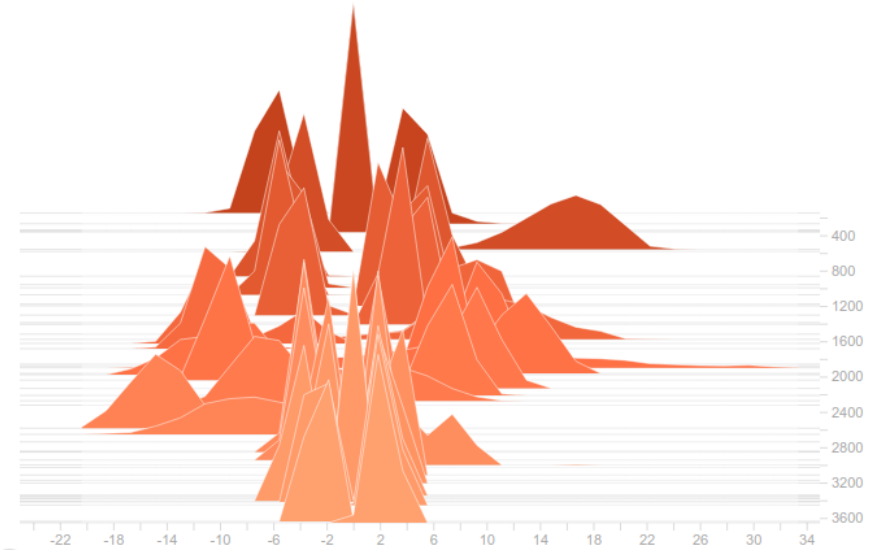

Figure 37. Weights fourth model after running it on the bitcoin dataset

combinations during training and the changes in weights are abrupt as can be seen in figure 37. Normally the weights would change slowly but this is not the case in this training run.

The results prove that price alone is not a good way to predict the price change percentage. The abnormal cost chart might be caused by the price movements which are abrupt and oftentimes parabolic for Bitcoin. Another factor might be the exchange from where the price data was obtained, where there are often speculative price movements that might reach 10% between 2 minute intervals. The poor results of this training run speak for the currently unpredictable nature of Bitcoin price and the many expectations and fears that are prevalent in the market.

Conclusions and Recommendations

This masters thesis focuses on predicting price data from textual sources. Twitter was used as the only textual data source for the experiments as it was the most suitable for such an analysis(see section 3.3). About 60% of twitter data are re-tweets and duplicates. This could be avoided by selecting more detailed filter criteria for data collection.
Most of the models that were created tend to show good results on the training data, but not validation data due to over-fitting.
The data source selection was not optimal, as the training and test results were quite low, with the trained model predicting the price change almost randomly (around 50%).
The best performing model out of all was the deep convolutional neural network with 2 hidden layers in section 3.9. It yielded 56.7% prediction accuracy on the test dataset. The worst performing model was the deep parallel convolutional neural network in section 3.10 with a test accuracy of 48%.
The price alone for is not enough to make good predictions on the price movement. For the period that the data was collected the price increased four-fold which skews the results. Minute intervals had flash bubbles and flash-crashes of up to 10% which is a result of unpredictable market conditions and data discrepancies.
Better data textual information sources are needed to either augment the twitter data or replace it completely. Specialized platforms would be a better option for textual data, even though they do no have as much information as twitter.
In minute by minute intervals tweets are often caused by price movement and not the other way around, therefore better data to price mapping needs to be found.

Plan for future research

The link between twitter data and bitcoin price is not clear and there are things than can be improved upon in this masters thesis.
Firstly, the quality of the data needs to be improved. The current data collection model has many flaws. The twitter data collection gets stuck at times for a few hours, leaving gaps resulting in increasingly worse predictions. The price data is also not consistent with multiple gaps and duplicates that need to be filled by averaging adjacent price points.
Twitter was proven to be really noisy, with many duplicated tweets. This could be changed by using a different textual data source. Stocktwits is an option, however the stream of data there is much lower than on twitter and it would take at least 1 year to collect as much information. A combination of various sources could be used, from news to social media, to price data in combination as inputs into the deep learning models.
Alternative crypto-currencies to Bitcoin could be used for prices predictions. Also a holistic crypto-currency model could be created, where all the movements in the market are evaluated, most other coins have a close price relationship to Bitcoin.
Hybrid architectures could be used in deep learning, like combining colvolutional and recurrent layers.

References

[1] Rodolfo C. Cavalcante, Rodrigo C. Brasileiro, Victor L.F. Souza, Jarley P. Nobrega, and Adriano L.I. Oliveira. Computational intelligence and financial markets: A survey and future directions. *Expert Systems With Applications J.*, (55):194—211, 2016.

[2] M.S. Checkley, D. Añón Higón, and H. Alles. The hasty wisdom of the mob: How market sentiment predicts stock market behavior. *Expert Systems With Applications J.*, (77):256–263, 2017.

[3] Victor W. Chua, Raymond K. Wonga, Fang Chen, Ivan Ho, and Joe Lee. Enhancing portfolio return based on sentiment-of-topic. (54):817—841, 2017.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[5] B. Shravan Kumar and Vadlamani Ravi. A survey of the applications of text mining in financial domain. *Knowledge-Based Systems J.*, (114):128—147, 2016.

[6] Yann LeCun, Yoshua Bengio, , and Geoffrey Hinton. Deep learning. *Nature J.*, (521):436—444, 2015.

[7] Xiaodong Li, Haoran Xie, Li Chen, Jianping Wang, and Xiaotie Deng. News impact on stock price return via sentiment analysis. *Knowledge-Based Systems J.*, (69):14–23, 2014.

[8] Thien Hai Nguyen, Kiyoaki Shirai, and Julien Velcin. Sentiment analysis on social media for stock movement prediction. *Expert Systems With Applications J.*, (42):9603—9611, 2015.

[9] Nuno Oliveira, Paulo Cortez, and Nelson Areal. Stock market sentiment lexicon acquisition using microblogging data and statistical measures. *Decision Support Systems J.*, (85):62–73, 2016.

[10] Venkata Sasank Pagolu, Kamal Nayan Reddy Challa, Ganapati Panda, and Babita Majhi. Sentiment analysis of twitter data for predicting stock market movements. *Power and Embedded System J.*, 2016.

[11] Juan Ramón Piñeiro-Chousa, M. Ángeles López-Cabarcos, and Ada María Pérez-Pico. Examining the influence of stock market variables on microblogging sentiment. *Journal of Business Research J.*, (69):2087—2092, 2016.

[12] Online resource. Coin market cap, 2017. https://www.blochchain.info.

[13] Online resource. Crypto currency market cap listing, 2017. https://coinmarketcap.com/all/views/all/.

[14] Online resource. Tensorflow homepage and documentation, 2017. https://www.tensorflow.org/api_docs/python/.

[15] Online resource. Website that provides historical minute by minute data, 2017. https://www.blochchain.info.

[16] Robert P. Schumaker, A. Tomasz Jarmoszko, and Chester S. Labedz Jr. Predicting wins and spread in the premier league using a sentiment analysis of twitter. *Decision Support Systems J.*, (88):86–74, 2016.

[17] Noam Shazeer, Azalia Mirhoseinia, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *Cornel University Library J.*, 2017.

[18] Junseok Song, Kyung Tae Kim, Byungjun Lee, Sangyoung Kim, and Hee Yong Youn. A novel classification approach based on naïve bayes for twitter sentiment analysis. *KSII transactions on the internet and information systems J.*, 11(6):2996—3011, 2017.

[19] Andrew Sun, Michael Lachanski, and Frank J. Fabozzi. Trade the tweet: Social media text mining and sparse matrix factorization for stock market prediction. *International Review of Financial Analysis J.*, (48):272–281, 2016.

[20] Steve Y. Yang, Sheung Yin, Kevin Mo, Anqi Liu, and Andrei A. Kirilenko. Genetic programming optimization for a sentiment feedback strength based trading strategy. *Neurocomputing J.*, (264):29—41, 2017.