Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

Impact of Social Media on Cryptocurrency Trading with Deep Learning

Scientific Students' Conference

Written by
Szabó Dániel Attila

Supervisor
Dr. Gyires-Tóth Bálint Pál

26th October 2017

Abstract

Due to the revolutionary increase in the amount of available data, the rise of high performance GPUs and the novel scientific results of neural networks, deep learning has received high attention among machine learning scientists. The numerous layers of deep architectures are able to extract and learn different abstractions of the input data and model them more efficiently than previous machine learning methods in many application fields. Deep learning is one of the main technology of language understanding, natural language processing and time series analysis.

One of the promising theoretical question is the feasibility of deep learning for weak coherent signals, where the feature extraction and therefore the analysis and modeling is difficult with classical methods. (For example market demand or price movement of a financial asset and the corresponding news sentiment, or signals of an IoT sensor network are weak coherent signals.

Digital currencies with an underlying peer-to-peer, decentralized payment system have seen a large growth recently. Most of the coins reward the so called miners, who are offering their computational capacity to enable to run and secure the decentralized services. This earning has a real values on open market, therefore the growing popularity of cryptocurrencies has an influence on the growing of the price tag of this rewards. The aftermath of this considerable presence on the market is the appearance of the exchange services for these currencies. These exchanges execute a huge amount of tradings with an opened API.The market analysis is difficult, because the volatility is high, the signal is noisy, and the lack of classical features cause the need of rethinking the well-tried strategies. Furthermore, cryptocurrency markets are not regulated.

Due to the large amount of public data from the blockchain and cryptocurrency exchanges, data driven machine learning approaches may explore pricing anomalies. Besides the trading data additional sources of information (like political, or financial news) may enhance the predictive performance of a machine learning model.

Twitter like short texts are typically within the modeling capacity of such analytical systems. As social media, including Twitter, become one of the primary source of news (eg. political, technological, financial news), it has a great influence on masses. Thus, it is critical to be able to model, classify, cluster these texts and eg. to extract sentiment and detect fake news.

I suppose that the price movement of cryptocurrencies and publicly available data on Twitter are weak coherent signals, therefore social media activity may affect the price of the cryptocurrencies. The goal of this work is to use the deep learning methods to analyze

the behavior of crypto markets, including the price movement and Twitter activities. I apply deep learning based natural language processing (NLP) techniques on Twitter feeds, I build a deep learning model for cryptocurrency price movement and complete this model with the NLP methods. The evaluation of the feasibility for weak coherent signals are also introduced.

Kivonat

A rendelkezésre álló adattömeg növekedésével, a magas számítási kapacitással bíró GPU-kal, és a neurális hálókkal elért tudományos eredményekkel a mélytanulás hatalmas figyelmet kapott a gépi tanulással foglalkozó tudósok körében. Rétegzett mély architektúrák sikeresen képesek a bemeneti adatok magasabb szintű absztrakcióinak kinyerésére és modellezésére. A mély tanulás az egyik főbb eszköze a természetes nyelvfeldolgozásnak és az idősoranalízisnek.

Ígéretes elméleti terület a mély tanulás vizsgálata gyengén koherens jelek esetén, ahol a jellemző tanulás (jellemző kinyerés) és a modellezés is nehezebb. Gyengén koherens idősor például a termékkereslet előrejelzése vagy egy adott pénzügyi eszköz és a kapcsolódó hírek tartalmának időbeli vektoros reprezentációi (beágyazásai) vagy IoT szenzorhálózatok jeleinek együttes modellezése.

A peer-to-peer, decentralizált digitális (kripto)valuták hatalmas növekedésen mentek keresztül az elmúlt években. Közel az összes kripto valutajutalmazza az úgynevezett bányászokat, akik számítási kapacitásuk felajánlásával teszik lehetővé a decentralizált szolgáltatás biztonságos működését. Ezen jutalmak valós értékkel bírnak, ami kihatással van a kripto valuták értékének növekedésére is. A meghatározó piaci jelenlét magával vonta különböző kereskedési szolgáltatások megjelenését. Ezen kereskedési felületeken hatalmas összegek forognak nap mint nap. Az árfolyam mozgás jellemzően nem mutat a klasszikus tőzsdéhez hasonló viselkedést (úgy mint trendeket, vagy napi áringadozást). A piac elemzése bonyolult, mivel a volatilitás magas, a környezet zajos, és hiányt szenved klasszikus mintákból, továbbá ezen piacok nem regularizáltak.

Sejtésem szerint a rendelkezésre álló pubilkus adatok segítségével egy adatvezérelt megközelítés felfedhet árazási anomáliákat. Ilyen rendszerek használhatóságát ígéretesnek látszik javítani különböző tématerületből származó adatok bevonásával (mint például politikai, vagy pénzügyi hírek elemzésével).

A Twitterre jellemző rövid szövegek előnyösek lehetnek egy adatvezérelt modell kialakításához. A közösségi média számos területen életünk első számú hírforrásává vált (mind politika, pénzügyi, vagy akár technológiai kérdésekben), így komoly hatással van nagy tömegekre. Ezért különösen érdekes lehet ezen hírfolyamok modellezése például piaci szentiment kinyerésre vagy álhírek detektálásánál.

Feltételezésem alapján a kripto valuták árfolyamadatai és a kapcsolódó Twitter csatornák gyengén koherens jeleknek tekinthetőek., Elképzelhető, hogy a közösségi média kihatással van az árfolyam mozgására, és az árban megjelenő ingadozások is megjelennek a bejegyzésekben.

A munkám célja mély tanulás eszközeit felhasználva vizsgálni a digitális valuták piacát, egy modellt építeni az idősorok viselkedésének modellezésére, és kiértékelni az eredményeket. Ezt a modellt természetes nyelvfeldolgozás alapú Twitter hírcsatorna elemzéssel egészítem ki. Dolgozatomban megvizsgálom az eredeti és a kibővített modell hatását a modellezés pontosságára, illetve elemzem a mély tanulás alkalmazhatóságát gyengén koherens jelek esetében.

Contents

Chapter 1

Introduction

Modern informational and decision support systems are built on the fact that there is public or accessible information and the corresponding methodology, which bridges the existing computation power and raw data to extract business value. In an informational system quality of the performance highly depends on the input data and the method how the system is processing it. Be it a decision making, a recommendation or a diagnosis system the overall performance depends on the input and the system's functionality. To ensure the progression of such systems new strategies are needed to aggregate, progress and use information. A promising way to enhance such systems is to use week-coherent signals, extract the connection between them and use this knowledge as a new business value. The analysis of these kind of signals can be a complex, challenging task. Specially, if the environment of the data source is noisy and its behaviour changes over time, there is a constant need for new algorithmic approaches and strategies. Due to the amount of accessible computational power, to the increase of public data sources and novel scientific results deep learning has become one of the most enhanced machine learning technique recently. The various applications of deep learning are present in our everyday life. Be it a image recognition, drug discovery, recommendation systems, or natural language processing based problems, this border family of machine learning earned great results in many fields. The theoretical question is straightforward:

Can be this new method a part of a informational system where the aim is to extract value from weak coherent signals in a noisy environment?

The goal of my "TDK" work is to evaluate the feasibility of deep learning in modeling of weak coherent signals. Therefore, in the first section I define the used terminology and the scope of this article. Next, I give a detailed picture about the chosen research environment, and about its special characteristics. After laying down the fundamentals I will introduce two different signals in this environment and techniques how can we process it with existing deep learning algorithms, highlighting the ones which had a great impact on my solution.

In the third section the feasibility of deep learning in this environment is described in details, and I also propose a possible deep learning solution.

Chapter 2

Terminology

Because the used terminology determines the readability and the intelligibility, it's important to define the scope of the paper, and to clarify the used definitions and language. As I stated before, this paper focuses on the problem of weak coherent signal processing. Weak coherence between two signals means that there is a connection between the elements of a multivariate signal, but the relation is not straightforward, therefore the detection, and the utilization of this behavior is problematic, especially in noisy environment. The main scope of this paper is to evaluate deep learning methods to analyse coherence between these signals. My work includes not only deep learning based methods and strategies, but some classical preprocessing steps as well. However, in my work I was focusing on using as few preprocessing methods as possible and utilize deep learning methodology for representation learning.

Machine learning methods can be classified into the following three categories:

- supervised learning: the learning data is labeled, the target values are provided, the task is to find the mapping from the input to the corresponding output, which generalizes well,
- unsupervised learning: the data is unlabeled, and the task is to find meaningful representations, and key features and clusters in the data,
- reinforcement learning: an AI agent interacts with a real, or simulated environment, and the agent improves its performance according to the feedback from the environment.

These three approaches involve a huge amount of different techniques and algorithms. This paper focuses on supervised learning.

To be able to examine weak coherent signals the environment plays a valuable role. A

relatively noisy environment with at least two signals is required. I chose the cryptocurrency market, because this field isn't regulated yet, the behavior of the actors on the market mostly uncertain, the price movement can be very volatile, thus the risk is high. For example the volatile index (averaged for 2017 Q3) of EUR/USD is circa 0.7%, while the Bitcoin/USD is 4.7%, and the Ethereum/USD is 5.6%, detailed historical data can bee seen on the figure 2.1 [1].

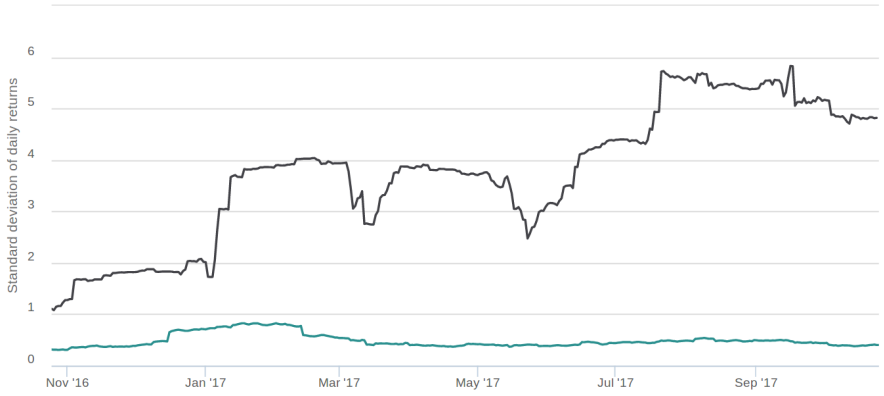

Figure 2.1: *Volatile index of USD/Bitcoin (black), and USD/EUR(green) from November 2016 to November 2017.*

The value of the cryptocurrencies is dependent of the demand and the supply, thus the price is act as a discrete signal, because of the transactions. The signal - consists of ask prices - is the mainly used signal in my work, and the aim is to find an another signal, which shows weak coherence with this, and to model the connection of these signals with deep learning.

The popularity of these currencies is reflected in the everyday life. Articles, books, and many rumors have appeared recently, the social media showed a huge activity around this field. As 2.2 figure shows the popularity of Bitcoin (currently the major cryptocurrency) grew and reached considerable attention compared to dollar or euro.

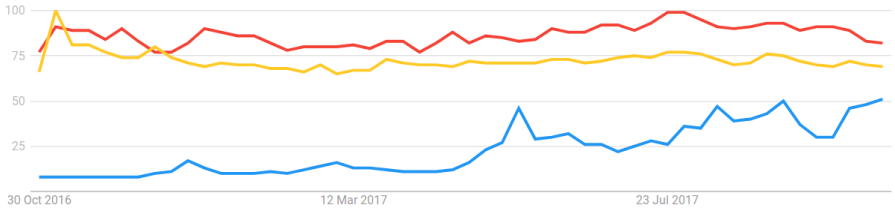

Figure 2.2: *Google search index for USD (red), EUR (yellow), and Bitcoin (blue) from 2016 November to 2017 November. The graph is made with data from Google Trends. The index is a relative measurement, where the highest point of interest is equal with 100, and the lowest with 0.*

The sentiment of each post varies, but the overall sentiment may have an impact to the actual price value of these coins. A challenging task is to analyse the published messages,

[1] Data is extracted from https://www.buybitcoinworldwide.com/volatility-index/ (20. October 2017)

articles, and posts about these cryptocurrencies.

There are several social media platform (for example Facebook, Twitter), and a lots of newsfeed (like BBC.com, Wired). In my current work I chose Twitter, because the frequency of cryptocurrencies news articles is low for a successive signal (still, it may have on impact), while there are a large number of related posts on Twitter (see figure 2.3). Another reason of choosing Twitter is that further social media platforms (eg. Reddit) has no such a short text oriented content like Twitter, and they may contain images, music or videos as well. In consequence, the second signal that I use in my current work is containing tweets from cryptocurrency related Twitter feeds, and it's meant to be to represent the sentiment. (In this terminology, we consider the "sentiment" as a time series.)

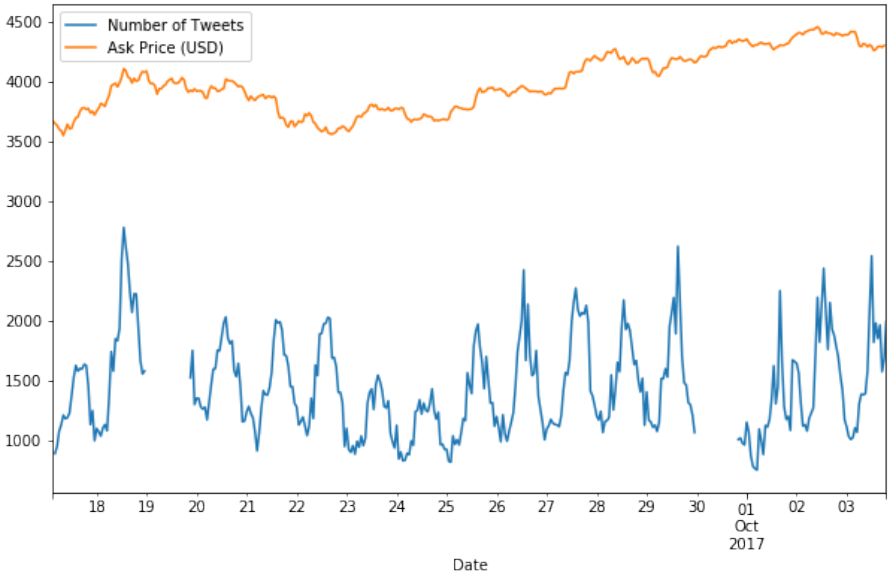

Figure 2.3: *Bitcoin price and the number of tweets that contains "Bitcoin" as a word. The daily swing in the tweeting frequency is clearly visible.*

These two signals, about which I suppose that are weakly coherent, are the ask prices of selected cryptocurrencies and Twitter posts (from which I will extract the sentiment) arranged in a temporal structure. The aim is to extract and learn features with deep neural networks and to model the combined behavior of these signals. The goal can be break down into two separate tasks: (1) a time series analysis and modeling and (2) a natural language processing (NLP) challenge.

Chapter 3

Related works

In this section I give a snapshot about the currently used deep learning methods, which are applied for time series analysis (TSA) and for natural language processing (NLP). The goal of this section is to investigate the directions of previous works in time series analysis and natural language processing with deep learning for financial domain, however it is not a complete review of the existing method

3.1 Time Series Analysis

In deep learning time series modeling is one of the main challenges. For example in speech recognition or synthesis, weather[36] and traffic forecast[25] the informational systems want to reach a higher level of certainty: better speech recognition, natural synthesized speech, trustful weather and traffic forecast. The underlying system is built upon one important hypothesis: there is a dependency between the current value of the signal, and the previous values, or with another words, the past has an impact on the present. This key feature can be extracted in many ways with deep learning, let us investigate the most important directions.

3.1.1 Recurrent solutions

The aim is to use the the causality between the values to forecast a future value. A regular feedforward multi layer perceptron architecture is not capable to extract complex temporal features. To address this deficiency it is recommended to use another architecture, namely recurrent[28] or convolutional solutions.

Long Short-Term Memory (LSTM)

In deep learning, one neural network consists of neurons, where in the case of a feedforward fully connected architecture, each neuron in a given layery is connected to all neurons in the next layer. Recurrent neural networks have a different dataflow, the direction of the data is controlled in a directed graph: each neuron receives not only the actual input, but previous values, so it can abstract the connection between the current output and series of inputs from the past. The learning procedure utilizes a modified version of back propagation, namely the back propagation through time (BPTT) algorithm[23].

There are several variants of reccurent neural networks. The most widely used is the Long Short-Term Memory cell (LSTM)[20]. In a simple recurrent neural network the output of each neuron is back-coupled to their input, it can typically abstract a short dependency, however, it is uncapable to catch long term dependencies. LSTM is explicitly designed to address this problem. One LSTM cell consists of four different elements. LSTM can remove, or add information to the cell's state by this additional elements, called gates. The behavior of these gates are learned from the data, so the final model knows which information is needed and can change the activation according to this state.

3.1.2 Convolutional Networks

These networks was mainly used for image and speech processing[1] [16], because a simple feedforward fully connected neural network can't handle the curse of dimensionality that comes from an image or speech. The way this architecture solves this problem is the usage of property encoding by convolutional layers with shared weights, and with pooling layers.[22]

One approach is to convert time series to image like datasets then feed it to a convolutional network. One way to convert time series to convolutional network compatible is frequency splitting and aggregation[3]. To do so, we first convert the signal to frequency domain and split it into parts by bandwidth. The number of bandwidth groups will be the depth of the data, just like red, green, and blue channel in a RGB image. Then we calculate a value for each bandwidth segment with a predefined function, for example with the integral of the signal magnitude. In this way we get a representation, which can be fed to a convolutional neural network.

3.1.3 Deep Learning Solutions for Financial Asset Price Modeling

Time series analysis is an important task for many fields in applied machine learning, for example: speech recognition[15][19], stock market forecast[11], market supply forecast[38], etc.

Heaton [17] uses deep learning hierarchical models for problems in financial prediction and classification. They demonstrated that deep learning has the potential to improve - sometimes significantly - on predictive performance in conventional applications. In their paper the models are using LSTM and fully connected layers.

Aaron Elliot, and Cheng[12] made four models (linear model, generalized linear model, recurrent neural network, and a model with online-to-batch algorithm) to predict stock market price using S&P 500 index as an input time series data. The best performing model was the recurrent neural network based solution. Besides the models they also created a trading strategy that can create win-win and zero-sum situations.

John Gamboa[13] investigated deep learning algorithms for time series analysis more generally. He reviewed different approaches for various tasks related to time series, let it be a regression, or a classification problem. He proved that stacking several independent neural network layers can produce better results than the already existing shallow structures.

Chong [11] made a study about deep learning in stock market analysis. They used high-frequency intraday stock returns as input data and they examined the effects of three unsupervised feature extraction methods principal component analysis, autoencoder, and the restricted Boltzmann machine on the network's overall ability to predict future market behavior. They pointed out, that when the network is applied to covariance-based market structure analysis the covariance estimation has improved.

During my literature review I definitely got the confirmation that for my proposed model, the best approach is to use recurrent neural architecture with Long Short-Term Memory cells. Several papers formulated the problem as classification, however, in my current work I focus on a regression model, as the predictive performance of such models already shows if additional, weak coherent data feeds have an impact.

3.2 Natural Language Processing

The goal of my work is to improve the prediction performance of cryptocurrency price models with considering social media activity. Therefore, I will use Twitter posts as data source, and natural language processing for feature learning/extraction. In the following I introduce current state-of-the-art methods using deep learning for linguistic problems.

One of the main tasks in natural language processing (NLP) is language modeling. The aim is to produce the best representation of text for computers to solve NLP based problems, such as speech recognition, translation, text generation, or text summarisation. The difficulty of this model building is the high-dimensional behaviour of the texts. Classical, statistical and rule based solutions are used even today for NLP tasks, but rule based system doesn't scale well, so it's hard to adapt for new datasets or for new domain. Still,

classical methods are used to improve the quality of machine learning systems. A deep learning based system can adapt to the dataset much easier thank to the representation learning capability,, but high-dimensionality introduces a challengeable task here. The order of words, or simply the diversity of our language causes high variance in our sentences, made the text to vector transformation an interesting, and challenging task. We need this vector representation because we want to store relationships that may exist between the individual words.

3.2.1 Words, and sentences to vectors

The question - "How can we represent our text as numbers?" - has a long history in natural language processing. The main idea behind the transformation is that we want to build a probability distribution over sequences of words, or characters. These models are called n-grams, where we predict the succeeding item with a likelihood knowing the preceding items. Words should converted to vectors, where tokens with similar meanings are "closer" to each other[30] (meaning that the distance between vector representation of two similar words are shorter than two irrelevant words). For example the distance between "book" and "novel" is shorter than between "refrigerator" and "dinosaur".

Due languages contain thousands of different words, and grammar can enlarge this number with affixes, we need to reduce the dimensionality of our matrix representation. There are several techniques to solve this problem, the most common one is to use just a small subset of our vocabulary, or change inflected words to one similar word. The selected strategy depends upon the natural language problem. For example, if we want to generalize text, we need to keep the exact same grammar model as it is used in the natural language, but for sentiment analysis we can capture useful features without that. Of course in general we want to capture as much features as much we can (or in deep learning, we want to let the model abstract this features), this is a trade-off between speed and accuracy.

The following section will introduce how one can build natural language models, and how can get rid of dimensionality problems with deep learning. Most existing models fall into one of free classes: bag-of-words models, sequence models, or tree-structured models, but it should be noted that all methods depend on the distributional hypothesis: words that appear in the same context share semantic meaning.

Bag of words

This model is an orderless representation of sentences and phrases, that means for the model hasn't got information about the relationship between nouns, or verbs. We have only information about the frequency of occurrence in the given context. There are two main subtype of this approach: continuous bag-of-word model (CBOW)[40], and skip gram

model.

CBOW based neural network model want to predict the likelihood of occurrence of a simple word in a window with predetermined size. During training there are words from past, and from future in the windowed buffer, so the model builds a log-linear classifier with 'n' words from history, and 'm' words from future, with the aim to classify the "middle" word.

For model architecture, see figure 3.1. At input, the words are encoded using 1-of-V coding, where V is the size of the vocabulary. Then the input layer is projected to a projection layer, means that the hidden layer output is the average of word vectors corresponding to context words at input.

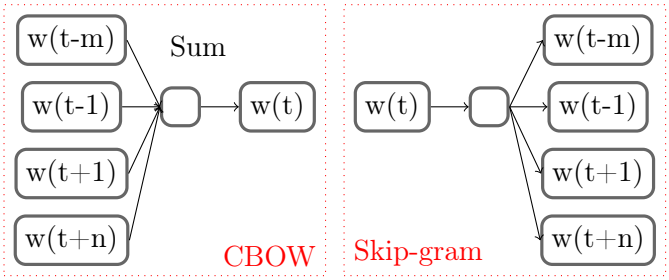

Figure 3.1: *CBOW, and Skip-gram arhitecture (modified[27]).*

Skip-gram model is similar to CBOW, but the direction of prediction is inverted, the model want to predict the words in the neighborhood of the input word. Worth to mention that increasing the range improves quality of the resulting words vectors in a manner, but it also increases the complexity.

The architecture (showing in figure 3.1) is similar to CBOW, but as I mentioned before, it is inverted. The error vector from output layers are summed up to modify the weights via backpropagation.

In the previous paragraphs the models used simple windowed buffer to get word vectors from a given context. The problem with this approach is that the size of the window is affect the outcome - because we always fill the buffer, we need to slice the text accordingly, even if the previous word doesn't affect the target word's meaning. This type of models have their own limitations, but because they are simple, there is no need for additional information, or data structure.

Sequence Models

One of the main problem in the aspect of NLP with vanilla feedforward neural networks is the following assumption: all the inputs are independent from each other. In natural language the order of the sequence also has meaning, therefore plenty models are using recurrent neural networks (RNN). With RNNs we also solve the fixed size input problem.

The most commonly used RNN type is Long Short-Term Memory networks (LSTM).

Sequence-to-sequence model is introduced by Cho et al.[10]. The base architecture (see figure 3.2 on page 13) contains two RNN in encoder-decoder structure The encoder's task is to map the input data to a different feature representation, the decoder's task is to map the previously created features to output (or back into the input data space), and the hidden state of the encoder is transferred to the decoder, this will be a fix sized vector representation. Note that the model is working with sequences with different length.

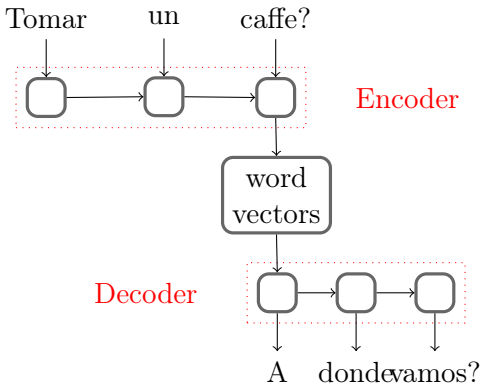

Figure 3.2: *Seq2seq autoencoder arhitecture (modified[10]).*

One problem with classical recurrent neural network is difficulty to learn long term dependencies. To solve this problem the model also can be built with LSTM or GRU[1] cells.

With this representation it's possible to build better models due it is order-sensitive, but we can't extract the full structure, and semantics because sequence models are still unable to capture different meanings as a result of different syntactic structure.

Tree Structured Models

Syntactic structure is built with rules governing the grammatical arrangement of words and morphemes in a language. These linguistic models are very similar to tree structures, so the following models are based on this tree schema.

In this Tree LSTM model (on figure 3.3 the representation of a sentence (or root node) is calculated by a recursive function of its children, and so on down the tree. The recursive function $\vec{p}$ is defined as:

$$\vec{p} = \sigma(W[\vec{c_1}, \vec{c_2}])$$

where σ is a nonlinear function, and $\vec{c_1}$ and $\vec{c_2}$ are the child representations, and the representation of the root node is calculated bottom-up.

[1] Gated Recurrent Unit

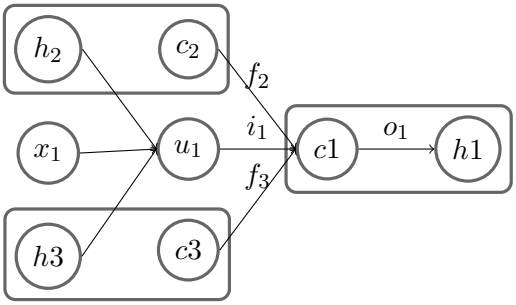

Figure 3.3: *Tree-LSTM cell with two children node[33].*

These architectures are based on LSTM generalisation to tree-structured network topologies. The tree-LSTM [33] composes its state from an input vector and the hidden states of many child units. Tree-LSTM cell contains one forget gate for each child, this allows the unit to selectively gather information from each child. From this viewpoint we can see the classical LSTM as a tree-LSTM, where each internal node has exactly one child.

However the previously mentioned tree-structured models are suffering from two things. Because each sequence has a different model structure, it's much harder to implement an efficient batching strategy due the variety and the size of the data. To address this problem we can use a syntactic parser to preprocess the input, but this approach drags one more dependency, and brings more complexity to the system.

Stack-augmented Parser-Interpreter Neural Network (SPINN)

To address the problems of the Tree-LSTM we need to change the representation which can support batching better. To produce parse structures from sequences in linear time the stack-augmented parser-interpreter (SPINN) neural network uses shift-reduce parsing [8].

Shift-reduce parser is a table-driven bottom-up parsing method, which starts with an empty reduction stack, and with an input string (or with stack of tokens). First we shift the first token from the input string to the reduction stack. After each shift we check the reduction stack, if the top elements fit to a predefined rule, we can apply the reduce operation. The reduce operation merges the elements, and pushes back the result to the stack.

In SPINN, the reduce operation simply combines the top two elements of the reduction stack into a single element with a standard feedforward recurrent neural network function: $\vec{p} = \sigma(W[\vec{c_1}, \vec{c_2}])$, called composition function.

This solution computes the same function as the recursive neural networks mentioned above. After the neural network based shift-reduce parsing is finished, we have a sequence representation, which can be used as an input to an another neural net.

1-Dimensional Convolutional Neural Network

Convolutional networks are commonly used in computer vision, because the input data source - 2 dimensional data source with different number of color channels - is to large for a fully connected neural network, but not for a convolutional neural network (ConvNet, or CNN). The aim with images is also the same as before: reduce the full image into a single vector. The ConvNet contains neurons arranged in 3 dimensions: width, height, depth - where the depth refers to the third dimension of an activation volume, for example in image processing the depth belongs to the number of color channels (red, green, blue). The architecture has 3 different types of layer: convolutional, pooling, and fully-connected layer. Further details of ConvNets are described in the corresponding paper.[22] In the following I will show how can we use it for NLP tasks.

1-dimensional convolutional refers to the width of the convolution's filter, it's the same as the length of the feature vector of the individual word encodings. Let's assume that we have one channel, this means that we have only one type of vectorized representation as input. If we want to extend the model we can use different vector representations as input, or different vectors from different languages, but first assume we have only one channel.

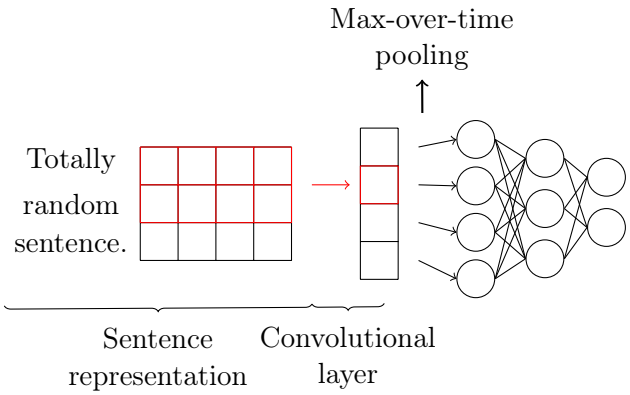

Figure 3.4: *Simplified 1-D Convolutional Neural Network Arhitecture.*

A very simplified version can be seen on figure 3.4 on page 15, where the inputs are static word encodings. The first part, the convolutional layer contains four feature map with different filters, and stride sizes. We are using wide convolutin with zero padding to down sampling in the filter dimension. After convolution we are using pooling layers to subsample the results with a simple max function. In our example we have a 1x4 pooling window. After the pooling operation we are feeding the tensors to a fully connected network.

The output of the pooling operation is feeded to a fully connected neural network.

Char Convolutional Neural Network

In this model we are using convolution not only on tokens, but in character level [39], therefore the input of the models is encoded characters. Then the sequence of characters is transformed to a vector with a fixed size, any character that are not in the predefined alphabet (for example blank characters) are transformed to a null vector. The character transformation order is reversed to help the fully connected layers to associate weights with the latest reading.

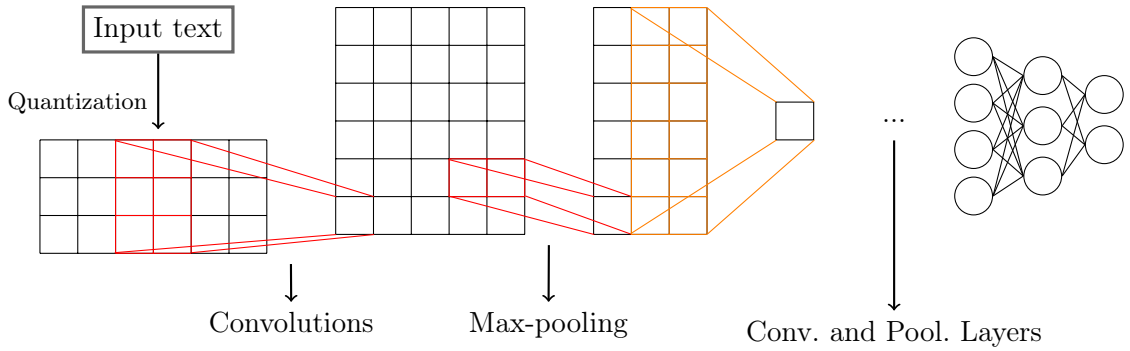

Figure 3.5: *Character based ConvNet.*

The input feature size in the architecture (on figure 3.5 on page 16) is equal to the size of the predefined alphabet due the character quantization method. The architecture is nine layers deep with six convolutional layers and three fully connected layers. The convolutional layers have stride size one, and the pooling layers are all non-overlapping.

3.2.2 Unsupervised Methods

Besides deep learning models, I also reviewed classical statistical models. Because some of the mentioned models are using vector representations from classical methods as initial values. These methods are unsupervised solutions.

3.2.3 Global Vectors for Word Representation

GloVe is a log-bilinear model with a weighted least-squares objective[30].

3.2.4 FastText

FastText is a library for creating efficient learning of word representations and sentence classification[6]. Facebook provides a C++11 based project to create word representations for a given text, also there are available representations for 294 different languages. The

model uses a logistic loss function $l : x \mapsto log(1 + e^{-x})$, and obtains the objective:

$$\sum_{t=1}^{T} \sum_{c \in C_t} 1(s(w_t, w_c)) + \sum_{n \in N_{t,c}} l(-s(w_t, n))$$

where w_t is a given word at position t, c is the context, and $N_{t,c}$ is a set of negative (which is not represented in the context) examples sampled from vocabulary, and s is the scoring function:

$$s(w, c) = \sum_{g \in G_w} z_g^\top * v_c$$

where $G_w \subset (1, ..., G)$ is the set of n-grams appearing in a given word w, it will associate a vector representation z_g to each n-gram g, and the representation of a word is the sum of vector representations of its n-grams.

3.2.5 Deep Learning Based NLP Solutions for Sentiment Analysis

Tang [34] created a system for message-level Twitter sentiment classification. Their model is built in a supervised learning freamwork, the test data was the Twitter2014 test set of SemEval 2014 Task 9. They combined the sentiment specific word embedding features (learnt by a deep learning network) with hand-crafted features. Their system was effective in both positive/negative/natural and positive/negative classification of tweets.

Gregoire Burel [9] made a model to capture the contextual information from tweets, and use it to identify the existence of an event, and event types. They can identify it with 79% accuracy (F-measure), which is competitive with classical machine learning models, such as SVM. The splitted the tweets into two main data source: word based, and a concept based information, then they created embeddings for it, and fed it to a Dual-CNN network.

Johan Bollen[7] investigated how can societies experience mood affect the collective decision making. They measured the sentiment of the crowd by using large Twitter datasets. They extracted the sentiment state from tweets, then they used neural networks to abstract the impact of the sentiment changes on DJIA values[2].

Yifan Liu [24] used stock market forums to retrieve sentiment informations to predict the behavior of Chinese market. They created a tool for generating sentiment weights from chinese posts (so the neural net is not making the abstraction itself), and an emotion model to process this data. They evaluated the price prediction on a simple recurrent neural network, and with a emotion model supported RNN. The prediction the prediction performance improves nearly 4%.

Previous methods shows that there is an opportunity to use text based information to

[2]Dow Jones Industrial Average

enhance prediction on stock market. The most related method is made by Yifan, however, it uses preprocessed text, and the model get a sentiment score as an input. In my solution I want to let the model abstract this features.

Chapter 4

Proposed Method

To address the goal of this paper, to research the feasibility of deep learning methodologies for weak coherence analysis, and Bitcoin price forecast, I define three different tasks:

- convert currencies ask prices to a signal, and build a model for it,
- natural language progressing task: convert tweets to a signal,
- connect the two models and evaluate its performance.

In the following subsections I give a guideline how to create a framework that models simultaneously the prices and the Tweets with deep learning.

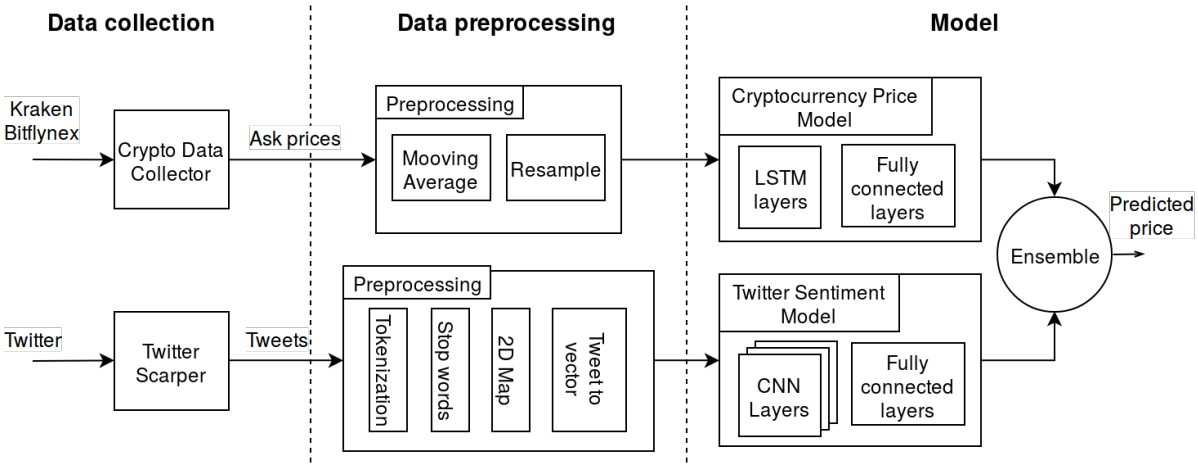

Figure 4.1: *Proposed system structure.*

The proposed system structure (in figure 4.1) consists of two parts: time series, and a natural language model. Each part has its own input, and they are functioning in parallel, however the training is done at the same time. First I separate these two model, evaluate the performance apart, and then I reconnect these models.

4.1 Cryptocurrency Price Model

The first part of the proposed method is the time series model. In this section I will show how I collected publicly available data from the web, I introduce the tools I created for this task, then I describe the proposed model. This model will be the baseline for the weak coherence analysis.

4.1.1 Price Data

Just like in classical exchange market the cryptocurrency market has several different measurement units, like:

- ask price
- bid price
- ask volume
- bid volume
- averaged ask, and bid prices (24 hours)
- opening price
- last closing price
- etc.

There are many existing and available dataset, but the resolution, and the mentioned available measurement units are different for each dataset. I need less dependency from the quality of these data sources, therefore I developed a tool to extract real time information from cryptocurrency market.

Data Collection

There are several existing application interfaces from where the tool can download data The disadvantage of these sites is that they aren't allowing custom queries for historical, therefore the only solution is to create a real time tool.

Although the prices for a specific currency are similar in these exchange services I used two exchange websites to reach robustness in the tool (I don't want to lose data, because the servers are down). Kraken, Bitflyer, and Bitflinex[1] transact on of the the biggest volumes daily[18] (see figure 4.2), so I used these sites.

[1] www.kraken.com, www.bitfinex.com, www.bitflyer.jp

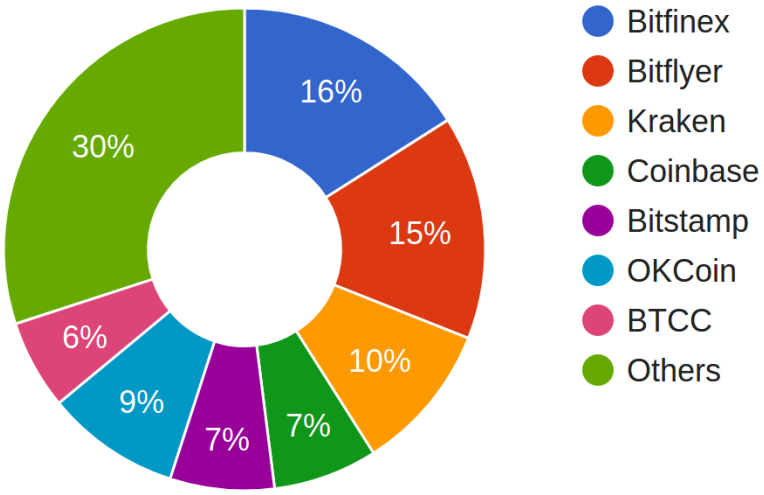

Figure 4.2: *Average market share between cryptocurrency exchange sites.*

To ensure high availability I deployed the tool to PaaS[2] provider, and I configured the environment to restart the virtual machine, or the tool if any problem occurs.

The application is an object oriented NodeJS based tool including 450 lines of code for extracting data from Kraken and Bitflinex. It saves the orderbook and the thicker information for every available crypto and classical currency pairs. The data is saved to comma separated values (CSV) files, each day to a different file. At this point it was not critical to store the data in a complex database, of course the migration is executable if it might be necessary in the future.

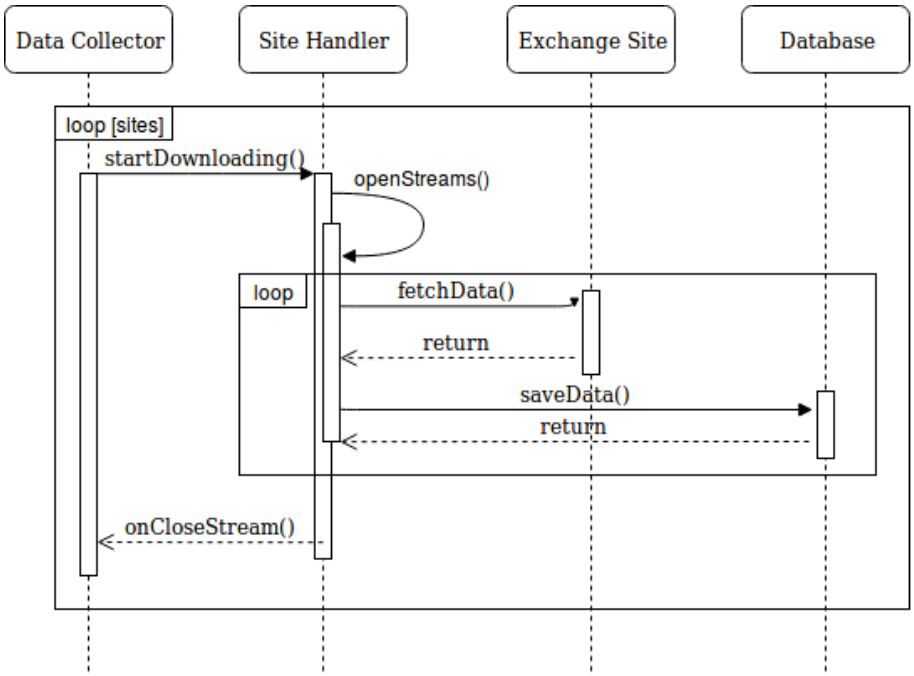

Figure 4.3: *Data collector sequence diagram.*

[2]Platform as a Service

The data collection process can be seen on figure 4.3. The tool is opening a different thread for each sites, than in these separated threads it starts a stream, the data source of the stream is a HTTP pooling to the target site. The returned object which is containing the important data then will be saved to the "database", in our case this database layer appends the data to a CSV file.

Data Preprocessing

Cryptocurrency market is not regulated, it is noisy and highly volatile, therefore a price prediction model seems too big task for the scope of this paper, if it is even possible. In classical stock exchanges moving average values are often used instead of raw price values to reduce the effect of high frequency noise.[5] Depending of the size of the moving average, the signal can keep short, and mid term trends.

Moving average is also functioning as a support, or resistance level[3]. Besides the expected performance increase, the signal keeps the original behavior.

The sampling rate of the original signal is six samples per minute, I resampled the data to 1 sample per minute frequency with Pandas[4] toolkit. The data is created from Bitcoin ask prices from 1st July 2017. to 20th October 2017. For the model I split the dataset into three part: train, test, and validation data, the sizes are 60%-20%-20%. The size of the training data period is kept rather small to be able to deal with the large number of Tweets later. The goal of this paper is to show weak coherence and not to build a complex predictive model.

To preprocess the data I created a library in Python according to object oriented principles, to ensure the robustness, flexibility and reusability of the code. To demonstrate the basic functionality I also made Jupyter[5] notebooks . The codebase for preprocessing is around 300 lines of code.

4.1.2 Model

The goal of the model is to predict the bitcoin ask price in a given timestamp from historical ask prices (input of the model), the number of input prices is 60, so the model predicts the price value from the activity in the previous hour, the output is the predicted price value.

The model's architecture (see on figure 4.4) is simple: recurrent neural network based solution with a fully connected part at the end.

[3]Support level acts like a floor for the signal, and the resistance acts like an upper level for the raw signal.[5]

[4]Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

[5]Open sourced web-based document sharing system that allows live code in the documents.

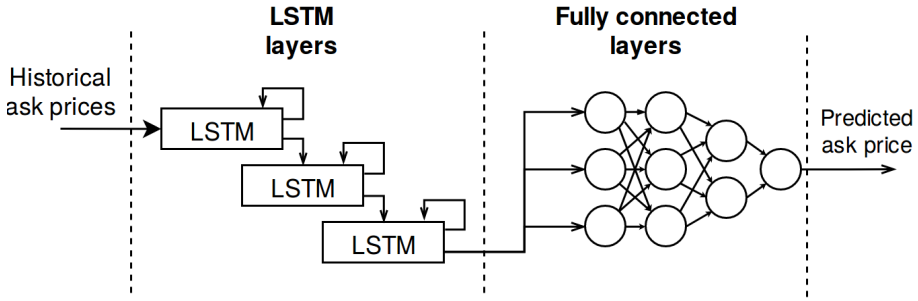

Figure 4.4: *Cryptocurrency Price Model.*

The first part of the model, the recurrent neural net, consists of three LSTM layers, which are initialized with Xavier's method[14]. I chose this initializer method over null, or random (Gaussian) initializer, because I want same variance after each layer, so the grandient will have a better convergence throughout backpropagation algorithm, and we can reach better learning, and deeper architecture. For activation function I used sigmoid function. The number of cells in each LSTM layer was a hyperparameter.

The second part of the model is a fully connected network, where the goal is to abstract higher level of features from the LSTM's feature set for regression. For activation function leaky rectified unit (LeakyReLU) is used, so the model allows non zero gradient, if the neuron is not active. Simple rectified unit (ReLU) was also tested, but the model's performance is sightly better with LeakyReLU, and also we avoid the dead ReLU problem with this (where a neuron will not activate ever).[26] The size of each layer was a hyperparameter. For regularization I used dropout after dense layers.[32]

For the loss function I used mean squared error, for optimization algorithm I used the adaptive Adam[21] method, as these methods shows competitive results in various deep learning tasks.

4.1.3 Evaluation

I made experiments with 8,16,32,64 number of cells within each layers. After hyperparameter optimization 32 number of cells showed the best results. The model contains three dense layers, where the first layer contains 512, the second 256, and the output layer contains 1 neuron, these are results of the hyperparameter optimization. Besides hyperparameter optimization I tested the model with different moving averaged input, where the parameter was the size of the window, the results are in the table 4.1. The best performing model has widow width of 30, I used this as a baseline model.

Table 4.1: *Baseline MSE results with different moving average window size*

Model \Window width	1	30	60	120
Simple Time Series Model	0.1573	0.044493	0.0994	0.1553

4.2 Twitter Sentiment Model

To prove the that deep learning is usable for weak coherence analysis I build a model which is intended to support price prediction in the cryptocurrency market. In this section I show how I collected and created the dataset, and I introduce the model which utilizes raw textual data (without further aggregated sentiment score from preprocessing).

4.2.1 Textual Data

Data Collection

There are two common ways to collect data from Twitter automatically: the original application interface provided by Twitter, or a programmed scarper. To understand which one is better for different applications I will introduce this two method in this section. The original application interface have three main types called Search API, Streaming API, and Firehose, and there is an additional method, where we downloading HTML pages, and automatically extracting data from it (called scarper).

Search API is the most simple one, this is a simple REST[6] interface where we can build a query and retrieve information. The only, and main drawback is the limitation of the maximal request in a given time slot. There are two initial buckets available for GET requests: 15 calls every 15 minutes, and 180 calls every 15 minutes.

This method is good for application, where we want to make single searches, read user profile information, or post Tweets.

Streaming API provides a low latency access to Twitter's global stream of data. Because it's stream based, there is no overhead associated with pooling a REST endpoint. Twitter provides different types of streams, the suitable stream for data mining is the public streams, which is a data stream contains the public data flowing through Twitter. The main limitation in that the streaming API only access to 1% of tweets, given in a random order, so if I want to build a detailed data set with all of the existing tweet, I can't do it with Streaming API.

[6]Representational State Transfer

Firehose works like streaming API, the difference is that the Firehose API access to all available public post, but the user have to pay for this service. However it's worth to mention that the streaming API is enough for scientific applications[29], if the completeness is not a requirement from the system.

Scarper The second main method to retrieve data from Twitter is to use a so called scarper. Scarpers are specific programs which can extract data from HTML pages according to predefined rules.

I used the Twitter's advanced search webpage as a source, and nodeJS as a programming environment, because JavaScript is the most suitable language to handle HTML pages, and HTTP requests. An another requirement was that the scarper app have to work asynchronously, like a stream. In a nodeJS it's easy to implement an lightweight scarper with nodemon module, and streams. My scarper is a command line tool, the input is the search query, the output is the retrieved tweets from the HTML page saved to a csv[7] file. For example the example command in list 4 on page 25 we will retrieve all tweets from user with displayname "Tensorflow".

```
1 npm start q=<search query> start=<start time> end=<end time>
2 // Example
3 npm start q=from:Tensoflow start=<2015-11-06> end=<2017-02-11>
```

list 4.1: *Scarper usage*

This tool is a 120 line length simple application, but because it's stream based, the downloading can be executed in many threads, which speeds up the downloading time. I used this tool on a PaaS provided virtual machine where the network bandwidth is not a bottleneck and I'm successfully downloaded all of the tweets for a simple search in a month (zirca one hundred thousand tweets per day) under twenty minutes.

Preprocessing

The number of tweets during a simple day is noteworthly. Just for a simple search with one query word can give more than 100 thousand tweets in a 24 hour interval [8].

In the beginning of studying cryptocurrency related tweets my purpose was to get an overall knowledge about the corpus, therefore splitted the tweets into words and transformed to GloVe pretrained vectors. I collected the unique words and with t-SNE[35] dimension reduction algorithm I transformed the 100D embedding to a 2-dimensional vector for each word embedding. I used 3 corpus: randomly downloaded tweets, tweets that have

[7] Command separated values.
[8] "Bitcoin" as query resulted averaged 100k tweets/day in the last 4 month.

been downloaded with search query "Ethereum" and tweets that have been downloaded with "Bitcoin" search query.

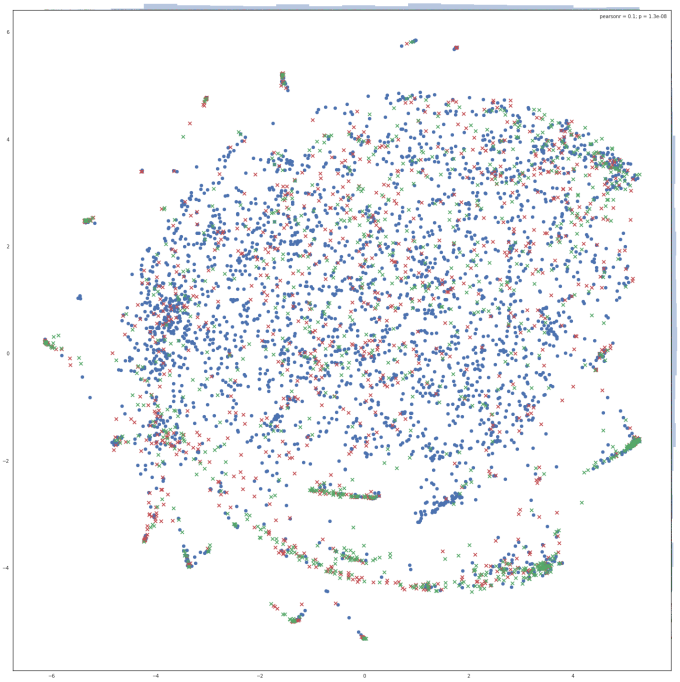

Figure 4.5: *Tweet corpus. The blue dot is the random corpus, the red cross is the Etherum corpus, and the green cross is the Bitcoin corpus.*

I plotted the results of the T-SNE algorithm and it is clearly visible that the three corpus share similar word usage, so during my future work I can use GloVe vectors.

Because the dataset is rather huge, it is also challenging as a big data problem, and not only as a machine learning / deep learning problem. To be able to effectively feed it with a deep learning system the data must be preprocessed.

According to section 3, there are several methods to process sentences, words, or texts, and transform it to a numerical representation. The best performed methods are based on tree structures: SPINN, and the convolutional network based architectures. SPINN is using pre-trained vectors for sentence modeling, however here I want to model not only sentences, but higher level of features based on tweets, therefore I used convolutional neural networks (CNN).

CNNs are used most for image processing, but it's also suitable for data processing, where the input is converted for an image like representation, for example word based CNN can extract sentence or word level features from the input texts.

However, the existing methods in cryptocurrency market analysis acts like a guideline. Because the dataset is huge, we have two choices:

- we can keep the timestamps and the order of the tweets, and with a recurrent archi-

tecture we can extract more features from the input,

- or we can throw away the information about the time distribution, and aggregate the data.

If we decided to go with the first solution, the model can be too deep, therefore (especially because the model is recurrent network) the speed of the learning will be low, and the model is more sensitive for noise However if we pick the second method, we will lose a lots of information, and it's questionable how can we split the dataset for aggregation. It's clear that the best solution will be to mix these ideas.

I have two proposal for this problem, I made a 2-dimensional tweet map. First I collected tweets with timestamps from Twitter for a query word. In this work I downloaded all the posts which contains "Bitcoin" in the last ten days.

Then arranged these tweets according to their timestamp and resampled them to hours, so I created a 2-dimensional list (figure 4.6), where the first dimension is the timestamp (with hourly resolution), and the second are the tweets. The maximal number of hourly tweets, and the number of hours - or in other words - the size of the map is a hyperparameter.

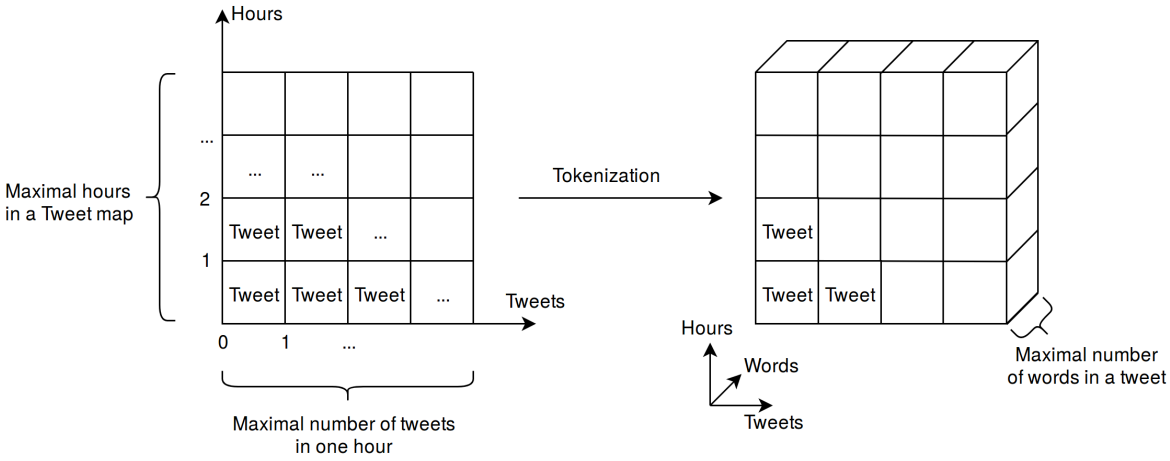

Figure 4.6: *Arranging tweet to hours, then tokenize the tweets. This method produces a 3D map.*

There are still texts in each cell in the map. I used classical natural language toolkit to remove stop words[9], the list of the stopwords are modified to ensure that I preserve the meaning of the tweet.[2] I also replaced URLs to the following token "url", because the full URL path is not interesting at this point, but I want to keep the information that the post contains a web link. I measured the averaged size after this process for bitcoin corpus, the results can be seen in figure 4.7. According to the result of this measurement I maximized the length of the tweet to 20.

To use this map for deep learning, I had to convert this data to a vector representation.

[9]Stop words are the most common words in a language, containing less information.

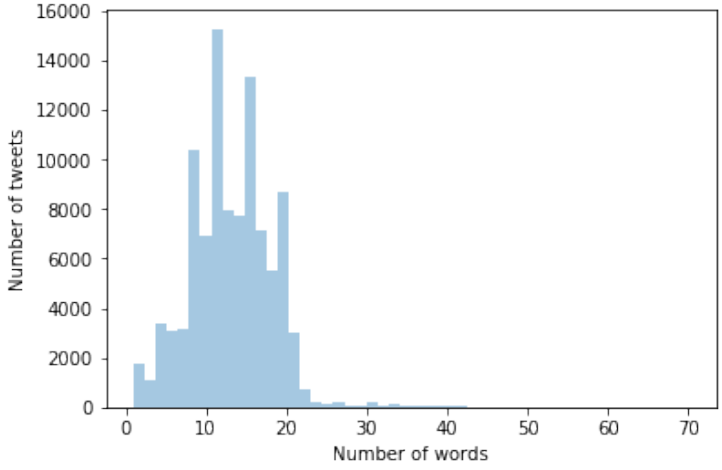

Figure 4.7: *Distribution of number of words in a tweet.*

For this purpose I used pre-trained GloVe vectors (trained on tweets), and after tokenization, I replaced the words to vectors. Now one tweet vector has two dimensions, one is the used embeddings, and the second one is the words in the tweet, so the resulting map has 3 dimensions now with depth size of the embeddings. To be able to use at most 2D convolution, I need to reduce this dimensionality (the computational demand of 3D convolution is too high).

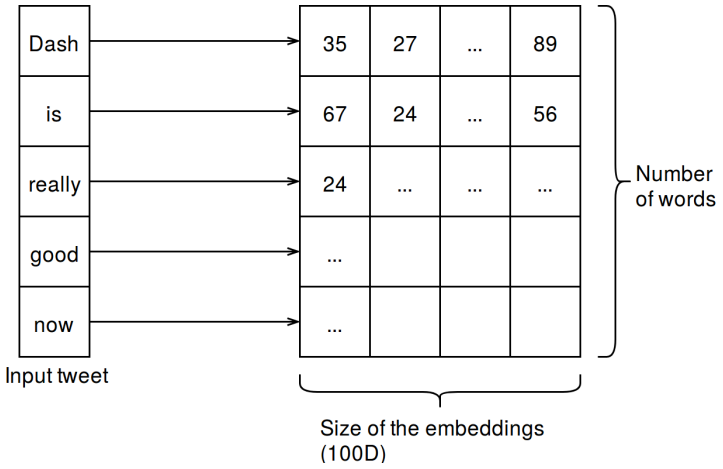

Figure 4.8: *Word to vector operation makes the tweet to a two dimensional array, which is too much for the proposed model. The shape of one tweet is equal to the (number of words, embedding size). If I replace the tweets in the map with this vectors I'll get shape of (number of hours, number of tweets, number of words, embedding size) which means four dimensions. This must be reduced to three.*

To solve this dimensionality problem, I trained an autoencoder (figure 4.9). The auto encoder is a data compression / feature learner algorithm, where the compression, and the decompression is lossy, data-specific, and it is learning automatically from examples in an unsupervised manner. Because this solution is data-specific, which means that they will only be able to compress data similar to what they have been trained on, it's needed to retrain the model in an informational system of novel domains for better performance.

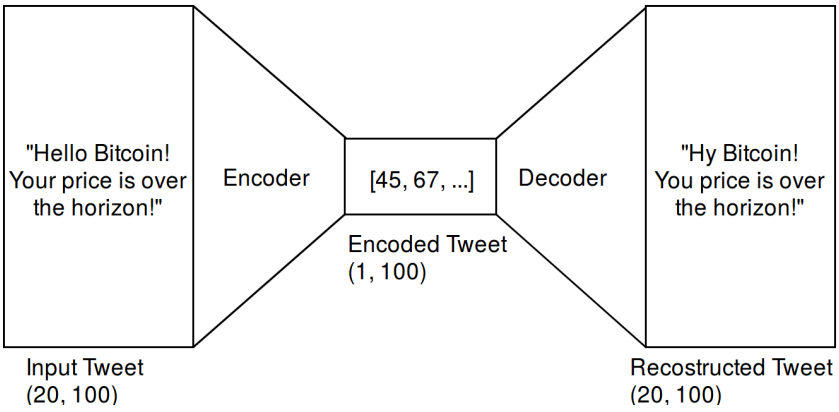

Figure 4.9: *Autoencoder. The inputs are vectors constructed with GloVe vectors from tweets, the output is the recostructed lossy tweet. The input, and the output are two dimensional with shape of (20, 100), but the encoded tweet is one dimensional with lenght of 100.*

Autoencoders consist of two parts: an encoder, and a decoder. The encoder will lower the dimensionality of the input tweet, and the decoder part is needed for the training in our case. The input, and the output of the autoencoder will be the same, the goal of the model is to compress the input, and decompress it to reach the same representation. The learned representation can be found at the bottleneck of the autoencoder.

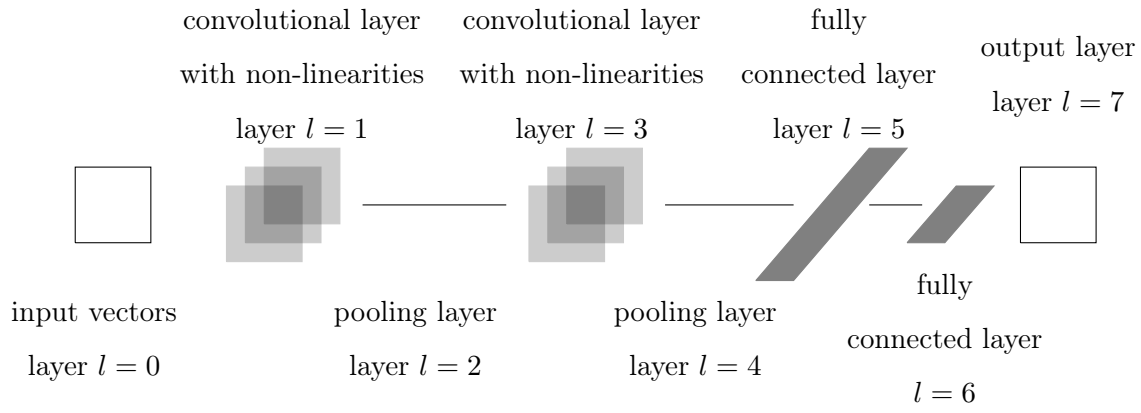

Figure 4.10: *Encoder. The first layer is a convolutional layer with 16 filters, which is sub-sampled with a pooling layer (window size: width=4, height=1), the second layer is a convolutional layer again with 16 filters, sub-sampled with a 4x1 pooling layer. The CNN is connected to a fully connected network, with two hidden layers: the first dense layer contains 256 neurons, where the second contains 126 neurons. For regularization the model uses dropout with 50% probability.*

My autoencoder is a CNN based network, where the encoder consists of convolutional, and pooling layers, with a fully connected part, and the encoder consists deconvolutional, and upsampling layers. The autoencoder is trained on 200 thousands of tweets, and the encoder (on figure 4.10 part is used for compressing a simple tweet to one 1D vector. The output of the encoder is a 1D vector with the same length as the used word embedding

size (in our case 100D).

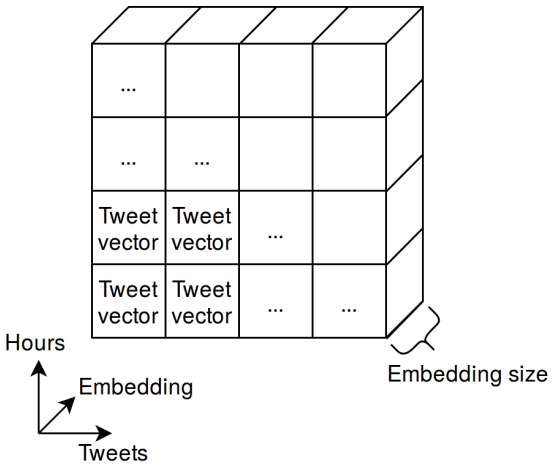

Figure 4.11: *Structure of the 2-dimensional tweet map*

With the encoder, the map has exactly 2 dimensions (on figure 4.11), with 100 depth (depth is the same number of channels in the input data, in RGB images, this value is equal with three: red, blue, and green channel).

As it turns out, the building of this preprocessing, and the 2D map creation was a time consuming task: on a Intel(R) Core(TM) i5-7500 CPU (on 3,4GHz) the preprocessing lasted for 31 hours, and it created 78,4 GB of data for one month[10]. Of course if I wanted to modify a hyperparameter related to the input data, I need to rerun this preprocessing.

Besides the drawback of the time consuming preprocessing, I have a promisingly better representation of the tweets. One of the question is, that how good my representation for this task. Can it preserve enough information for weak coherence abstraction, and price prediction?

4.2.2 Model

After preparing the 2D map I created the neural network. The architecture contains two 2D convolutional layers, and a fully connected part, see on figure 4.12. The output was a moving averaged price value, the input was a 2D map, where the size of the map is a hyperparameter.

[10]The size of the dataset is equal with the size of the saved .npy files.

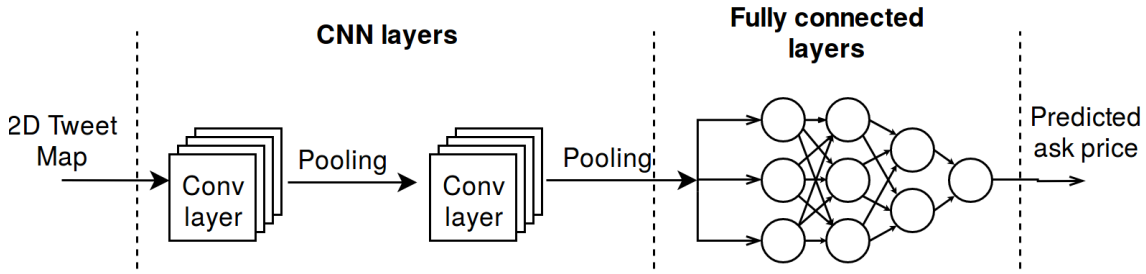

Figure 4.12: *Twitter sentiment model.*

The convolutional part contains two stacked convolutional, and pooling layer. The width of convolutional window is equal with the total number of tweets in a hour (aka the width of the 2D map), and the height is a hyperparameter. The number of filters is a hyperparameter. I used sigmoid activation function, and I used zero-padding to preserve the size of the map. The pooling layer's window size is a hyper parameter.

The fully connected part uses ReLU activation, and dropout for regularization. For the loss function I used mean squared error, and for optimization the model uses Adam algorithm.

4.2.3 Evaluation

The results shows (in table 4.2) that this model can't predict the price values just from the tweets.

Table 4.2: *NLP model results with different hyperparameters*

	20 tweets/hour	100 tweets/hour
MSE	0.723	0.927

I made a experiments with 20, and 100 tweets per hour (another hyperparameters is optimized with an algorithm, I will write about it later), and it turned out that the model's performance is not enough for this task. It doesn't mean that the idea is totally wrong, maybe the error was the shallow architecture, or the lack of classical preprocessing. An another explanation can be that the model get too few tweets as input, and it can't abstract any features from it.

My true question is: Can this representation, and this model enhance the results of the time series model?

4.3 Ensemble Model

The core idea is to use the overall social media sentiment to enhance predictions in crypto market, or in other words: use the weak coherence between these two signals.

4.3.1 Data

Because the combined model consists of the previously showed time series, and natural language model, the input data, and the output are similar. The model has two different inputs, one for the historical time series data, and one for the 2D tweet map. The goal is to predict the price of the currency, so the ooutput is the predicted price value.

The dataset consists ask prices, and tweets from 1. July 2017. to 20. October 2017.

4.3.2 Model

This combined modes shares similarities with previous models, the first parts are the same, but the top layers, the fully connected part is modified. I cut off the last fully connected layers, and amend these two part with a new fully connected part, as you can see in the figure **??**..

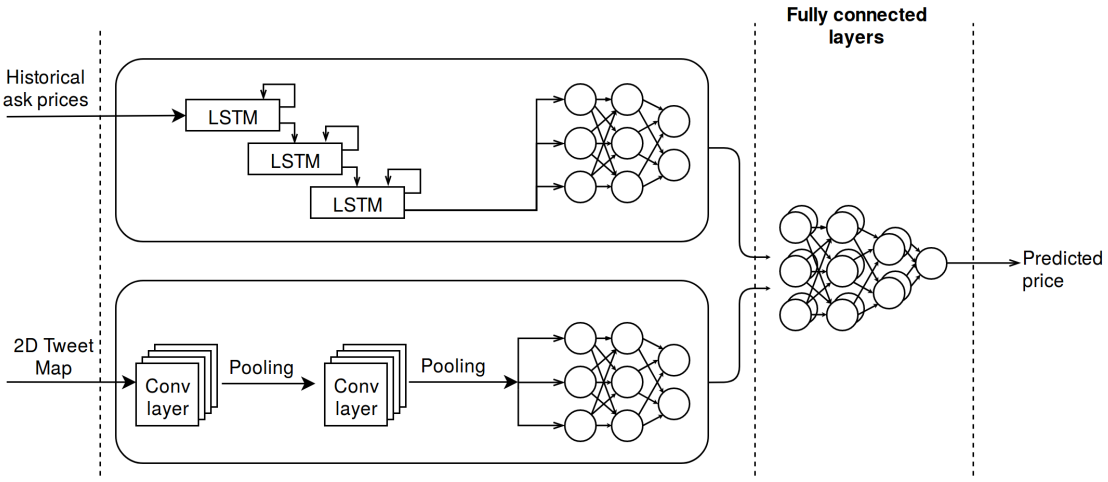

Figure 4.13: *Ensemble model structure.*

The model is compiled again with Adam optimizer, and the loss function is mean squared error. The weights in the twitter sentiment part, and in the crypto price part is not random initialized. The weights are imported from previous models. With this transform learning approach[37][31] I wanted to improve the learning of my model.

Table 4.3: *Ensemble model results. The baseline model earned 0.044493 with MSE loss function. Smaller MSE results means better predictive performance.*

Tweets / hour	20	40	100	200
MSE	0.045514	0.044272	0.04293	0.04289
Results	+0.047%	-0.5%	-3.5%	-3.6%

4.3.3 Evaluation

The main hyperparameter was the number of the tweets in one hour (or with other words, the width of the 2D map). The results are in the table 4.3, the best performing model uses 200 tweets per hour, and 24 hours in the input, results 4800 tweet in a map which is the 4% of the averaged daily tweets. The models with smaller input maps have worse performance, because in this case the natural language input is to small for a good representation and it is likely to be just noise for the system.

The earned 3.6% performance growth is showing that there is a weak coherence between these signals, and this abstraction can enhance the price prediction.

The size of model, training logic, and the preprocessing code is equal with cca. 800 lines of code. I used university resources, and virtual machines from a cloud provider so I can speed up the training time.

4.3.4 3D Model

After the 2D Twitter map, I also created a 3D map (on figure 4.14), which is consists of stacked 2D map. As it can be seen on figure 4.11), the 2D map has a shape of (hours, tweets, embeddings)[11], so if I'm stacking these layers I'll get the shape (layers, hours, tweets, embeddings).

[11]Where the structure of the shape is the following:(first dimension, second dimension, depth).

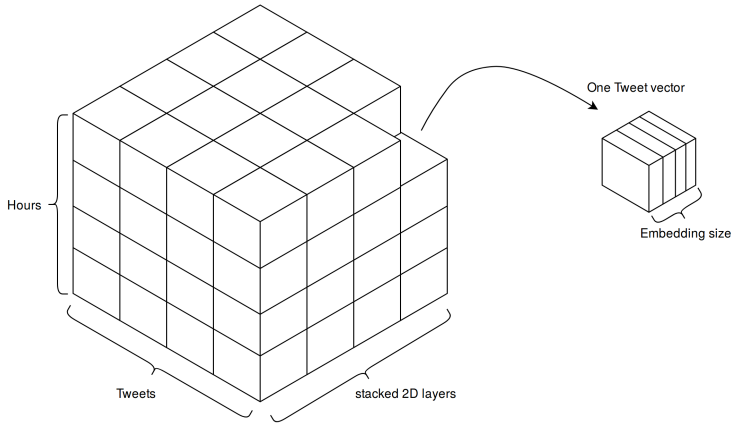

Figure 4.14: *Structure of the 3D Twitter map. It's hard to visualize the 3 dimensional map with a depth value (practically it is 4-dimensional), therefore on the figure first I illustrate the first three dimension, then I grab one cube from the map, and show that it has a depth value equal with the used embedding size.*

The deep learning structure for this 3D map is the same as before but instead of 2D convolution I used 3D convolution. With hyperparameters: 100 tweet per hour, 24 hours in one 2D layer, and 5 stacked layer.

The model is promising, but the results showed that this method is not enough for this task. The model's performance is more than 200% worse than the baseline model (loss value is equal with 0,0127) , which suggests that I only introduced noise to the system.

4.4 Optimization

The model has a plenty of hyperparameters. Number of neurons in a dense layer, size of the input map, convolutional window size, pooling window size, activation functions, optimizers, etc. To find the best performing model, I need to optimize this parameters, because manually this is time consuming, I used a third party library for it, called Hyperas.

I build the architecture, but for each hyperparameter I used the Hyperas[12] directive, then when I started a hyperas task, the library change the parameters according to the given list, and run the model, after evaluation, it will change some parameters, and run the training again on the new model. Hyperas has two output, the best model, and the evaluation results. Hyperas is using tree-structured parzen estimator[4] for optimization. This library can speed up model building, so it's recommended to use for hyperparameter optimization.

[12]https://github.com/maxpumperla/hyperas

4.5 Software Stack

For data collection I used NodeJS and Javascript.

For data preprocessing I used Pandas[13] and Numpy[14] Python libraries.

For deep learning implementations I used Keras[15] with Tensorflow[16] backend. These libraries are written in Python.

[13] http://pandas.pydata.org/
[14] http://www.numpy.org/
[15] https://keras.io/
[16] https://www.tensorflow.org/

Chapter 5

Conclusions

During my work I reviewed the related literature about deep learning models in general and in financial analysis. The most popular approach was recurrent architectures to model the behavior of the stock market. However, I made experiments on cryptocurrency exchanges, therefore I need to adopt these techniques for a slightly different dataset. The dataset is also created by me, not even just the dataset, but the underlying tools are created during my work to collect, and filter those informations.

After data collection I created a baseline model for price prediction with LSTM network. For weak coherence analysis I proposed a hybrid model with LSTM based part for time series analysis, and a convolutional network based part for extracting features from tweets, and auto-encoder to transform tweets to vectors. The hybrid model can abstract the weak coherence between the two signal. I successfully enhance the performance of the baseline model with natural language modelling.

5.1 Future Work

The goal of my work was to use deep learning for weak coherence abstraction, and during this time I also have a chance to work with interesting datasets. This cryptocurrencies, and the blockchain technology got my attention. The behavior of cryptocurrency markets is very novel and has a lot more to investigatestrange.

In the next steps I want to optimize the existing models, and build deeper architectures, but besides that I want to enhance the input data with tweet popularity, tweet popularity and frequency, exchange market volumes, bid prices.

Now I'm working on a regression problem in the deep learning model, but it's not the only approach to analyse these currencies. Also a promising, and valuable outcome if we

predict, that the price will grow or decrease (classification problem).

The behavior of crypto markets is strange. To understand how it is working it's a good idea to detect the influencers behind it, and analyse the extracted data.

Chapter 6

Acknowledgments

I would like to thank my thesis advisor Dr. Gyires-Tóth Bálint Pál of the Department of Telecommunications and Media Informatics at Budapest University of Technology and Economics. He consistently allowed this paper to be my own work, but steered me in the right direction whenever he thought I needed it. He supported me greatly and were always willing to help me even at late hours or at weekends. I wish to express my sicere thanks for providing me all the necessary resources for my work.

This project was partly supported by EFOP-3.6.2-16-2017-00013.

Bibliography

[1] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, and Gerald Penn. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4277–4280. IEEE, 2012.

[2] Alexandra Balahur, Erik van der Goot, Piek Vossen, and Andres Montoyo. Proceedings of the 7th workshop on computational approaches to subjectivity, sentiment and social media analysis. In *Proceedings of the 7th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, 2016.

[3] Pouya Bashivan, Irina Rish, Mohammed Yeasin, and Noel Codella. Learning representations from EEG with deep recurrent-convolutional neural networks. *CoRR*, abs/1511.06448, 2015.

[4] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[5] Zvi Bodie. *Investments*. McGraw-Hill, 2013.

[6] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016.

[7] Johan Bollen, Huina Mao, and Xiao-Jun Zeng. Twitter mood predicts the stock market. *CoRR*, abs/1010.3003, 2010.

[8] Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. *CoRR*, abs/1603.06021, 2016.

[9] Grégoire Burel, Hassan Saif, Miriam Fernandez, and Harith Alani. On semantics and deep learning for event detection in crisis situations. 2017.

[10] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[11] Eunsuk Chong, Chulwoo Han, and Frank C Park. Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies. *Expert Systems with Applications*, 83:187–205, 2017.

[12] Aaron Elliot, Cheng Hua Hsu, and Jennifer Slodoba. Time series prediction: Predicting stock price. *arXiv preprint arXiv:1710.05751*, 2017.

[13] John Cristian Borges Gamboa. Deep learning for time-series analysis. *arXiv preprint arXiv:1701.01887*, 2017.

[14] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[15] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013.

[16] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 2017.

[17] J. B. Heaton, N. G. Polson, and J. H. Witte. Deep learning in finance. *CoRR*, abs/1602.06561, 2016.

[18] Garrick Hileman and Michel Rauchs. Global cryptocurrency benchmarking study. *Cambridge Centre for Alternative Finance*, 2017.

[19] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[24] Yifan Liu, Zengchang Qin, Pengyu Li, and Tao Wan. Stock volatility prediction using recurrent neural networks with sentiment analysis. *CoRR*, abs/1705.02447, 2017.

[25] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. Traffic flow prediction with big data: a deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):865–873, 2015.

[26] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[27] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[28] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[29] Fred Morstatter, Jürgen Pfeffer, Huan Liu, and Kathleen M. Carley. Is the sample good enough? comparing data from twitter's streaming API with twitter's firehose. *CoRR*, abs/1306.5204, 2013.

[30] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[31] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, abs/1403.6382, 2014.

[32] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[33] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *CoRR*, abs/1503.00075, 2015.

[34] Duyu Tang, Furu Wei, Bing Qin, Ting Liu, and Ming Zhou. Coooolll: A deep learning system for twitter sentiment classification. In *SemEval@ COLING*, pages 208–212, 2014.

[35] Laurens van der Maaten and Geoffrey E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[36] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015.

[37] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.

[38] G Peter Zhang and Min Qi. Neural network forecasting for seasonal and trend time series. *European journal of operational research*, 160(2):501–514, 2005.

[39] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626, 2015.

[40] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1-4):43–52, 2010.