Masaryk University
Faculty of Informatics

Prediction of Financial Markets Using Deep Learning

Bachelor's Thesis

Jiří Vahala

Brno, Spring 2016

Masaryk University
Faculty of Informatics

Prediction of Financial Markets Using Deep Learning

Bachelor's Thesis

Jiří Vahala

Brno, Spring 2016

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Jiří Vahala

Advisor: doc. RNDr. Tomáš Brázdil, Ph.D.

Acknowledgement

I would like to thank my supervisor doc. RNDr. Tomáš Brázdil, Ph.D. for all provided support and for leading my research very closely. I would also like to thank industrial partner Michal Kreslík, who gave me this great opportunity. The biggest thanks go to Katarína Kejstová, my girlfriend who supported me during writing this thesis.

Abstract

We tested possible usage of fully connected neural networks for predicting FOREX market. More specifically we focused on generalization providing methods used in neural networks. In order to facilitate an easy grid search over the target domain, the general framework around TensorFlow and Keras framework was built to provide flexibility for executing automatically generated experiments. Because TensorFlow and Keras do not implement autoencoders for pretraining, four implementations of different autoencoders were created and successfully merged into official TensorFlow repository. We also introduce an algorithm for finding threshold with chosen confidence filter since predictors does not have to make predictions for all inputs and confidence filtering marginally increases their success. Finally, we proposed few models and deployed them into the real trading platform where they were marked as successful predictors.

Keywords

Neural networks, deep learning, prediction, FOREX, time series, TensorFlow, prediction confidence

Contents

List of Tables

List of Figures

1 Introduction

Artificial neural networks [10] have passed from their creation in the fifties through several waves of interest and decay which lead to overfunding or on the other side extinction in practical usage. The main motivation behind neural networks attempts to simulate an animal brain. Neural networks are basically simplified models of neurons and their mutual connection, which can be adjusted by using various algorithms to obtain a desired result for a specific task. Mostly such tasks can be solved by minimization of error function based on the problem. The biggest issue of neural networks was always their speed and ability to learn using gradient descent [10]. Because of it, the neural networks were pushed by other machine learning methods into the background. However in 2006, an algorithm for fast training of neural networks with more hidden layers was invented. Its basic idea is based on smart initialization of connections between neurons using restricted boltzmann machines [10]. Thanks to this smart initialization neural networks experience a new renaissance.

Adaptation of connections between neurons in neural networks is computationally very difficult problem and on usual hardware it, might take over days. That is why new frameworks which gain from graphic cards's performance with CUDA (Compute unified device architecture) technology were lately introduced. Because all modern games are based on vectorized graphics, graphic cards tend to be faster in vectorized calculations and CUDA technology comes with tools providing such performance for programmers. TensorFlow is one of the frameworks which uses CUDA and provides tools for effective defining, optimization and evaluation of mathematical tensor based calculations including multidimensional vectors and realizing automatic differentiation on top of them. Also, automatically perform these calculations in parallel on classical CPU architectures but also on GPU architectures.

More often implementations of neural networks are used in advanced applications such as image recognition, diseases diagnostics or autonomous driving cars. Another possible application is trading on online markets, where neural networks can replace trader in making decisions and trading strategy. The goal of this thesis is research and

development of training algorithms of neural networks for predicting financial time series where speed and precision of predictions are crucial for success.

For purposes of this thesis an experimental environment above TensorFlow and Keras frameworks was developed for implementation of data processing, deep learning architectures and evaluation of results. We introduce concepts which filters some predictions of predictor out in order to increase the precision.

2 Time series Forecasting

Time series are sequences of data-points observed in a behavior of some process P. We can define sequence as generated by process P as:

$$\mathbf{x}^{t+1} = P(t) \tag{2.1}$$

where t is time parameter of process P. P has its own hidden state which cannot be observed.

The data vector $\mathbf{x}^{t+1}$ is treated as random vector. The time series can be continuous or discrete. In case of continuous series, observations are taken in each time instance [5]. In discrete case observations are taken at discrete, not necessarily following(time continuum is retained), points in time.

2.1 Forecasting model

Time series forecasting is finding a model based on observations of specified sequence and then applying calculated model on that sequence to predict its future values with minimum error if possible.

There are two basic linear models proposed in literature Autoregressive and Moving Average [5] and its combinations and modifications like Autoregressive Moving Average (ARMA), Autoregressive Integrated Moving Average (ARIMA), Autoregressive Fractionally Integrated Moving Average, which generalizes ARMA, and ARIMA models. There are also many other models which extend these models in specific, domain based, problems.

Since linear models are easy to understand they tend to be often implemented, however many practical real-world time series show non-linear patterns. For such time series there have been suggested non-linear models such Autoregresive Conditional Heteroskedasticity (ARCH) which is generalized by Generalized ARCH (GARCH). More proposed models, linear and non-linear, can be found at [5].

To predict process P we need to define sequence of last k vectors:

$$\mathbf{x}^{t},\ \ldots, \mathbf{x}^{t-k} \tag{2.2}$$

at time t, which we call lookback. Lookback is cosnidered as real-valued vector and it is composition of k last sampled data-vectors.

The goal is to find model M which minimizes error E:

$$E = \sum_{t=t_0}^{t^*} E_t \tag{2.3}$$

where:

$$E_t(M(\mathbf{x}^t, \ldots, \mathbf{x}^{t-k}), P(t)) \tag{2.4}$$

where $E_t : (\mathbb{R}^n, \mathbb{R}^n) \to \mathbb{R}^n$ is error for prediction at time t.

In this thesis, we approximate model M using neural network.

3 Neural networks

Artificial neural networks (NNs) are machine-learning formalism inspired by the animal brain with basic structural and functional unit - artificial neuron. The idea is to reverse-engineer how animals learn to perform some tasks. Despite the fact that NNs are inspired by an animal brain, they are far from real biological neural networks. The basic unit of the model is single neuron which represents some linear mathematical function. It is easy to see that NN lacks a lot of details and it is very simplified. Closer models to biological brains than NNs are so called spiking neural networks which simulate neurons as pulses of energy[9].

There are multiple types of neural networks and they differ in tasks which they can perform and also in their training algorithms. All models are based on fundamental units - neurons, which are mathematically described as operation:

$$y = \sigma(\mathbf{w}\mathbf{x} + b) \tag{3.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is input vector, $\mathbf{w} \in \mathbb{R}^n$ is weight vector, $b \in \mathbb{R}$ is bias and $\sigma : \mathbb{R} \to \mathbb{R}$ is called activation function.

3.1 Fully connected neural network model

Consider matrix $\mathbf{W} \in \mathbb{R}^{n \times m}$ and vector $\mathbf{b} \in \mathbb{R}^m$. Furthermore let us extend σ to be defined for vector argument as:

$$\sigma((a_1, a_2, ..., a_n)) = (\sigma(a_1), \sigma(a_2), ..., \sigma(a_n)) \tag{3.2}$$

then we can define simple transformation $\mathbb{R}^n \to \mathbb{R}^m$ as:

$$\mathbf{y} = \sigma\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right) \tag{3.3}$$

If σ is identity function we have linear transformation. For non-linear σ we get non-linear model. We denote such functionality as l-th layer which we use as a building block for layered models:

$$\mathbf{y}^{(l)} = \sigma^{(l)}\left(\mathbf{W}^{(l)}\mathbf{y}^{(l-1)} + \mathbf{b}^{(l)}\right) \tag{3.4}$$

Let us define fully connected neural network model with L layers. Call $\mathbf{W}^{(0)}, \ldots, \mathbf{W}^{(L)}$ as weights and $\mathbf{b}^{(0)}, \ldots, \mathbf{b}^{(L)}$ as biases where both, biases and weights, are parameters of model, furthermore let us call $\sigma^{(0)}, \ldots, \sigma^{(L)}$ activation functions, then:

$$\mathbf{y}^{(0)} = \sigma^{(0)}\left(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right) \tag{3.5}$$

$$\mathbf{y}^{(l)} = \sigma^{(l)}\left(\mathbf{W}^{(l)}\mathbf{y}^{(l-1)} + \mathbf{b}^{(l)}\right) \tag{3.6}$$

is a function $T_N(\mathbf{x}) = \mathbf{y}^{(l)}$ represented by neural network and is considered as transformation $T_N : \mathbb{R}^n \to \mathbb{R}^m$. It is common to mark topology as list of numbers, where each number represent count of numbers in one layer, so for example neural network with 100 input neurons, 10 hidden neurons and 20 output neurons can be marked as [100-10-20].

It has been shown that fully connected network with sigmoidal activation functions and two hidden layers can approximate any continuous function on subset of $\mathbb{R}$ [16]. Fully connected networks are the simplest models of neural networks which are used today for performing classification and regression tasks, yet they perform well.

3.1.1 Activation function

For many decades, neural networks were using strictly sigmoidal functions with typical examples of sigmoid or hyperbolic tangent. Sigmoid function:

$$y = \frac{1}{1 + e^{-x}} \tag{3.7}$$

Hyperbolic tangent:

$$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.8}$$

Problem with sigmoidal functions is that they relatively quickly saturate in its limit values. It was shown in [1] that if we take infinite number of copies of one neuron with shared weights $\mathbf{w}$ and bias b and activation sigmoid function, but each copy gets an artificial offset from $-0.5, -1.5, -2.5...$ sequence, then the sum of all such neurons converges to value $ln(1 + e^x)$:

$$\sum_{i=1}^{N} \frac{1}{1 + x^{-(x-i+0.5)}} \approx ln(1 + e^x) \tag{3.9}$$

This function is called softplus and it does not suffer from vanishing gradient problem. Softplus is usually approximated by so called rectified linear unit(ReLU):

$$y = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{3.10}$$

Empirical experiments show that ReLU performs better than softplus [10]. Many other modifications of ReLU exists, such as Leaky ReLU, noisy ReLU, and others, but we do not consider them in this thesis.

3.2 Loss function

One of most the important concepts of neural networks is definition of loss(error) function which is being optimized. It is crucial to show to the NN how well it approximates desired function with given data and how it should adjust its parameters(weights and biases) for better performance in next step of learning. The choice of loss function strongly depends on application of NN with respect to given labels. With given data set

$$\mathcal{T} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ \ldots, (\mathbf{x}_p, \mathbf{y}_p)\} \tag{3.11}$$

where p is number of pairs and $\mathbf{x}_k \in \mathbb{R}^n$, $\mathbf{y}_k \in \mathbb{R}^m$ are k-th input and output vectors of neural network respectively. Let $\mathbf{y'}_k = F_N(\mathbf{x}_k)$ and call it prediction.

Then we can define many different loss functions:

Mean squared error is one of the most common loss functions used.

$$MSE = \frac{1}{2n}\sum_{k=1}^{n}\sum_{i=1}^{|\mathbf{y}|}(y_{ik} - y'_{ik})^2 \tag{3.12}$$

Mean absolute error

$$MAE = \frac{1}{n}\sum_{k=1}^{n}\sum_{i=1}^{|\mathbf{y}|}|y_{ik} - y'_{ik}| \tag{3.13}$$

Mean absolute percentage error

$$MAPE = \frac{1}{n}\sum_{k=1}^{n}\sum_{i=1}^{|\mathbf{y}|}\left|\frac{y_{ik} - y'_{ik}}{y_{ik}}\right| \tag{3.14}$$

Binary cross-entropy

$$crossentropy = \sum_{i=1}^{n} -(y_i \log(y'_i) + (1 - y'_i)log(1 - y_i)) \tag{3.15}$$

Categorical cross-entropy

$$H = -\sum_{i=1}^{n} y_i log(y'_i) \tag{3.16}$$

3.2.1 Gradient

Minimizing the loss function leads to the desired model on given data. By adjusting neural network's parameters we can optimize the loss function. Since loss function is defined, we can differentiate it and perform gradient descent to find loss function optimum.

Gradient descent is a basic approach to optimizing functions. Gradient is defined for some function $F(\mathbf{x})$ as:

$$\nabla F(\mathbf{x}) = \frac{\partial F}{\partial \mathbf{x}} \tag{3.17}$$

In order to perform gradient descent for function defined by neural network, we can write gradient as:

$$\nabla E(\mathbf{w}) = \frac{\partial E}{\partial \mathbf{w}} \tag{3.18}$$

where $\mathbf{w}$ is considered as vector of parameters (weights and biases). To optimize neural network's parameters, we have to define update rule which adjusts parameters with respect to negative gradient:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Delta \mathbf{w}^{(t)} \tag{3.19}$$

where:

$$\Delta \mathbf{w}^{(t)} = -\epsilon^{(t)} \nabla E(\mathbf{w}^{(t)}) \tag{3.20}$$

where $\epsilon^{(t)}$ is parameter called learning rate at time t.

3.2.2 Gradient calculation

BackPropagation is name of an algorithm which is used for calculation of partial derivatives of $E(\mathbf{w})$ which propagates error calculated on the output of the network into hidden layers, layer by layer, so all partial derivatives are calculated and model can adjust parameters with respect of negative gradient of loss function[10].

Consider general loss function E. Denote w_{ij}^l is weight of connection between neuron i in layer l and neuron j in layer $l+1$. Also let b_i^l be bias for neuron i in layer l. Then derivation of loss function is:

$$\frac{\partial E}{\partial w_{ij}^l} = \sum_k \frac{\partial E_k}{\partial w_{ij}^l} \tag{3.21}$$

$$\frac{\partial E}{\partial b_i^l} = \sum_k \frac{\partial E_k}{\partial b_i^l} \tag{3.22}$$

Where E_k is error on k-th update and for each k applies:

$$\frac{\partial E_k}{\partial w_{ij}^{(l)}} = \frac{\partial E_k}{\partial y_{ik}^{(l)}} \sigma'^{(l)}(\mathbf{W}^{(l)} \mathbf{y}_k^{(l-1)} + \mathbf{b}^{(l)}) y_{jk}^{(l-1)} \tag{3.23}$$

$$\frac{\partial E_k}{\partial b_i^{(l)}} = \frac{\partial E_k}{\partial y_{ik}^{(l)}} \sigma'^{(l)}(\mathbf{W}^{(l)} \mathbf{y}_k^{(l-1)} + \mathbf{b}^{(l)}) \tag{3.24}$$

Where for all hidden neurons:

$$\frac{\partial E_k}{\partial y_{jk}^{(l)}} = \sum_r \frac{\partial E_k}{\partial y_{rk}^{(l+1)}} \sigma'^{(l)}(\mathbf{W}^{(l+1)}\mathbf{y}_k^{(l)} + \mathbf{b}^{(l+1)}) w_{rj} \tag{3.25}$$

Gradient descent is the basic approach used in neural networks, but it has many weaknesses like local minima. Other alternative techniques were proven to be more successful, i.e. conjugate gradient [15].

3.2.3 Optimizers

Optimizer is general concept of calculating $\Delta\mathbf{w}^{(t)}$. There are many ways how to calculate new update of weights. The simplest one, which we already defined, is gradient descent but due to its problem with getting stuck in local optima there have been invented methods alternating gradient descent.

Momentum method is based on gradient descent. Next step partially takes into account the previous step. This might help to avoid local extremes.

$$\Delta\mathbf{w}^{(t)} = -\epsilon^{(t)} \nabla E(\mathbf{w}^{(t)}) + \alpha^{(t)} \Delta\mathbf{w}^{(t-1)} \tag{3.26}$$

Where α is parameter called momentum at time t and $0 \leq \alpha < 1$ [11].

Nesterov's momentum method computes the gradient of the error function and then does the cumulative parameter update. At first are updated parameters according to direction of the old step, after which a new gradient is computed and then the real new step is computed using the new gradient [11].

$$\Delta\mathbf{w}^{(t)} = \alpha^{(t)} \Delta\mathbf{w}^{(t-1)} - \epsilon^{(t)} \frac{\partial E}{\partial(\mathbf{w}^{(t-1)} + \alpha^{(t)} \Delta\mathbf{w}^{(t-1)})} \tag{3.27}$$

Conjugate gradient improves basic gradient descent and respects gradient from the step at the time $(t-1)$. In basic gradient descent

direction of each new step is perpendicular to the direction of the previous step, which might be considered as inefficient [10].

Consider:

$$\Delta \mathbf{w}^{(t)} = \epsilon^{(t)} \mathbf{d}^{(t)} \tag{3.28}$$

where $\mathbf{d}^{(t)}$ is direction of step at time t. Then:

$$\mathbf{d}^{(t)} = -\nabla E(\mathbf{w}^{(t)}) + \beta \mathbf{d}^{(t-1)} \tag{3.29}$$

and β is parameter calculated as:

$$\beta = \frac{(\nabla E(\mathbf{w}^{(t)}) - \nabla E(\mathbf{w}^{(t-1)}))\nabla E(\mathbf{w}^{(t)})}{\nabla E(\mathbf{w}^{(t)})\nabla E(\mathbf{w}^{(t)})} \tag{3.30}$$

Rprop is a heuristic which is trying to balance sizes of steps between weights at NN by assigning learning rate to each weight.

$$\epsilon_{ij}^{(t)} = \begin{cases} \eta^{+}\epsilon_{ij}^{(t-1)} & \text{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \eta^{-}\epsilon_{ij}^{(t-1)} & \text{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ \epsilon_{ij}^{(t-1)} & \text{otherwise} \end{cases} \tag{3.31}$$

where $0 < \eta^{-} < 1 < \eta^{+}$, then:

$$\Delta w_{ij}^{(t)} = \begin{cases} -\epsilon_{ij}^{(t)} & \text{if } \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ +\epsilon_{ij}^{(t)} & \text{if } \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.32}$$

It was shown at [12] that Rprorp significantly reduces learning steps compared to original gradient descent algorithm.

There are other widely-used algorithms for optimization like RMSProp, Adagrad or Adam [10]

3.2.4 Batch

The parameters can be updated with different frequencies. There are three possibilities how to update parameters of NN:

1. Full-batch: the gradient is calculated on all provided data instances so the gradient calculation is statistically precise. Disadvantage of full-batch approach is poor performance.

2. Online: the gradient is calculated after each data instance provided to NN. If data instances are randomly picked from dataset then we call this approach stochastic gradient descent. Disadvantage of stochastic gradient descent is poor estimate of gradient for each weight update. On the other hand, thanks to inaccurate gradient descent, local extremes might be easily avoided.

3. Mini-batch: is a method combining online and full-batch approaches. The gradient is calculated from the stochastically picked subset of data instances which provides a better gradient estimate and also a better performance with slightly inaccurate gradient descent[13].

Chosen frequency of weights updates is crucial and significantly affects optimizing algorithms.

3.2.5 Dropout

Dropout is a technique where, with probability p, randomly selected neurons are switched off during training steps in order to improve generalization. Dropout causes that during each iteration of training selected subset of neurons is switched off leaving each training step different subnetwork functional. [3]

$$\mathbf{r}^{(l)} \sim \text{Bernoulli}(p) \tag{3.33}$$

$$\mathbf{y}^{(l+1)} = \sigma^{(l+1)}(\mathbf{W}^{(l+1)}(\mathbf{y}^{(l)}\mathbf{r}^{(l)}) + \mathbf{b}^{(l+1)}) \tag{3.34}$$

3.3 Initialization and autoencoders

The neural network is very sensitive to its initialization, which can strongly affects the progress and output of training process. There were proposed many different layers initializing algorithms in [25] which take into account number of neurons in each layer so the average

calculated potential of each neuron in hidden layers tends to be around zero at the begining of training. In recent years, the key of neural network research is so called deep-learning which was started with pre-training of networks. Pre-training is the initialization of weights and biases between two sets of neurons using weights and biases found with feature extractor. It was shown in [4] that pre-training used as initialization of each layer in the neural network can lead to better results than random initialization of weights. Such process is called greedy layer-wise pretraining with fine tuning. On the other hand there is an opinion that with activation functions which do not suffer from vanishing gradient signal pre-training networks is not necessary since weights from a greedily pre-trained network are not equal to weights from training all layers together.

3.3.1 Autoencoder

Autoencoder is a type of fully connected neural network where input has the same dimension as output $F_A : \mathbb{R}^n \to \mathbb{R}^n$ and error function is defined as:

$$MSE(\mathbf{x}_k, F_A(\mathbf{x}_k)) \tag{3.35}$$

Autoencoder can be trained with back-propagation in the same way as typical fully connected neural network defined earlier in this thesis. The key idea is to find a transformation which can encode information obtained from input in smaller number of dimensions with the strongest features and then reconstruct (decode) it back with minimal error [10].

Autoencoder as function F_A can be written more specifically as composition of two functions:

$$\mathbf{z} = \text{encoder}(\mathbf{x}) \tag{3.36}$$

$$\mathbf{y} = \text{decoder}(\mathbf{z}) \tag{3.37}$$

$$F_A = \text{decoder}(\text{encoder}(\mathbf{x})) \tag{3.38}$$

where encoder and decoder are fully connected neural networks and where encoder : $\mathbb{R}^n \to \mathbb{R}^m$ and decoder : $\mathbb{R}^m \to \mathbb{R}^n$. A usual form of autoencoder has only one hidden layer. There are variations

of autoencoder such as denoising autoencoder using standard noise or masking noise [10] and variational autoencoder[18]. Trained autoencoder can be then used as data transformation (feature extractor) using only encoder function.

4 Computational tools

The speed of implementation of NN is crucial for its training and evaluating and with increasing importance of NNs in the machine-learning fiel. Since NNs revival after 2006 many developer groups have started creating frameworks implementing tools for building neural networks. In recent years, there have been developed many frameworks supporting neural network training. There are two major approaches to implementation of such framework, one which compiles symbolically modeled calculation using fast math libraries and second one which maps calculations on the already pre-compiled source code.

4.1 Theano

Theano framework is one of the first frameworks which combines advantages of a computer algebra system with advantages of an optimizing compiler. It was developed by the research group at University of Montreal, but it has been open-sourced on github repository [1]. Theano provides a way to model symbolic calculation and then compiling it using mathematical libraries like BLAS or LAPACK. Theano gains from systems with GPUs wich can be used in parallel computing. Since Theano was introduced in 2011, it has a large community of researchers and active users. For scientific community, Theano comes with support of Python language.

4.2 TensorFlow

TensorFlow(TF) is open-sourced framework developed by Google based on experiences with framework DistBelief which was first one, that Google had developed for purposes of Google Brain project. The main research field of Google Brain is the use of very-large scale deep neural networks in Google's products. User defines dataflow-like model and TF maps it onto many different hardware platforms from Android platforms to massively distributed systems with hundreds of CPU/GPU units. [14] The latest release of TF is 0.8.0 version, which

1. Theano repository `www.github.com/theano`

means that final API is not fixed yet and there might be some changes. The sources can be found at github repository [2]. Unlike Theano, TF provides public API not only for Python but also for C++, which makes it easier to deploy models into business applications.

4.3 Deep Learning for Java

Deep Learning for Java (also Deeplearning4j or as abbreviation DL4J) is, according to creators in Skymind company, first commercial framework for deep learning written in Java for Java and Scala environments. With its integration of Hadoop and Spark, DL4J aims to usage in business applications rather than usage as a research tool. DL4J is also open-sourced [3] as maven[23] project so it is easily integrated into production. Internally DL4J runs on JBlass and provides automatic data parallelism on multiple GPUs.

4.4 CNTK

Microsoft has its own tool for neural networks called CNTK. It supports both C++ and Python for modeling and evaluation. The created models can be also executed from C# code. CNTK can scale-up up to 8 GPUs, which is second highest number of supported parallel GPUs at the same time. CNKT is open-source project released on web site [4] in April 2015.

4.5 Torch7

Is a scientific computing framework for LuaJIT. The initial release was in 2002. Many huge companies like Facebook or Twitter uses Torch7 on daily basis for data mining and machine learning. It is based on tensor-like calculations modeling in Lua. Thanks to it, it is very modular and provides an easy way to create computations on CPU and GPU. As almost all other frameworks Torch7 is open source project based on [5].

2. TensorFlow repository `www.github.com/tensorflow`
3. DL4J repository `https://github.com/deeplearning4j/`
4. CNTK repository `https://github.com/microsoft/cntk`
5. Torch repository `https://github.com/torch`

4.6 Caffe

Caffe was developed on Berkeley College and today has many contributors on Github. Caffe contains the implementation of fast convolutional neural networks in C/C++. Caffe supports Python API for modeling networks, but it is only specialized in image processing. Caffe cannot handle other tasks such time series or text processing.

4.7 Frontend for frameworks

There are many other tools and frameworks for training neural network models. They differ in supported languages. environments and also most importantly in speed of computations.

Keras Due to complexity of some frameworks, there has been created front-end frameworks for easier modeling of neural network graphs. One of them is Keras which front end in Python provides layer-wise modeling of neural networks. It was initially developed as front-end for Theano framework, since Theano's mechanics might be considered complicated but recently was extended to be generic front-end for TF too. Keras has huge community around for its simplicity, usability and Python support.

Others The latest boom of neural networks started many projects aiming to neural networks. There are many other frameworks specially for neural networks. Bigger list of neural networks frameworks and tools can be found at [24].

4.8 Implementation of autoencoder model variations in TensorFlow's model repository

TensorFlow framework has been released during work on this thesis in version 0.5. As was said it is general tool for tensor calculus but it was lacking many different algorithms as examples. After communication with authors, there was implemented Autoencoder [17], Denoising Autoencoder using standard noise[10], Denoising Autoencoder using

vector masking[10] and Variational Autoencoder [18] for needs of this thesis using TensorFlow framework. All these implementations were successfully merged into official TensorFlow Github repository [6] as example models.

6. https://github.com/tensorflow/models

5 FOREX market

FOREX is an abbreviation for **For**eign **Ex**change which is also commonly known as Forex Trading, Currency Trading, Foreign Exchange Market or shortly FX. Forex is international, decentralized market system for trading currencies world wide [6] and is the biggest and most liquid market in the world[7]. Currencies are traded in pairs of two currencies a and b with some ratio r commonly marked as $a/b = r$. The currency a is called base currency the b currency is called quote currency. The notation stands that one unit of a is equal to r units of b, conversely rb units are equal to one unit of a. Typical example is pair EUR/USD = 1.23456 which means that one Euro is equal to 1.23456 U.S Dollars.

FOREX trades 24 hours a day, 7 days a week as trading moves across borders and around globe with the time. Estimated daily volume in April 2013 was $5.3 trillion [8], an exact volume is unknown since there are many market places and some might be even not known. All subjects that exchange currencies such as banks, traders, investors, companies but also individuals going on vacation participate on Forex Trading which makes FOREX decentralized and non-controlled market.

5.1 Trading Mechanics

Pip Pip is the smallest price movement that one currency pair can make. Different currency pairs may have different size of Pip.

Tick A single tick represents smallest possible update of price on FOREX market. Frequency of tick updates depends on many factors like daily and hourly liquidity, liquidity during news, number and quality of connected FOREX marketplaces etc. Frequency can vary from minutes to microseconds. Each tick updates price of traded currency by trader.

Asks, bids and spread Markets are created when two or more subjects have a disagreement on value and an agreement on price. The

actual price is not one number but two numbers changing constantly. These numbers are called bid and ask and represent the maximum price that a buyer is willing to pay for and minimum price that a seller is willing to sell for, respectively. A trade is done when ask equals bid so buyer and seller agree on the price of the currency pair. Usually, ask is higher than bid and difference between those two numbers is called spread which is the key indicator of liquidity of currency pair - the smaller spread is, the better liquidity. [6]

Leverage Leverage is widely known concept associated with trading. Leverage enables trader to trade significantly larger amounts of volume even if he does not possess the capital equivalent to the nominal traded size. Missing capital is provided by broker. The leverage increases profits but deepens losses.

Commissions Theoretically, FOREX is a zero-sum game. Every single profit of one trader is taken as loss of another one. The problem are commissions which must be, together with the spread, overcome to make the profit on the trade.

5.2 Trading markets

Although FOREX is decentralized there are some main currency exchange markets: London, New York, Sydney and Tokyo. During working week, there is always at least one main currency exchange market opened. During weekends, all main marketplaces are closed so liquidity drops down. It is possible to trade on markets online thanks to intermediary companies called brokers. The broker is subject that enables trading using its trading platform for which is usual that trader gets higher spreads as payment for using broker's services.

5.3 Forecasting FOREX

Each market can be observed and predicted on different time levels. It would not make sense to observe difference between bid and ask if predicting one month ahead, on the other hand, if predicting few

seconds into future, the difference between bid and ask can play a huge role in the prediction because its impact is immediate.

Financial market, in general, can be seen in its simplest form as the sequence of numbers where each number is the average of lowest ask and highest bid price, as two numbers representing bid and ask prices in parallel or as market depth.

Market depth is a list of all bids and asks opened as a trading position at each time stamp. It is the full information about one specific marketplace. Distribution of orders (bids and asks) can be build up from market depth.

For purposes of following the section, let us define some basic notation:

$$P_a(t) = \text{lowest ask price at time } t \tag{5.1}$$
$$P_b(t) = \text{highest bid price at time } t \tag{5.2}$$
$$P(t) = (P_a(t) + P_b(t))/2 \tag{5.3}$$
$$Ext(t_1, t_2) = \text{general feature vector from time } t_1 \text{ to time } t_2 \tag{5.4}$$
$$\tag{5.5}$$

Time based discretization The most common way to discretize market is to make market info snapshots at some specific time. This specific time can be as small as seconds to minutes, hour, days or even months. It is the simplest discretization of market and many financial plots, like candles or bars, are based on it. Feature vector in time based discretization with smallest time unit n can be written as:

$$\mathbf{x} = Ext(t - n, t) \tag{5.6}$$

Usually

$$t \mod n = 0 \tag{5.7}$$

Price change based discretization Another possible manner of visualizing the market is to make market info snapshots after predefined delta price. It is not that usual as time based discretization. Feature vector in price change based discretization with smallest price delta d can be written as:

$$\mathbf{x} = Ext(t_1, t_2) \text{ when } P(t_2) > P(t_1) + |d| \tag{5.8}$$

if d is equal to one pip, then we get sequence of pure ticks.

5.3.1 Data sampling

We can define sampled i-th pair p of data vector $\mathbf{x}_i$ and target vector $\mathbf{y}_i$ as:

$$p_i = (\mathbf{x}_i, \mathbf{y}_i) \tag{5.9}$$

where $\mathbf{x}_i$ is a composition of last k data vectors and $\mathbf{y}_i$ is data vector from next time interval:

$$\mathbf{x}_i = Ext(t_{i-1}, t_i), Ext(t_{i-2}, t_{i-1}), \ \ldots, Ext(t_{i-k-1}, t_{i-k}) \tag{5.10}$$
$$\mathbf{y}_i = Ext(t_i, t_{i+1}) \tag{5.11}$$

Vector $\mathbf{x}$ is example of lookback defined earlier. With given data pairs let $\mathcal{T}$ be set of sampled pairs:

$$\mathcal{T} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ \ldots, (\mathbf{x}_p, \mathbf{y}_p)\} \tag{5.12}$$

where p is number of valid sampled pairs and $\mathbf{x}_k \in \mathbb{R}^n$, $\mathbf{y}_k \in \mathbb{R}^m$, same as defined earlier.

6 Application on FOREX

The practical part of this thesis has been created by industrial partner of faculty and its target is to predict FOREX based time series, more precisely to predict very short trends on FOREX market.

6.1 System

Experimental environment was created on freshly installed Linux operating system, more specifically Ubuntu 15.10, with gcc version 5.2.1, running on Linux kernel 4.2.0-19 generic. A workstation powering all experiments includes Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz processor, 32GB of DDR3 RAM and two GeForce GTX 960 graphic cards both with 4GB of GDDR5 VRAM.

6.1.1 Computational framework

When deciding what framework we would use for running computations, TensorFlow(TF) was released on version 0.5.0 and with its more simplicity than Theano (which was our second candidate because its wide community and many already implemented models) we decided to use TF. Huge plus for TF are facts that Google, Inc. is behind it and its ability to distribute calculation over more GPUs at once, however this feature was not available in 0.5.0 but was promised by officials and released in version 0.8.0.

We also gained from advantages of Keras framework which was updated to support TF shortly after TF's release. Keras is highly modular, neural networks front-end running on top of TF and originally Theano. Its approach to neural networks is layer-based, which means, that each new functional layer is applied on top of already defined layers.

6.1.2 Experiment core

We built our own system for experimental purposes which surrounded Keras implementation of multi layer perceptron. Since Keras does not

support pretraining methods, we had to implement them using TF. Here is example of experiment setup:

```
with tf.device("/cpu:0" if len(sys.argv) <= 1
                        else sys.argv[1]):

    data_params = {
        "path": {
            "dirPath": "/data/path",
            "X_train": "lb_train.csv",
            "X_valid": "lb_valid.csv",
            "X_test": "lb_test.csv",
            "Y_train": "labels_train.csv",
            "Y_valid": "labels_valid.csv",
            "Y_test": "labels_test.csv"
        },
        "centralize": True,
        "balance_train_classes": "cut_down",
        "standard_noise_scales": None,
        "cut_training_set_ratio": 0.0,
        "targets_alternator":
            ["EpsTargetAlternator", {"eps": 0.02}],
    }

    model_params = {
        "model": "MLP",
        "topology": [1024, 128, 64, 32],
        "dropout": [0.5],
        "hidden_activations": ["relu", "relu", "relu", "sigmoid"],
        "hidden_init": ["pretrain"],
        "output_activation": "linear",
        "output_init": "zero",
        "loss_function": "mse",
        "optimizer":
            ["Adam", {"learning_rate": 0.001}]
    }

    learning_params = {
        "device": "/cpu:0" if len(sys.argv) <= 1
                           else sys.argv[1],
```

```
        "pretraining_params": {
            "pre-training_batch_size": 2 ** 8,
            "nb_pre-training_epoch": 100,
            "nb_pre-training_tracking_epoch_index": 1,
            "pre-training_machine":
                "masking_autoencoder",
            "pre-training_optimizer":
                ["Adam", {"learning_rate": 0.0001}],
            "dropout_probability": 0.8,
            "visualize_space": False
        },
        "training_batch_size": 2 ** 7,
        "testing_batch_size": 2 ** 17,
        "tracking_epoch_index": 1,
        "nb_training_epoch": 5,
        "confidence_calculator": ["MaxDiff", {}],
        "filter_to_basis_point_target":
            [9000, 7000, 5000, 2000, 1000, 500],
        "comparator": KappaComparator(),
        "use_business_labels": False,
        "keras_verbosity": 2
    }
    results_params = {
        "storing_path": "/results/path/",
        "file_path": __file__,
        "save_train_log": True,
        "save_valid_log": True,
        "save_activations": True,
        "track_data_flow": True,
    }

    start_time = time.time()

    execute_experiment(
        data_params = data_params,
        learning_params = learning_params,
        model_params = model_params,
        results_params = results_params)
```

Experiment setup takes all important parameters for experiment execution so user does not have to care about any functional code. It is

layer by layer wise so each layer can have different setup. All features from Keras 1.0.2 are supported.

6.1.3 Experiment executor

With two GPUs in the workstation, we wanted to use them both in parallel. We have developed a simple python script called Scheduler for purpose of mapping experiments on GPUs. Scheduler takes the queue of python scripts and executes them individually on GPUs. We have ended up with one process per one GPU.

It is possible to run more processes on one GPU using partial VRAM allocation which TF framework offers, but TF in the newest version 0.8.0 still lacks method for calculating real VRAM consumption before the actual allocation of the tensor graph, which causes a lot of program fails. Inability to get real memory consumption led us to safer solution with mapping only one process per GPU but there is still a chance that the process will fail on memory allocation.

6.1.4 Experiment setup generator

For grid search purposes we have created python script called JobGenerator. The JobGenerator was automatically generating independent python scripts for automatic execution. JobGenerator is meta-program which has string template of experiment setup, default value for each parameter and domain values, which is only part that user must care of for grid-search. It fills template's parameters and generated new python file with the name according to filled domain values. With automatic execution on GPUs provided by Scheduler it is very powerful tool.

6.2 Data

The industrial partner provided three data-sets, training set, validation set and test set, all sampled from FOREX market in temporal order. All provided data are anonymous since they contain sensitive information, so a source of the data and all features of data vectors are unknown.

One data vector consists of 460 features and its label consists of 3 values - UP, MID, DOWN. Figure 6.1 shows the visualization of projec-

Table 6.1: Final data set sizes

set	size
training	881889
validation	49947
testing	17638

tion of data into two dimensional space using Variational Autoencoder with five hidden layers. Used topology was [460-500-400-2-400-500-460]. Figure 6.2 shows the same technique applied on the MNIST dataset for comparison purposes. The visualization shows how inconsistent and noisy our problem is compared to widely known and, for comparing models, commonly used MNIST dataset.

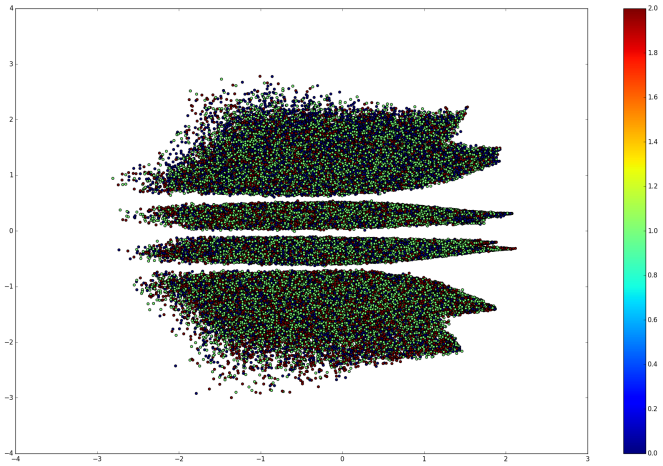

Figure 6.1: Data visualized by Variational Autoencoder

6.2.1 Missing values

Some data vectors were missing values. We have decided to get rid of such vectors since random noise or average values might be problematic in such application.

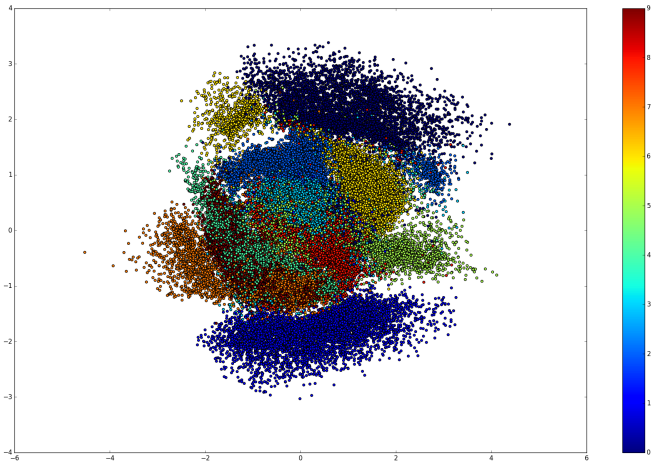

Figure 6.2: MNIST example using same settings as transformation in Figure 6.1 for comparison with well known dataset

6.2.2 Normalization

Since many inputs in dataset differ rapidly in mean and standard deviation, we used usual transformation of input data:

$$X'_{ij} = \frac{X_{ij} - \mu_j}{\sigma_j} \tag{6.1}$$

where:

$$\mu_j = \frac{\sum_{i=1}^{k} X_{ij}}{k} \tag{6.2}$$

$$\sigma_j = \frac{\sum_{i=1}^{k} (X_{ij} - \mu_j)}{k} \tag{6.3}$$

to prevent dominating of some features over others.

6.2.3 Balancing classes

The majority of labels belongs to MID class. We were experimenting with no alternating counts of data and it turns out that balancing

classes on same number of occurrences helps. Such operation can be done only on the training set. Let **c** be number of all given classes, then balancing classes is done by sampling new dataset from given one with calculated probability **p** as:

$$c_m = min(\mathbf{c}) \tag{6.4}$$

$$\mathbf{p} = c_m / \mathbf{c} \tag{6.5}$$

for each vector in each class.

6.2.4 Target alternation

In this thesis, we are using the standard sigmoid function as the last activation function in the neural network. We have encountered a problem with saturation of standard target values and created a target alternator module which transfers target values by adding some ϵ to low target values and subtracting ϵ from correct target value. For example it transforms vector $(1.0, 0.0, 0.0)$ to vector $(1.0 - \epsilon, 0.0 + \epsilon, 0.0 + \epsilon)$ where ϵ was experimentally set up to 0.02 for sigmoid function.

Another method is to have only linear units at the output layer and multiply target vector by constant. The idea is to spread neuron activations over a wider interval.

6.2.5 Confidence and filtering

The industrial partner has informed us that our predictor does not need to make predictions for all cases in the testing set. We introduced calculators of confidences and algorithm for finding the proper threshold for which predictor ignores all prediction signals and its output is emtpy value. The idea behind confidences is to minimize missed predictions and obtain better precision for each class in confident data predictions.

Max confidence Max confidence calculator is very easy and basic confidence calculator. It takes maximum from activations of all output neurons.

$$\text{confidence} = \max(\mathbf{y}) \tag{6.6}$$

Max diff confidence Max diff confidence calculator takes the two highest activations of output neurons and returns their difference

$$\text{confidence} = \max(\mathbf{y}) - \text{second\_max}(\mathbf{y}) \tag{6.7}$$

Ternary mid diff confidence Calculating confidence of UP and DOWN classes from MID class is another possible filter. Basic idea is to predict only UP and DOWN since MID is not interesting class because the market is not moving in any direction. Ternary mid diff confidence calculator can also be interpreted as MID class when neither UP nor DOWN is predicted.

$$\text{diff}_1 = y_1 - y_2 \tag{6.8}$$
$$\text{diff}_2 = y_3 - y_2 \tag{6.9}$$
$$\text{confidence} = \max(\text{diff}_1, \text{diff}_2) \tag{6.10}$$

Let $\mathbf{Y}$ be activations of output neurons on validation set. Then we can calculate the vector of confidences:

$$\text{confidence}_k = \theta(\mathbf{Y}_k) \tag{6.11}$$

where θ is confidence calculator such that $\theta : \mathbb{R}^m \to \mathbb{R}$ and m is number of output neurons.

Finding a threshold of confidence for picked calculator is done on validation set. Since it is not iterative process it does not need to be optimized on two sets like training of model does. Consider objective O which we want to optimize by adjusting the threshold. O can be for example precision over a number of cases. To adjust the threshold, all confidences are sorted from the highest one to the lowest one. Then we drop one by one vector with the lowest confidence and recalculate statistics and O. The threshold, which maximizes O is then picked and ready to test on the testing set. Algorithm 1. is pseudo code for finding the threshold.

Optimizing threshold and finding best confidence calculator can be in general considered as another machine-learning problem where input values are output activations from neurons and where target labels stays the same with new error function O. For that we need five datasets instead of three.

Algorithm 1 Threshold optimization algorithm

```
1:  function CalcConfidences(Y)
2:      confidenceVector ← {}
3:      for vector in Y do
4:          confidenceVector.add(θ(vector))
        return confidenceVector
5:  end function
6:  function FindThreshold(Y)
7:      confidenceVector ← CalcConfidences(Y)
8:      confidenceVector ← Sort((confidenceVector, Y))
9:      O ← CalcStats(Y)
10:     bestConfidenceThreshold ← 0
11:     for i ← confidenceVector.length downto 1 do
12:         Remove(Y, i)
13:         Remove(confidenceVector, i)
14:         O_tmp = CalcStats(Y)
15:         if O_tmp is better than O then
16:             O ← O_tmp
17:             bestConfidenceThreshold ← confidenceVector[i]
        return bestConfidenceThreshold
18: end function
```

Threshold can also be included as a parameter in learning process of original model with new loss function O. We did not yet experimented with last variant since threshold significantly affects results and it would most certainly become the most important parameter in the whole model.

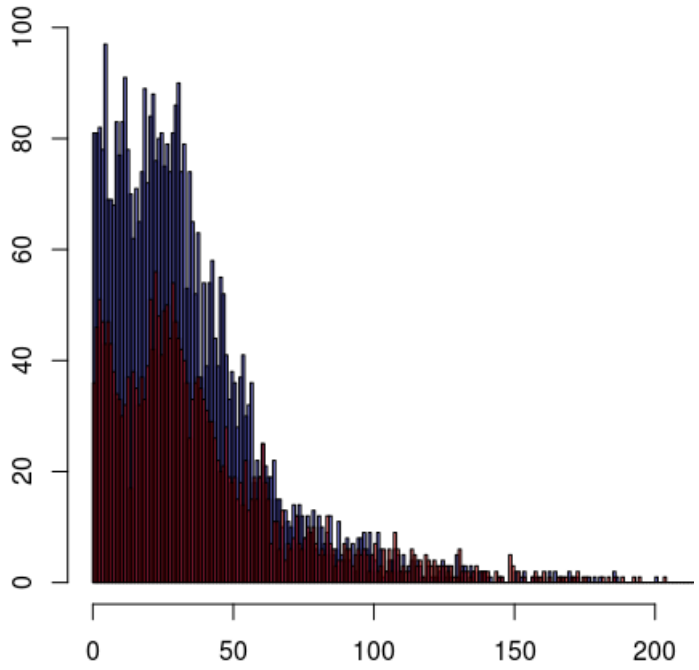

Figure 6.3: Frequency of confidences of neural network for all UP labels. UP(Red) vs SELL and MID (Blue)

In Figure 6.3 we can see positive effect of filtering by network's confidence. Figure 6.4 shows detailed look on already filtered predictions.

6.2.6 Cohen's kappa coefficient

Evaluation of models for their comparison were performed by using Cohen's kappa coefficient (κ). κ is a measure of difference between observed agreement on classes from the expected agreement on classes, standardized to lie in $\langle -1, 1 \rangle$, where 1 is perfect agreement, 0 is com-

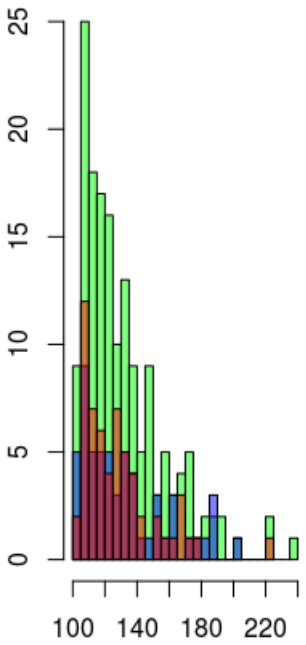

Figure 6.4: Frequency of confidences of neural network after filtering with threshold confidence = 100.

pletely randomized agreement and -1 is agreement worse than chance, ie. potencial systematic disagreement. [2] Kappa is calculated as:

$$\kappa = \frac{1 - p_o}{1 - p_e} \tag{6.12}$$

where p_o is observed agreement and p_e is expected agreement.

6.3 Results

Using JobGenerator and Scheduler, we were able to generate and execute over 3000 experiments in few months, providing local grid search during research. After running a significant number of experiments we have fixed some parameters which will not be changed from now on.

The grid-searched domain was: topology, dropout, hidden initialization and the pre-training number of epochs. Such domain was

Table 6.2: Fixed parameters for experimental purposes of this thesis

hidden activation	ReLU
last layer initialization	zeros
output activation	sigmoid
target alternator	additive $\epsilon = 0.02$ (6.2.4)
loss function	MSE
optimizer	Adam($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$)
batch size	2^8
training epochs	100
confidence calculator	Ternary mid diff (6.2.5)
filter target optimization	10 %
pre-training batch size	2^8
pre-training machine	autoencoder
pre-training optimizer	Adam($\alpha = 10^{-5}$, $\beta_1 = 0.9$,
	$\beta_2 = 0.999$, $\epsilon = 10^{-8}$)

picked in order to search for neural network setup which leads to generalization after data filtering, since given data are considered as very noisy. Using ReLU units, which were proven as functions with no vanishing gradient signal [10], we wanted to find optimal topology with initializing weights and biases by pre-training using simple Autoencoders and without it using Glorot normal initialization [25]. Topology itself is very important for generalization. Huge networks can very easily overfit but it is believed that deeper networks can perform better feature extraction. Dropout has been developed as very a effective technique to reduce overfitting of neural network, we have decided to see the difference between neural networks without dropout and with dropout as [3] suggests.

Example of confusion matrix calculated on unfiltered testing set in Table 6.4 shows that process is able to find a solution which is better than random. Cohen's kappa coefficient calculated from Table 6.4 is 0.122 which is clearly better than random (kappa 0.000). Precisions of the classes are: UP - 0.390, MID - 0.500, DOWN - 0.370. Besides kappa,

Table 6.3: Grid search domain

Topology	64
	64 - 32
	64 - 32 - 16
	128
	128 - 64
	128 - 64 - 32
	128 - 64 - 32 - 16
	128 - 64 - 32 - 16 - 8
	256
	256 - 128
	256 - 128 - 64
	256 - 128 - 64 - 32
	256 - 128 - 64 - 32 - 16
	256 - 128 - 64 - 32 - 16 - 8
Dropout	0.0
	0.5
Pretrain epoch	8
	16
	32
Learning rate	10^{-3}
	10^{-4}
	10^{-5}
	10^{-6}
Initialization	Glorot normal
	pretraining

Table 6.4: Example of unfiltered testing set

	UP	MID	DOWN
UP	2458	1163	2123
MID	1967	2252	2158
DOWN	1870	1088	2559

Table 6.5: Example of filtered testing set

	UP	MID	DOWN
UP	528	0	431
MID	277	0	371
DOWN	331	0	613

the precision of each class is also a very important quality indicator of results.

After filtering, the confusion matrix might radically changed. Change depends on picked confidence calculator. In Table 6.5 we used Ternary mid diff confidence calculator resulting into vanishing MID class. We can observe that kappa equals 0.118 which is worse than kappa score on unfiltered test result. This is caused by Ternary mid diff. Precision of UP and DOWN classes rises on 0.464 and 0.433 respectively.

With right filtration, we were able to get precision over 0.7 and 0.64 for UP and DOWN classes respectively.

Learning rate

Learning rate occurs in all described gradient descent algorithms and it can significantly affect results of the learning process. We have tested learning rates 10^{-3} , 10^{-4}, 10^{-5} and 10^{-6}. The best results were achieved with learning rate 10^{-4}. Learning rate 10^{-3} showed unstable learning and on the other hand, learning rates 10^{-5} and 10^{-6} showed tendencies to dominate over one class. There were cases when such small learning rates worked but in general performance was worse than 10^{-4}. This might be fixed by more epoch given to the training algorithm.

Dropout

Dropout is a relatively new concept in neural networks and should prevent overfitting model on the training set. We tested 0.0 and 0.5 levels of dropout which means that during training 0% and 50% of neurons are randomly switched off. For neural networks with relatively small number of connections and one or two hidden layers the dropout 0.5 has no significant effect on unfiltered validation and test set, however, combined with filtering dropout 0.5 ended up worse than neural network without dropout (dropout 0.0). On the other hand, neural networks with five or six hidden layers seemed to gain from dropout in unfiltered and filtered cases. Dropout defect with smaller networks could be caused by smaller number of parameters so model cannot get overfitted easily.

Topology

Topology is one of the most discussed parameters of neural networks. It highly affects the performance of learning and final model. We have tested topologies from one hidden layer up to 6 hidden layers also with different widths from 64 to 256 neurons in first hidden unit. Each next hidden layer was two times smaller than the previous one, so one possible topology has 128, 64, 32 neurons in first, second and third hidden layer respectively. The biggest network had over 160 thousands of connections. We were not able to run bigger networks due to memory issues. Results for topologies differs rapidly and there is a strong correlation based on used dropout and already mentioned problems between dropout and number of connections. Wider networks with no dropout has similar kappa and class precision as deeper networks with 50% dropout.

Weights initialization

The important question is if initialization of neural network by stacking autoencoders [10] improves the performance of the neural network. We used simple autoencoders with different number of training epochs (8, 16, 32) and compared results with static the Glorot normal [25] initialization. Different epoch numbers were used due to lack of early

Table 6.6: Confusion matrix with best kappa reached with proposed grid search

	UP	MID	DOWN
UP	410	0	266
MID	206	0	233
DOWN	236	0	409

stopping for pretraining. Unlike statically initialized networks, pretrained networks tends to be more stable when dropout for supervised training is used in combination with deeper topologies, on the other hand, some network instances ended up with no usable results, predicting only one class for all cases. There were also network instances which were able to train deeper architectures without pretraining.

If we consider kappa as only quality indicator of model, after described grid search, we were able to reach kappa 0.144, UP and DOWN class precision 0.481 and 0.450 respectively as Table 6.6 shows. The final target was set up on 9.978% of the total test set. Property of the model was topology 256-128-64-32-16-8, using pretraining with 16 epochs as initialization of weights, dropout 0.0 and learning rate 10^{-4}.

Despite the fact that model with the highest kappa was found using pretraining, pretraining itself seems to bring no actual quality value to trained model. Models without pretraining were also capable of similar results. It is possible that 100 learning epochs is not enough and more epoch with lower learning rate may suffice. Pretraining most likely does not work because pretraining is considered as generalization[10] technique and pure generalization is not enough for given problem.

6.4 Discussion and future work

6.4.1 Data

Most of the models had similar performance and many models had no problem to overfit training set almost perfectly. It might be signal that proposed metodic still have reserves and the problem is with noisy data. Introducing more different features might help with noise, on the

other hand, some features might be dropped out. We have picked one of the successful models and analyzed means and standard deviations of weights for input neurons.

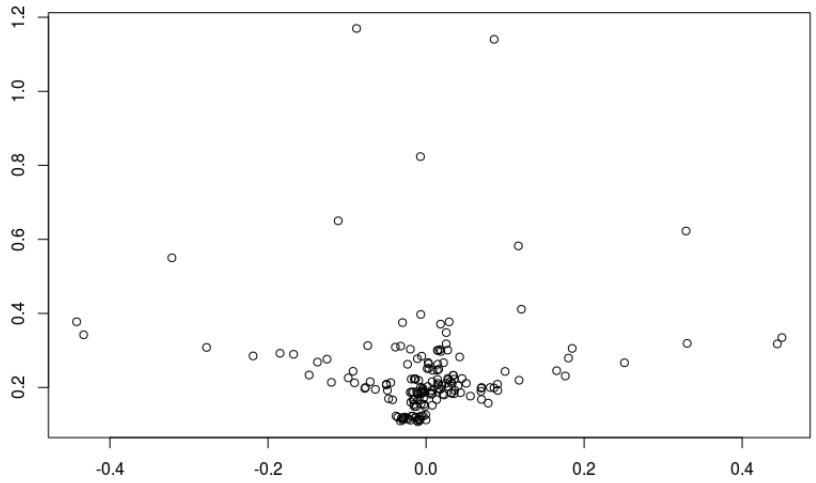

Figure 6.5: Means (x) and standard deviations (y) of weights of first 46 input neurons

Figure 6.5 shows that many features have marginally smaller standard deviation than others, also their means are very close to zero. We can deduct from such picture that many features could be avoided, which might lead to clearer data and training mechanisms could find better models.

MID class avoiding

It is better to predict MID class instead of DOWN class when the true class is UP and vice versa. In this thesis, we used three classes for predictions. Since predicting MID class is not necessary, getting rid of all MID class vectors might help trainers to find the best function separating UP and DOWN classes and then applying confidence threshold on such function might get better binary prediction UP vs DOWN.

6.4.2 Ensemble

Ensembles are considered stronger than single models. We did not try to build ensemble models in this thesis. One way to build ensemble is simple voting of more models and picking the elected class. From back analysis of activations we know that proposed models makes predictions at different input vectors which means that models are actually different and since all models are better than random guessing, voting of many models will lead to the more successful ensemble.

Since predictor does not have to make predictions for all input vectors, there are many alternatives how to build the ensemble. One way is to have set of models where all models already have its own threshold. Another possibility is to have a set of models without the thresholds and find one final threshold for the sum of activations of all models. Combination of both approaches is also possible.

Another way to build better predictor might be AdaBoost algorithm which combines many so called weak learners into one model.

6.4.3 Filtering optimization

Proposed methodics for optimizing filtration might be considered as wrong since the confidence threshold is optimized on the validation set where is also optimized neural network itself. Model optimization is stopped on validation set which means that model is little overfitted not only on the training set but also on the validation set. We observed that the left percentages of the testing set were unstable and for 10% targets we sometimes got +- 4%. It is very likely due to optimizing filter on the same set that the model was.

In order to find best the filter, we might consider it as another machine learning process with ideally two new data sets. Such model could take activations from predictor and optimize filtered predictions using different loss function. Especially with more detailed categorization of output classes (UP high, UP low, MID, DOWN low, DOWN high) filtering becomes nontrivial task which might be easily solved using machine learning techniques.

6.4.4 Deployment

Based on the experiences from all the experiments we proposed few models which were successfully deployed for our industrial partner into production and the partner confirmed that models were successfully predicting the market, yet there are many things to improve.

7 Conclusion

Financial market prediction is one of the hardest machine learning problems. Some financial markets, like FOREX, for example, are even considered as Brownian motion. For machine learning process it means that the data are very noisy and it is very hard to find some inefficiency of market (inefficiency of market = anything that is not entirely randomized) in the data.

We trained fully connected neural networks to predict time series sampled from FOREX market. Prediction consisted of classes UP, DOWN, MID. Trained models were compared using Cohen's kappa coefficient and class precision. We have executed over 3000 experiments grid-searching for useful settings of neural networks. In this thesis, we have focused on the analysis of the influence of dropout, topology, pretraining with a different number of epoch, static initialization and learning rate.

For massive grid-search, we had to build our own system for creating and executing experiments. The system was built on top of frameworks TensorFlow and Keras, leaving the user with the only configuration file to care of. The system was able to run parallel tasks on graphics cards for better performance.

Since it is not necessary to make prediction for all vectors, we have introduced confidence calculators and filtering method to filter out predictions that were not confident enough.

We have derived from experiments that the neural network setup is not crucial for model success and it is more important to work with data and post filtering methods. There were no huge differences between topologies of neural networks, although deeper architectures sometimes had problem with getting some useful results. The bests results were attained with static initialization same as with pre-training.

After few months of experimenting, we proposed few models which were successfully deployed for our industrial partner into the production and tested by making predictions for the real trading system.

Bibliography

[1] NAIR Vinod, Geoffrey HINTON. *Rectified Linear Units Improve Restricted Boltzmann Machines* [online]. 2010. http://www.cs.toronto.edu/~fritz/absps/reluICML.pdf [cit. 2016-05-15]

[2] VIERA Anthony J., Joanne M. GARRETT. *Understanding Interobserver Agreement: The Kappa Statistic* [online]. 2005. http://virtualhost.cs.columbia.edu/~julia/courses/CS6998/Interrater_agreement.Kappa_statistic.pdf [cit. 2016-05-08]

[3] SRIVASTAVA Nitish, Geoffrey HINTON, Alex KRIZHEVSKY, Ilya SUTSKEVER, Ruslan SALAKHUTDINOV. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* [online]. 2014. https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf [cit. 2016-05-03]

[4] ERHAN Dumitru, Yoshua BENGIO, Aaron COURVILLE, Pierre-Antoine MANZAGOL, Pascal VINCENT. *Why Does Unsupervised Pre-training Help Deep Learning?* [online]. 2010. http://www.jmlr.org/papers/volume11/erhan10a/erhan10a.pdf [cit. 2016-05-03]

[5] ADHIKARI Ratnadip, R. K. AGRAWAL. *An Introductory Study on Time Series Modeling and Forecasting* [online]. 2013. http://www.realtechsupport.org/UB/SR/time/Agrawal_TimeSeriesAnalysis.pdf[cit. 2016-04-27]

[6] Forex Tutorial: The Forex Market. *Investopedia*. [online]. http://www.investopedia.com/university/forexmarket/ [cit. 2016-03-28].

[7] Co je FOREX?. *Svět obchodování na FOREXu* [online]. http://www.fxstreet.cz/co-je-forex.html [cit. 2016-04-23].

[8] Bank for International Settlements. *Triennial Central Bank Survey of foreign exchange turnover in April 2013 - preliminary results released by the BIS* [online]. 2013. http://www.bis.org/press/p130905.htm [cit. 2016-04-23].

[9] VREEKEN Jilles. *Spiking neural networks, an introduction*, 2002. http://eda.mmci.uni-saarland.de/pubs/2002/spiking_neural_networks_an_introduction-vreeken.pdf [cit. 2016-05-18]

[10] GOODFELLOW Ian, Yoshua BENGIO, Aaron COURVILLE. *Deep Learning* 2016. http://www.deeplearningbook.org/ [cit. 2016-05-18]

[11] SUTSKEVER I., J. MARTENS, G DAHL, G HINTON. *On the importance of initialization and momentum in deep learning* [online]. 2013 https://www.cs.utoronto.ca/~ilya/pubs/2013/1051_2.pdf [cit. 2016-04-05].

[12] REIDMILLER Martin, BRAUN Heinrich. *A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm* [online]. 1993. http://deeplearning.cs.cmu.edu/pdfs/Rprop.pdf [cit. 2016-04-05].

[13] HINTON Geoffrey, SRIVASTAVA Nitish, SWERSKY Kevin. *Overview of mini-batch gradient descent* [online]. 2014 http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf [cit. 2016-04-05].

[14] ABADI Martin, AGARWAL Ashish, BAHRAM Paul, BREVDO Eugene et. [online]. *TensorFlow: Large-Scale Machine Learning on heterogenous Distributed Systems* [online]. 2015 http://download.tensorflow.org/paper/whitepaper2015.pdf [cit. 2016-03-30].

[15] LECUN Yann, Leon BOTTOU, Genevieve B. ORR, Klaus-Robert Müller. *Efficient BackProp* [online]. 1998. http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf [cit. 2016-04-03].

[16] CYBENKO. G., *Approximations by superpositions of sigmoidal functions* [online]. 1989. https://www.dartmouth.edu/~gvc/Cybenko_MCSS.pdf [cit. 2016-03-30].

[17] BALDI Pierre, *Autoencoders, Unsupervised Learning, and Deep Architectures* [online]. 2012 http://www.jmlr.org/proceedings/papers/v27/baldi12a/baldi12a.pdf [cit. 2016-04-28].

[18] KINGMA Diederik, Max WELLING, *Auto-Encoding Variational Bayes* [online]. 2013. http://arxiv.org/pdf/1312.6114v10.pdf [cit. 2016-04-28].

[19] BASTIEN F., P. LAMBLIN, R. PASCANU, J. BERGSTRA, I. GOODFELLOW, A. BERGERON, N. BOUCHARD, D. WARDE-FARLEY and Y. BENGIO. *Theano: new features and speed improvements*. NIPS 2012 deep learning workshop.

[20] BERGSTRA J., O. BERULEUX, F. BASTIEN, P. LAMBLIN, R. PASCANU, G. DESJARDINS, J. TURIAN, D. WARDE-FARLEY and Y. BENGIO. *Theano: A CPU and GPU Math Expression Compiler"*. Proceedings of the Python for Scientific Computing Conference (SciPy) 2010. June 30 - July 3, Austin, TX (BibTeX)

[21] BERGSTRA James, Olivier BREULEUX, Frédéric BASTIEN, Pascal LAMBLIN, Razvan PASCANU, Guillaume DESJARDINS, Joseph TURIAN, David WARDE-FARLEY, Yoshua BENGIO. *Theano: A CPU and GPU Math Compiler in Python* [online]. 2010 http://www.iro.umontreal.ca/~lisa/pointeurs/theano_scipy2010.pdf [cit. 2016-04-28].

[22] Deep Learning *Theano at a Glance* [online]. http://deeplearning.net/software/theano/introduction.html [cit. 2016-04-29]

[23] Apache Maven Project *Apache Maven Project* [online]. https://maven.apache.org/ [cit. 2016-04-29]

[24] Teglor *Deep Learning Libraries by Language* [online]. http://www.teglor.com/b/deep-learning-libraries-language-cm569/ [cit. 2016-04-30]

[25] GLOROT Xavier, Yoshua BENGIO. *Understanding the difficulty of training deep feedforward neural networks* [online]. 2010. http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf [cit. 2016-05-03]

A Examples

A.1 Experimental core

Experiment

1. Make sure that numpy, sklearn and scipy libraries are installed.
2. Install TensorFlow 0.8.0 by following installation instructions[1]
3. Install Keras from[2]
   (a) git clone https://github.com/fchollet/keras
   (b) git checkout 1.0.2
   (c) python3 setup.py install
4. Provide any data for classification. Make sure that it is divided into three datasets (train, valid, test). Put them into directory.
5. Setup proper data path and file names in sub-dict "path" of "data_params" dictionary.
6. Setup storing path in "results_params" dictionary.
7. Change all wanted parameters. Not all setups make sense.
   - It is possible to have different dropouts, activation functions and initializations for each layer.
   - By setting list of "filter_to_basis_point_target", more models with different filtering are created.
   - Supported features of Keras
   - Switching between random initialization and pre-training is done by setting hidden_init as "pretrain"
8. Run Experiment.py in Python 3 interpreter.

---

1. TF Installation `https://github.com/tensorflow/tensorflow/blob/master/tensorflow/g3doc/get_started/os_setup.md`
2. Keras repo: `https://github.com/fchollet/keras`

A.2 Autoencoder examples

1. Install TensorFlow by following installation instructions[3]
2. Clone repository [4]
3. Go into "models" directory
4. Run any runner file using Python 2 interpreter.

Experiment generator Experiment generator is meta-program.

1. Change default arguments.
2. Setup SearchParamsDomain dictionary for grid-search.
3. Run in Python 3 interpreter.

3. TF Installation `https://github.com/tensorflow/tensorflow/blob/master/tensorflow/g3doc/get_started/os_setup.md`
4. TF Models `https://github.com/tensorflow/models`