Vincent Mary School of Science and Technology

IT 4291, CS 3200

Senior Project I

Topic: Exchange rate prediction [1621]

Submitted to

Dr. Thitipong Tanprasert

A. Tapanan Yeophantong

Dr. AnilKumar Kothalil Gopalakrishnan

Submitted by

Kasperi Reinikainen	5818014
Hein Htet Naing	5818035
Asnai Narang	5815228

Semester 1/2017

Table of content

1. Introduction

For the first senior project our group was assigned to build an application that would utilize Machine Learning techniques to predict movements in currency exchange (Forex) markets.

This project topic immediately attracted high interest among our group members for reasons that it provides a great learning experience in the world of Machine Learning, in highly volatile commercial application use-case.

Our group's foundations for the project relied on three students in their third year of studies in Computer Science or Information Technology. None of the members had prior experience of Machine Learning, but were equipped with sharp focus and strong interest toward the field.

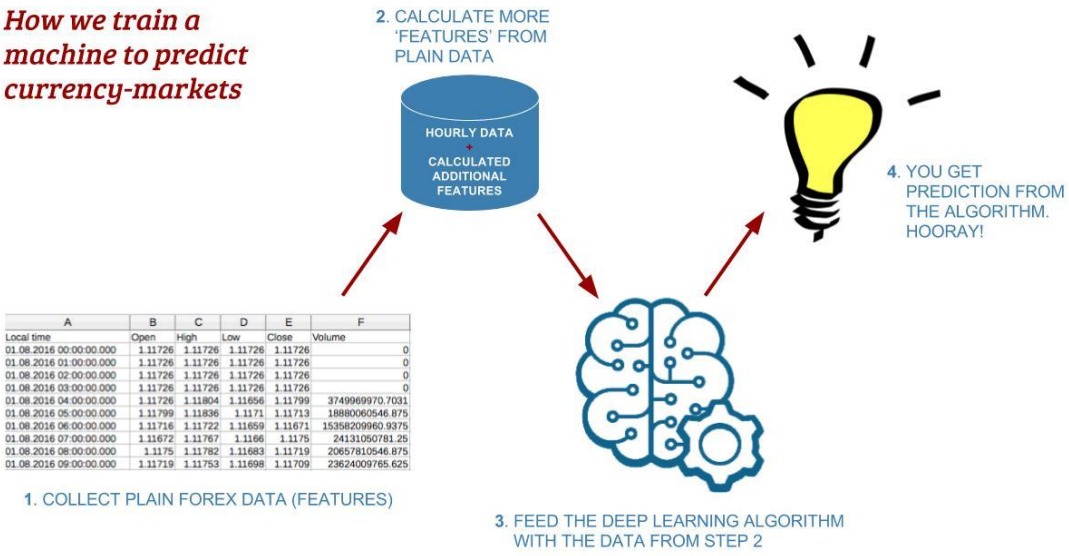
To begin with, our group was provided a reference study [1] by University of Nagoya. The idea was to initially follow the case study's methods of constructing a Deep Learning - model for Forex prediction, and after case study's result had been matched, try to improve the method in systematic ways.

As the case study was in a presentation format with very limited description of exact methods, and no published research of the authors were found, we had to make some assumptions regarding their methods. Several other researches and articles were relied on as well for both learning- and implementation purposes and they will be referenced as they are referred along the way.

An actual application was also built for the purpose of this project. The application showcases a potential use-case for the data that our findings produce. It is provided as a hybrid-mobile application

where user can follow currency markets and follow trends of the currencies of their interest and also get a prediction of possible future trend.

In this report, we will introduce the general concepts of Artificial Neural Networks, Forex markets and the depths of our work and the development process that was encountered.



	A	B	C	D	E	F
	Local time	Open	High	Low	Close	Volume
	01.08.2016 00:00:00.000	1.11726	1.11726	1.11726	1.11726	0
	01.08.2016 01:00:00.000	1.11726	1.11726	1.11726	1.11726	0
	01.08.2016 02:00:00.000	1.11726	1.11726	1.11726	1.11726	0
	01.08.2016 03:00:00.000	1.11726	1.11726	1.11726	1.11726	0
	01.08.2016 04:00:00.000	1.11726	1.11804	1.11656	1.11799	3749969970.7031
	01.08.2016 05:00:00.000	1.11799	1.11836	1.1171	1.11713	1888060546.875
	01.08.2016 06:00:00.000	1.11716	1.11722	1.11659	1.11671	1535820996O.9375
	01.08.2016 07:00:00.000	1.11672	1.11767	1.1166	1.1175	24131050781.25
	01.08.2016 08:00:00.000	1.1175	1.11782	1.11683	1.11719	20657810546.875
	01.08.2016 09:00:00.000	1.11719	1.11753	1.11698	1.11709	23624009765.625

Figure: simple illustration of our projects problem

2. Our motivation and background

Our motivation to do research on forex market with machine learning technology is to find out how to apply machine learning technique on the real life use case. Our team consists of 3 members and each of us have high interest towards modern machine learning techniques. We were keen to find out how to apply what we know in term of ML technique in the forex market.

Before doing this project our knowledge of machine learning was next to none. After we got assigned to this project we started to read other academic research we found online and also our main research path was following Nagoya University's research paper on the forex market by using Machine Learning technique.

The reason why we choose forex market as a target field to apply machine learning technology is due to the nature of forex market liquidity and the mount of trading volume that is going in the world today. The other reason is due to its popularity in every field of industry today.

2.1. Team members:

All of our team members are 3rd year students. This report was first senior project.

Kasperi Reinikainen, CS student. A student who is highly interested in programming and wanted to further challenge to dive in another field study, machine learning.

Hein Htet Naing, IT student. A student who has high interest in programming and moderate knowledge in statistics.

Asnai Narang, CS student.

3. Forex markets

The foreign exchange market (Forex market) is a global market as a medium for the exchange of foreign currencies. The market allows all participants to sell and buy of any foreign currencies pairs and exchange currencies at current determined rate. It is believed to be the largest trading volume in the current world.

The main participants of the forex market are large commercial banks, central banks, commercial companies and investors. The currencies are always traded in pairs (e.g. USD/THB), the forex market does not set absolute value for one currency but rather the value of exchange of market is determined by its relative value with another currency. (e.g. 1 USD = 32 THB).

Machine learning in our use-case with forex market data is used to be trained with historical data of one currency pair and then applied to predict whether the next point of time the currency pair exchange value is going to increase or decrease. The motivation of applying machine learning technology with forex market is to use the data from the past to train our machine for finding the pattern from the past data and building model according to how the machine found the pattern within the past data. The data being used in the machine learning training involved open, close, high & low and others extracted feature data. With the predicted result from machine, it is believed to help experts in their daily trading decision.

4. Artificial Neural Networks in brief

As famous author Tom Mitchell describes in his book 'Machine Learning' [2], the neural network learning methods provides a robust approach to approximating real-valued, discrete-valued, and vector-valued target functions. This means that neural networks are generally applicable for multiple use-cases, with variable desired outcome types and varied complexities of the learning problem.

The development of artificial neural networks (ANNs) has been inspired in part by the observation of how very complex webs of interconnected neurons are built in the brain. Roughly, artificial neural networks are built out of an interconnected set of simple units, where each unit takes a number of inputs (often the outputs of other units) and produces an output (which may be the input next layer of units) [2]. These units or 'nodes' are represented as neurons in the network. Most common versions of ANNs are 'feed-forward' networks, where the graph of neurons is directed and connection flows only toward the output layer.

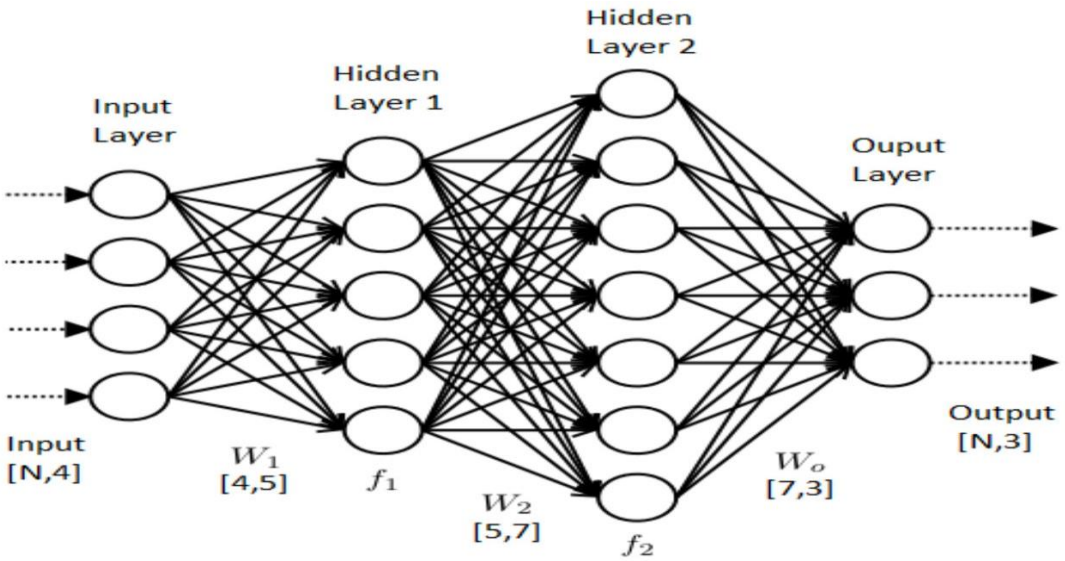


Figure: 3 layer feed-forward artificial neural network

In recent years, ANNs has attracted much attention and 'buzz' following recent advances in Artificial Intelligence in the fields like image recognition, voice recognition and natural language processing,

following successful implementation of ANNs. Because of their power to model highly nonlinear functions they are frequently implemented in the financial market prediction tasks and this makes them most suitable machine learning technique for our project as well.

There are multiple types of neural networks and they differ in tasks which they can perform and also in their training algorithms. However, some components rarely differ from ANN type to another and we try to cover the most common fundamental components that makes ANN such powerful tool.

3.2 Computation model of the neuron

Almost regardless of the ANN type in question, they all follow the same principle for integration function [3], where each neuron calculates the linear relationship from the input into the output. This is our first component of the computation model and it can be mathematically described as:

$$\boldsymbol{Wx + b}$$

Where W is a vector of weights, x is a vector of inputs and b is a scalar (or bias).

A weight represents a connection between two neurons and is essentially the main parameter to adjust in the task of finding most valuable neuron connections that maps to the desired output. The other parameter here is b and together W and b are often seen as □.

Let's consider a neuron n with multiple connections from previous layer. Each of the neurons from the previous layer would send an output that would together be considered as an input vector x. Each of these connections has an assigned weight (that describes the value of the connection) that would be together considered as weight vector W. In our neuron n these weight and input products would then be summed

and after this scalar b would be added. This can be represented as:

$$y = b + \sum (w_i x_i)$$

Second fundamental computational part is the activation function. From the first part, the integration function the output given from the equation is then sent to the activation function. The purpose of activation function is to map the value from previous step into a common format to be used with conditions. Some common activation functions are step-function and logistic-function.

In initial design of our project, we use sigmoid function which is one of the most common logistics functions. Sigmoid function quite simply just maps any input value to the positive real-number range of [0, 1]. This operation is represented with the following formula:

$$1 / (1+e^{-z})$$

(Where superscript z is the input.)

Since the output of sigmoid function is guaranteed to be between 0 and 1, we can use it as it is, apply conditions or use more formatting techniques to produce an output y of a neuron. Say this particular neuron lies at the output layer and the network is supposed to produce binary classification, either 0 or 1. We can set condition where our output y is 0 for all values below threshold 0.5 and output y is 1 for equal and greater values than the threshold.

Putting the first two parts together we get the following computational graph for a single neuron

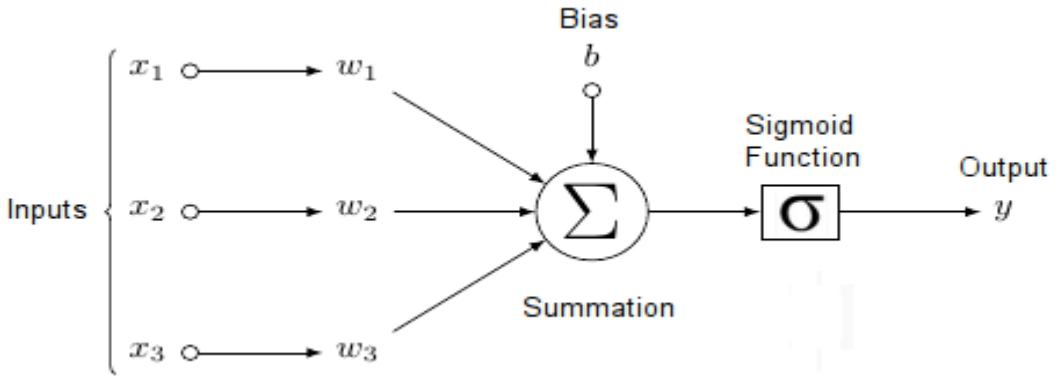


Figure: computational graph of a single neuron [3]

The third critical computational unit to make up a Neural Network is the optimization algorithm that is responsible of adjusting the Weight- and Bias-parameters. The parameter optimization is the actual 'learning' that occurs in the network and is therefore the single most important and essential idea behind ANN's and is based on the 'error' between the produced output and the real 'class'-label. Commonly in Artificial Neural Networks the most common optimizer algorithm has been Back Propagation, which 'propagates' backwards from the output layer after calculating the true error in the output layer and then calculates the expected values for the hidden layers and adjusts the parameters accordingly. In more recent times especially with the widespread use of Deep Neural Networks (like in our work) the optimizer algorithms vary and comes in different forms depending on the type of Deep Neural Network in use.

5. Reference studies and common methodologies

In the beginning of our project we were provided a reference study by University of Nagoya [1] as a guideline, which is provided in a format of PowerPoint presentation. The study introduces Deep Neural Networks (DNN) as the machine learning method for a problem of time-series forecasting both directly (regression) and by classification.

Forex market being the financial market with highest liquidity of trade each day, it attracts research in high numbers, and studies trying to find good prediction results in forex markets using machine learning techniques are not hard to find. We took a close look into several researches conducted by universities all around the world and intuitively we can list a few points common to all of them:

- ❖ They all use some form of Artificial Neural Networks.
- ❖ Features are pre-defined and selected highly intuitively based on various statistical formulations of the 'raw' input data
- ❖ Prediction accuracy is relatively low (ranging between 40-60 % for classification problems)

The approach for forex prediction can be listed as a sequence of steps in quite straight forward manner, which is common to all studies we encountered, regardless of the actual size of the unit (hour, day, month):

1. Find and utilize a data source (that includes market open, close, high, low and volume data)
2. Calculate additional statistical and technical fields based on the 'raw' data and add them to be features of the dataset. (this is the first differentiating factor between studies)
3. Find or pre-define optimal 'time-frames'. Time-frame is the sequence of data that leads to certain event (being predicted) (this is the second differentiator)

4. Define, train and test the ANN (the third differentiator)
5. Evaluate. Some use a ‘trading strategy’ simulating the potential profit they would generate using their model and others prefer to simply measure their ‘accuracy’ in various statistical ways.

5.2 Training conditions of the case study

Our case study from University of Nagoya originally has two initial approaches for the problem: direct value prediction (regression) and binary classification (going either Up (1) or Down (0)).

They list only one assumption, which is; future trend consists of past information. We interpret this as meaning: The future value is a result of a sequence of past occurrences, where the length of the past and the order matters.

The case study applies Deep Neural Networks, which is a type of Artificial Neural Networks that traditionally consist more than 3 hidden layers and because of this, it can adjust to more complex functions. As the activation function they have selected a Logistic function (Sigmoid) to map the output of a neuron to a range of [0, 1] which supports both regression and classification problems.

The dataset in use in this study includes 97,362 instances, in format of hourly exchange rate data between Japanese Yen and US-Dollar from 1991 until 2014. The ‘raw’ data has the following values: date, time, open price, high price, low price, close price and transaction volume.

The ten core features they use in this study are the six raw-values (datetime, open, close, high, low, volume) and four additional statistical descriptive features (14-day moving average, RSI, Stochastic RSI, Williams %R). After processing this, the study suggests that they concatenate features with time-dimension to produce 100-dimensional feature-space. However, this part was very unclearly explained in the

PowerPoint and has no similarities with methods in other research we came across, which forced us to leave concatenation in our own use-case.

The case study used pre-training for the original model in the form of Restricted Boltzmann Machines with model parameter settings including learning rate: 0.002, momentum: 0.9, batch-size: 128 and epoch: 30. The reason for using pre-training for quite extensively trained model like theirs is unknown, and other researches of the field did not use pre-training frequently.

Model training parameters that the study mention include learning rate: 0.00006, momentum: 0, batch-size: 128, epoch: 50. Number of layers they mention to be 5, but do not specify whether it includes the input and / or output layer (commonly: # layers = hidden + output). Number of nodes (neurons) in the middle layers they declare to be 256 within 5 layers.

5.3 Testing conditions of the case study

The input for the testing round is the features of a certain point in time, and the output is for regression: the next time closing value, and for classification: the next time trend (Up or Down).

According to the case study's picture illustration (on page 24) it seems like the strategy for training/testing split is to first train from 0 to n point in time and then test (predict) n+1. They do this iteratively training the same model and moving one point forward with each

iteration so that they end up testing multiple instances in sequence without using those instances in training before they are being tested.

The case study didn't explicitly specify the number of instances they use for each iteration for training, but based on the illustrations from page 24 onward we estimate it to be some 46 451 instances for testing on the first experiments. This is the result of the fact that they draw an arrow pointing from 0 to 6th point in a graph that is in 1: 10^4 scale and as they mention, has 31 hours instead of 24. Then they declare the number of tests being 51516 of 96366 instances. Based on this we can find that 96366 - 51516 = 46 451 which is quite close to the ratio of an arrow on the graph.

This test setting is quite unusual for DNN use-cases but is actually the correct one to use based on the assumption. The nature of time-series forecasting sets some limitations and forces the train-test setting to be done in different manner if you believe that time series data points are not independent but results of previous occurrences.

5.4 Outcomes of the case study

After running exhaustive testing rounds with up to 51516 tests, the study was able to reach 50.40 % - 53.46 % accuracy on binary-classification case predicting the close-value of the next data movement. On regression they didn't specify accuracy but graph movement was showing high in-accuracy with the initial settings.

The case study then moved to other direction in their train-test settings. They shifted to predict only short term trend transitions predicting only binary classification with shorter time-frames, but didn't really specify any information of the actual changes in the

settings other than graphs. They did mention a value ‘threshold’ that they set to 0.8 for ‘Up’ and 0.2 for ‘Down’. How we see this is, if output y is anything between 0.8 and 0.2 there is no prediction, and when it hits these thresholds, the prediction has enough confidence and will be conducted with the corresponding binary class. With this sort of setting they were able to predict with up to 94 % accuracy with much lower number of test-iterations.

Since the shift in train-test settings doesn’t have any clear documentation, it is hard to try to replicate this case’s train-settings.
So for the latter part of their study there is less comparison we can conduct between our studies.

5.5. Initial approach

In the first stage of our project we wanted to follow the University of Nagoya's case study as closely as possible to be able to draw conclusion of both their and our work's accuracy and similarities. After that on the second stage we would branch off and systematically try to find better solutions to build a better model for Forex prediction.

Since several critical parts in the settings were not clear in the case of referred case study, we had to make some assumptions along the way. Also, some settings like the selected dataset is not necessary to be exact same, as the Artificial Neural Network model should adjust to describe the patterns of any sequential currency data roughly the same way.

A major difference we wanted to draw to the original study is that we are not trying to perform direct prediction (regression), but only binary classification (Up: 1 or Down: 0). This is drawn from the fact that case study didn't perform well at all for regression problem and the latter part of their study even they choose to stick to just binary classification. Handling a regression problem generally is a more complex problem and has higher requirements for parts like feature selection. As the time wasn't enough to dig deeper into that field.

5.5.1 Data preprocessing

Our data was provided by a private exchange broker called Dukascopy, a Swiss forex bank and market place, and it consisted of hourly data generated between 13.2.2017 and 13.10.2017 including a total of 5833 instances.

Few pre-processing modifications were necessary before continuing with this task. First one was to convert datetime column into different format. In our case, exact date and time format doesn't serve

our purpose as valuable training feature, since they are all unique. We are using less than year's span of data, so even removing only the year doesn't serve our purpose as all months make all instances to become unique again. In order to let our neural network to detect patterns between time-points on different weeks (we assume that weekday's and hour's has common movement patterns) we encoded the datetime to represent only the week on hand and the hour on hand.

Encoding was conducted like this:

- Weekday (mon-sun) ranges between integer values 0-6
- Hour value ranges between 00-23
- As we remove year and month data, the data gets a format of WHH where W is single digit representing weekday and HH is two digits representing hour. Example value on Sunday 23:00 would be represented as 623

Next preprocessing that was needed was to remove the high frequency of noise in the data. We found out that the broker that provided the data had listed all the trading hours even outside of their active hours during the week and weekends. This resulted the service to have a high frequency of instances with 0 - transaction volume and no change in the price movement. We tried experiments with this data which accounted as very good accuracy (up to 70 %) because our training settings was able to detect these trends given their high frequency in the data.

After removing this 0-volume data we are left with 3785 training instances. Then we calculate four statistical descriptive features derived from the raw-data to better represent the time-series movement (following the example of the case study). These features are namely 14-days moving average, RSI, Stochastic RSI and Williams %R.

Class label is then given to each row depending whether the ‘close’ price in the next instance is higher or lower than the ‘close’ price of the current instance.

5.5.2 Training conditions for our initial approach

Following assumptions defined our initial approach for the forex classification problem:

- Fundamental assumption of the prediction problem: Future value is a result of a sequence of past occurrences, where the length of the observed past and the order matters.
- Number of layers and distribution of neurons between layers: the case study stated only that they are using 5-layers with 256 neurons in the middle layers. Commonly when talked about number of layers, it includes the hidden-layers and the output layer which means 4+1 = 5 setting in this case. 256 neurons were distributed evenly between hidden layers which left us with 4 hidden layers each having 64 neurons in this initial setting.
- Training-testing split:

  the case study had data consisting a total of 96366 instances. They seemed (unclear, so we have to assume) to use 46451 for training before each test, which means they used roughly 48 % of the total dataset for training.
- Some noise (or misclassification):

  we are only classifying test cases under 2 classes, Up and Down. In reality the change between two day’s close prices can also be 0, which means the price actually ‘stalls’. We classify 0-changes under the label ‘Down’. This adds little ‘misclassification’ noise but is in fact small in numbers.

The type of classifier in use here is a 5-layer DNN classifier with 4 hidden layers and an input and output layer. Each of the hidden layers has an even distribution of 64 neurons per layer. Activation function in use here is a sigmoid function that maps its output to be in range between [0, 1].

Our dataset differs from case study. As applying this classification problem in Thailand for the purpose of a Senior Project in a University, it would be more suitable to localize it to consist local currency. Our data is exchange rate between US - Dollar and Thai Baht from start of 2017 until autumn of 2017 consisting of 3785 instances preprocessed as described in above section. For training we used 1821 instances.

The features we use for training are the 10 core features described in the pre-processing description. We did not further concatenate the features with time-dimensions following the case study, as it was left unclear what they exactly meant by that. We didn't pre-train the data either.

Following training parameters were specified according to case study:

- Learning rate ⇒ 0.00006
- Batch-size: ⇒ 128
- Number_epoch ⇒ 50
- No momentum

Comparison between case study and our initial training settings:

T1	Instances in dataset	Train / % train	Features	Layers	Neurons (total)
Nagoya University	96,366	46,451 / 48%	10 (concat to 100)	5	256
Assumption University	3785	1821 / 48%	10	5	256 in hidden layers

T2	Activation	Pre-training	Learning-rate	Momentum	Batch_size	No-epoch
Nagoya University	Sigmoid	RBM	0.00006	no	128	50
Assumption University	Sigmoid	no	0.00006	no	128	50

5.5.3 Testing conditions for our initial approach

For conducting the test we followed the case study. The test settings were actually the appropriate kind for this kind of problem setting. In the test setting we iterate *t* times training the model with the training settings, each time using *n* instances for training, after which we test (predict) the label for instance *n*+1. The number of iterations *t* is the number of tests which in our case was 400 for one test, but we

repeated the test 4 times to cover 4 discrete ranges in the dataset. Therefore in total we conducted 1600 tests.

5.5.4 Outcomes of our initial approach

The case study was initially able to reach accuracy of 50.40% to 53.46 % with test sets ranging from 744 to 51516. We ran 4 tests with initial test settings, each having 400 test instances and testing different parts of the dataset. We were able to get classification accuracy between 50.50 % and 54.75 %.

T3	# test instances	% accuracy
Nagoya University	744 - 51516	50.40 % - 53.46 %
Assumption University	400 - 1600	50.50 % - 54.75 %

5.6 Conclusion of initial approach

As the previous section shows, we were able to match University of Nagoya's case study almost exactly. This happened even though we had a few intentional changes (like the dataset) and some limitations we had following uncertainty of case study's methods.

Even though the results follow almost exactly the case study, they can be concluded as being bad results for the actual classification problem. Accuracy of up to 54.75 % in binary classification can be

compared to accuracy of tossing a coin on each round, which would produce roughly the same results.

It was clear even during the construction of the initial study that we wanted to do changes and optimize our training conditions several ways. The results from this part provides us a good comparison point to improve the accuracy in our actual classification problem.

From now on, we are not going to follow the University of Nagoya's example but try to find higher prediction accuracy in our novel systematical ways.

6. Development process

We keep the following assumption throughout the following development stages:

- Fundamental assumption of the prediction problem:
  Future value is a result of a sequence of past occurrences, where the length of the observed past and the order matters.
- Number of layers:
  We assume 5 layers provides a good approximation for our problem's complexity and we do not intend to change this

6.1 Finding the optimal number of neurons

Since our artificial neural network works like a 'black box' where the hidden layers job is to model the problem concept by learning from labelled examples, the setting of the network's layers is probably the single most important configuration in our network.

The first development step for us is to find the optimal settings for the number of neurons in each layer. We are not going to change the number of layers, and the number of neurons in each layer will be something from the set {4, 8, 16, 32, 64, 128}.

Our method for finding this is simply to run an independent, standard test case for each possible permutation in this setting. It is critical to underline here that here, it has to be permutations, as the order of neurons matters greatly. The process will be explained in more detail in section 8.

6.2 Finding the optimal number of training instances

The number of training-instances to use for each test is another interesting differentiator in our learning problem. If we approach the problem with intuition, we might say that for predicting one instance ahead from past x instances, the size of x should rather be shorter than longer. But we don't really know that so we have to test it and draw the conclusion based on the facts.

We decided to base the test on constant parameter settings, and run equivalent training-testing cycles for 16 different lengths of training instances, namely {30, 60, 90, 120, 150, 180, 250, 300, 400, 500, 750, 1000, 1250, 2000, 2500, 3000}. The numbers are selected intuitively rather than based on some permutation or other selection method like in the process of selecting the neuron-settings per layer. The process is explained in more depth in section 8.

6.3 Finding optimal time for prediction with optimized number of training instances

This test is focusing on finding out finding out the accuracy of predictions at each point of the time within a day with different number of training instances set. We are expecting to see if the predictions is consistent within some hours of the days.

We decided to run a sample of 500 sample with different instances, namely {30, 60, 90, 120, 150, 180, 250, 300, 400, 500, 750, 1000, 1250, 2000, 2500, 3000}. The test will be focusing on the hourly predictions from each training instances.

6.4 Research findings (and value of the study)

The research stage began with building a stable model. To begin with, we built the model by using Estimators from TensorFlow™. tensorflow.estimator is a high level API which is meant to be used for machine learning programming. It is enclosed of training, evaluation, predictions and export for serving. The classifier we used for our experiment is DNNClassifier which is inherent from Estimator.

6.4.1 Optimizing number of neurons

In order to find out the optimize number of neurons for our neural network, we've decided to set the layers to be four and running the tests by replace number of neurons in each layers. We came out the systematic approach to for changing number of neurons in each layer. It will start from {4, 4, 4, 4} neurons in each layers. For each round of the test the number of neurons will be change to 8,16,32,64 up to 128 neurons in each layers. The possible round of the test is:

$$\textit{Permutation (6, 4)} = \frac{□!}{(□ - □)!}$$

6.4.1.1 Setting for the test

- Permutations (6,4) ⇒ 360 possible rounds of testing
- Dataset ⇒ 3,785 instances
- Training set ⇒ 100 instances
- Testing set ⇒ 100 instances per one permutation
- Optimizer ⇒ Gradient Descent
- Activation func. ⇒ ReLU
- Number of epoch ⇒ 50
- Batch size ⇒ 38
- Optimization steps ⇒ (100 / 38 * 50)= 198 steps

6.4.1.2 Findings and analyses

The accuracy of the tests are being measured as how many predictions are being correct in the sample size of 100. Our predictions involved whether the next point of time if the *close price* of currency pair is going to increase or decrease.

The total duration to run the tests with above setting for all possible rounds took roughly 35 hours. We split the possible combinations of permutation into three sets and ran with 3 computers. Each computer took around 11 hours. The result of tests are summarized in the graph below.

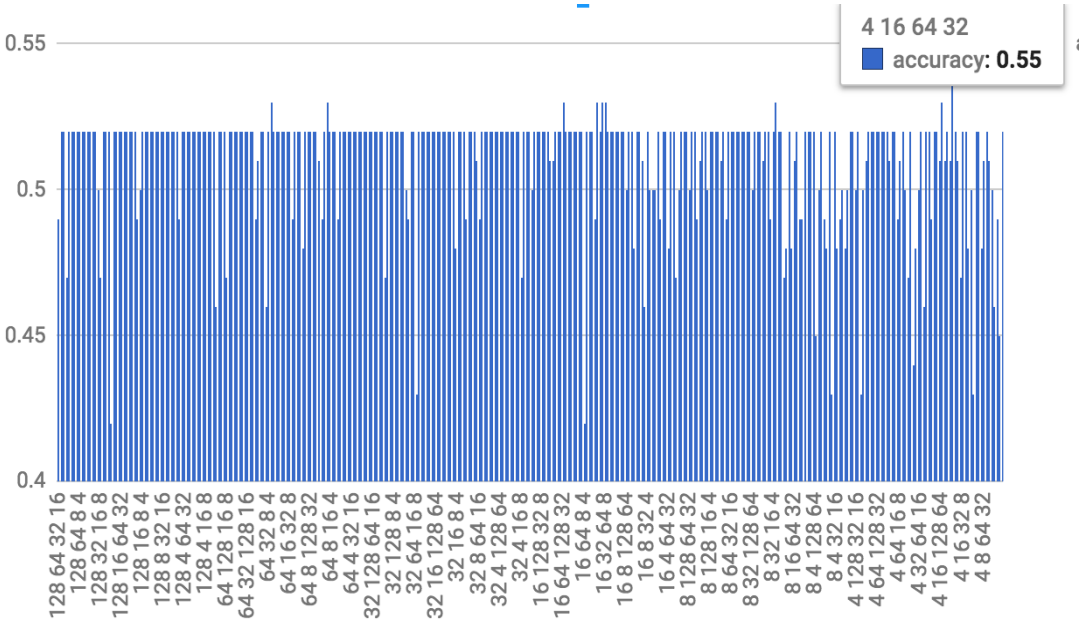


Figure: Result for optimizing number of neurons

- **Basis statistic**
  - ➢ Mean ⇒ 0.51
  - ➢ Mode ⇒ 0.52
  - ➢ Range ⇒ 0.42 ~ 0.55
  - ➢ Best Result ⇒ { 4, 16, 64, 32} with 55.00%

From the statistic number we obtained above, {4, 16, 64, 32} set of neurons gave the best result with 55.00% accuracy. We also noticed

that the order of layers took an important role in the accuracy of the result which means the order in {4, 16, 64, 32} can't be changed. It has to be in the same order as mentioned above. We will be using the neuron set with {4, 16, 64, 32} for all the future testing.

6.4.2 Optimizing accuracy with number of training instances

After we found out the neuron set of {4, 16, 64, 32} gave us most accuracy of result with 55.00% accuracy. We wanted to improve our result as much as we can. We have modified a few setting of parameters to run the test again.

For this test the main focus is how many of training instances should we use to obtain optimized results. We ran the tests with following number of training instances:

Training instances: [30, 60, 90, 120, 150, 180, 250, 300, 400, 500, 750, 1000, 1250, 1500, 2000, 2500, 3000]

6.4.2.1 Setting of the test

- Number of rounds ⇒ 17 rounds with each training instance
- Dataset ⇒ 3,785 instances
- Testing set ⇒ 400 instances of sample size
- Optimizer ⇒ ProxmialAdagradOptmizer
- Activation func. ⇒ ReLU
- Learning rate ⇒ 0.00006
- Number of epoch ⇒ 50
- Batch size ⇒ 128

6.4.2.2 Findings and analyses

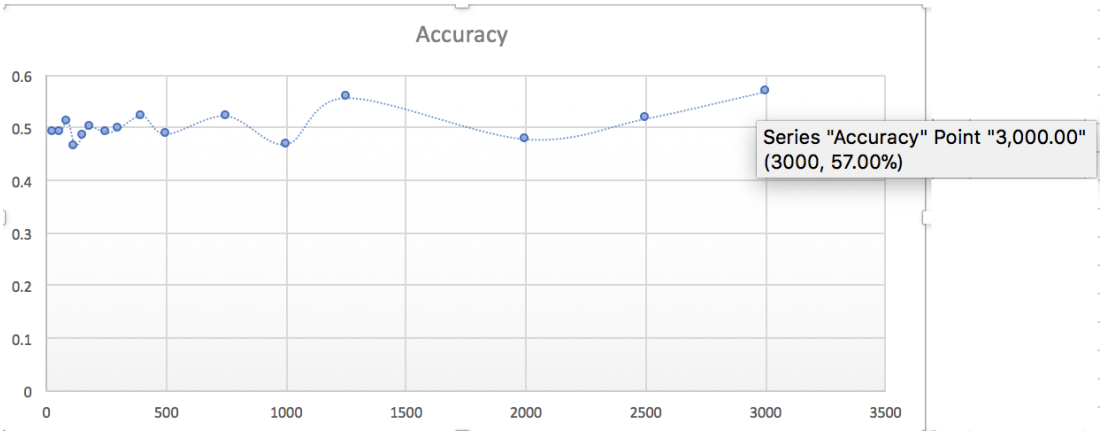


Figure: Optimizing accuracy with training instances

- Basis statistic
  - Mean ⇒ 0.50
  - Mode ⇒ 0.49 & 0.52
  - Range ⇒ 0.46 ~ 0.57
  - Best Result ⇒ ***3,000 with 57% accuracy***

From the statistic results that we've obtained, we were able to get 57% accuracy with 3,000 training instances. 57% accurate was a breakthrough result from what we have been trying but in reality 57% is not good enough. We were trying to look back at all the settings from this test to find out what else could we change to make higher accurate predictions. From this test, we've notice that the learning rate is relatively low as compared to the default learning rate (0.1) set by TensorFlow™ group. Some of these parameters doesn't seem to be well optimized for our use-case and intuitively it seems like the best result from this round is drawn most likely by the simple reason that 3,000 training instances provides quite a lot more rounds for parameter adjustment.

6.4.3 Training parameter optimization based on intuition and previous test findings

We had some presumptions about some of the training parameters that did not seem to match our expectations and of good design to model the problem. These parameters are namely epoch, batch-size and learning rate. We ran several unsuccessful tests after the last reported test (8.2 - optimizing the number of training instances), and based on the last reported test and the unsuccessful ones, there was a few patterns to detect:

- Adjusting the named parameters don't much improve accuracy
- Along with adjustment, optimal number of training instances becomes smaller

As the value of the above mentioned attributes are not the most critical factor in of our neural network design, we didn't want to go for exhaustive tests optimizing these parameters, but instead optimize the parameters intuitively to get the optimal number of training instances to get somewhere closer to the mid-range of our training-instances.

The following values for the concerned parameters were used in this round:

- Num_epoch ⇒ 250
- Learning rate ⇒ 0.0006
- Batch size ⇒ 38

The intuition driving for the selection of these values was mainly that from the multiple previous tests, it seems like shorter steps are better for learning than longer steps and that the network needs more

steps for learning (optimizing) the parameters than what was previously provided (derived from the case study).

Simply put, we wanted to provide our network more time to adjust the weights but at the same time make sure the optimization steps are not too large and over time affect negatively for the learning process.

6.4.4 Finding optimal time for prediction with optimized number of training instances (hybrid approach)

In this approach we wanted to find if there is some specific time-periods with higher accuracy than in the other time-periods. This is particularly interesting due to the fact that our data set has some 'gaps' after removing hours without any trading volume. We assume that during those 'gap'-hours the trend signal from the previous known data will weaken with the length of the 'gap' and therefore result in worse prediction.

As we had not disclosed the absolute optimal number of training instances, and to run more test-cases for better reference, we decided to run the same test of finding the 'hourly' prediction for multiple different numbers of instances.

6.4.4.1 Test setting

- Tested train-instance numbers: [30, 60, 90, 120, 150, 180, 250, 300, 400, 500, 750, 1000, 1250]
- Dataset ⇒ 3,785 instances
- Testing set ⇒ 500 tests (for each train-instance test)

- ❖ Optimizer ⇒ ProxmialAdagradOptmizer
- ❖ Activation func. ⇒ ReLU
- ❖ Learning_rate ⇒ 0.0006
- ❖ Number of epoch ⇒ 250
- ❖ Batch size ⇒ 38

6.4.4.2 Findings and analysis

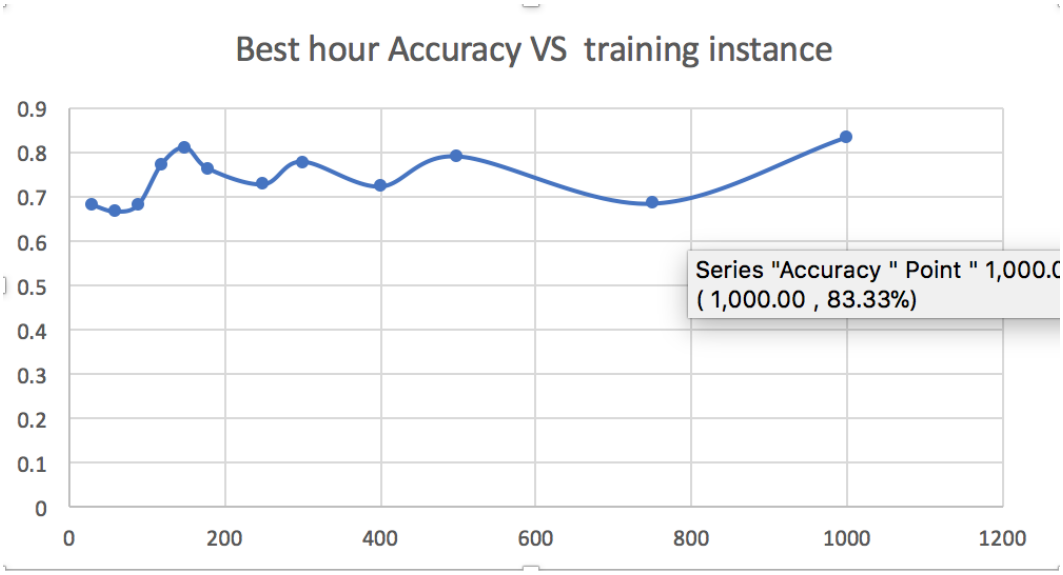


Figure: Best hour prediction from set of training instances

This test has some very interesting outcomes. Firstly, as expected, the optimal number of instances kept moving towards smaller number along with higher epoch for relatively small learning rate. Secondly, each number of training instances (except for the first, 30) produced overall accuracy of higher than 50 % which is in general more consistent than before in our tests. However, these two are just more of side-notes than main observations compared to the true outcome.

One major note is that, by dividing the predicted results with the corresponding hours, we were able to find some regions in the time-frame where accuracy was particularly good. Out of which there were 2 cases where the predictions were over 80%. One set with 150 instances which produced 80% accuracy at 19th hour from the sample and another testing set with 1,000 instances which has produces 83.33% at 23rd hour from the testing sample. There is one another good finding where a 3-hour time-frame between hours 17-19 where average accuracy was over 68 % (with 66 tests). These findings suggest that as hoped, our classifier is able to detect consistent trends in the market.

To conclude this test, we were able to find out that the best accuracy predictions occurred at 19th and 23rd hour of the day with over 80% accuracy. Due to the reason we used a data provider from Switzerland to train and predict, the timing are presented in Switzerland as well.

Training instances	Hour of day	Sample size	Correct predictions	Accuracy
30.00	13	22	15	68.18%
60.00	8	21	14	66.67%
90.00	17	22	15	68.18%
120.00	16	22	17	77.27%

150.00	19	21	17	**80.95%**
180.00	8	21	16	76.19%
250.00	11	22	16	72.73%
300.00	4	18	14	77.78%
400.00	5	18	13	72.22%
500.00	2	19	15	78.95%
750.00	2	19	13	68.42%
1,000.00	23	18	15	**83.33**%

7. Evaluation & assessment

Our work has had three core goal-components in it. The first goal was to learn and apply machine learning in the problem-setting of currency exchange prediction following a case study of the field. The second goal was to then branch-off and find ways to improve the results and find interesting outcomes by making independent tests. Third goal was to apply this concept to some real-life use-case to show the applicable value of the work.

7.1 First ANN-learning problem and the case-study approach

After extensive study of the topics from academic literature, practical tutorials and research papers including our case-study, we were able to break a problem like this into parts to understand the core-components of the task, and the techniques, in order to develop our first ANN-learner. This followed as closely as possible the case study we

had assigned, but due to the highly cryptic nature of their presentation, several assumptions and more thorough understanding of the concepts were required.

We were able to build a system that is almost exactly the same range in terms of accuracy as our case study. The results in range 50-55 % were as expected and provided good foundations for deeper level experiments for the next rounds.

7.2 Improvements towards better prediction accuracy

After the first stage was check-marked, we wanted to move to more systematical testing with already existing attributes that would reveal more about the behavior of our learning problem in different settings.

We purposefully left features untouched. Even though the features probably are (and we are aware of it) the single most important part of a machine learning problem, the foundations the three of our group members had of Forex markets, didn't provide sufficient enough base to really dig into really valuable analysis of different technical analysis and statistical evaluation of currency movements. Someone else had done the groundwork for this (our case-study) and with the given time, our additional contribution to this would have added no value.

The results were delightfully up-trendy. In very short time we were able to simulate dozens different test-scenarios revealing value like the best neuron-settings per layer and the somewhat optimal number of training instances and based on them good valued tests were drawn. Especially after making the assumption of our dataset including 'gaps' that lead us to make observations based on each hour.

Some hours were proved to have really impressive prediction accuracy. For some individual hours there was even over 80% accuracy shown, and even for 3-hour frames we could find 68% accuracies. This sort of accuracy can be valuable even for commercial purpose, and should be considered a good point for motivation for further study of the field in the future as well.

7.3 Application use-case

The third goal was to develop an application that can actually utilize this kind of use-case. We developed an app as a combined course-effort (with same team-members) to be shown in our Hybrid Mobile Application course and the Senior Project 1 assessment.

8. Architecture of application use case

To show our machine learning algorithm in real life application, we have developed an application to track currency exchange rates, providing customization in both settings and view, and provides a trend prediction for 5 different currency pairs. The application consists of two parts: 1. Python back-end to produce and update the latest data; 2. Ionic-framework front-end to provide the user interface and functionality to utilize the data we produce.

8.1 Back end program

The back-end program consists of a few python scripts to retrieve and update simple ‘daily’ data, a machine learning - model training and testing, and the google firebase cloud - database to keep all this updated data available for the front-end program.

The python back-end has a script to just update firebase db’s ‘historical’ price details which consists only of adjusted daily prices from last 7 days. When this script is triggered, it retrieves the 10 hard-coded currency pair’s data from the past 7 days and overrides Firebase NoSQL historical-price datastring with those values.

The machine learning-part of our back-end utilizes the exact same latest version of the model that we have been using for our experiments, but in the application use-case we are not using hourly, but daily data, which is absolutely just as well applicable to our scenario, but not as extensively proved to belong the same accuracy ranges.

After the DNN-model has been trained in the python-backend, the program stores the model (for all 5 provided currencies) as a file

and then it can be tested and further trained daily. With each daily trigger of the test, the latest predictions for all currencies will be updated directly to another firebase NoSQL data string to be retrieved by the front-end.

Firebase serves as the direct API for our front-end app and handles a few back-end services beyond the already mentioned ones. It takes care of the authentication, providing social auth Facebook and Google to users to secure their data and provide customization options. It keeps three main NoSQL data strings for use, namely: historical charts, predictions, and user settings. Firebase backend services are also utilized as the host provider.

8.2 Front end program

The front of our application is built on top of Ionic framework that is meant for hybrid mobile application development, meaning that the application is built as a web-app with the common technologies but it provides native-application integrations with the device and could be distributed as a native application.

Our front-end application is intentionally made to communicate only with one core firebase source-db instead of multiple distinct API's. This simplifies the construction of the application interface and streamlines the content delivery.

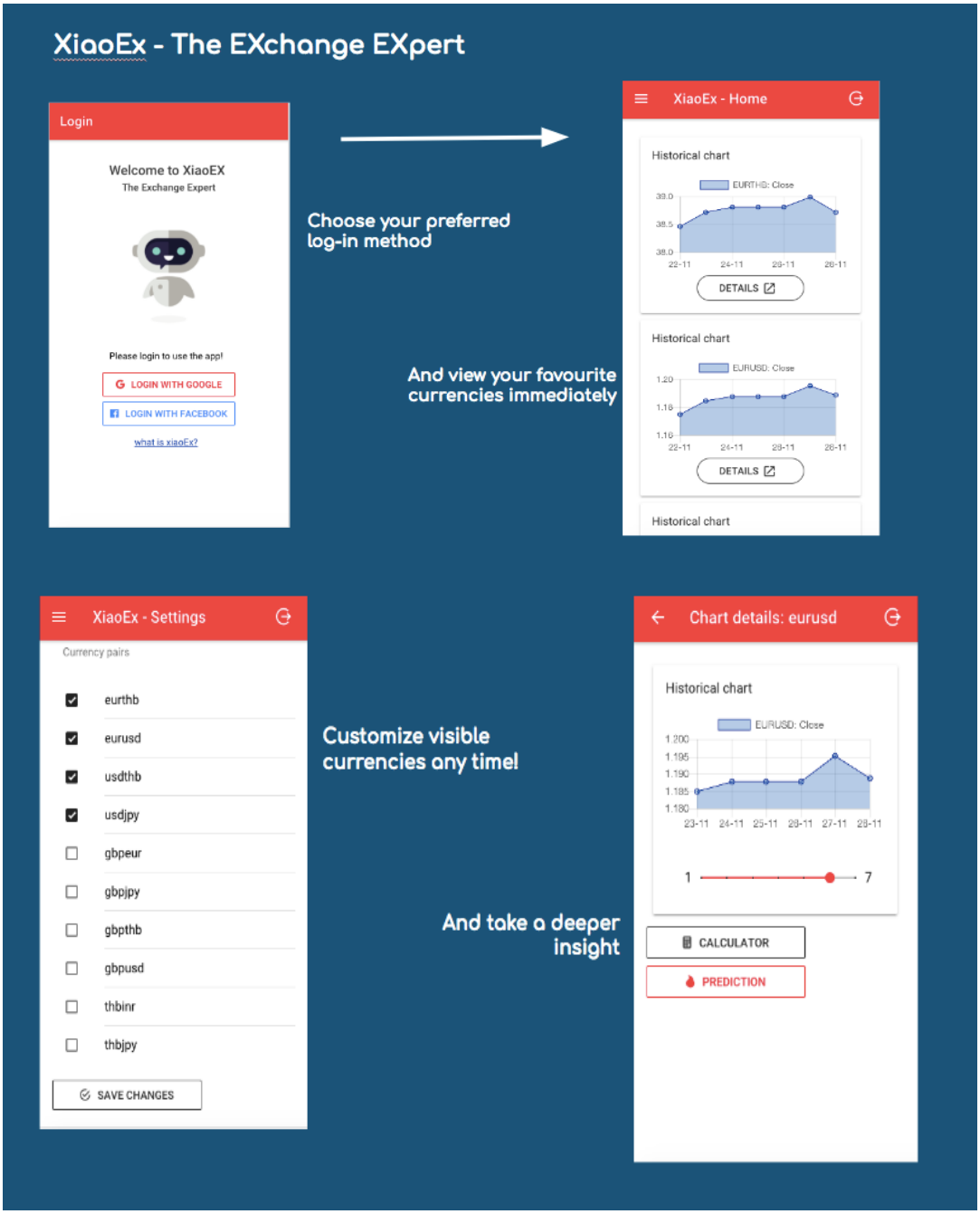
The functionalities we provide in the application:

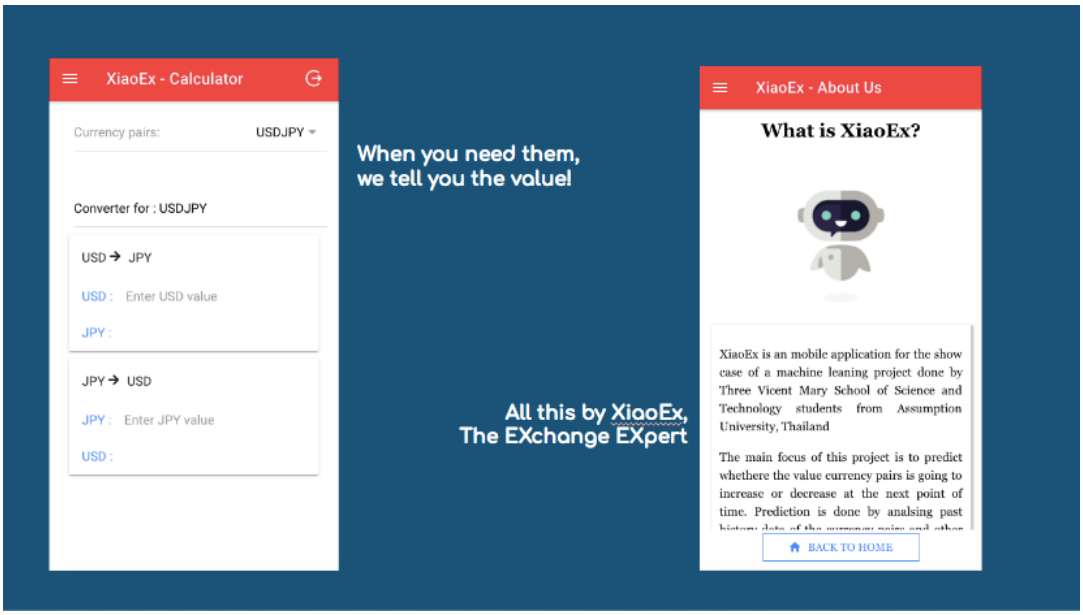
- ❖ login (app is not accessible without logging in)
- ❖ snapshot of selected currency charts (historical price chart)
- ❖ details of selected currency pair (scalable chart, options)

- currency calculator for selected pair
- prediction for 5 different currencies
- customizability (any number of the 10 pairs can be selected)

The application utilizes the main data, currency predictions, in the chart-details page where, if one of the pairs with available prediction, the screen will pop up a message telling the trend where it is moving next.

Appendix A - Application visualization




XiaoEx - Calculator
Currency pairs:
USDJPY
Converter for : USDJPY
USD → JPY
USD : Enter USD value
JPY :
JPY → USD
JPY : Enter JPY value
USD :
When you need them,
we tell you the value!
All this by XiaoEx,
The EXchange EXpert
XiaoEx - About Us
What is XiaoEx?
XiaoEx is an mobile application for the show case of a machine leaning project done by Three Vicent Mary School of Science and Technology students from Assumption University, Thailand
The main focus of this project is to predict whethere the value currency pairs is going to increase or decrease at the next point of time. Prediction is done by analsing past
BACK TO HOME

10 . References

- Prediction of Exchange Rate Using Deep Neural Network | @tachyeonz. (2016, August 09) from Nagoya University.
- Machine Learning By Tom M. Mitchell | Publisher: McGraw-Hill Science/Engineering/Math; (March 1, 1997)
- TensorFlow. (n.d.). Retrieved July 08, 2017, from https://www.tensorflow.org/
- Pandas 0.21.0. Retrieved December 06, 2017, from https://pypi.python.org/pypi/pandas