Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

Praca dyplomowa

Design, Implementation and Tests of an AI-Based System for Currency Exchange Rate Prediction

Zaprojektowanie, implementacja i testy bazującego na sztucznej inteligencji systemu do przewidywania wartości kursów walut

Author: *Dawid Dzhafarov*
Degree programme: *Information and Communication Technologies*
Supervisor: *PhD. Eng. Michał Grega*

Kraków, 2022

Table of contents

1 Introduction

1.1 Thesis Description

This work aims to extract the best features from similar experiments and use them in designing, implementing and testing new machine learning model architectures, that will be used to predict future values of ForEx exchange rates. Multiple configurable parameters will be taken into account during the design step, resulting in the most optimized model for chosen data and task. The models' performance will be measured by a loss function and test cases, which involves calculating correct predictions for the next day and 30 day-horizon, also with performance comparison against the baseline model.

1.2 Time-Series Prediction

Although there exist multiple theoretical financial models that help to understand how the exchange rates, stock market prices, crypto-currencies work and what aspects of social, political and economical situation they depend on, it is still impossible to achieve high accuracy when it comes to predicting them. Even the most sophisticated models that had been developed for years could not reach satisfactory results. Years go by, technology advances at an enormous pace, but it is still unlikely to correctly predict financial markets. What is the reason for it? The answer to this question is simple and rather disappointing - it is impossible. At least for now, with current knowledge of economics and possibilities offered by technology.

Time-series prediction of exchange rates (and similar issues) based on previous values have no fixed pattern that can be extracted from historical data by a machine learning model. There is no repetition in the data, so there is no way for any algorithm to learn patterns that even might not exist (or they are unknown at the current moment) [1]. What about adding some variables into that equation, like mentioned social, economical indexes? The problem with it is that adding them creates deeper dependencies that are not understood by models - this results in adding meaningless data and over complicating the model. Moreover, this type of data is seldomly updated in the same way as exchange rates - i.e. Consumer Price Index (CPI) is most often published as monthly type of data, so during one month the exchange rates of any pair of currencies may vary significantly but at the same time the CPI is constant. What is more, it is hard to represent the political situation of the world in numbers and adjust it in a very dynamic environment. These indexes might change even due to newspaper titles, posts on social networks, comments from influential people and many different things that people are not aware of.

1.3 History

The first attempts to predict exchange rates or stock market prices date back not long after the birth of machine learning, when the phrase was first used in 1959 by an IBM employee Arthur Samuel. At first, most of these works were based on applying strict mathematical and statistical models to predict future values. After the time when the term "Machine Learning" was first used by an IBM employee Arthur Samuel in 1959, the progress in this field majorly improved. When the algorithms started to be automated and easily applied due to the evolution of technology, the only obstacle was to come up with a "good" algorithm, that would meet researchers' expectations or exceed them. Since then, there have been many solutions proposed that can be split into three main categories:

- Classical Time-Series Models
- Supervised Machine Learning Models
- Deep Learning Models

Each of them has its own strengths and weaknesses, depending on what kind of task they are given. However, in the past few years, the last category attracted the most attention. In classical models, the most successful and recognized models are:

- Autoregression (AR)
- Moving Average (MA)
- Simple Moving Average (SMA)
- Exponential Moving Average (EMA)
- Autoregressive Integrated Moving Average (ARIMA)

The characteristic feature of all the above methods is that they can all be combined into one, standalone solution. As for supervised machine learning models used for time-series prediction, these are the most outstanding:

- Linear Regression
- Support Vector Machine (SVM)
- Random Forest

Proceeding to the very last category, deep learning algorithms for time-series forecasting that proved to be successful are:

- Long-Short-Term-Memory (LSTM)
- Gated Recurrent Unit (GRU)
- Radial Basis Functions (RBF)

Further findings in this thesis are based on applying algorithms from the last category, with an emphasis on LSTM networks.

1.4 Goals

This work's aim is to create a machine learning model for ForEx exchange rates time-series prediction that could beat the baseline average model. This baseline model predicts future value as the average of previous ones. It can calculate this value based on the whole available history, or it can be reduced to the desired time scope, i.e., 30 days.

It may be odd, but this solution is actually really hard to beat. The model design, creation and optimization will be executed on one pair of currencies: USD/PLN and then tested against the mentioned benchmark. It will also undergo a prediction test that relies on calculating possible gain (or loss) from predicting values in a certain period of time. There will be two types of this test, one with short-term prediction horizon (1 day) and the other with a longer prediction horizon (30 days). Lastly, the created model will be used to carry out the same tests but on different pairs of currencies (EUR/PLN and EUR/USD). The method that this work will consider is Artificial Neural Networks (ANN), more precisely Recurrent Neural Networks (RNN), such as Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU).

1.5 Overview

This work is composed of 7 chapters. The first chapter is an introduction to the time-series prediction problem and states the goal of this work. Chapters 2 and 3 give a depiction of related studies and contain theoretical information about used components. Chapter 4 in short presents used tools. Chapter 5 describes the implementation steps taken to create the final model, whereas chapters 6 and 7 cover the evaluation, results and conclusions.

2 Previous Works

Nowadays, it is not a secret that Recurrent Neural Networks deal with time-series prediction better than most other methods used in machine learning. It is due to the scalability and flexibility of RNNs that enables to build and develop solutions specific to certain problems, even the most unique ones. There are so many variations, tools and extensions that can be applied on top of previous findings that make the combinations almost endless. There are new algorithms invented frequently and besides that, the "old" ones are refreshed and redesigned to solve current problems.

The first attempts to predict exchange rates or stock market prices date back to the 1980s. As the possibilities of automating and achieving high computation were limited due to not sufficient hardware capabilities, the first reliable approaches that tackled this problem using machine learning were documented in the 1990s. [2] was one of the most interesting and groundbreaking propositions for using neural networks in such a task. Although it used a basic perceptron network with careful choice of a squashing function and parameters, it allowed to achieve promising results. The next big milestone was the invention of LSTM neurons in 1995. This triggered a wave of continuous discoveries and improvements in this field, as it is still happening to this day. Most of the articles and publications that proposed new methods in time-series forecasting were (and still are) based on LSTM.

Most recent works, such as [3],[4], prove that the invention from 1995 is still regarded as a leading and state-of-the-art method that deals with the most current problems. In these articles, built networks are capable of predicting gold prices and the production of oil. This suggests to what extent LSTM networks can be used. With the right choice of tools, data and environment, these types of models can be built from scratch, without specific knowledge of underlying complexity. The amount of configurable parameters and features, number and type of layers and neurons make LSTM-based networks versatile and dynamic, allowing to introduce new capabilities in a wide range of topics.

However, despite such great innovations and possibilities, there is one big problem that every researcher should be aware of - with current knowledge of economics and technology, it is yet impossible to achieve results that would guarantee a profit or very high accuracy (or even guarantee not to lose). Such a model does not exist and probably never will. Thinking about what would happen if it did, maybe it is for the best. But that does not stop the next generations of researchers from seeking better algorithms that could prove to be useful in other aspects of human life, if not in time-series forecasting.

One of the examples of a useful application of the time-series prediction model

is the work submitted in 2020 by Zahra Karevan and Johan A.K. Suykens.[5] The model they have proposed is used to forecast the weather based on history. In this kind of problem, one important thing has to be highlighted - most recent days have more impact on the future value, which is why the LSTM they have used is called "transductive". The results show that applying this methodology proved to be better than other similar methods, like inductive LSTM or a standard one.

Another application of time-series forecasting is presented in [6], where it is used to predict patient volume in a hospital. This kind of research could help medical institutions to prepare better for such occasions, i.e., providing sufficient equipment, enough medical staff or more availability during the most occupied time. The proposed ARIMA model outperformed the baseline model and proved to be a valid and correct solution.

Stock market prediction is a very similar dilemma to the one investigated in this thesis. There were and will be many new attempts at predicting those values. In the work [7] different models are used to forecast stock market prices in the Indian market. Different validation methods are used and the results confirm that the existing models can be better than the simple average.

2.1 SVM

Support Vector Machine is the type of supervised machine learning model, mainly used in classification problems, that was developed in the 1990s. Although it has achieved great results in the mentioned field of machine learning, fewer people are aware of the extension of this method, called Support Vector Regression (SVR). As the name suggests, it is based on regression that can be applied in the multi-dimensional environment - meaning it can do much more than ordinary regression models. As Aleksander Palikuca and Timo Seidl showed in their research [8], SVR-based models can be as good as deep learning methods. The key to achieving these results is adequate preparation of data for the certain time horizon and tuning parameters. This process can be lengthy, as there are many possibilities. In their case, the data points were of high frequency, since the time horizon of a prediction did not extend 60 seconds. These short time intervals usually are not linked to big fluctuations in currency rates, so the work focuses on achieving high precision, rather than long-term predictions.

2.2 Back-propagation

Back-propagation is one of the most important algorithms that pushed machine

learning further, to the point as it stands now. It was used in the early 1980s, though the idea behind it dates to the 1960s. The algorithm uses computation of the gradient of the loss function from the last layer back to the first one, considering each weight for each combination of input-output neurons. Then it adjusts the weights and biases in order to minimize the loss function.

The method became very popular and for quite some time, it was considered one of the best for certain kinds of problems. It is the base mechanism in most of the developed algorithms, since there has to be a way for a deep learning network to propagate current and previous information across all the neurons, so the appropriate weights and biases are adjusted.

In the paper [9], the author uses a basic back-propagation algorithm in the prediction of exchange rates and compares it to other methods, such as the LSTM network, Radial Basis Function and SVR. The results show that it is outperformed by every other method, especially during the comparison of errors each method made. The reason why this is happening is that all other methods are equipped not only with basic propagation of information, but mechanisms that can carry more previous information to the next data points, which enriches the decision-making process. Basic back-propagation does not have the capability to retain distant history and use it in the future; it is only based on the most recent observations.

2.3 LSTM

Long-Short Term Memory is an implementation of a neural network that uses LSTM neurons. As of 2021, according to *Google Scholar*, it was the most cited term within the previous year, in many different aspects of human life, including healthcare. This shows how much popularity this solution has gained over the last 20 years.

It was first announced to public knowledge in 1995, in the technical report written by Sepp Hochreiter and Jürgen Schmidhuber. In 1997 the main LSTM paper was released by the same authors [10]. By introducing Constant Error Carousel (CEC) units, LSTM dealt with the vanishing gradient problem, which was the biggest obstacle for neural networks. The problem is that during the back-propagation phase, the further the information goes through the network layers, the gradient becomes lower and lower, to the point that it has no impact on the weights and biases, so they can not be properly adjusted. With this enhancement, this solution was used more and more often, to the point where big technology companies started to use it in their businesses.

As an example, Google started using an LSTM network trained for speech recogni-

tion on Google Voice. Also, Facebook performs around 4.5 billion automatic translations every day using long short-term memory networks. OpenAI, the leader of global Artificial Intelligence companies, used LSTM network trained by policy gradients to beat humans in the complex video game of Dota 2. The same technology was used to control a human-like robot hand that manipulates physical objects with unprecedented dexterity. Another AI-focused company, DeepMind, used LSTM to excel at the video game of Starcraft II.

2.4 LSTM-GRU

One of the strengths of LSTM is that it is very versatile when it comes to using it with other types of neurons. All the trainable parameters, together with hyperparameters, allow to create models that can be used in the vast majority of fields. The experiment carried out by M.S. Islam and E. Hossain [11] presents a model that consists of LSTM and GRU units in a hybrid network. The results described in this article state that the custom model outperforms standalone LSTM and GRU models.

Although only one architecture was evaluated, it shows that there are many possibilities to create different models with different architectures, in order to find a better solution. What is worth mentioning, when tackling such tasks, one has to bear in mind that not only architecture or algorithm is the most important factor in the potential success of a model. Data processing and the data itself play an important role. Before making a decision regarding the structure of a network, one has to answer several crucial questions: What kind of data to use? What frequency of data points to use? Where to get from reliable data? How to prepare the data to feed into the network? Going through these dilemmas first may save one's unnecessary effort and waste of time in later stages of research.

3 Recurrent Neural Networks

The biggest difference between implementations of Recurrent Neural Networks (RNN) and densely connected networks or convolutional networks is that RNN units have memory. In convolutional networks, in order to train the network, one has to feed the network with whole data point - i.e., whole picture. Then this data point is processed at once. In RNNs, there is an internal loop, that connects outputs to the previous inputs, allowing the network to process a single data point iteratively. In the end, RNN does not process a single data point at once, but rather it loops through the elements of each data point and persists its state. If the next data point is provided, the state from the previous processing is used in the calculation of the next state and the whole algorithm starts again. Figure 1 presents the general flow of data in the network with recurrent connection, while figure 2 depicts how it looks like "inside" the network, that is, between the corresponding neurons.

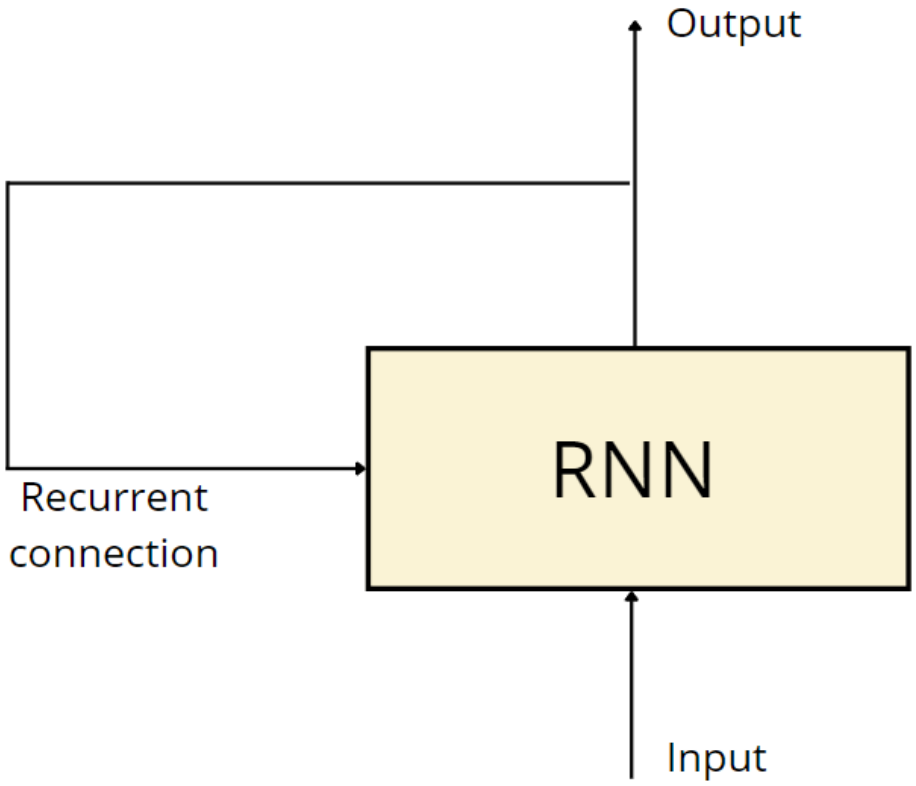


Figure 1: Simple RNN

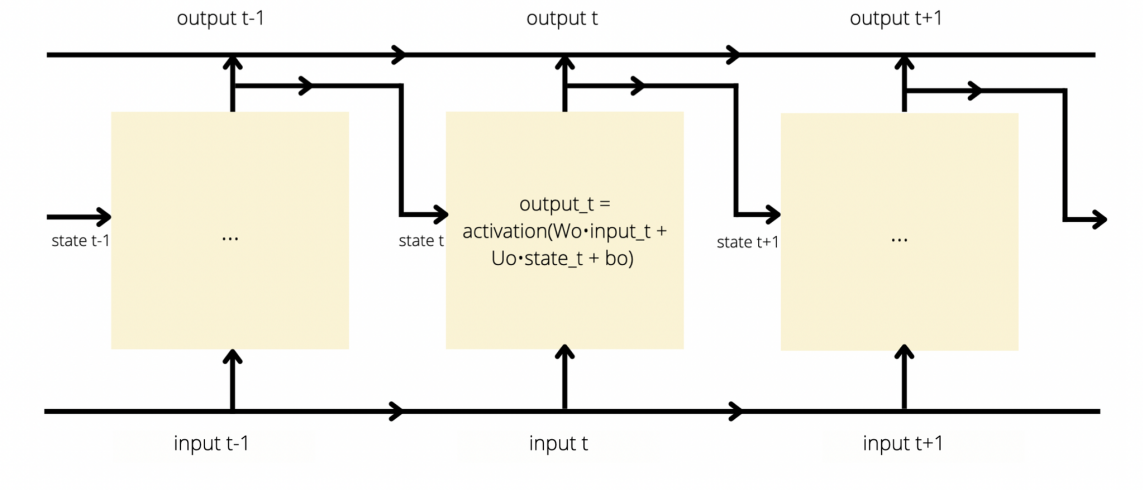


Figure 2: Unrolled RNN

But the basic implementation of an internal loop between corresponding inputs and outputs of layers will not do a groundbreaking job. It is due to the vanishing gradient problem, which was studied in 2001, by Bengio and Frasconi, along with the same two authors of LSTM [12]. This is not an RNN problem only, as it also appears in feed-forward networks during the back-propagation phase. Adding more layers to the network with more units per layer, results in a network that is very hard to train, or is not capable of doing so. The reason behind it is that as the network grows, important features or information are lost during processing, before it exits at the output layer. That is why the LSTM and GRU were created. However, they are not the only existing solutions that can eliminate or minimize this problem.

3.1 LSTM

As mentioned earlier, the same authors that tackled the vanishing gradient problem, came out with the solution [10] of developing and using the LSTM algorithm, which proved to be a big milestone in machine learning history. In fact, how does it work and why it became so important? It is an implementation of the recurrent neural network, but it does not use standard neurons, but LSTM units. Such network may consist of many layers of different dimensions, additionally with different types of neurons between them. This makes such solution very powerful - it can be very dynamic, suitable for different kinds of problems. The simplest RNN cell calculates its output based on the given input and state from the previous cell. What LSTM

does, it adds a connection (carry track), from the first layer to the last, that carries data along the way and its state, which varies in time. Now the output within a cell is calculated using this additional information, which greatly impacts the next output and next state. This allows network to "remember" some part of a feature that occurred a long time ago. It allows to design network that computes the next values not only based on a data point that occurred just before the current one, but it enables it to make a prediction based even on the first data points that appeared at the very beginning of training. Figure 3 presents connected LSTM units along with the carry track.

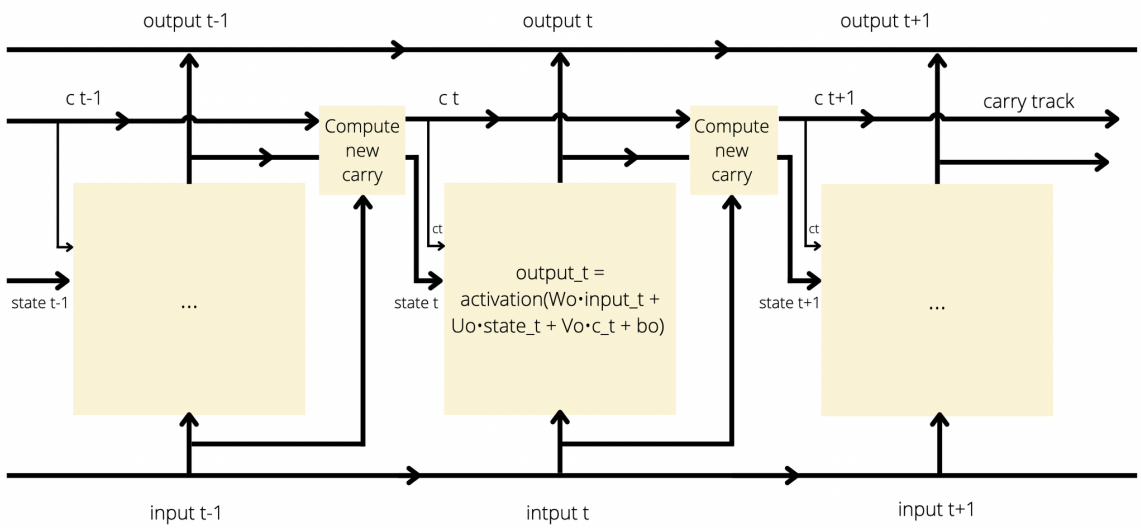


Figure 3: LSTM

This is especially useful in time-series prediction problems, where, in some cases, the whole history carries important information, meaning distant data points may have a great impact in the future. By carefully choosing the parameters of a model, that uses this algorithm, one can define which type of information the network should persist and which one to forget. For example, one may configure a network not to remember rare occurrences of very high spikes in currency rates, assuming that those have no impact, and one wants to disregard this information. As for the other types of units, LSTM neurons provide a great amount and possibilities of configuration, that can be optimized in many various ways to tackle different kinds of problems. An important role is played by hyper-parameters that define how the learning process behaves. Precise tuning may be a "make or break" for every machine learning model. What is more, LSTM work perfectly in combination with different units, meaning

they can be used with others, i.e., GRU or standard ones. The hardest task in developing well-suited architecture for given problem is finding the correct ratio between the amount of layers and dimensions of each layer, along with the right combination of different types of units and models' parameters. This work mainly focuses on finding such solution, that can be used to achieve the desired results.

3.2 GRU

Gated Recurrent Unit is a fairly new invention, as it was proposed back in 2014 [13]. It is very similar to LSTM, considering its structure, but it lacks an output gate. It also has update and reset gates, like LSTM does, which allows to "remember" important information and "forget" irrelevant one. Figure 4 depicts the basic GRU unit.

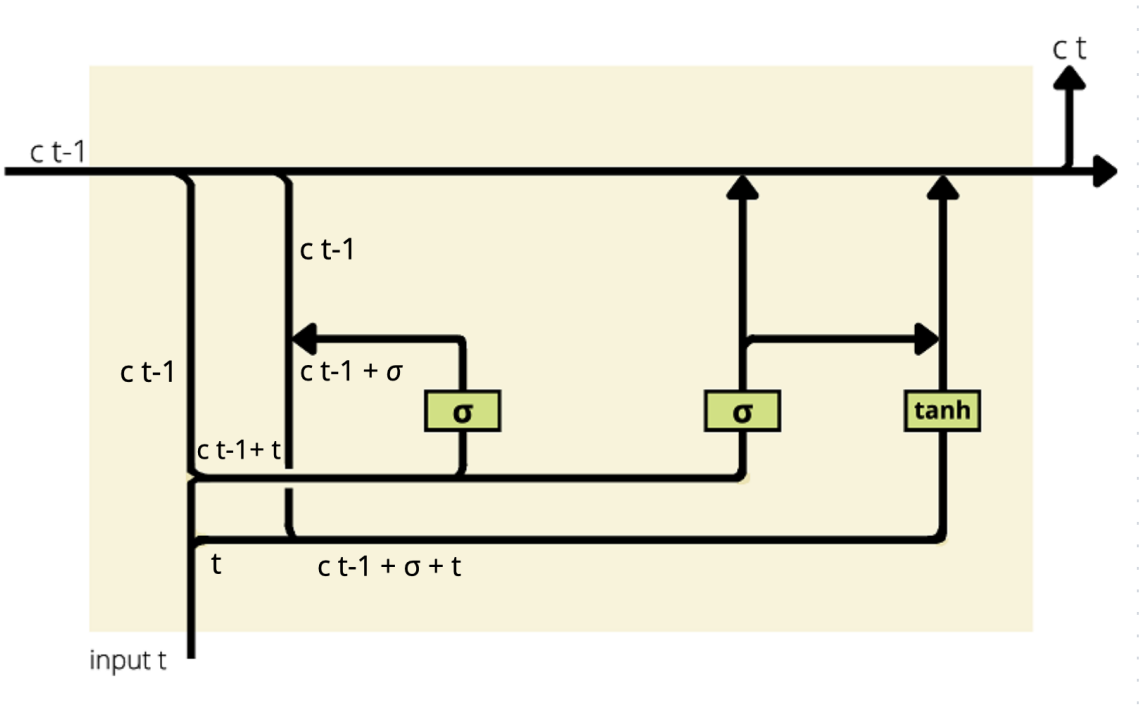


Figure 4: GRU

So if it is so similar to LSTM, why is it so effective? If LSTM neuron has more configurable features than GRU, why is it used at all? What has proved to be the advantage of such a unit is that it can outperform LSTM-based networks on

certain kinds of problems, especially in speech recognition and language processing, but in a specific environment. This environment usually means small data sets or a lack of data points in a used set. As it is less "powerful" than LSTM, it may achieve great performances on less complex problems. Looking at the architecture of this unit, it also has a carry track that contains historical information that is carried through to the next neurons. In a speech recognition task, performed at the University of Southern Queensland [14], GRU performance was measured against LSTM on different types of data. Results show that GRU can perform at least as well as the LSTM model and in some cases, it can beat it. Small data sets and the amount of information is the key factor in choosing the solution. The flexibility of both algorithms in certain environments and important details ultimately contribute to the final solution, which may have a slight or tremendous advantage over another one.

4 Used Tools

The complexity and characteristic of this study require the usage of many different libraries, the selection of the environment and of the programming language to process the data. This chapter briefly describes the chosen tools and explains why they were picked.

4.1 Google Colab Notebooks

For the environment in which the model is developed, Google Colab Notebook is chosen. It allows one to write any expressions in Python programming language. One may import many available third-party libraries to use them during model creation. Besides that, it is free when one is using it for a certain amount of time during one session (12h) and under certain hardware limitations. It also permits to use any available GPU under the tier that is in use. In the case of this study, what Google Colab offers is well sufficient. In addition to mentioned features, it is easy to use and maintain created resources and data, which can be easily exported and shared, i.e., via Google Drive. When used with other libraries and extensions, it proves to be a very powerful tool when it comes to machine learning oriented tasks.

4.2 Python

As Python is imposed by the usage of Google Colab, it is not the only reason. This language provides a huge number of libraries that one can use. There are many extensions developed especially for data science, machine learning, mathematical and statistical computation. What is more, it is one of the easiest languages to learn and use. It also supports well-known libraries that support complex figure printing, which is useful for time-series prediction problems.

4.3 TensorFlow

On top of that, this research uses TensorFlow [15], which is a library that provides end-to-end machine learning support. It was created by Google in 2015 and gained an immense amount of popularity in the machine learning community. From within this package, one may use provided data, structures, models, layers and many more useful tools in order to tackle different kinds of problems, whether it is image classification, speech recognition or time-series problem. This package provides inner packages that can handle more specific tasks. Additionally, it engages a great amount of users in

the TensorFlow community, which makes the usage even easier, as one may reach out to other developers for help.

4.4 Keras

Keras [16] provides APIs, that are used in this study. It allows to create a model from scratch, defining the type of it, number of layers, number and type of neurons in each layer, activation functions, dropout rates, loss functions and metrics, optimizers and many more. It is responsible for feeding the model with data and the learning process. By using it, one may also configure hyperparameters to control the learning phase. This package also provides clever solutions that can help in achieving optimal model parameters - like KerasTuners, especially HyperTuner, which will try to find the best parameters for a given task and exit the search under certain, specified by user conditions.

4.5 NumPy

NumPy [17] is a python scientific computation package. It provides support for multi-dimensional arrays and makes their implementation easy. One may perform many complex operations on specific array structures. In this study, it is used to create arrays compatible with the Keras model to use. Also, it is used during the evaluation, when dynamic management of an array is required in the process of learning from length-variable data.

4.6 Pandas

Pandas [18] is a handy Python package that helps to process the data. One can operate on multidimensional matrices of data, performing a variety of operations on each data point, column or row. In this study, it is used to create *Data Frames* from imported data and to strip it from unwanted columns. When needed, it might be useful to disregard or fill the missing data points with dummy data that fits the surrounding information and this can also be handled by the Pandas package.

4.7 Matplotlib

The Matplotlib [19] package is worth mentioning, as its module, *pyplot* was used in this work to create each of the plots and figures representing the data. It is a powerful visualization tool that one may configure in various ways, depending on

the type of graph one wants to get. It allows to create interactive, customized and publication-quality plots.

5 Implementation

The model creation process is a time-consuming operation. Each small change, whether it is in the data features, hyper-parameters or parameters of a model, requires re-creating data sets from generators, learning and evaluation process starting from scratch. Although there are some tools that can automate some parts of the work, the longest time that is spent on learning stays the same - this process can not be shortened. The implementation was divided into two parts. The first one consisted of discovering possibilities of modules and corresponding structures. It was crucial to learn the basic dependencies of some of the parameters and to recognize their impact on the network's performance. Extending the learning time, increasing the learning rate, removing certain features from the data - from all of these and many more changes, conclusions were made that helped in developing the final model. The second step was, after coming up with the initial model, to examine the impact of different parameter values step by step. Choosing the number of layers and the number of neurons in each layer was a starting point. Then the activation functions, dropout rates, learning rates, optimizers and regularizers were explored. All of these possibilities were carefully tested and evaluated.

5.1 Data Gathering and Pre-processing

The exchange rates data used in this study was gathered from *ForEx HistData* site [20]. Three currency pairs were used: USD/PLN, EUR/PLN and EUR/USD. Data provided by *histdata.com* contains six pieces of information: date, hour, opening price, closing price, minimal and maximal price. Since the provided data is divided into the monthly set, a merge tool was used to combine data from consecutive months, to create one continuous and chronological series of data. The data interval is one minute, so it had to be converted to one-day-based data. Using the script that is available under *GitHub* page [21], where all the resources created during this study are stored, the USD/PLN data was converted as follows: each value of exchange rate under a certain day was added to corresponding exchange rate under the same date, for each day respectively and then divided by a number of data points for the current day. The outcome of this operation is data stored in a one-day format, rather than one minute. After computing day-based values and adding additional features, this is how the final data set looks like:

- Opening Price - opening price for each day, calculated as a mean of all the opening prices for a day

- Closing Price - closing price for each day, calculated as a mean of all the closing prices for a day
- High Price - the highest value encountered on each day (maximum)
- Low Price - the lowest value encountered on each day (minimum)
- Momentum - the difference between opening and closing prices (opening price - closing price)
- Range - the difference between high and low prices (high price - low price)
- OHLC Price - sum of opening, closing, high and low prices, divided by 4

One last thing that has to be considered regarding the data is which feature the model should predict. In the case of this problem, it may be any one of them. For this work, the Closing Price was chosen to be the one.

From the OECD data publications [22], CPI and interest rate data were added to the final result for two countries, USA and Poland. What is important to note, the provided data is in a monthly format, meaning the same values appear for every day of the month. This creates a problem that will be discussed later in the chapter. Finally, the resulting data was exported to a *csv* format that was later used during network training. Figures 5, 6 and 7 present the data that was gathered and examined in over a 10-year scope. Each data set's beginning date is 11.15.2010 and the ending date is 31.03.2022, containing 3542 data points. The values that are printed on the figures are Closing Prices, which are going to be predicted by models.

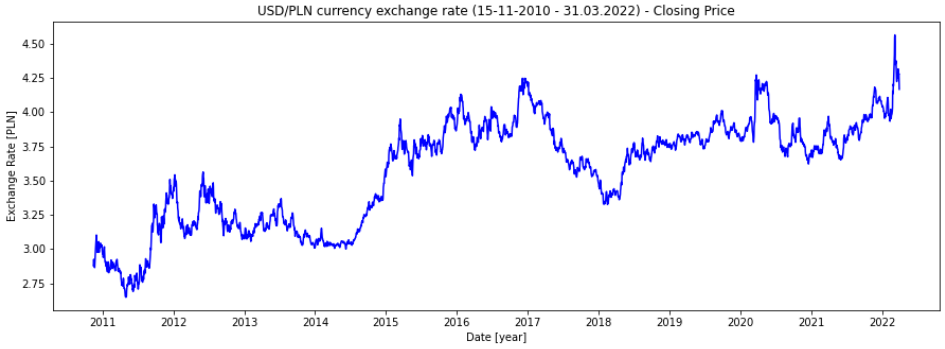


Figure 5: USD\PLN exchange rate history

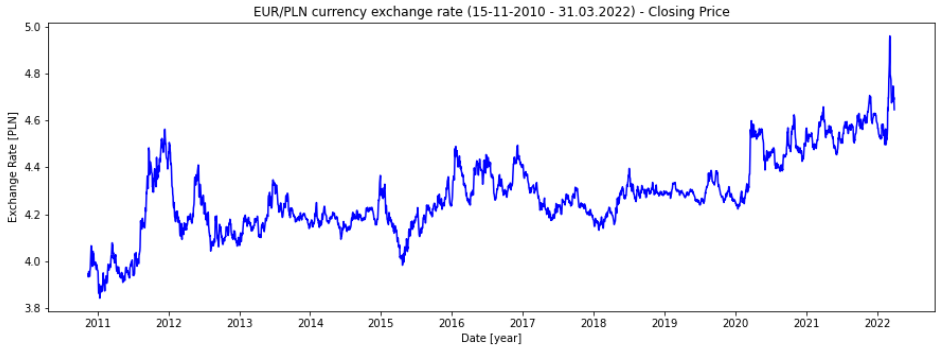


Figure 6: EUR\PLN exchange rate history

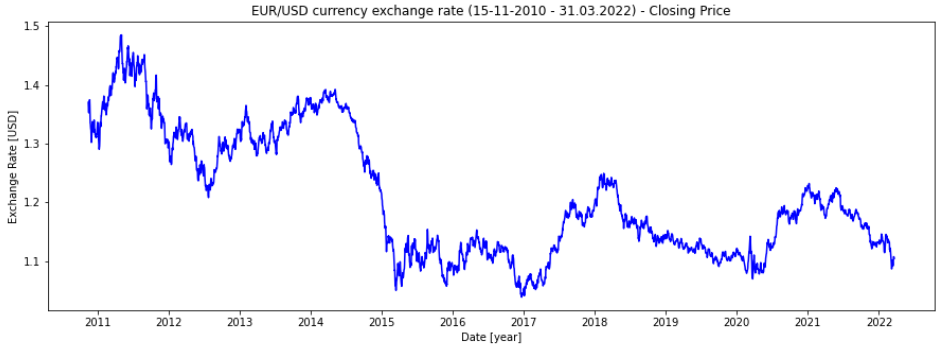


Figure 7: EUR\USD exchange rate history

5.2 Workflow

Finding the correct architecture with a combination of many different parameters is a long process and in this kind of study, requires re-creating each model in an independent way. This leads to the extraction of some actions that are performed repeatedly, but for different-valued parameters.

5.2.1 Data Division into Training, Evaluation and Test Sets

Each time a new model was created and tested, the data had to be split into three sets. The largest of them, the training set, is used by a network to train - in order to allow the network to understand and find possible correlations within the

data, the size of this set, in this case, must be sufficient enough and definitely bigger than the other sets.

Validation data is used after each epoch of learning to see how the model is performing along the way and to see if it is improving or not. The network calculates the loss function based on the evaluation results and tunes weights and biases, trying to achieve optimal configuration. The size of the loss during validation is a great indicator of the performance of the model and informs about one of the crucial problems during neural network development - overfitting and underfitting.

The test set is unknown for the network, as it was not used during the training process. It is used after the model learning is finished to measure how it performs on never-seen-before data and is an definite, final indicator of the model's performance. In all researched models, the data was divided as follows: the number of data points for all of the currency pairs is 3542. The training set contains the first 2500 samples, validation data contains the next 501 points and the rest (541) samples are used to create the test set. The ratio between the sets is approximately 70:15:15.

5.2.2 Data Scaling

Scaling is the next step in preparation for feeding the data into the network. Without it, the values that are larger than others might be wrongly understood by a network and more preferred than the smaller values. This might result in a network that completely disregards parts of data and misses potentially valuable information. In order to achieve faster convergence, scaling is a must-have. In this study, it is done by a *MinMaxScaler* from *sklearn.preprocessing* package. Firstly, the scaler is fitted to the training and validation data and based on that knowledge, it scales the whole provided data accordingly to the calculated scale. Why is the training set data not included in this calculation? Simply because the aim of the model is to predict future values without the knowledge of it, so including it might give a hint about these values. The resulting model could perform greatly on provided data, but when fed with the data it actually had never seen before, one might be hugely disappointed with its performance.

5.2.3 Data Generators and Parameters

In order to feed the data to the network, apart from the correct format, a few things have to be defined. In the time-series prediction problem, one of the most important parameters is the time horizon, which the network uses to look at the previous values. This parameter is often referred to as a loopback. It is the number of data points that are fed into the network during one learning iteration, from which

the network makes a prediction. The decision is made based on a defined number of previous values and the whole learning history. Using the generator proposed by François Chollet in the book [23], three more parameters have to be defined:

- delay - how many data points should the network predict into the future - 0 meaning the next value, just after the last received data point
- step - which data points to choose from the data set - 1 meaning each data point will be used, without skipping any
- batch size - the amount of generated, ready to use training parts, containing training data and expected (desired) values

This generator generates key-value pairs from given data: the key being the look-back number of consecutive data points and the value being the expected currency value, depending on specified lookback, step and delay parameters. Based on the batch size, these structures are grouped together into batches which the network uses in the learning process. If there are 100 samples available, lookback is set to 10, step to 1 and delay to 0, the generator generates 90 pairs - the first key starting from index 0 to 9 with expected value at index 10 and the last pair's key starting from index 89 to 98 with expected value at index 99. Sample generated key-value pair is depicted in figure 8.

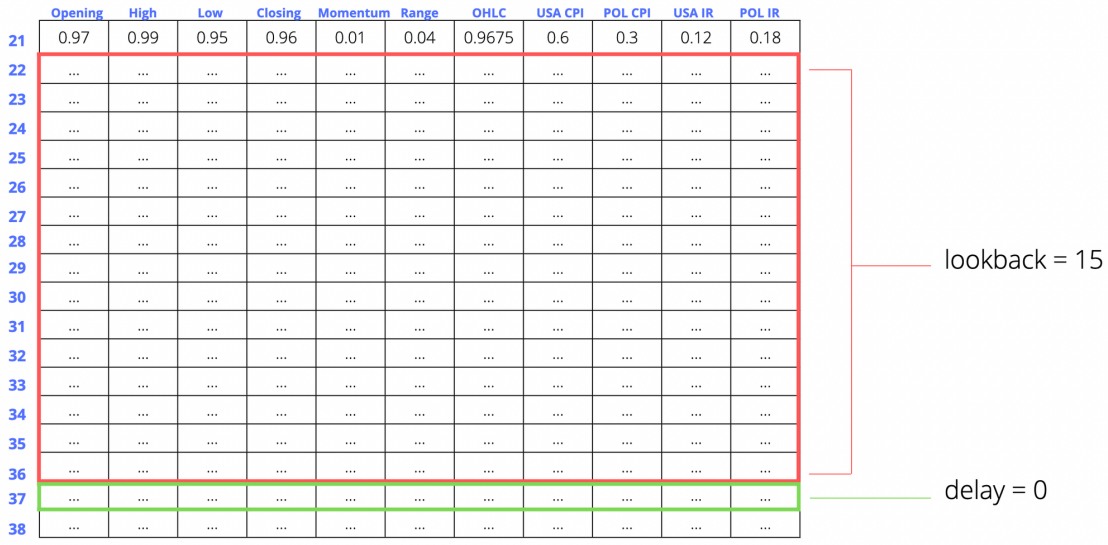


Figure 8: Generated data sample

The same rule applies to the evaluation and test sets. This means that for the training data set, there are 2485 samples, for the evaluation set, there are 486 samples and for the test set, there are 526 samples. As the model architecture is created with the aim to predict next-day values, it will also be tested on a longer time horizon - 30 days. For this case, the lookback parameter will be increased to 30, with the delay parameter set to 30. This means that all of the samples for the data sets will be decreased by 60.

5.2.4 Model Definition and Learning Process

The next step was the definition of a model. Here, the model architecture has to be specified, along with the learning process parameters. Each time any change was introduced, this time-consuming process had to be repeated. This step allows one to define number of layers, type and quantity of neurons in each layer, additional optimizers, dropout techniques, regularizers and management of the learning process - number of epochs, steps per epoch and loss function.

5.2.5 Testing

The last part of the workflow, after the model training, is to evaluate it. Each time, it was carried out on the test data set which the network did not see before. The numerical value achieved after the test was the ultimate indicator of model performance. Throughout the study, Mean Absolute Error (MAE) was used as the metric to determine the model's position among others.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{1}$$

where:

n = number of samples
y = predicted value
x = actual value
i = sample number

5.3 Architectures

The architecture of a model describes its' structure - number of layers, type of neurons and number of units in each layer. Keras tuner allows one to check different setups, combinations in an automatic way. There are a few types of them, HyperBand being the most often used. In the scope of this study, many architectures

were examined with many different variations. Each architecture score was saved and compared with others, resulting in choosing the best-performing setup.

5.4 Dropout

Dropout is a very powerful method to prevent models from overfitting. Overfitting occurs when the network is large (in a number of neurons or layers) and the data set is relatively small. The model perfectly learns the training data pattern but fails to achieve the same level of results on evaluation or test data. What dropout does, it picks random neurons from layers and ignores its outputs. This results in neighboring units having to adapt in each iteration in a different way. The network benefits from this by learning from these so-called side effects. Each iteration causes different random neurons to be turned off - one may configure the dropout rate parameter to decide how much of all the available units should be dropped out. Recurrent dropout is an extension to the regular one, differing in the dropped values. Recurrent dropout removes connections between recurrent units, instead of their outputs and inputs. It may also improve the model performance and make the network more robust. During finding optimal parameters, the usage of dropout proved to be a worthy solution, as it resolved the overfitting problem. Although usage of the Keras Tuner proved to find architectures that do not succumb to overfitting, it was crucial in the beginning phase of model design, which helped to better understand the relation between the sizes of the network and data.

5.5 Activation Functions

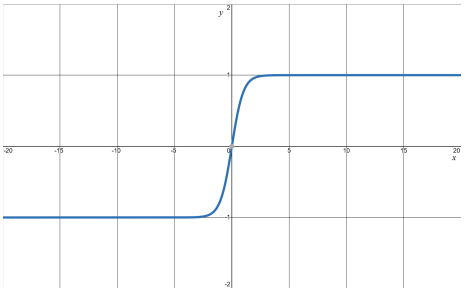
The neuron's activation function, also called the transfer function, decides if the neuron should be activated based on input values. The output of this function might be used as an input to the next layer neurons. These functions are usually divided into two groups:

- Linear Functions
- Non-linear Functions

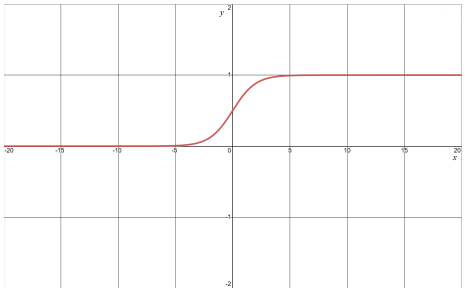
The usage of linear function means that the output may be in any range limited only by a function definition. Since time-series forecasting is basically a probability problem, the activation output should be in the range of 0 to 1. That is why the most often used functions are non-linear, as the model might adapt better to the difference between input and output value. There were several functions taken into consideration, for example:

- tanh
- sigmoid
- relu
- softmax
- softplus

Once again, the Keras Tuner helped in choosing the most suitable activation function, which turned out to be sigmoid. Very closely behind performance-wise were the relu and softmax functions. Sigmoid, as well as relu and softmax functions, limit the range of the output values to <0;1> range, while i.e., tanh limits it to the <-1;1> scope. Any probability function results in values ranging from 0 to 1, so for this particular problem, an activation function with this value range is better suited. Figure 9 (a) presents the plot of the *tanh* function, while figure 9 (b) depicts the *sigmoid* function.



(a) tanh function



(b) sigmoid function

Figure 9: Activation functions

5.6 Other Parameters

There are many more parameters that one may configure, as the Keras API implements many features and algorithms. There are several more that were taken into account during this research, such as:

- Regularizers
- Optimizers
- Lookback
- Number of Epochs
- Number of Steps per Epoch
- Loss function

There are two most often used regularization methods in machine learning, that being L1 and L2 regularization. This is another way to prevent the model from overfitting. It is primarily used in more simple models, that tend to overcomplicate the solution. This method regularizes the data, flattens any data points that are not similar to the rest. For the time-series problem, it means that it removes potentially very important information, for example, high peak in currency value just after a low point. What the researched model should do is the opposite of the regularization, it should try to remember these "anomalies" as they may carry a crucial piece of information. That is why any of the models that applied L1 or L2 regularization did not achieve great results, they all performed worse than without this technique.

One of the most important parameters when it comes to machine learning models is an optimizer. This algorithm is actually responsible for the learning part of a network. It is used to minimize the loss function by adjusting weights and biases in a way that produces better performance. Optimizer, whichever one may choose, has one particular goal - to find a local (or sometimes global) minimum of a loss function, meaning the best possible fit. Since there are many possible algorithms ready to use in the Keras library, tuners were used to test most of them. RMSprop proved to be the best solution for this task. The second-best performing algorithm was Adam, with the likes of SGD (Stochastic Gradient Descent), AdaMax, AdaDelta and Adagrad being further behind. RMSprop resolves one of the obstacles that proved to be hard to handle - when there are multiple gradients that vary in values. It tackles the vanishing and exploding gradient problems. What this algorithm also does, instead of using fixed learning rate, it adapts its value over time, trying to fit to the data accordingly. It does not perform well on very large data sets, but since the training data size is 2500 samples, it is a great fit.

Lookback, as mentioned earlier in the thesis, is the number of data points that the network is fed at once. It makes predictions based on this amount of received values. For the prediction horizon of the next day, values between 5 and 50 have been tested, with 15 producing the best results. For the horizon of 30 days, the optimal lookback parameter equaled the same value, being 30 samples.

The number of epochs are how many times the model will loop through the entire data set in the training process. At the beginning of the study, relatively low numbers were tested, being in the range of 15 to 40. The results were not great, so it was decided to increase it to 100. This enabled to achieve better results, which were maximized with the Keras HyperBand tuner. It allows to test multiple numbers of epochs as a parameter, but it is often a very time-consuming process. In the end, 200 epochs were chosen.

As for steps per epoch, this ideally should be the result of the division of the size of training data between the batch size. That was not the case in this study, as with according value each iteration ended prematurely, without learning actual dependencies within the data.

The loss function calculates the difference between predicted and actual values. It is an indicator of how well the given model fits to the data. During the learning process, this value should decrease in each iteration. For the test set evaluation, it can be used as an ultimate performance indicator of the model. Two loss functions were tested from the regression losses class from Keras, that being Mean Squared Error (MSE) and Mean Absolute Error (MAE). The second one was chosen as a loss function, because of a faster convergence. What is worth mentioning, any of the loss

functions might be used as a metric for a model. This means that besides presenting actual loss function values, it is also possible to calculate it by a different algorithm and print it during the learning process.

5.7 Final Models

After considering all of the previously mentioned parameters, creating and evaluating many different combinations, three architectures have been chosen to be tested against the prepared test cases. Table 1 presents architectures that achieved the best results. They form three different models: LSTM, GRU and Hybrid model.

Layer/Model	LSTM	GRU	HYBRID
Layer 1 (input)	7	7	7
Layer 2	120 (LSTM)	140 (GRU)	120 (LSTM)
Layer 3	160 (Dense)	120 (Dense)	5 (GRU)
Layer 4	1 (Dense)	2 (GRU)	100 (Dense)
Layer 5	-	1 (Dense)	1 (Dense)

Table 1: Proposed models architectures

Table 2 showcases parameter values, that have been applied within particular models.

Parameter/Model	LSTM	GRU	HYBRID
Activation function	sigmoid	sigmoid	sigmoid
Model type	sequential	sequential	sequential
Optimizer	RMSprop	RMSprop	RMSprop
Learning Rate	0.01	0.001	0.01
Dropout Rate	0.1	-	-
Recurrent Dropout Rate	0.1	-	-
Loss Function	MAE	MAE	MAE
Lookback	15/30	15/30	15/30
Batch size	128	128	128
Step	1	1	1
Delay	0/30	0/30	0/30

Table 2: Models parameters

6 Evaluation

Each architecture was tested using three different currency pairs. Since all of the designs were based on the USD/PLN currency rate, the same models had to be trained for the remaining two pairs. Then each model for each currency pair was evaluated on the test data, which was not previously seen by the model. Comparison against the baseline model was carried out, as well as trading tests and the impact of retraining, which will be covered in the next sections.

6.1 Baseline Model

The baseline model is commonly used for benchmarking any developed model that should be beaten to prove that the actual performance of the studied model is better. This is a milestone that should be achieved. Since the prediction is only based on an average from the amount of *lookback* days, it is impossible to retrain such model. Baseline model, along with others, had undergone trading test referred to as the "investment game". For this test, three concepts have to be defined:

- current value - it is the exchange currency rate value at the day the prediction is made
- predicted value - it is the exchange currency rate value being predicted by the model
- actual value - it is the true exchange currency rate value in the day that it is being predicted

If the predicted value is higher than the current value and the actual value is higher than the current - it is counted as a correct pick. The same situation goes for the case where the model predicts lower value than the current one, and the actual value is also lower than the current. All other cases are considered as wrong picks. Additionally, the difference between the predicted and actual values is calculated for each prediction and depending on whether the pick was correct or not, this value is added or subtracted from a starting value of 0. This metric indicates if, after the whole prediction horizon, the model gained or lost "virtual money" and is purely an additional metric that one may use to see if the model performed well for the given test data. It does not mean that for the different test data, the model would perform in a similar way.

Table 3 presents the evaluation results and the trading test results for USD/PLN currency rate that the baseline model achieved for next-day prediction and 30 days

prediction. Tables 4 and 5 contain the same set of information but for EUR/PLN and EUR/USD currency rates, respectively.

USD/PLN baseline		
	next day	30 days
Evaluation result	0.03689	0.06768
Correct picks	239	249
Value	-91.9	309.2
Total number of picks	541	511

Table 3: Baseline model test results for USD/PLN currency rate

EUR/PLN baseline		
	next day	30 days
Evaluation result	0.04927	0.08744
Correct picks	230	293
Value	-97.4	945.6
Total number of picks	541	511

Table 4: Baseline model test results for EUR/PLN currency rate

EUR/USD baseline		
	next day	30 days
Evaluation result	0.03209	0.06868
Correct picks	259	245
Value	-35.2	-126.6
Total number of picks	541	511

Table 5: Baseline model test results for EUR/USD currency rate

6.2 Evaluation Tests for the Next Day Forecast

In this section, results for next-day forecast are presented. The results are grouped into three parts, for the corresponding currency pairs. Table 6 presents results for USD/PLN, table 7 for EUR/PLN and table 8 for EUR/USD.

USD/PLN			
	LSTM	GRU	HYBRID
Evaluation result [MAE]	0.01515	0.01451	0.02775

Table 6: Evaluation results for USD/PLN currency rate - next day

EUR/PLN			
	LSTM	GRU	HYBRID
Evaluation result [MAE]	0.05017	0.06223	0.07225

Table 7: Evaluation results for EUR/PLN currency rate - next day

EUR/USD			
	LSTM	GRU	HYBRID
Evaluation result [MAE]	0.01350	0.01233	0.01420

Table 8: Evaluation results for EUR/USD currency rate - next day

6.3 Evaluation Tests for the 30 Days Forecast

Next, the evaluation tests were performed on the same architectures, but models were trained to predict 30 days in the future, based on the 30 last days. Tables 9, 10 and 11 contain the achieved performance results.

USD/PLN			
	LSTM	GRU	HYBRID
Evaluation result [MAE]	0.05456	0.06342	0.05260

Table 9: Evaluation results for USD/PLN currency rate - 30 days

EUR/PLN			
	LSTM	GRU	HYBRID
Evaluation result [MAE]	0.38198	0.46445	0.45654

Table 10: Evaluation results for EUR/PLN currency rate - 30 days

EUR/USD			
	LSTM	GRU	HYBRID
Evaluation result [MAE]	0.04012	0.04680	0.04717

Table 11: Evaluation results for EUR/USD currency rate - 30 days

6.4 Trading Tests

The next test, that each model underwent was a trading test. Each day in the test data, the model had to make a prediction (for the next day or for the 30 days). Tables 12, 13 and 14 present indicators mentioned for the next-day prediction and corresponding models.

USD/PLN			
	LSTM	GRU	HYBRID
Correct picks	281	298	275
Value	43.1	96.5	-9.5
Total number of picks	541	541	541

Table 12: Trading test results for USD/PLN currency rate - next day

EUR/PLN			
	LSTM	GRU	HYBRID
Correct picks	294	296	273
Value	10.1	63.8	7.2
Total number of picks	541	541	541

Table 13: Trading test results for EUR/PLN currency rate - next day

EUR/USD			
	LSTM	GRU	HYBRID
Correct picks	275	283	257
Value	25.9	22.2	-18.0
Total number of picks	541	541	541

Table 14: Trading test results for EUR/USD currency rate - next day

The same tests were performed for 30 days horizon, but because of that, the total number of picks is decreased by 30. Table 15 shows the results for USD/PLN currency rate, table 16 for EUR/PLN and table 17 for the EUR/USD.

USD/PLN			
	LSTM	GRU	HYBRID
Correct picks	284	269	297
Value	-141.5	-184.1	446.7
Total number of picks	511	511	511

Table 15: Trading test results for USD/PLN currency rate - 30 days

EUR/PLN			
	LSTM	GRU	HYBRID
Correct picks	223	220	222
Value	-1191.8	-1213.6	-1220.4
Total number of picks	511	511	511

Table 16: Trading test results for EUR/PLN currency rate - 30 days

EUR/USD			
	LSTM	GRU	HYBRID
Correct picks	243	292	267
Value	-236.1	282.1	15.4
Total number of picks	511	511	511

Table 17: Trading test results for EUR/USD currency rate - 30 days

6.5 Trading Tests with Retraining

In the next tests, retraining was used to find out whether retraining the model during the prediction phase would help in achieving better results. It was performed as follows: after 150 days of making predictions from test data, the model was retrained with the most recent 150 days, allowing it to persist most current values. Then the next 150 predictions were made. It adds up to 3 full retrain processes during

the execution of the test. Smaller values for the gap between retraining phases were studied, but did not provide any improvements. Table 18 present the results for the USD/PLN, table 19 for the EUR/PLN and table 20 for the EUR/USD currency rate trading test for the next day with the retraining.

USD/PLN with retraining			
	LSTM	GRU	HYBRID
Correct picks	265	279	272
Value	31.0	53.6	-6.4
Total number of picks	541	541	541

Table 18: Trading test results for USD/PLN currency rate - next day with retraining

EUR/PLN with retraining			
	LSTM	GRU	HYBRID
Correct picks	286	301	278
Value	14.1	65.1	77.1
Total number of picks	541	541	541

Table 19: Trading test results for EUR/PLN currency rate - next day with retraining

EUR/USD with retraining			
	LSTM	GRU	HYBRID
Correct picks	286	273	283
Value	26.2	8.3	25.0
Total number of picks	541	541	541

Table 20: Trading test results for EUR/USD currency rate - next day with retraining

The same calculations were performed for the longer time horizon. Table 21 shows the results for USD/PLN, table 22 for EUR/PLN and table 23 for EUR/USD currency rate trading test for the 30 days with retraining.

USD/PLN with retraining			
	LSTM	GRU	HYBRID
Correct picks	257	274	278
Value	-370.6	-59.0	169.8
Total number of picks	511	511	511

Table 21: Trading test results for USD/PLN currency rate - 30 days with retraining

EUR/PLN with retraining			
	LSTM	GRU	HYBRID
Correct picks	306	255	232
Value	530.2	-332.2	-1051.9
Total number of picks	511	511	511

Table 22: Trading test results for EUR/PLN currency rate - 30 days with retraining

EUR/USD with retraining			
	LSTM	GRU	HYBRID
Correct picks	172	238	196
Value	-961.8	-282.9	-573.0
Total number of picks	511	511	511

Table 23: Trading test results for EUR/USD currency rate - 30 days with retraining

7 Conclusions

After gathering all of the evaluation and trading test results, figures were drawn in order to help visualize the performance. Results are divided into five categories: best evaluation results, trading test results for the next day and 30 days prediction and trading test with retraining for the next day and 30 days prediction.

7.1 Evaluation Comparison

Starting with evaluation results for models that predict currency rate for the next day, from figure 10, one may discover that for USD/PLN and EUR/USD currency pairs, each of the architectures has beat the baseline model. For the EUR/PLN pair, the baseline model emerged just a little ahead of the LSTM model. The cause of this situation is that all the model architectures were trained firstly for the USD/PLN pair, so each architecture performs better than the baseline model. What is interesting, EUR/USD prediction for each architecture is on a very similar level. This is due to the data characteristics. For this pair, currency oscillates between 1.30 - 1.05 value with no major dispersion, hence predicting this value, does not come with a risk of missing the prediction by huge amount. However, for the EUR/PLN pair, one may observe that each model, even the baseline one, does not perform on the same level as others. This means that there were multiple situations in which the value for the next day suddenly dropped or spiked and no model could learn it from history.

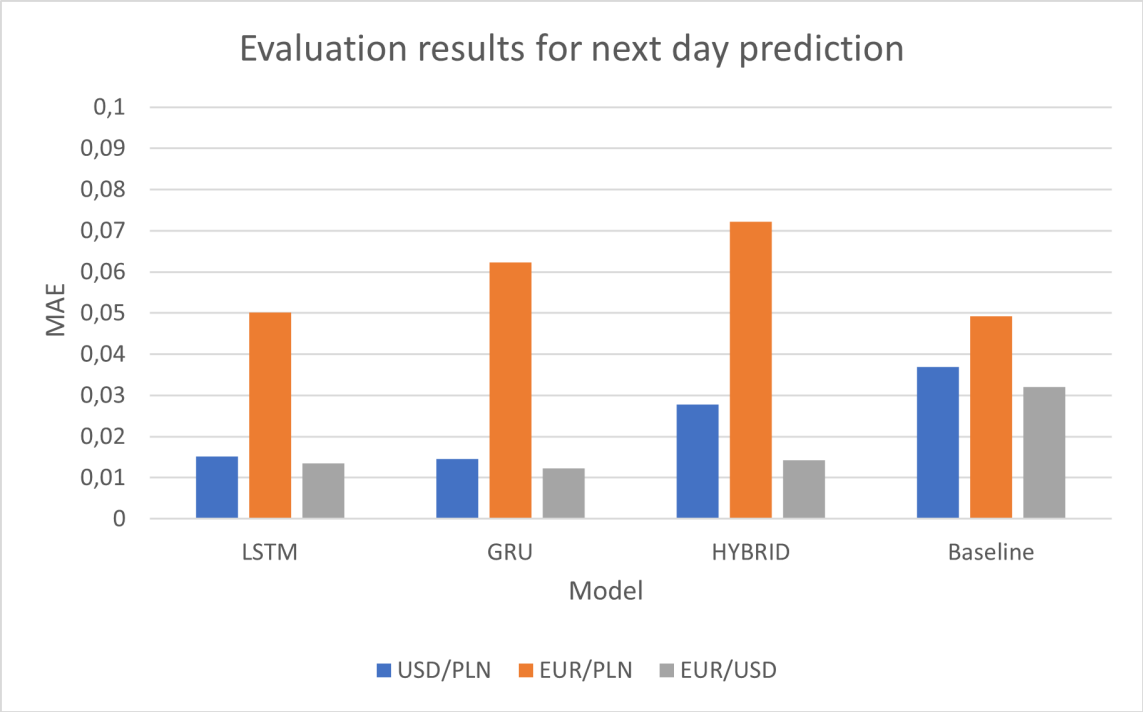


Figure 10: Evaluation results for next day prediction

The same plot was obtained for the 30 days prediction horizon and is presented in figure 11. Although 30 days time prediction horizon was not a primary goal for this thesis, it is valuable information about model performance. It is even harder to beat by a machine learning model, compared to the baseline model. Baseline model will calculate a mean from previous days and predict it for the 30th day. This value will not be very far from the actual value, because of the nature of time-series, such as currency rates (unless there is some kind of economic crash on the global market). This is the cause of a dramatic decrease in models' performance when it comes to the EUR/PLN pair, where, as previously has been mentioned, many "anomalies" occurred. However, for the remaining pairs, the developed models still manage to beat the baseline one, proving that they are not making predictions blindly.

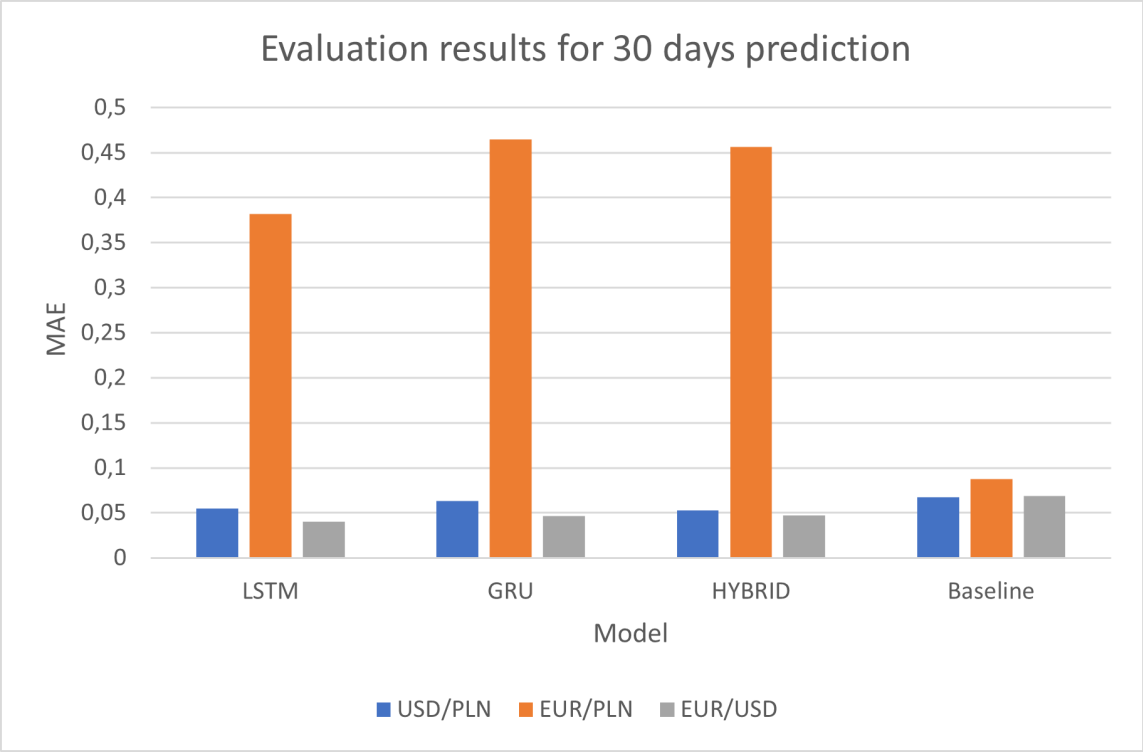


Figure 11: Evaluation results for 30 days prediction

When it comes to choosing the best model based only on evaluation results, for next-day prediction, it would be LSTM. This is because it manages to match the performance of the baseline model for the EUR/PLN currency pair, and performs better than the Hybrid model. Worth mentioning is GRU, that even beats the LSTM by a very narrow margin, but falls short when it comes to predicting EUR/PLN currency rate.

The baseline model performs better in 30 days prediction for the EUR/PLN currency pair than the rest of the models. However, even for this time horizon, it is beaten by LSTM, GRU and Hybrid models for the USD/PLN and EUR/USD pair. Since the currency pair that the training was done for, has different characteristics than the EUR/PLN pair, models could not extract correct information. Perhaps different architecture could be used to train the model specifically for EUR/PLN currency pair to obtain better results.

7.2 Trading Tests Results Comparison

Trading test results are divided into two categories: without and with retraining. Figure 12 presents these results for the next-day prediction without retraining. As one may observe, only once the percentage of correct picks is under the 50% mark. What is more, besides this one case, each time, all three developed architectures managed to beat the baseline model. The best results are obtained by the GRU model, with LSTM and Hybrid models behind. These results are no surprise, they were expected to be around 50%, but they actually are slightly higher. Reasons, why they cannot be even higher, are going to be described later in the thesis.

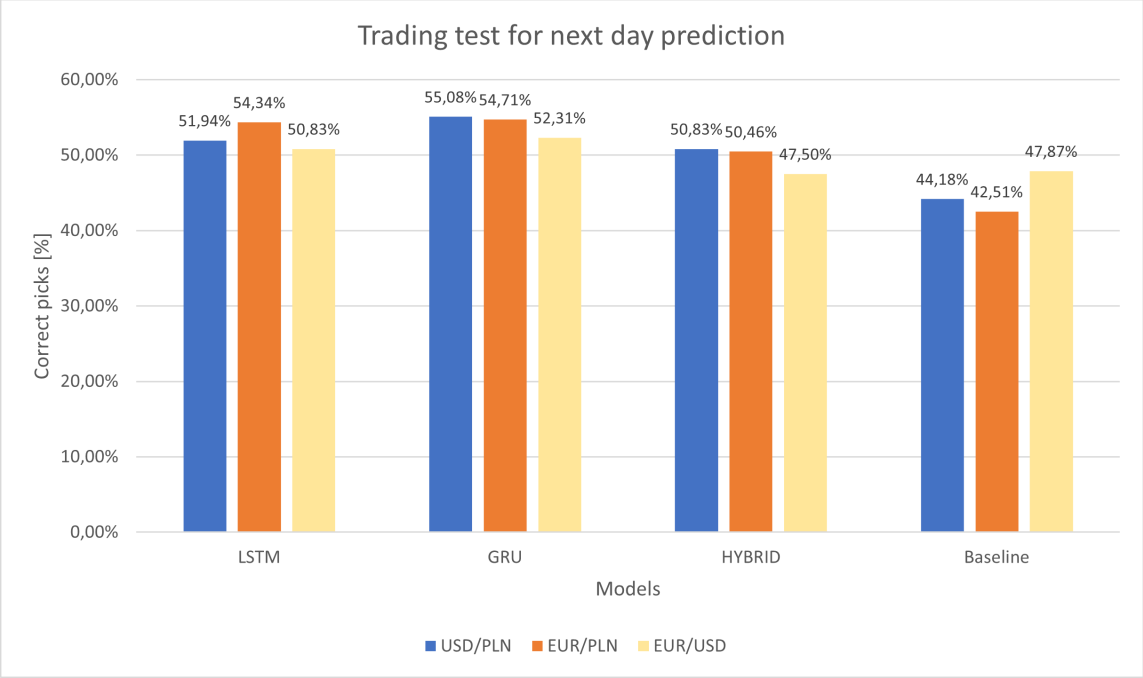


Figure 12: Trading test results for next day prediction

The same calculations were performed for the 30 days horizon and are presented in figure 13. These results highlight randomness in the prediction of time-series such as currency rate. One time the total number of correct picks is below 50%, reaching 43% and one time, achieving 58%. However one can still notice that for the EUR/PLN pair, the developed models make similarly wrong predictions, on the other side, however, the baseline model performs better than on average. There is no concrete pattern in this set of results, since the prediction for 30 days in the future is a harder task than predicting it for the next day. Moreover, each architecture was developed specifically for the prediction of the next day's value, not the 30 days period. If the architectures were developed strictly for the later case, results would be very different and possible dependencies between architectures could be extracted.

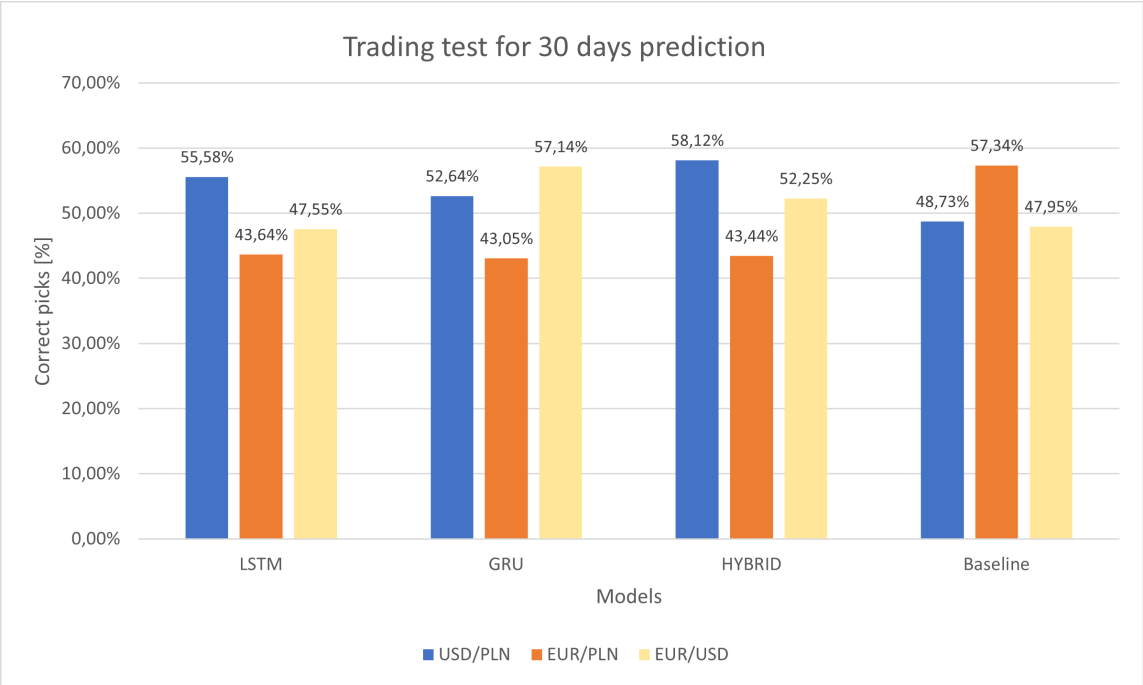


Figure 13: Trading test results for 30 days prediction

Next, the impact of the retraining process is discussed. Figure 14 presents the same trading test results, but with retraining for the next day's prediction. Since the baseline model cannot be retrained or trained at all in any way, the same results have to be used. Compared to the results visible in figure 12, the impact of the retraining process is different regarding each model. Hybrid model performance is increased for every currency pair. The biggest improvement is achieved for the Hybrid model and EUR/USD pair, from 47.50% to 55.38%. Also, GRU model performance for EUR/PLN currency pair is distinguishable from the rest, reaching close to the 60% mark. The overall performance of the models stays at the same level or is improved.

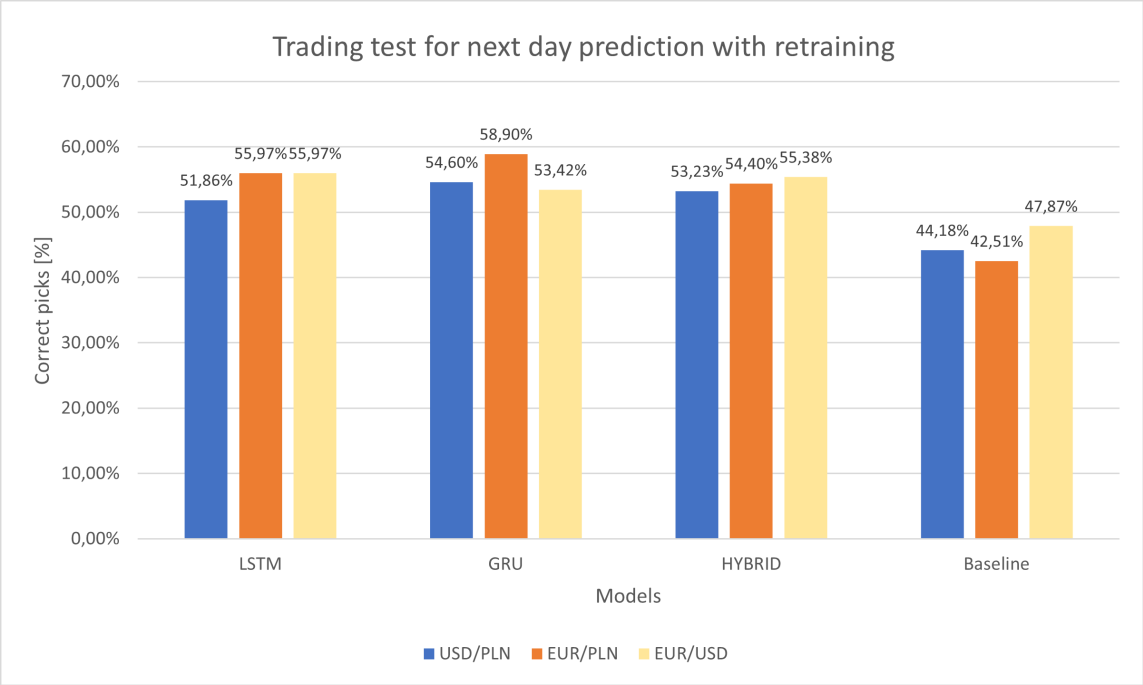


Figure 14: Trading test results for next day prediction with retraining

The remaining trading test results for the 30 days prediction with retraining are shown in figure 15. One may observe that the impact of retraining for the EUR/PLN currency rate for each architecture allows achieving higher correct picks ratio. The biggest improvement is noticed for the LSTM model, from 43.64% up to 59.88%. Again, however, for some of the models, the performance worsens, once more showcasing randomness in the prediction of such values. For the main currency pair, which is USD/PLN, the results beat the baseline model.

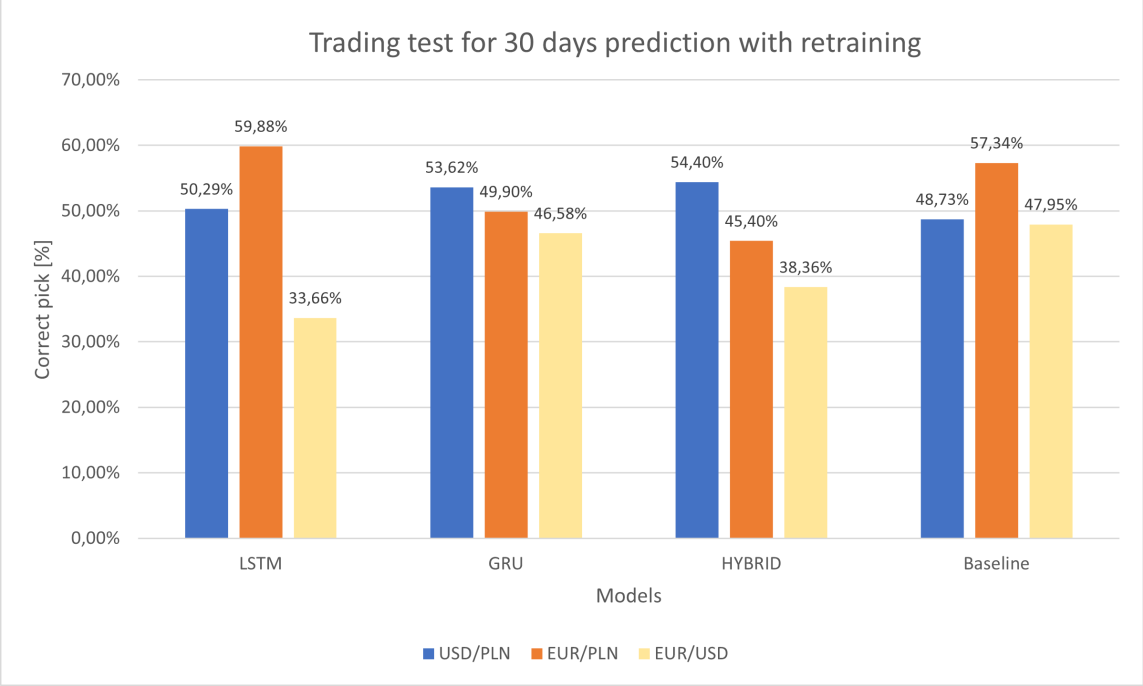


Figure 15: Trading test results for 30 days prediction with retraining

7.3 Discussion of Results

The goal of this thesis was to implement and test machine learning models that should beat the baseline model. Because all of the architectures were developed based on USD/PLN currency pair, one may notice that for all of the performed tests, evaluation and trading, the developed model beat the baseline model. Moreover, 17 out of 18 results for the next-day were better than the baseline model when it comes to trading tests. The only scenario, when the reference model achieved significantly better results is for the 30 days prediction for the EUR/PLN currency pair. As stated at the beginning of the thesis, the 30 days prediction is an additional test that the models have undergone and none of them were specifically trained for this task. As a result, one may observe that the results obtained in this part of the tests are "more or less" random. When the model is "unlucky" it predicts wrong values and scores around 43% of correct picks. However, this also works the other way around, allowing to score almost 60% of correct picks.

So the question is: why cannot the model be more accurate? As mentioned in the first chapters, such time-series are unpredictable. There are no concrete patterns in the data history. Even adding economic data did not help. Partially because the format of this data is not suitable for the daily-based currency rates, but also because there are much more factors that result in currency rate changes. There is no statistical or mathematical model that would explain those dependencies. If one had existed, the global market would be corrupted since the financial mechanisms would be predictable. Currency rates, as well as stock market prices or similar, may be linked to the chaos theory, which states that a very small change in, in this case, currency rates, might have a great impact on future values. We may not even be aware of such connections. As the example of 30 days prediction showed, currency rates characterize complex and chaotic behavior, that is perceived as random. With the current knowledge in this field, it is impossible to state that, i.e., certain changes in currency rate of specific pair 20 years ago might have had a great impact on future values or even on different data, such as different currencies, stock market prices or prices of products.

In conclusion, this thesis provided insight into developing machine learning models to predict time-series, such as currency rates. The primary goal was achieved and additional tests were carried out, to ensure the correctness and usefulness of such models.

7.4 Further Steps

Time-series prediction using machine learning has a very large scope of possible solutions. One may use existing architectures and combine them with others, tune the parameters, change the data, create their own algorithm from scratch or add different factors that might have a positive impact on the model's performance. There are many useful and interesting articles providing insight into such solutions. In this work, the main goal was to predict currency exchange rates for the next day but the tests that checked the performance for the longer time horizon showed the randomness of such approach. It would be valuable to focus specifically on the 30 days forecast.

List of Figures

List of Tables

References

[1] Burton G. Malkiel. *A Random Walk Down Wall Street*. New York, NY: WW Norton, 1973. ISBN: 9781324002185.

[2] A N Refenes, M Azema-Barac, L Chen, and S A Karoussos. "Currency exchange rate prediction and neural network design strategies". In: *Neural Comput Applic* (1993). DOI: https://doi.org/10.1007/BF01411374.

[3] Alaa Sagheer and Mostafa Kotb. "Time series forecasting of petroleum production using deep LSTM recurrent networks". In: *Neurocomputing* 323 (2019), pp. 203–213. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.09.082. URL: https://www.sciencedirect.com/science/article/pii/S0925231218311639.

[4] Ioannis E. Livieris, Emmanuel Pintelas, and Panagiotis Pintelas. "A CNN–LSTM model for gold price time-series forecasting". In: *Neural Computing and Applications* (2020). ISSN: 1433-3058. DOI: 10.1007/s00521-020-04867-x. URL: https://doi.org/10.1007/s00521-020-04867-x.

[5] Zahra Karevan and Johan A.K. Suykens. "Transductive LSTM for time-series prediction: An application to weather forecasting". In: *Neural Networks* 125 (2020), pp. 1–9. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2019.12.030. URL: https://www.sciencedirect.com/science/article/pii/S0893608020300010.

[6] Kibaek Kim, Changhyeok Lee, Kevin O'Leary, Shannon Rosenauer, and Sanjay Mehrotra. "Predicting patient volumes in hospital medicine: A comparative study of different time series forecasting methods". In: *Northwestern University, Illinois, USA, Scientific Report* (2014).

[7] Sheikh Mohammad Idrees, M. Afshar Alam, and Parul Agarwal. "A Prediction Approach for Stock Market Volatility Based on Time Series Data". In: *IEEE Access* 7 (2019), pp. 17287–17298. DOI: 10.1109/ACCESS.2019.2895252.

[8] Aleksander Palikuca and Timo Seidl. "Predicting High Frequency Exchange Rates using Machine Learning". In: *Master Thesis dissertation* (2016).

[9] WAZIR Mohammadi. "Currency Exchange Rate Forecasting Using Machine Learning Techniques". In: *Graduate School of Applied Sciences. Near East University* (2019).

[10] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[11] M.S. Islam and E. Hossain. "Foreign exchange currency rate prediction using a GRU-LSTM hybrid network". In: *Soft Computing Letters* 3 (2021), p. 100009. ISSN: 2666-2221. DOI: https://doi.org/10.1016/j.socl.2020.100009. URL: https://www.sciencedirect.com/science/article/pii/S2666222120300083.

[12] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies". In: (2001).

[13] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. "On the properties of neural machine translation: Encoder-decoder approaches". In: *arXiv preprint arXiv:1409.1259* (2014). DOI: 10.48550/arXiv.1409.1259.

[14] Rajib Rana. "Gated Recurrent Unit (GRU) for Emotion Classification from Noisy Speech". In: *arXiv preprint arXiv:1612.07778* (2016). DOI: 10.48550/arXiv:1612.07778v1.

[15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[16] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[17] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[18] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[19] J. D. Hunter. *Matplotlib: A 2D graphics environment*. 2007. DOI: 10.1109/MCSE.2007.55. URL: https://matplotlib.org/.

[20] *HistData*. URL: https://www.histdata.com/.

[21] Dawid Dzhafarov. *GitHub repository*. 2022. URL: https://github.com/dawiddzhafarov/Forex-Exchange-Rates-Prediction.

[22] Organisation for Economic Co-operation and Development. *OECD Data*. URL: https://data.oecd.org/.

[23] François Chollet. *Deep learning with python*. New York, NY: Manning Publications. ISBN: 9781617294433.