Warsaw University of Technology

FACULTY OF
MATHEMATICS AND INFORMATION SCIENCE

Bachelor's diploma thesis

in the field of study Computer Science and Information Systems

Application project for predicting the
value of stocks on the stock exchange

Michał Filipiak

student record book number 305778

Kamil Jakoniuk

student record book number 305781

thesis supervisor

dr hab. inż. Jerzy Balicki, prof. PW

WARSAW 2023

Abstract

Application project for predicting the value of stocks on the stock exchange

This thesis explores and describes the applications of deep neural networks in the domain of financial forecasting. The result of our work is a sandbox-type web application for training Long Short-Term Memory and Convolutional neural networks. It includes an intuitive interface for customizing, assessing, and comparing various machine-learning models. The paper covers an extensive theoretical overview of the underlying concepts, documentation of the application, and the results of numerical experiments performed using the said application.
Keywords: regression, forecasting, stock markets, LSTM, CNN

Streszczenie

Projekt aplikacji do przewidywania cen akcji na giełdzie

Poniższa praca ~~zgłębia i opisuje zastosowania~~ głębokich sieci neuronowych w dziedzinie przewidywania wartości akcji na rynkach finansowych. Rezultatem jest aplikacja internetowa stworzona do trenowania sieci LSTM i CNN. Składa się ona z intuicyjnego interfejsu służącego do modyfikacji, oceny i porównania różnych modeli sztucznej inteligencji. Dokument pokrywa również obszerny teoretyczny przegląd używanych pojęć z zakresu uczenia maszynowego, dokumentację stworzonej aplikacji i wyniki eksperymentów numerycznych przeprowadzonych przy jej użyciu.
Słowa kluczowe: regresja, prognozowanie, rynki finansowe, LSTM, CNN

Contents

Introduction

Stock market prediction is a challenging task that has attracted the attention of researchers and investors alike. Accurately predicting the future price movements of stocks can provide significant financial benefits. However, traditional methods of stock prediction, such as fundamental and technical analysis, have shown limited success in achieving this goal. Being consistently profitable as an individual market participant is very hard. There are hundreds of big, institutional players exploiting every market irregularity by using advanced statistical models and techniques such as high-frequency trading.

Recent advancements in machine learning and artificial intelligence have led to the development of new and more effective methods for stock prediction. One such method is the use of neural networks, particularly Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) Neural Networks. Our application provides an intuitive interface for the creation, customization, and training of neural network models as well as the comparison of their performance on various financial markets and tickers.

Stock markets are complex systems that are affected by a wide range of factors, such as global economic conditions, political developments, and company-specific events. These factors can produce a large amount of raw and unstructured data. CNNs are particularly effective at analyzing and interpreting this type of data. They are able to handle non-linear relationships and can model complex systems effectively. On the other hand, stock markets also exhibit temporal dependencies, which refer to the dependence of future price movements on previous price movements. These dependencies can span long periods of time and are difficult to capture using traditional methods. LSTM networks are specifically designed to capture and analyze long-term temporal dependencies in data. They are able to maintain a memory of past events and use this information to make predictions about future events.

Our application enables individual investors to create their own customizable neural networks without previous programming knowledge. It offers a wide range of financial data and

provides numerical metrics that can be used for comparing and evaluating the performance of the models. This paper consists of several chapters that describe the functionalities of said application and explain underlying problems.

The first chapter characterizes the most important stock exchanges in the world. It provides a brief background and introduction to the topic of the thesis. The second chapter explores the topic of regression analysis which is a crucial concept directly correlated with the problem of financial forecasting. It includes a general outline of the subject and a comparison of different types of regression. It also talks from a theoretical standpoint about two neural network models used to tackle the regression problem. These are convolutional and long short-term memory neural networks. The next chapter includes the details of created application. It discusses the business goal, functional and non-functional requirements, and the design of the most important parts of the project. The last chapter is devoted to numerical experiments performed using the application. It summarizes the observations and gives context to the results. It also describes numerical metrics used to compare the results.

1. A profile of chosen stock exchanges

1.1. Warsaw Stock Exchange

The Warsaw Stock Exchange, abbreviated WSE, is one of the most important stock exchanges in Eastern and Central Europe. WSE is the largest and most liquid of all the stock exchanges in its region, with a market capitalization of PLN 1.12 trillion as of June 2022. The exchange in its current form was established in 1991 after the collapse of communism in Poland. The founding of it was one of the efforts to broaden privatization in post-communist Poland. Since then it has grown immensely to become one of the most important centers for capital markets in its region. The stock exchange is open every workday from 8:30 am to 5:05 am and offers a wide range of products, including equities, bonds, derivatives, commodities, and ETFs. The most commonly traded stocks are those of banks and energy companies, while the largest bond issuers are the government and state-owned companies. WSE, like every other stock exchange, depends on both the current economic situation of the country of its origin and the companies which are listed on it (primarily the American Stock Exchange in New York (NYSE) and stock exchanges in Western Europe).

1.2. New York Stock Exchange

The New York Stock Exchange often referred to as NYSE, is one of the largest and oldest stock exchanges in the world. Its roots can be traced back to 1792 when 24 stockbrokers and merchants signed the Buttonwood Agreement, which laid the foundation for the formation of the NYSE. The exchange is home to some of the world's biggest and most successful companies, such as Apple, Microsoft, and Berkshire Hathaway as well as several important market indices including the S&P 500 and Dow Jones Industrial Average (DJIA). NYSE is known for its strict listing requirements, which guarantee that only the most prominent and financially stable companies are listed on the exchange. A wide variety of trading products ranging from equities and options to bonds, ETFs, and commodities such as oil and gas is offered on the exchange. NYSE utilizes

state-of-the-art technology to provide consistency and performance, even under the most volatile circumstances. As of October 20222, the total equity market capitalization of NYSE was around 22.1 trillion USD, making it the largest exchange in the world. It is the financial center of the world and as such, has an immense impact on markets all around the world.

1.3. National Association of Securities Dealers Automated Quotations Stock Market

The Nasdaq Stock Market is the world's first fully electronic stock market. It is the most active US exchange by trading volume and after NYSE, the second largest exchange in the world by total market capitalization. Listing requirements on Nasdaq are less rigorous than those on the NYSE, making it the perfect place for startups and smaller companies to go public.

1.4. London Stock Market

The City of London in England is home to the London Stock Market (LSE), a stock exchange. One of the world's oldest stock exchanges, it was founded in the sixteenth century. With access to more than 3,000 firms from around the world, the LSE is one of the top financial markets in the world. In terms of market capitalization, it is the biggest stock exchange in Europe and the third largest in the world.

1.5. Remarks and conclusions

The stock exchanges are of great importance to the global economy. They serve as a medium to buy and sell securities for everyone interested. With the continuously growing popularity of trading stocks, more and more people are becoming intrigued by it and trying to do it themselves. The application created by us serves as a great tool for beginners to enhance their understanding of stock price prediction and eases their introduction into the world of finance and machine learning.

2. Selected regression models

2.1. Regression analysis overview

Regression is a technique of estimating the relationship between a dependent variable, often called the outcome, or in the context of machine learning - the label, and one or more independent variables also called predictors or features. It is a wide term encompassing a set of statistical methods and a variety of machine-learning techniques and processes used for predictive modeling.

There are several distinct types of regression, the most basic one being linear regression, in which case one assumes that the dependent variable is related to the independent variables in a linear fashion. The problem of linear regression is solved by finding a linear combination that most precisely describes the correlation in the data, i.e. the sum of squared differences between the real data and the predicted line is minimized. The simplicity of the model is its biggest advantage and makes it a common tool for tasks such as forecasting or computing the trend line in the data. Although perfectly capable in many situations, the method makes several assumptions, which can lead to poor results. For one, it assumes homoscedasticity, which means that the variance of the errors is said to be constant and independent of the values of the features. The errors are also presumed to be normally distributed and not dependent on each other. Some techniques and extensions of the method address these problems, but in its basic form, linear regression may be too simple of a model to tackle more complicated data.

Another form of regression is polynomial regression. It is a more complex model, better suited for problems in which the relationship between the variables is non-linear. In this case, the expected value of the dependent variable y is modeled as an n-th degree polynomial in terms of the independent variable x. While in many cases the polynomial model can be beneficial, it is prone to overfitting, especially when the degree of the polynomial is high, which implies it may not generalize well outside of the original data. One can use regularization techniques, such as Lasso (L1) or Ridge (L2) to address this issue. Since polynomial equations can have more than

one solution, there might be multiple polynomials that fit the data. The impact of the individual coefficients of the polynomial on the whole model is also fuzzy because the corresponding monomials can be closely correlated. This may lead to difficulties with the overall interpretability of the model, which is another common downside of using complex polynomial regression models.

2.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of artificial neural network. They are commonly used in computer vision and image processing but can deliver promising results in financial forecasting problems as well. In this case, the linear data needs to be transformed into a 2D representation which is then used for convolution.

Convolution is a process of applying various filters, also known as kernels, to the input data. A kernel is simply a matrix of numbers that is slid across the 2D data. At each step, a dot product is computed and the result is the output of the convolution. In general, convolutional neural networks are very good at handling data that include some spatial or continuous coordinate information. In the case of time series, this information can be obtained from the order of the data, or time index. Differences between adjacent data points should be interpreted in completely separate way than same differences between the data points that are far away in the coordinate space. This fact makes this particular type of networks appropriate and matching to our problem.

2.2.1. Network architecture

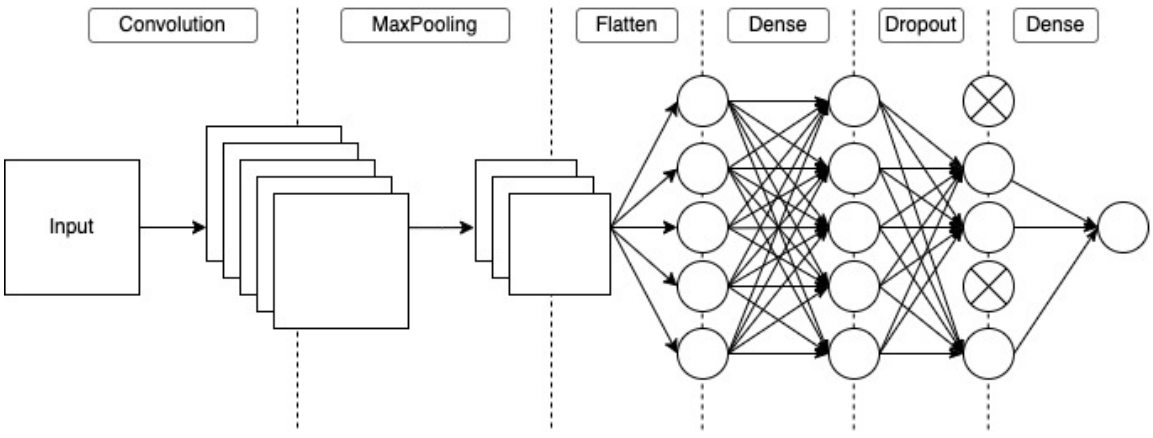

Figure 2.1: Visualization of the CNN implementation

The network in the application consists of several layers. The first one is the Convolutional layer responsible for performing the convolution on the input data. It is followed by a MaxPooling layer which reduces the dimensionality. The data is then reshaped into a one-dimensional vector using a Flatten layer. Next, a fully-connected Dense layer is inserted followed by an optional Dropout layer. The last Dense layer returns a single neuron which is the output of the network.

2.3. Long Short-Term Memory Networks

Long Short-Term Memory Networks (LSTMs) are a special type of recurrent neural networks, meaning they have additional feedback connections. These connections enable the network to not only use its current input, but also data from the past. On the other hand, the standard feed-forward networks do not have any memory and thus cannot use the previous results. Because of this, RNNs are able to process sequences of data, as opposed to only single data points. However, standard recurrent neural networks suffer from what is known as the vanishing gradient problem.

2.3.1. Vanishing gradient problem

The training is a process in which the network repeatedly updates its weights to obtain the minimum possible value of the loss function. To achieve that, the network internally utilizes the gradient, which holds the information on how the loss function changes with respect to each parameter. The gradient is computed using backpropagation, which involves calculating partial derivatives of the loss function with respect to all parameters. The computation is performed using the chain rule and commonly leads to the gradient tending to zero. This in turn results in infinitesimal changes to the weights of the network and very slow training. LSTMs were designed to tackle this specific problem.

2.3.2. Architecture of an LSTM unit

A standard LSTM unit consists of three gates that control the flow of the data and two types of memory - a long-lasting memory referred to as cell state, which is unique to LSTMs, and the short-term working memory called hidden state, also found in vanilla RNNs.

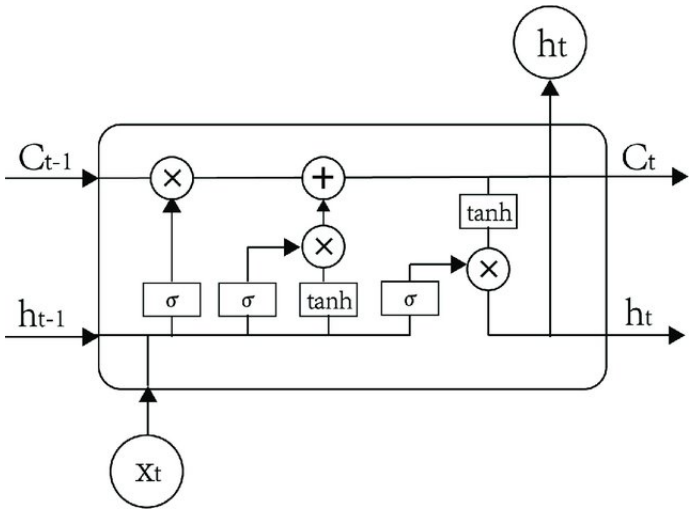

Figure 2.2: The structure of an LSTM unit

The goal of the forget gate is to determine how much of the information should be remembered and how much needs to be thrown away. It computes a linear combination of the current input of the cell and the hidden state passed from the previous unit with their respective weights. The result is then transformed using the *sigmoid* activation function to a value between 0 and 1 and used as a multiplier for the long-term cell state passed from the previous unit. Intuitively, the forget gate determines the percentage of long-term memory to be remembered.

The input gate is responsible for updating the cell state with new data. First, it performs a very similar operation as the forget gate, but with its own set of weights. The weighted input of the cell is added to the weighted hidden state and passed into the *sigmoid* function. The result of this operation serves as a multiplier for the output of the second part of the input gate. This part takes a linear combination of input and hidden state with yet another set of weights and this time, passes it to a hyperbolic tangent activation function. The function returns a number between -1 and 1, which then gets multiplied by the output of the *sigmoid* function from the first part and added to the long-term cell state.

The output gate performs a similar set of computations. The cell state modified by the previous gates is normalized using *tanh* function. The hidden state and input with their respective weights are again transformed by the *sigmoid* function. The two are then multiplied and the result is the new short-term memory or hidden state returned by the cell.

2.3.3. Network architecture

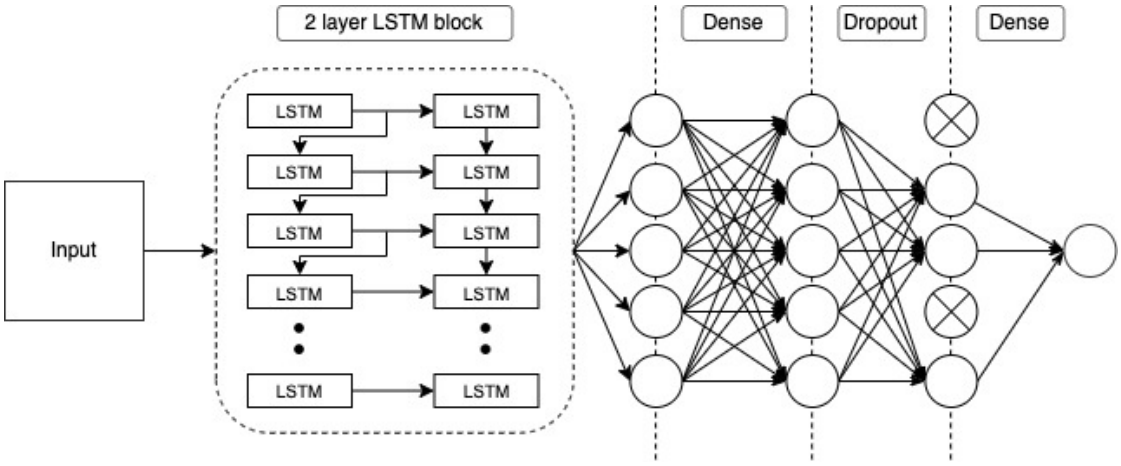

Figure 2.3: Visualization of the LSTM implementation

The network architecture in the application is built upon several consecutive layers. At first, the input data goes through 2 LSTM layers, each containing a set of units described above. Next, there is a fully-connected Dense layer and after that an optional Dropout layer. Finally, the last Dense layer outputs a single prediction.

2.4. Remarks and conclusions

Deep neural network models are relatively new, but powerful tools with applications in regression analysis. They are particularly suited for processing financial data, such as stock prices. Although forecasting is far from trivial, neural networks are well-suited for this task. A convolutional variation uses a technique adapted from the biological structure of the visual cortex and the LSTM modification uses a novel memory type that can remember long-term dependencies which solves the vanishing gradient problem. An understanding of the mechanisms that power these networks is immensely beneficial.

3. Group Project Documentation

3.1. Business goal and requirements

3.1.1. Business goal

Stock trading and machine learning are gaining in popularity. Many people hear and read about the possibilities of earning by selling and buying stocks or incorporating artificial intelligence to help them with various tasks. However, it is hard for a person without previous experience or education in these topics to begin their journey and they may feel overwhelmed. The goal of the project is to create an application, which allows the exploration of different methods for stock price prediction. While being consistently successful on the stock market is not the easiest, the application offers an introduction to the world of financial forecasting and neural networks via an accessible interface.

3.1.2. Functional requirements

As a user, I want to:

- choose the data to perform training on
- choose which artificial neural network I want to use
- adjust the hyperparameters of the model
- display the graph with the results of the prediction
- display multiple graphs to see the comparison
- display all the results with details in a table
- see the evaluation metrics
- clear the graph
- download the trained model
- upload the previously downloaded model

3.1.3. Non-functional requirements

Environment	The backend works on Windows 10 or later and MacOS 12.6 or later
	The application can be opened on Google Chrome and Safari
	The backend does not take more than 5GB
Usability	Intuitive user interface
	The application works on a standard 1920 x 1080 screen resolution
Scalability	The application is written following optimal coding standards
Maintainability	The application is created using modern technologies
Accessibility	The application can be used using only a keyboard
Supportability	The user guide is available for the user

Table 3.1: Non-functional requirements

3.2. Module design

The application consists of several modules nested inside each other. On the frontend side, we have an input handler that offers a visual interface used for parameter tuning. It is used for customizing and interacting with the model. The data from the interface is processed by the query builder and sent to the backend.

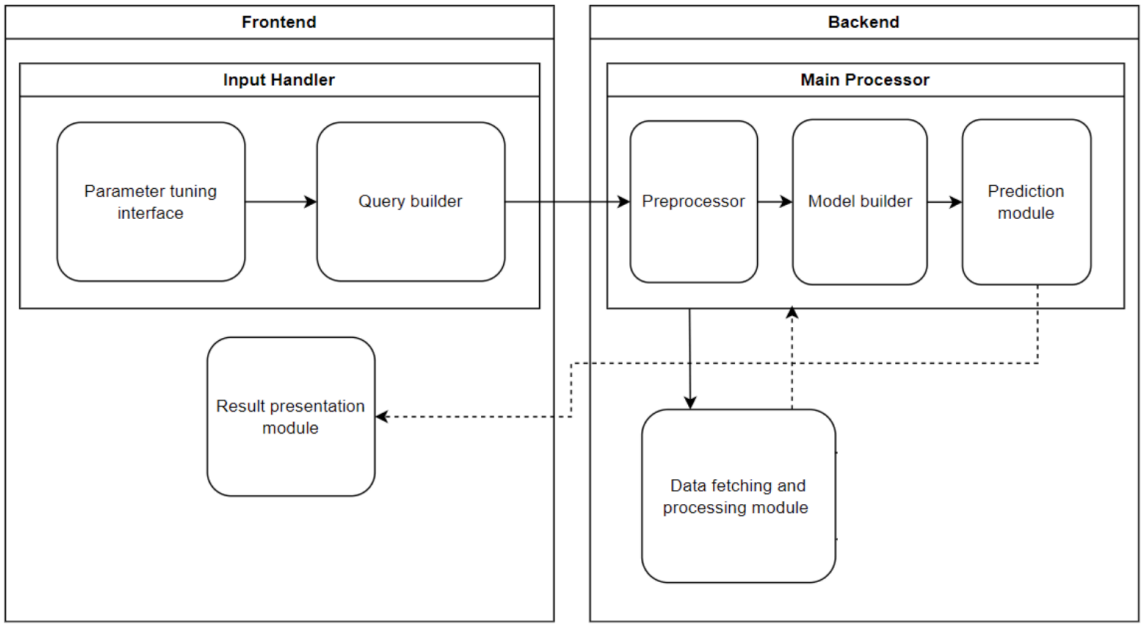

Figure 3.1: Visualization of the modules

After the request is received on the backend side, there is a separate module that fetches the data for a specified stock. The data then enters the module with the main business logic.

The main processor consists of three components:

- Preprocessor, which prepares the data for modeling, separates it into training, validation, and testing sets, and performs any necessary cleanup. It splits the data into windows of a size specified by the user and transforms and reshapes it to match the format expected by the neural networks.
- Model builder, which creates an actual model of the network, based on the query received from the frontend and the data from the preprocessor.
- Prediction module, where the actual training and forecasting takes place. Training is the most time-consuming component of the application, but the module is also capable of using preloaded models to serve the predictions and metrics on the fly. The results from this module are then sent back for display.

The last piece in this flow is a result presentation module. It displays the graphical representation of the results together with a set of useful numerical metrics. It also provides a separate page for the comparison of different models.

3.3. Parameters overview

Data selection

The interface offers a number of fields for selecting and preparing the data. The user can choose the ticker on which the prediction is performed. The tickers are sourced from a preloaded list based on the data source, which is another selectable field. There are two supported data sources: yahooFinance and stooq. They cover a majority of the most important stock markets and the companies listed on these exchanges. Data obtained from the external APIs contains the opening, high, and closing prices together with the volume for each day. The models use only the closing price for the forecasting. The user can also specify the starting and ending date using an intuitive date picker.

The ratio of training to testing data is also customizable, with the application automatically selecting the validation data ratio. Choosing the right train-to-test ratio has a substantial impact on the final results of the prediction. If the training ratio is very high, i.e. 0.8 or 0.9, the model might perform very well on the training data, but it may generalize poorly and the results on the unfamiliar data might be mediocre. On the other hand, if the data is split too much in the favor

of the testing, the overall performance of the model can drop, even though the generalization can be relatively better. It is thus crucial to choose this parameter with caution and experiment until the ratio is balanced.

Window size

Window size is also one of the most important factors when it comes to time series forecasting. It describes how many observations, in our case - past closing prices, are used as input for predicting future values. For example, if the window size is set to twenty, we transform the linear series of closing prices into two-dimensional vectors, each one holding twenty prices which are later used as inputs, and the twenty-first value as an output. The window is moved one by one and a new frame is created and appended to the set. The data in this form is then fed into the neural networks.

It is vital to find the window size that fits the selected data. Each stock and company has different characteristics and factors such as seasonality or long and short-term volatility need to be taken into account. When the window size is small, the model might not have sufficient information and can also overlook long-term trends and patterns. On the opposite, if the window is large, the model will capture these tendencies better but may be insensitive to short-term fluctuations. The size of the window should also be relatively small compared to the total number of available observations to avoid overfitting.

Model type

Activation function

The user can choose from several activation functions. Supported functions include:

- Rectified Linear Unit (ReLU), defines as:

$$f(x) = x^+ = max(x, 0) \tag{3.1}$$

- Sigmoid, defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} \tag{3.2}$$

- Softplus, defined as:

$$f(x) = \log(1 + \exp(x))) \tag{3.3}$$

- Softsign, defined as:

$$f(x) = \frac{x}{1 + |x|} \tag{3.4}$$

- Softmax
- Tanh
- Exponential
- Exponential Linear Unit (ELU)
- Scaled Exponential Linear Unit (SELU)

The activation is the function applied to the input of the neuron in the network layer. It transforms the value using some mapping and returns a result. Although it can be a simple linear function, usually more complicated functions are used to detect complex relations in the data.

Loss function

A loss or a cost function plays a significant role in the process of model training. One can view the training of a neural network as an optimization problem. The goal is to find the best possible weights for the connections between the neurons, which in turn determine how important the output of a given neuron is. To find them, one needs a way to measure the performance of a given set of weights. The loss function does exactly that, it computes the difference between the values predicted by the model and corresponding real values. To solve the optimization problem, one needs to minimize the output of the loss function. This is usually achieved using a gradient descent algorithm or another common optimizing method.

The application allows the user to choose from a set of popular loss functions including:

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- Mean Absolute Percentage Error (MAPE)
- Mean Squared Logarithmic Error (MSLE)

- Huber
- Logarithmic Hyperbolic Cosine
- Poisson

Optimizer

Optimizer is a method used to update the weights of the model in order to minimize the loss function. It is an algorithm used to perform the optimization described above. The choice of the optimizer is crucial for the performance of the model because it determines the manner in which it responds to the gradient of the cost function when looking for its minimum.

- Stochastic Gradient Descent (SGD)
- Adam
- RMSprop
- Adamax
- Adagrad
- Adadelta
- Ftrl
- Nadam

Each optimizer has its pros and cons and the choice ultimately depends on the characteristic of the data and a specific task.

Learning rate

The learning rate is an important parameter concerning the weight optimization process. It controls how much the parameters of the model are shifted in the direction of the negative gradient of the cost function. In other words, it describes the size of the steps toward the minimum of the loss or the speed at which the model adjusts itself. The common values for this parameter are 0.1, 0.01, 0.001, and 0.0001 and these are the values available from the interface.

A higher learning rate means the model converges quickly, but it may overshoot the global minimum. Respectively, choosing a low rate increases the chances of reaching the absolute minimum, but if one selects too small of a step the model might get stuck in a sub-optimal local minimum.

Epochs

An epoch is a full pass over the entire training data. The number of epochs is an important hyperparameter that impacts the overall performance of the model. A higher number of epochs means more iterations which generally means improved results. However, after too many passes, the model may adjust too much to the particular training data set, and it will not perform well on different data. This can be observed when the performance of the model on the validation set starts to degrade after a large number of epochs. To strike a balance between avoiding overfitting and giving the model enough time to train, it is vital to experiment with this parameter.

Batch size

The batch size refers to the number of observations handled by the model before the weights are readjusted. When this number is equal to 1, the process is called online learning. There are several trade-offs between this method and training in larger batches. The former means adjusting the weights more often, which reduces the bias when computing the gradients, but this method is more sensitive to the noise in the data. The latter is generally more computationally efficient, but since the weights are adjusted only after a large number of samples is processed, the complex relationships in the data may be overlooked.

Dropout

Dropout is a popular regularization technique used to avoid overfitting while training a neural network model. It accomplishes it by setting some input units in the layer to zero. Shutting off part of the neurons reduces the dependence on any particular input and makes the model more adaptable. It may also be useful to reduce the complexity if the model is very complicated. The percentage of zero-ed neurons is adjustable and the optimal value depends on the data and specific case.

3.4. Communication

The project is a full-stack web application and as such uses a typical request-response messaging pattern. The client layer is responsible for recording the inputs provided by the user in the interface and sending appropriate HTTP requests to the server. The possible requests may concern the change of the data, adjustment of various parameters, and saving or uploading of a pre-trained model. This layer is also responsible for displaying the results received back from

the server.

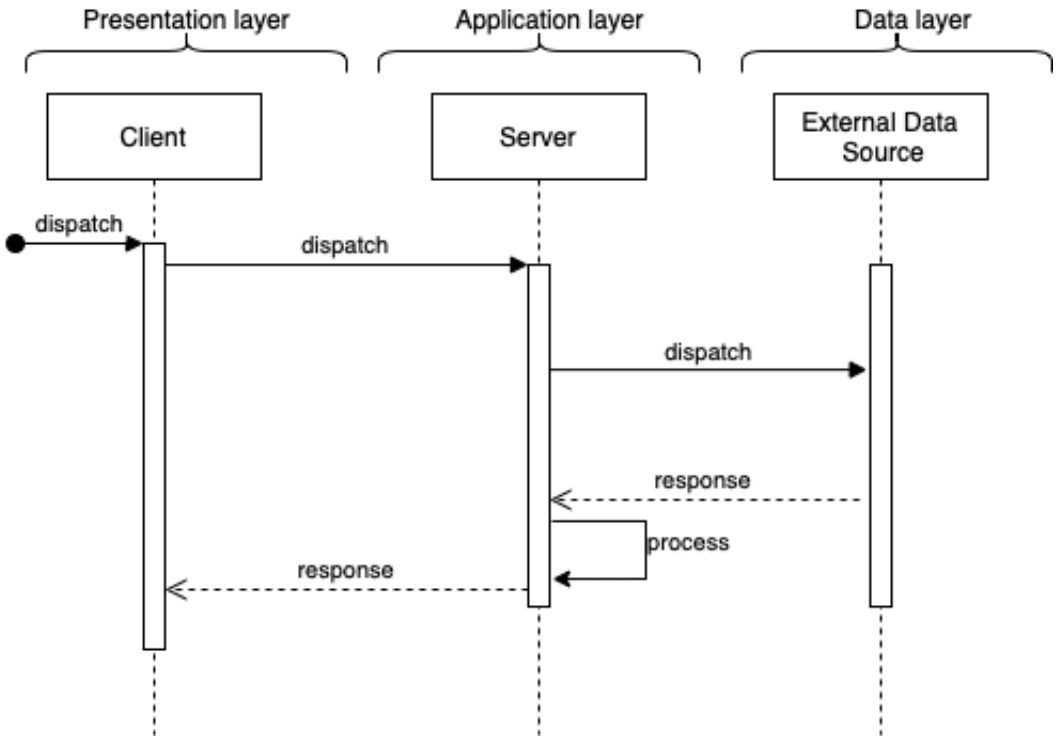

Figure 3.2: Communication diagram

The application layer handles all requests by either performing computations, sending back the responses, managing the data, or fetching and processing the data from an external source. After receiving the request to train a new model, it makes a request to the data layer to obtain it. After successfully fetching the data, the model is created and trained. The results are then passed back to the presentation layer for display. Processing the request to perform a prediction on a pre-loaded model is done in a similar way, but the steps concerning the model itself are skipped which reduces the waiting time to a minimum. If the user sends a request to download the model, the entire network including its architecture, weight values learned during the training, compilation information, and optimizer state will be serialized into TensorFlow SavedModel format, compressed, and downloaded.

The last layer is the data layer. It handles the server's request for financial data. The source of historical data is yahooFinance or stooq, accessed vi pandas_datareader package. The data is fetched with each request from the client layer, processed, split, and used for computation.

3.5. Technology selection

The application was realized as a web client connected to the server which exposes REST API and is capable of training and serving machine learning models. This choice required a range of technologies including:

- Python
- TensorFlow
- FastAPI
- TypeScript
- React

The frontend part of the project was created using Typescript and React and the backend used Python with FastAPI and TensorFlow.
The choice of Typescript over more popular Javascript was motivated by the number of advantages it provides over its competitor. Static typing has helped greatly in fixing bugs during the development as well as making the project much easier to maintain and expand. Moreover, as Typescript is an object-oriented language, it was more comfortable to write in as opposed to the prototype-based Javascript.
The React library was selected as writing the user interface in pure HTML, CSS, and Typescript would be strenuous and unnecessary. The decision to choose React and not any other framework was its great popularity with extensive community support and online resources.
Python was decided on because of its immense applicability in the field of machine learning. The part of the project responsible for creating and training the models was done using TensorFlow as it enables easy and efficient building, training, and evaluation of the deep learning models. The communication between the server and client side required a simple, yet robust web framework - this made FastAPI an excellent choice.

3.6. Work Division

No	Task	Author(s)
1	Project planning	
1.1	Choice of the topic, research	Kamil Jakoniuk, Michał Filipiak
1.2	System design, spike	Kamil Jakoniuk, Michał Filipiak
2	Server side implementation	
2.1	Module design, technology selection	Kamil Jakoniuk, Michał Filipiak
2.2	Processing the data	Michał Filipiak
2.3	Preparing neural networks	Michał Filipiak
2.4	Creating APIs	Michał Filipiak
3	Client side implementation	
3.1	Module design, technology selection	Kamil Jakoniuk, Michał Filipiak
3.2	Creating prediction, upload pages	Kamil Jakoniuk
3.3	Styling components	Kamil Jakoniuk
3.4	Creating endpoint handlers	Kamil Jakoniuk
4	Testing + experiments	
4.1	Unit testing	Kamil Jakoniuk, Michał Filipiak
4.2	Manual, acceptance testing	Kamil Jakoniuk, Michał Filipiak
4.3	Performing experiments using application	Michał Filipiak
5	Thesis	
5.1	A profile of chosen stock exchanges	Kamil Jakoniuk, Michał Filipiak
5.2	Selected regression models	Kamil Jakoniuk, Michał Filipiak
5.3	Group project documentation	Kamil Jakoniuk, Michał Filipiak
5.4	Numerical experiments using project application	Michał Filipiak

Figure 3.3: Division of the work

4. Numerical experiments using project application

Several experiments were performed to demonstrate the usage of the application and its possibilities. The experiments use closing prices of various stocks from dates between the 1st of January, 2018, and the 1st of January, 2022.

4.1. Comparing the performance of LSTM models with varying batch sizes

The first experiment concerns the impact of batch size on the performance of the LSTM model. The ticker used for forecasting in this experiment is AAPL listed on the Nasdaq stock market. We use ReLU activation with the MSE cost function and the Adam optimizer, the number of epochs is fixed at 50. We start with the online learning, batch size set to 1 meaning the weights are adjusted after each pass. The next two predictions are made using mini-batch learning, with batch size set to 10 and 50 respectively. The last prediction is performed using a batch size of 200, which is relatively high in the context of the particular data we are using.

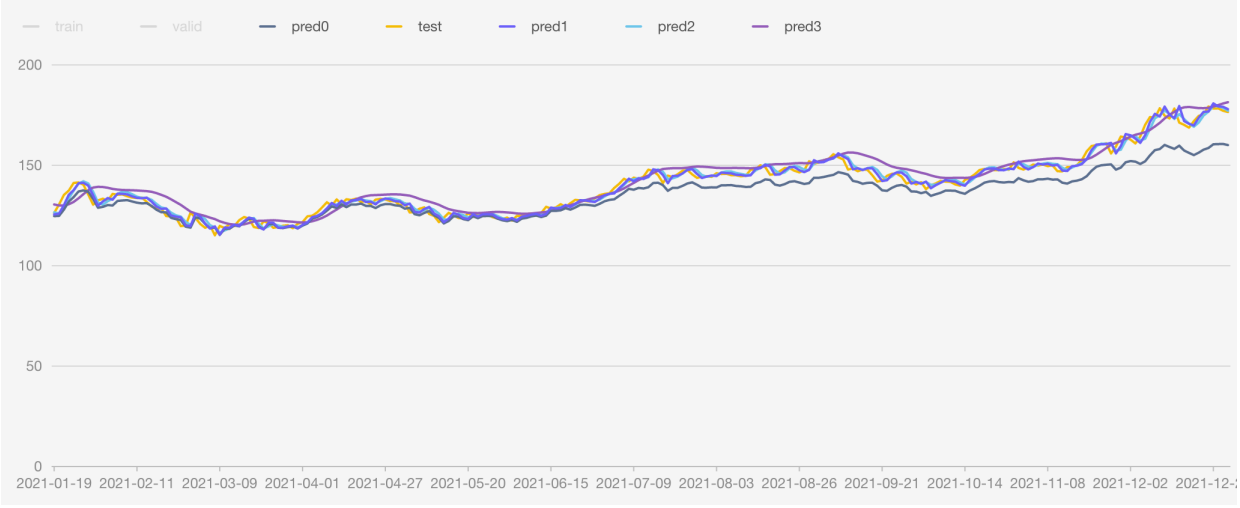

Figure 4.1: Graphs of LSTM predictions with different batch sizes

The results of prediction performed using iterative learning, visible in the figure as *pred0*

seem to respond to the data well, but the frequency and the magnitude of the steps are too aggressive. The results obtained by using mini-batch trained models *pred1* and *pred2* are more accurate. They trail the testing data almost perfectly. The plot of *pred3*, which is the result of learning with batch size set to 200 is completely different than the other three. It follows the general direction of the real data but is much smoother. This is expected because the frequency with which the model updates the weights is lower compared to the previous ones.

	NAME	MAE	MAPE	RMSE	Training time
+	pred0	5.28	3.723	6.878	319.868
+	pred1	1.692	1.21	2.202	113.081
+	pred2	1.869	1.338	2.409	81.984
+	pred3	3.455	2.427	4.252	74.323

Figure 4.2: Metrics of LSTM predictions with different batch sizes

The numerical metrics provided after each prediction confirm these observations. The first model not only was the worst out of the four in terms of performance, but it also took the longest to train. The high-batch-size model performed slightly better and the training was the shortest. Mini-batch models' results were very similar to each other, with a noticeable trade-off between training time and performance - the one with batch size set to 10 had slightly better results but took longer to train than the other.

4.2. Comparing the performance of the optimizers in CNN models

The next experiment explores how the choice of the optimizing algorithm affects the performance of a convolutional neural network. For this experiment, the predictions are performed on the AMD stock, which was chosen because of its historically high volatility which adds a layer of complexity. We also try three different learning rates (0.1, 0.01, 0.001) for each optimizer and use the best one for later comparison of the algorithms.

4.2.1. Stochastic Gradient Descent (SGD)

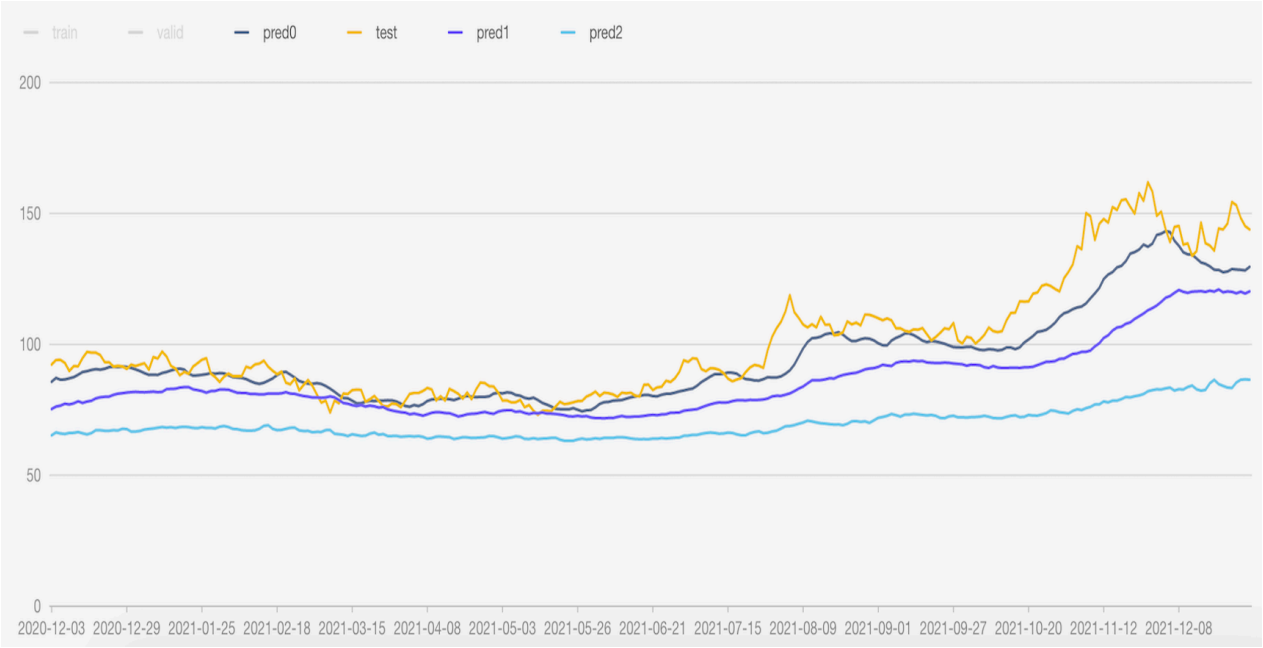

Figure 4.3: Graphs of CNN predictions with SGD optimizer using different learning rates

Stochastic gradient descent is a common algorithm, that serves as a base for many other optimizers. It first initializes the parameters of the model with random values. Then computes the gradient of the loss function with respect to each parameter and adjusts them with the goal of reaching the minimum of the cost function. The step size or the magnitude of this change is controlled by the learning rate.

	NAME	MAE	MAPE	RMSE	Training time
⊕	pred0	6.674	6.596	9.481	12.726
+	pred1	14.977	16.639	18.751	26.192
+	pred2	30.952	43.046	35.416	63.671

Figure 4.4: Metrics of CNN predictions with SGD optimizer using different learning rates

The results for a learning rate equal to 0.1 were fairly good, with RMSE equal to 9.5. The other values didn't perform that well, and the training took much longer.

4.2.2. Adam

After experimenting with different learning rates, we found that the optimizer performed the best with a learning rate set to 0.0001. This is consistent with the sources commonly pointing to values around $3e{-}4$ as the best option for this algorithm.

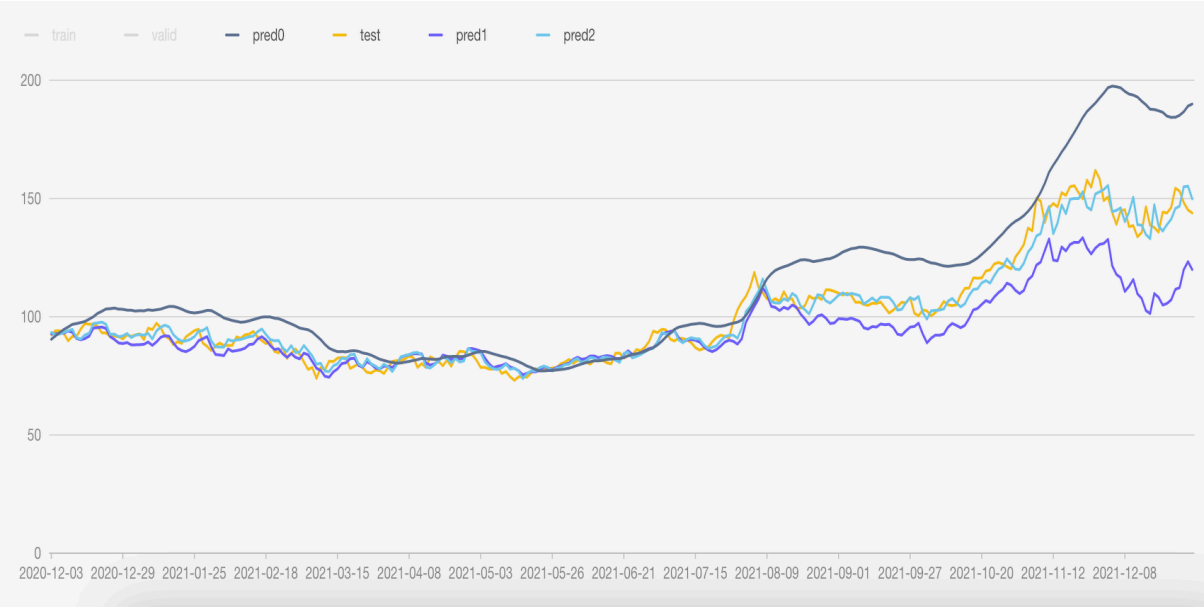

Figure 4.5: Graphs of CNN predictions with Adam optimizer using different learning rates

Adam optimizer is an extension of the stochastic gradient descent algorithm, which utilizes a concept of momentum. It means that the optimizer remembers the magnitude of the previous adjustment and uses this information for the next update. It also incorporates an adaptive learning rate technique, that takes a starting learning rate and treats it as another parameter for optimization during each iteration.

	NAME	MAE	MAPE	RMSE	Training time
+	pred0	12.739	9.768	17.949	22.352
+	pred1	7.987	7.716	12.189	16.017
+	pred2	3.266	3.132	4.328	17.373

Figure 4.6: Metrics of CNN predictions with Adam optimizer using different learning rates

The performance of the Adam optimizer was very good, especially with the correct initial learning rate. The best prediction obtained an RMSE score of 4.3, with satisfactory training time.

4.2.3. Adagrad

The Adagrad which is a modified version of Adam performed the worst in the preliminary comparison. The optimal learning rate obtained from this test was 0.1.

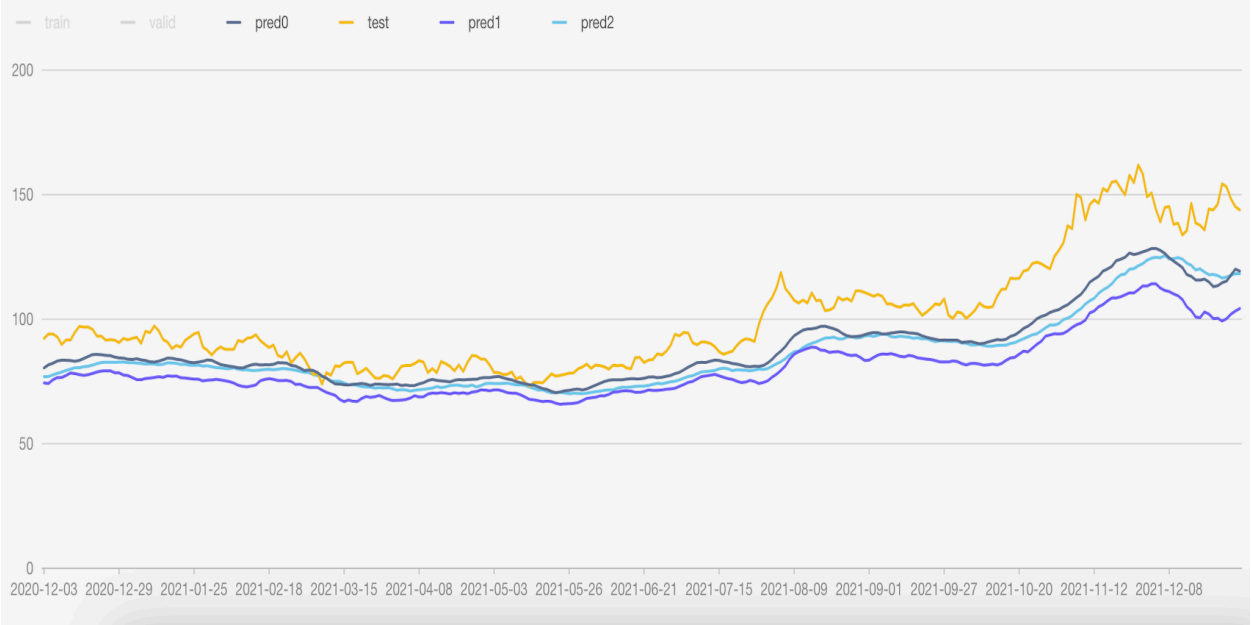

Figure 4.7: Graphs of CNN predictions with Adagrad optimizer using different learning rates

The main difference between Adagrad and SGD is that it uses per-parameter adaptive learning rate adjustment. The global learning rate is still an important hyperparameter and is treated as a base, but is updated during the optimization.

	NAME	MAE	MAPE	RMSE	Training time
+	pred0	12.167	12.94	14.968	13.432
+	pred1	19.939	23.632	22.862	21.113
+	pred2	14.415	15.973	17.232	49.233

Figure 4.8: Metrics of CNN predictions with Adagrad optimizer using different learning rates

The results of the Adagrad were significantly worse, with the best RMSE score of 14.9.

Comparison of optimizer algorithms with selected learning rates

Finally, we can compare the optimizers, each one with an optimal learning rate obtained during the above testing. The models are trained for 100 epochs, with a window of size 10, and

a mini-batch approach with batch size set to 10. The activation for fully-connected layers is the ReLU function and the MSE is used as a loss function. The Dropout layers are omitted in this specific example, but the results with Dropout before the last Dense layer were very similar.

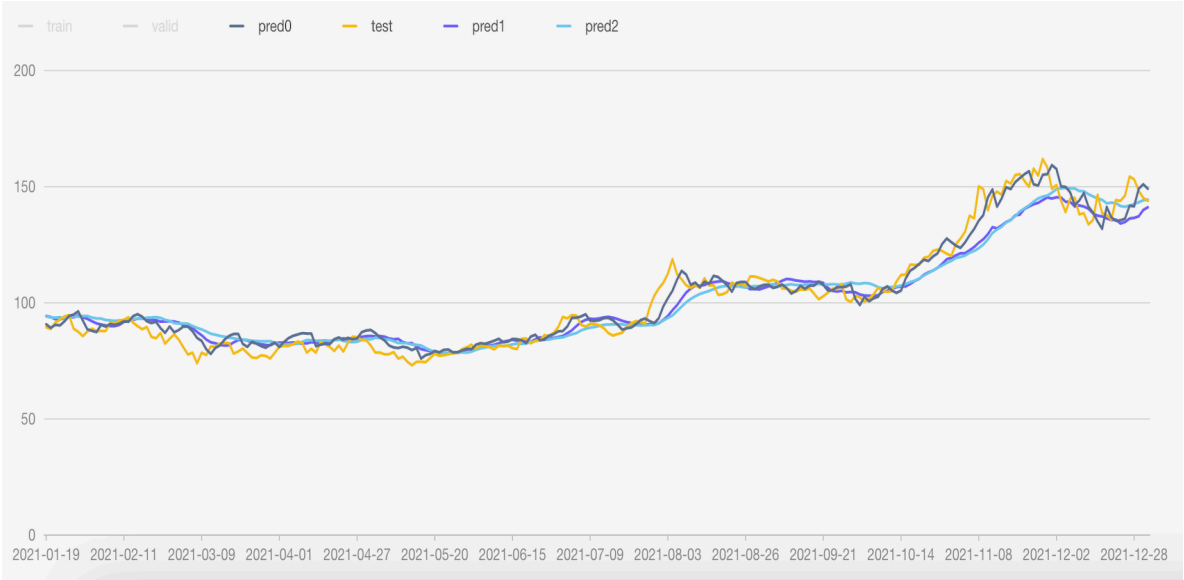

Figure 4.9: Graphs of CNN predictions with different optimizers and learning rates

Adam was the best-performing optimizing algorithm out of the three, with a real mean squared error of 4.85. Stochastic gradient descent was the next one with a score of 6.86, followed by Adagrad at 7.16.

This matches the literature on the subject, pointing to the Adam algorithm as a very well-rounded and robust optimizer. The overall performance scores of the models using the Adam optimizer during this and other tests were the most consistent, compared to all the other available options.

	NAME	MAE	MAPE	RMSE	Training time
+	pred0	3.767	3.674	4.852	18.88
+	pred1	4.995	4.792	6.855	13.448
+	pred2	5.295	5.097	7.161	14.37

Figure 4.10: Metrics of CNN predictions with different optimizers and learning rates

4.3. Numerical metrics and comparison tools

The application offers several metrics, which are displayed on the prediction, and comparison pages. They offer a numerical way to measure the performance of the model and the accuracy of the forecasting.

4.3.1. Mean Absolute Error (MAE)

Mean absolute error is a very popular metric for problems concerning regression. It is a measure of the average error magnitude of predicted values. It can be defined as:

$$MAE = \frac{\Sigma_{i=1}^{n}|y_i - x_i|}{n} \tag{4.1}$$

where y_i is the predicted value and x_i the real value. To obtain the mean absolute error we simply need to divide the sum of all absolute errors by the size of the sample.

Using mean absolute error to measure the network's performance has several advantages. It is relatively insensitive to outliers, which is useful when there is noise in the data or the data was not properly prepared for the prediction. It is also a very simple metric, which makes it easy to explain and interpret. However, it is worth noting that it is not a scale-invariant measure, meaning it depends on the scale of the considered data, which renders it useless for comparison with series that use a different scale.

4.3.2. Mean Absolute Percentage Error (MAPE)

Mean absolute percentage error is a scale-invariant counterpart of MAE. By dividing the difference between the actual and predicted value, by the actual value itself, it is possible to use it to compare the performance of the models on different data sets making it more versatile. It is expressed by the following formula:

$$MAPE = \frac{1}{n}\Sigma_{i=1}^{n}\left|\frac{x_i - y_i}{x_i}\right| \tag{4.2}$$

where x_i and y_i are actual and predicted values respectively. MAPE is a very intuitive measure - it describes an average percentage error of the prediction. Still, there are several downsides one needs to be aware of. It is not defined for zero, and for near-zero values, MAPE tends to infinity - this usually is not the case in the context of financial time series, but it can become an issue when performing the forecast on penny stocks. The measure also puts a bigger penalty on negative errors than on positive ones, i.e. it may underestimate the errors where actual observation is

greater than the predicted value.

4.3.3. Root Mean Square Error (RMSE)

Root mean square error also called root means square deviation is another common measure for time series problems. It is defined as:

$$RMSE = \sqrt{\frac{\Sigma_{i=1}^{n}(y_i - x_i)^2}{n}} \tag{4.3}$$

where y_i is a predicted value, x_i is a corresponing real value and n is a number of observations. Root mean square error is a good general-purpose measure, but like MAE it is scale-dependent, which makes it useful for comparing the performance of different models on the same data. When computing RMSE, the errors are squared, which makes this measure more sensitive to outliers than for example MAE. The outputs are also not as easy to interpret because the computation is slightly more complicated. Nonetheless, it is a very robust and informative numerical measure of the model's performance.

4.3.4. Training time

The application also displays the training time of a given neural network. This is probably the simplest, yet descriptive metric of the training. It does not provide information about the sole performance of the model but adds another important piece for comparison.

4.4. Remarks and conclusions

In this chapter, we presented the results of a couple of experiments utilizing our application. We examined the impact of varying batch sizes on the performance of convolutional neural networks and provided possible explanations of the observed phenomena. We also compared various optimizing algorithms and their performance in LSTM networks. We tested the effects of choosing different learning rates on the optimizers and explored the rationale behind their behavior. We described available numerical metrics and their advantages. Finally, we demonstrated the usability of the application in a practical environment.

5. Summary

In this thesis, we cover the problem of forecasting the prices of stocks in financial markets using two types of deep neural networks. The thesis provides the theoretical background needed to understand the problem of regression, the architecture of used neural networks, and underlying concepts. It also gives an overview of several stock markets from a historical and economical standpoint.

Additionally, we introduce the application developed to assist with creating, training, and comparing easily-customizable neural network models. It uses an accessible web interface, allowing users with a non-technical background to construct their own models with a no-code approach. The business motivation behind the application, its design, components, and justification for utilized technologies are contained in a separate chapter.

The thesis also covers several numerical experiments performed using the created application. They demonstrate its usage and applicability in the domain of financial forecasting. The observations obtained from the experiments are described and complete with explanations, that give insight into the reasons behind the behavior of the models.

Bibliography

[1] Sidra Mehtab, Jaydip Sen *Stock price prediction using LSTM, RNN and CNN-sliding window model*, IEEE, 2021

[2] Staudemeyer, Ralf C. and Morris, Eric Rothstein, *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks*, 2019.

[3] Max Drummy, Mag Gardner, Joanne Quinn, Joanne McEachen, Michael Fullan, Corwin Press, *Dive into Deep Learning*, 2019

[4] Aileen Nielsen, BPB Publications, *Practical Time Series Analysis: Prediction with Statistics and Machine Learning*, O'Reilly Media, 2019

[5] Ivan Gridin, BPB Publications, *Time Series Forecasting using Deep Learning*, 2019

[6] Andreas C. Müller, Sarah Guido *Introduction to Machine Learning with Python*, O'Reilly Media , 2016

[7] Yehezkel S. Resheff, Itay Lieder, Tom Hope *Learning TensorFlow: A Guide to Building Deep Learning Systems*, O'Reilly Media , 1973

[8] Michael Becket *How the Stock Market Works: A Beginner's Guide to Investment*, 2002

List of symbols and abbreviations

~~nzw.~~	~~nadzwyczajny~~
*	star operator
~	tilde

List of Figures

List of tables