Capiston Project - Investment and Trading

Machine Learning Engineer Nanodegree

Project Definition

Project Overview

Investment firms, hedge funds and even individuals have been using financial models to better understand market behavior and make profitable investments and trades. A wealth of information is available in the form of historical stock prices and company performance data, suitable for machine learning algorithms to process. Accurate prediction of stock market returns is a very challenging task due to volatile and non-linear nature of the financial stock markets. Since a lot of investors use AI models to work with stocks it's become very important to predict price for making decisison about future investment or porfolio rebalancing. For those who use Reinforcement Learning it's also quite important since agent should know when to hold, sell or buy new assets to expect better reword for the action in future.

In this project price predictor will be created which will be able to predict stock market price of the next day by providing stock preces or other featrures of the previous days. "Transformer" model is selected for this project since it's show good results on sequential data. Model will be trained using data provided by Yahoo! Finance. The project was inspired by my interest to quantitative finance and machine learning, and a lot of resources available around this topic nowadays.

Problem Statement

Build stock price predictor which will use historical data to predict stock price for given date. It's quite clear for me that it's impossible to predict future sock movement, becouse it depends on a lot of different factors. For example:

- social activity
- company performance
- natural desisters
- etc.

Some of this factors can be predicted from experience but some of them still hard to predict, like natural desister. Hence the main goal of this project is to create predictor which try to find patterns in the past stock movement and help with decision making process together with some other information. Ideally there should be many other predictors which use different information for predictions, for example market news or satellite images, etc. All this predictors should be combined in ensamble model for better prediction, but this require a lot of time and efforts to be implemented. In this project we try to reduce scope and build small part of bigger system and do small research if it worth to make this type of predictions at all.

Metrics

To measure model performance mean squared error (MSE) is used, better model should have lower error. When best model selected it will be tested against real stock data with measuring error. To check how well it perform we have to check result against similar models, for that reason similar researches was found. Please check links below.

According to read researches it's possible to acheive quite good accuracy on some markets. For this project try to acheived MSE <= 0.002 for markets with low volatility since markets with high volatility more risky and difficult to predic.

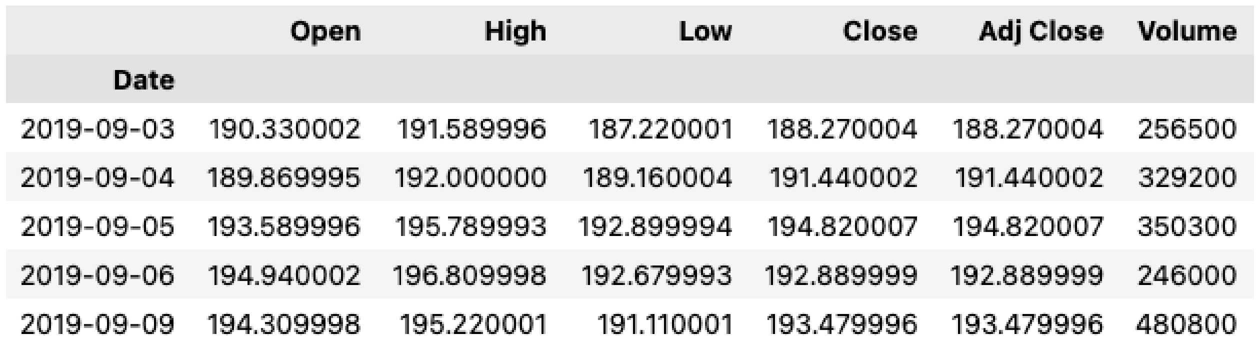
Research articles:
- Stock Market Analysis + Prediction using LSTM | Kaggle
- Stock predictions with state-of-the-art Transformer and Time Embeddings

Analysis

Data Exploration

Data for this analysis will be taken from Yahoo Finance API. It contains daily stack prices. For this project data from past 5 years will be taken.

Finance data contains following features:
- Date - price date (string)
- Open - stock open price (float)
- High - stock high price (float)
- Low - stock low price (float)
- Close - stock close price (float)
- Adj Close - stock adjusted close proce (float)
- Volume - stock trading volume (float)



Date	Open	High	Low	Close	Adj Close	Volume
2019-09-03	190.330002	191.589996	187.220001	188.270004	188.270004	256500
2019-09-04	189.869995	192.000000	189.160004	191.440002	191.440002	329200
2019-09-05	193.589996	195.789993	192.899994	194.820007	194.820007	350300
2019-09-06	194.940002	196.809998	192.679993	192.889999	192.889999	246000
2019-09-09	194.309998	195.220001	191.110001	193.479996	193.479996	480800

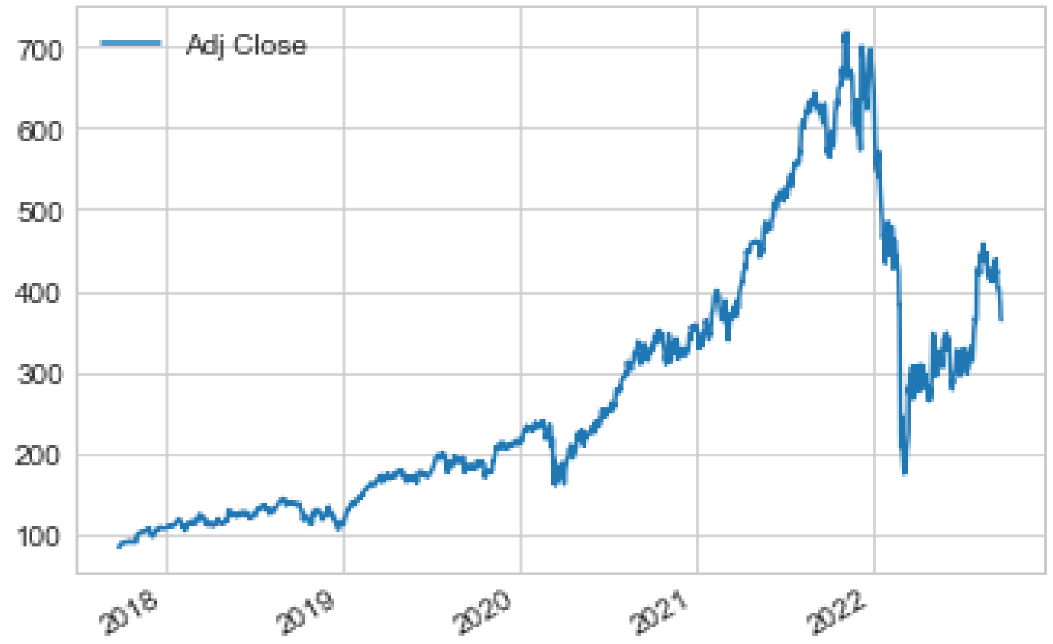
For this project EPAM stock data will be used. For the last 5 years. "Adj Close" will be used as target to predict base on other features or calculated indicators.

Data Visualization

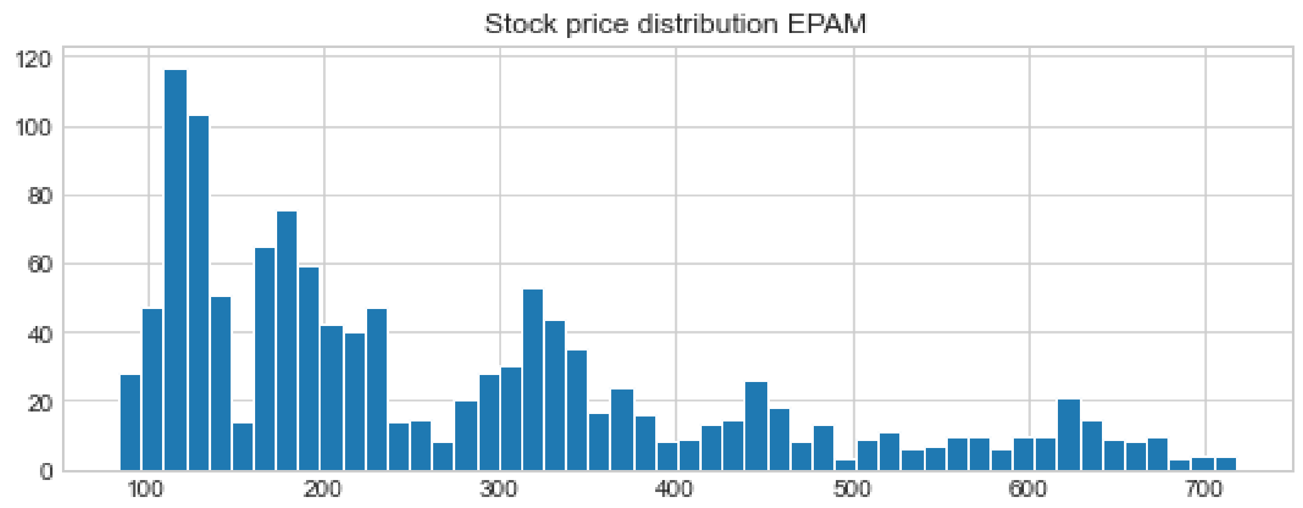
All charts build with the data for the last 5 years.

Stock price visualization

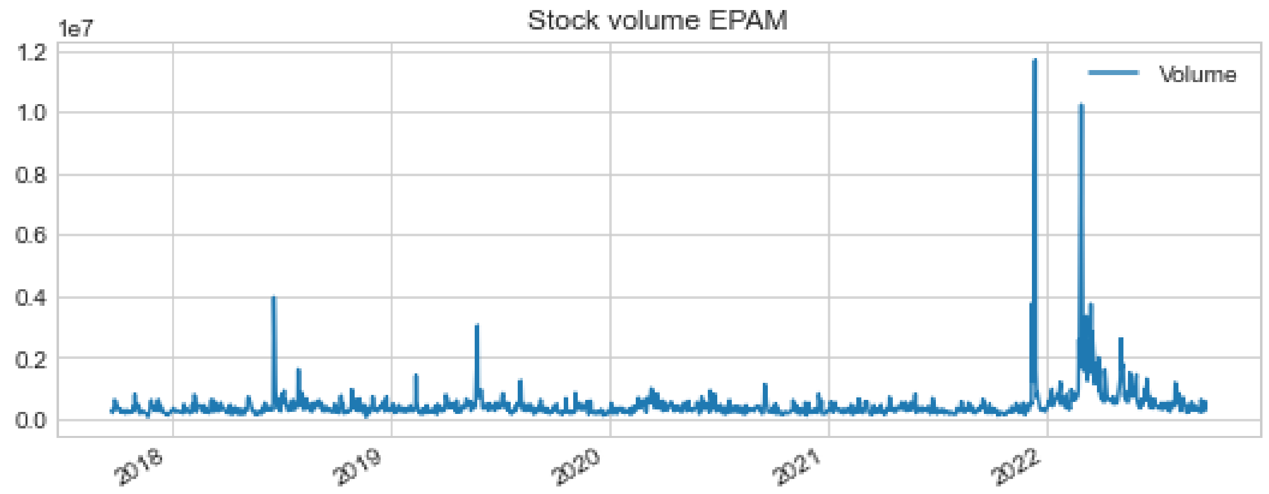
Stock price visualization show us stock behaviour. It might be useful when we try to analyze predicted values. To see were the model is struggle to predict stock price. We see from the plot that for the model it might be difficult to predict stock drop (around 2022 year) which is not usual behavior.



From this chart we see that starting from 2018 up to 2022 stock it growing and then drop significantly. For this dataste network might struggle to predict this drop, because it's not typical behaviour for it.



Stock volume visualization

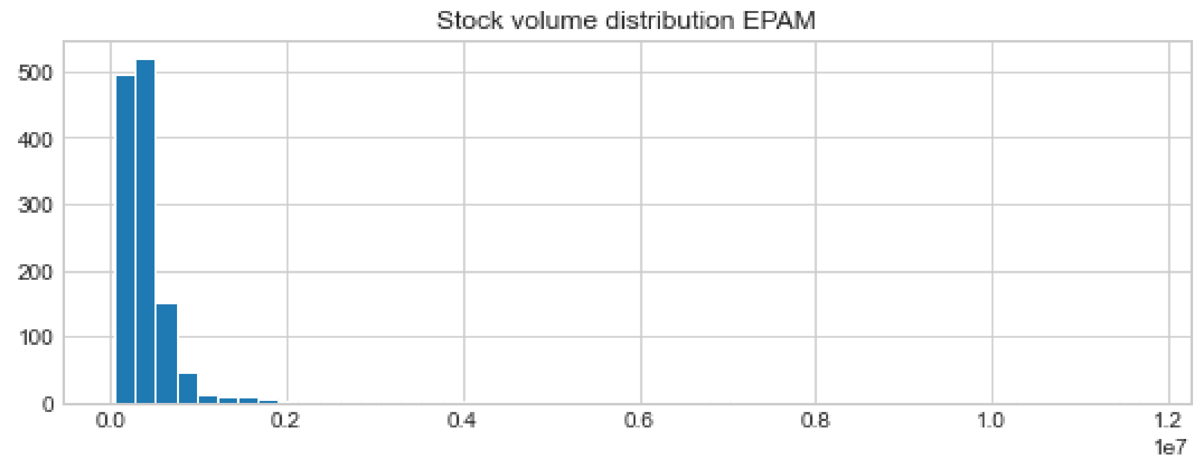


At the volume chart we see two peeks which can be considered as outliers and need to be reduced in size to the mean, otherwize model will struggle to make predictions since most of the time volume signal will be to low comparing to the volume peeks. For our case it’s better to adjust this anomalies to some meaningful value, because thouse peeks doesn’t means anything in isolation.

It can have 3 scenario which is lead to this result:

- Someone sell a lot
- Someone buy a lot
- Someone sell and buy a lot

Since our model won’t be able to predict a future and we try to use it to predict price base on some patterns, market behaviour, for us it will be enough to know that trading volume at this day was really high. And “high” value will be defined during data feature scaling phase.



We can clearly see from the volume distribution that most of the values sits at the beginning of the chart. This can lead to biased prediction.
On this project "Adj Colose" and "Volume" sequences will be used as features and "Adj Colose" following sequence will be used as a target.

There are few reasons why other trading indicators will not used:

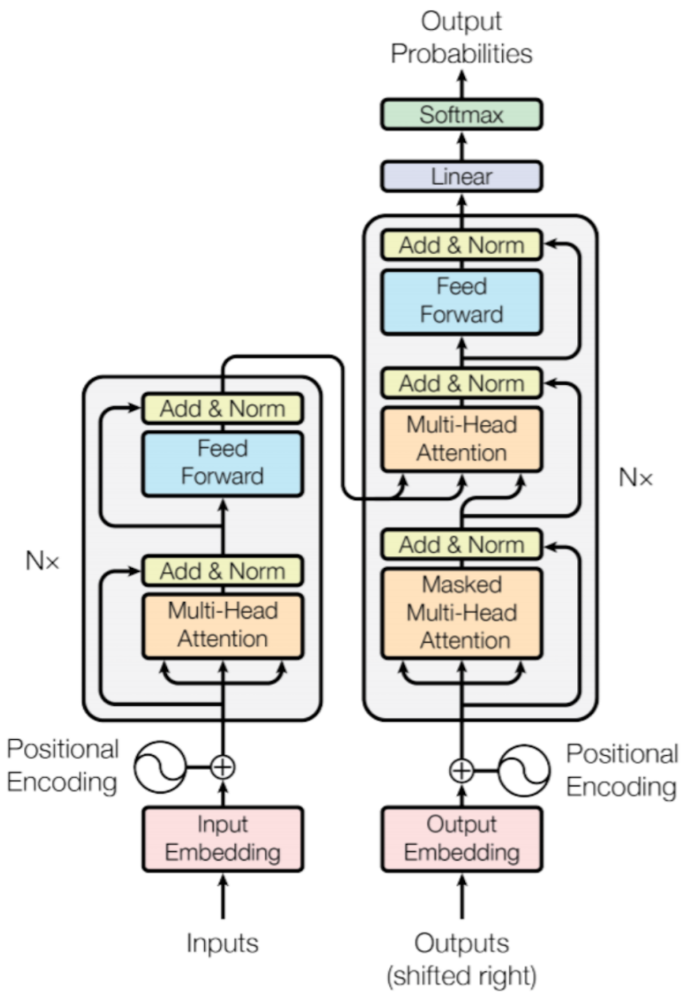
- Model simplification, almost any trading indicator like MA (moving average), ROC (price rate of change) has it's own parameters to calculate, like period and others. Hence using it in price prediction required find an optimal parameters, which is required additiona investigation. Since ML algorithm by it's nature consists of neurons which is adjusted according to the provided features I think that it should be enough to use row price and volume for prediction;
- Price movement vanishing, some trading indicators like MA and VWAP (volume weighted average price) for example, smoothing price hance might vanish some important patterns of stock movement;

Probably if we try to predict sell or buy signals or try to predict daily price based a lot of trading data (like 1 minute candle) it's good idea to use technical indicators for that. But for current task where we try to predict next day price use previous price data it will be a good idea to use raw data.

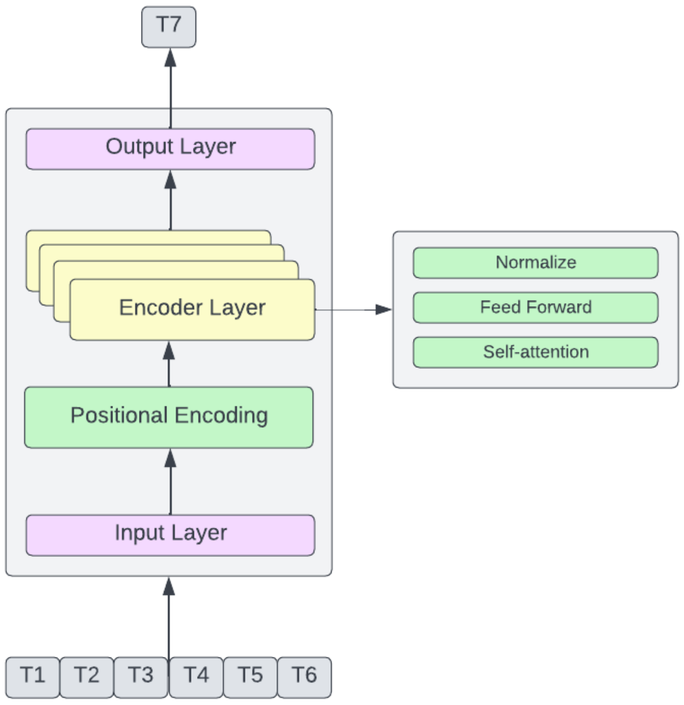
Algorithms and Techniques

The classifier will be Transformer neural network which is the state of the art neural network for time series prediction. It was first proposed in the paper "Attention Is All You Need".
Tranformers has several benefits comparing to other algorithms like LSTM or GRU networks. For example, gradient vanishing problem solved in transformer and that the input sequence can be passed parallelly. Transformers mostly used in NPL area but was successfully adopted for timeseries predictions.

Transformer is encoder-decoder architecture based on attention layers. Original transformer architecture proposed in "Attention Is All You Need" is shown below.



But for timeseries prediction we need to adjust this architecture. Our predictor architecture will looks like this.



1. Input layer - embedding layer which is transform input timeseries into vector;
2. Positional encoding - encode element position in timeseries, since timeseries passed in parallel we have to provide information about how this elements connected between each other;
3. Encoder layer - encoder layer consists of self-attn and feedforward network. This standard encoder layer is based on the paper “Attention Is All You Need”;
4. Output layer - decode encoder output into stock price.

Provided architecture will be implemented using PyTorch framework.
1. Input layer - nn.Linear
2. Positional encoder - custom layer implemented with Time2Vec algorithm according to following paper Time2Vec: Learning a Vector Representation of Time .
3. Encoding layer - nn.TransformerEncoderLayer
4. Output layer - nn.Linear

Model training parameters:
- Number of epochs;
- Learning rate
- Sequence length
- Weight decay

Model parameters:
- Input layer size
- Output layer size

- Transformer encoder parameters
    - Number of heads in the multiheadattention models
    - Dimension of the feedforward network model
    - Number of expected features in the input

Methodology

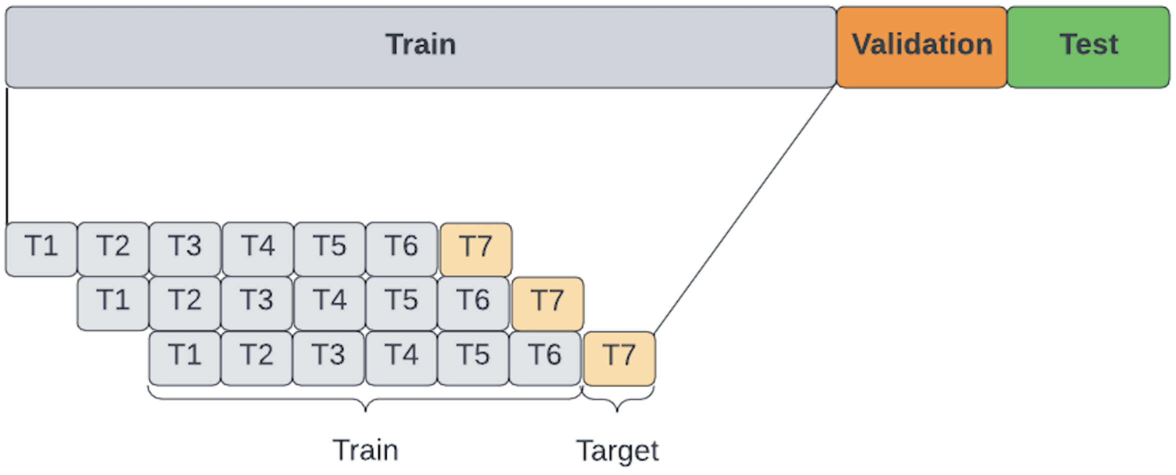
Data Preprocessing

Data preparation consist from the following steps:
- Split data into training, validation and test sets;
- Scale data;
- Split each set into sequences.

Data split

Data splitted into 3 datastest:
- Train
- Validation
- Test

Each dataset splited into sequences of length **L**. Each sequence splitted into training part with length **L-1** and target part the last element of the sequence. Each other sequence created by shifting initial sequence to one element.



Data scaling

In this project two types of scaling will be used:

- Normalization
- Standartization

Most features like stock price will be scaled with normalization and only features with outliers like trading volume will standartization scaler.

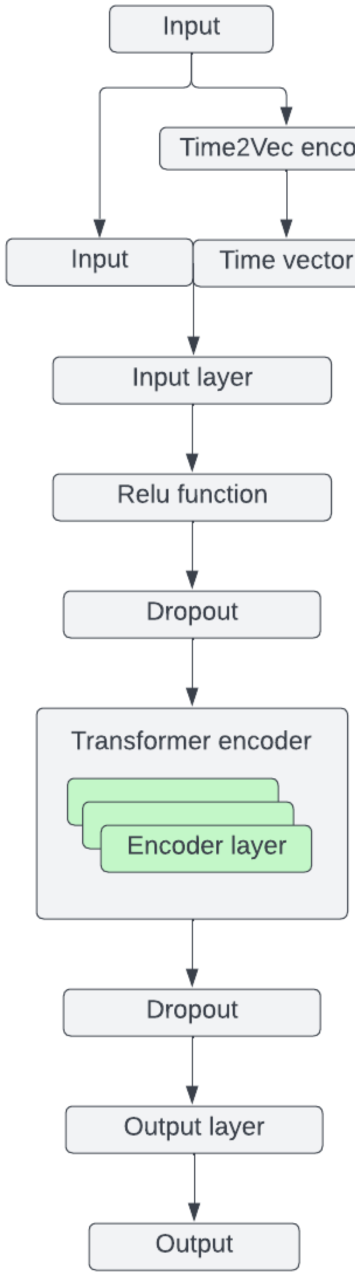
Implementation

The implementation consist of the following steps:
- Define network architecture (Jupiter Notebook);
- Validate network architecture, train first model (Jupiter Notebook);
- Train and tune model on stock data (Jupiter Notebook, AWS).

Define network architecture

To define network architecture PyTorch Framework and Python is used. To create network architecture default following PyTorch components is used:
- TransformerEncoder
- TransformerEncoderLayer
- Linear

Network architecture is following:



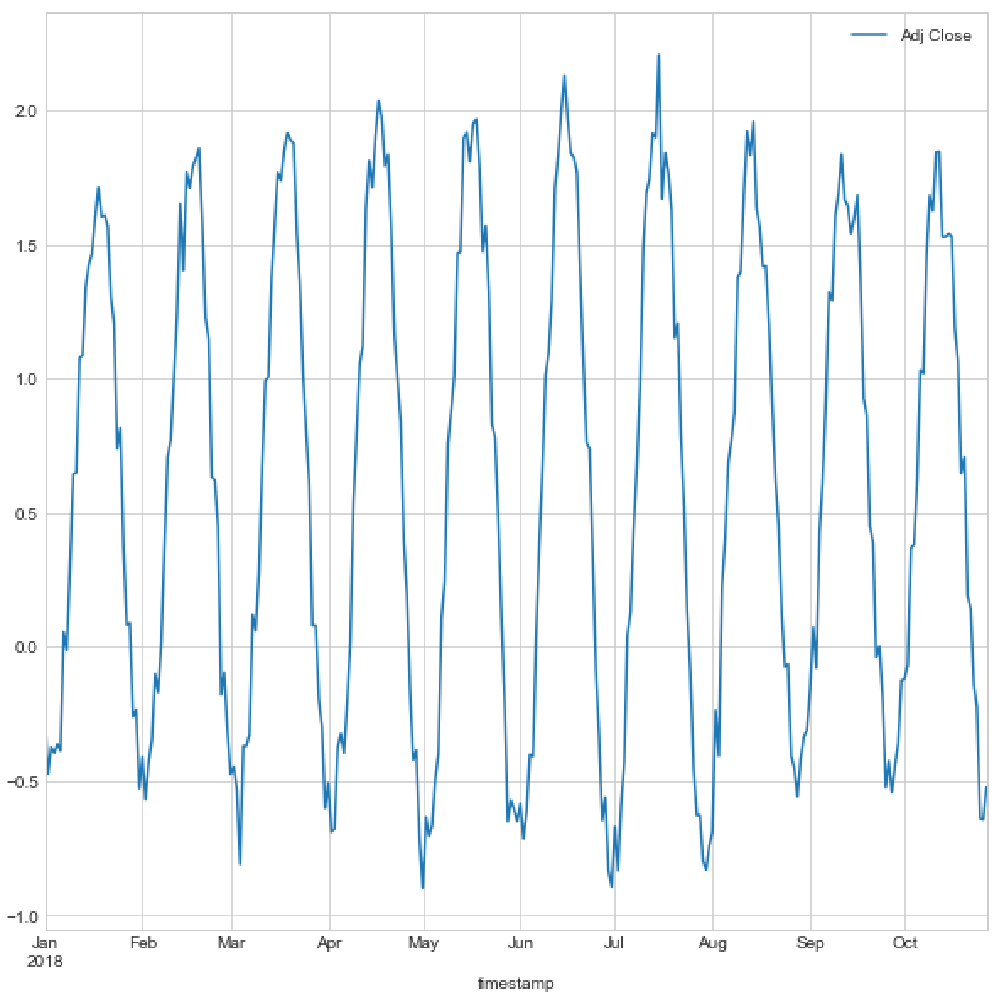
1. Input - input data, which is one of the data sequence;
2. Time2Vec encoder - used as positional encoder but with more generic approach;
3. Input + Time vector - input data which is including time component in it;
4. Input layer - required to transform input data to required number of inputs to transformer encoder layer;
5. Dropout layer - used twice in proposed architecture, to protect it from overfitting;
6. Transformer encoder - encapsulate several encoder layers which is implementing attention mechaniszm from the paper Attention Is All You Need;

7. Output layer - used as simplified version of transformer decoder layer, to provide predicted stock price;
8. Output - predicted stock price value for one day ahead.

Daily stack price data splitted into sequences for 7 days. Model trained on first 6 days and last day used as a target. This interval was selected as natural interval of a week, because it might include seasonality.
Since transformer trained parallely on all sequence it’s required to provide element positions in a sequence. In classical transformer which works with words and sentences positional encoder is used, but for our case with timeseries Time2Vec approach works better, since it’s more generic way to represent time. Time2Vec algorithm transform input data to a time representation and after that this time vector concatenated with original input and transferred to input layer.
Input layer implemented based on PyTorch Linear layer and required to transform input data into required encoder input dimention.
As we saw from the Attention Is All You Need paper classical Transformer consiste from encoder and decoder layers and both of the widely used in translation task. For the translation they both needed because first sentences from one language encoded in kind of universal language and after that decoded into translated language which is understandable by human. But in our case we don’t need decoder since we don’t need to decode output in some other way. Actually we can use decoder for current implementation but it will be overkill for this case. Output layer is a simplified version of decoder which is interpret encoder output into actual stack price value.

Validate network architecture

All the results of testing might be found in this in architecture-validation.ipynb. The main goal of the given step is to test provided architecture performance and find initial parameters to start with.
Since it’s quite difficult to evaluate model performance on real data decided to do this research on generated data sequence. Generated sequence represent periodical sequence with some fluctuations. See sample of generated data below:



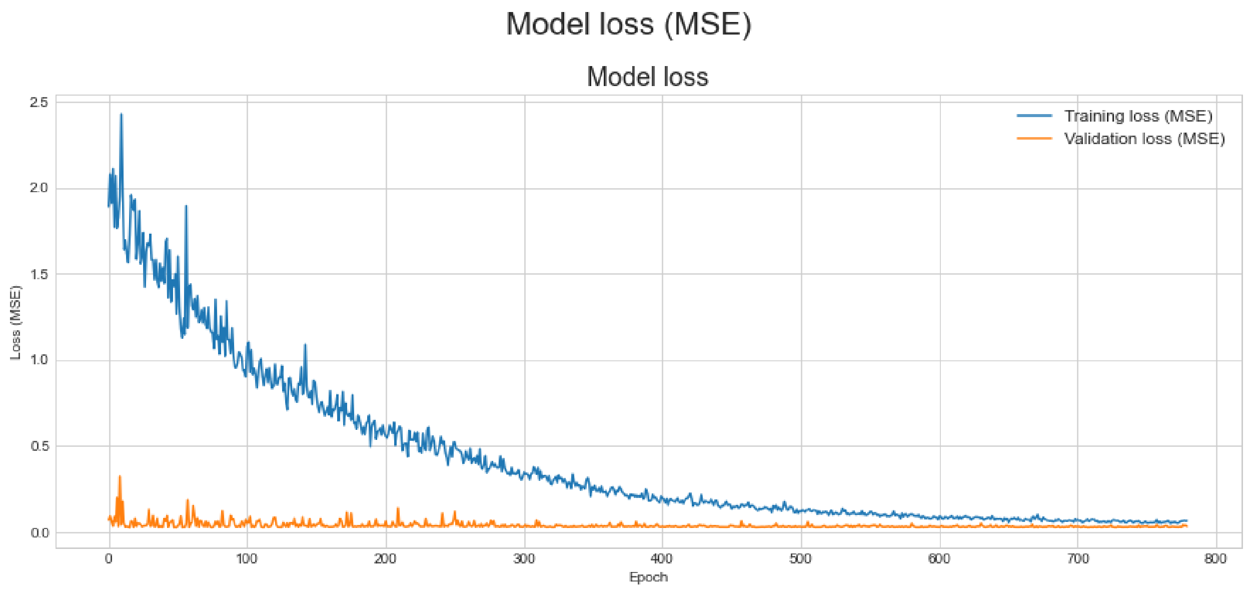
Proposed model training works quite good on synthetic data.

Training parameters:

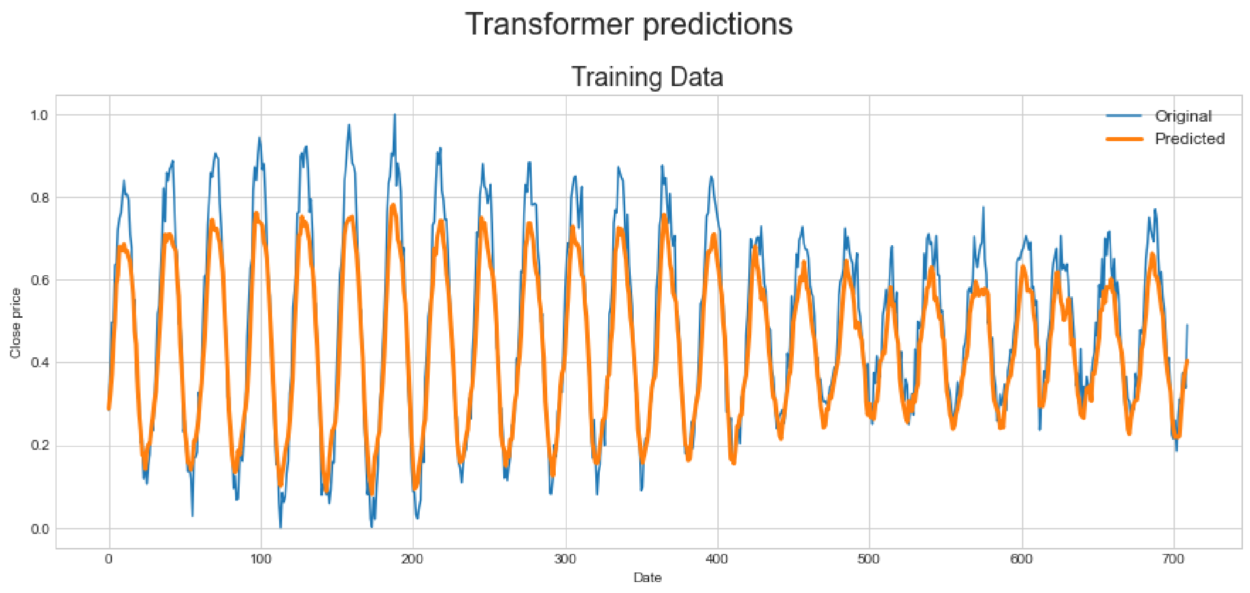
- Learning rate: 0.00001
- Number of epochs: 800

Result:

- Training loss: 0.064933
- Validation loss: 0.031749



Predicted result was quite accurate, see chart below.



Train and tune model on stock data

All the results of training and tunning model can be found in model-training.ipynb. The main goals of this steps are:

- Tunemodel parameters, to find the best;
- Train model on stock data.

Select optimal instance

To select optimal training instance several experiments was done on different instance types. Experiment results provided in the table below.

Type	Price, hour	Training time, sec	Total price	CPU load
ml.c5.xlarge	0.204	3348	0.19	100%
ml.c5.2xlarge	0.408	1900	0.22	100%
ml.c5.4xlarge	0.816	1609	0.36	97%
ml.g4dn.xlarge	0.736	851	0.17	50%

From the result above best instance to train model is ml.g4dn.xlarge with GPU. This instance price comparable with ml.c5.xlarge and ml.c5.2xlarge but training time less up to 4 and 2 times respectively. As an alternative ml.c5.2xlarge and ml.c5.xlarge CPU only instances can be used with more or less the same price but significantly longer training time.
To get even more price benefits estimators was switch to AWS Spot instances which is cheaper up tp 80% than on-demand instances. In experiments training savings was 70% for ml.g4dn.xlarge and 47% for ml.c5.2xlarge.

Tune model hyperparameters

By starting hyperparameters tunning job was discovered that two types of tasks was quite close by mean square error but with difference in huperparameter use mask. In order to understand how mask influence on model prediction decision was made to train both models.
You can see hyperparameters tunning results in table below.

	learning_rate	use_mask	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	TrainingElapsedTimeSeconds
3	0.003787	"False"	pytorch-training-221107-2256-009-5b091a3b	Completed	0.089791	2022-11-07 23:19:16+01:00	2022-11-07 23:27:19+01:00	483.0
0	0.000876	"True"	pytorch-training-221107-2256-012-9b75ae38	Completed	0.005049	2022-11-07 23:27:50+01:00	2022-11-07 23:35:55+01:00	485.0
5	0.000110	"False"	pytorch-training-221107-2256-007-638843a2	Completed	0.005005	2022-11-07 23:19:12+01:00	2022-11-07 23:26:46+01:00	454.0
6	0.000098	"False"	pytorch-training-22110	Completed	0.003014	2022-11-07 23:10:43+01:00	2022-11-07 23:18:46+01:00	483.0

			7-2256-006-faf767ad					
1	0.000056	"True"	pytorch-training-221107-2256-011-eeb5fdad	Completed	0.002783	2022-11-07 23:27:48+01:00	2022-11-07 23:35:47+01:00	479.0
2	0.000050	"True"	pytorch-training-221107-2256-010-0dee42de	Completed	0.002684	2022-11-07 23:27:46+01:00	2022-11-07 23:35:20+01:00	454.0
10	0.000176	"True"	pytorch-training-221107-2256-002-008648f0	Completed	0.002536	2022-11-07 22:57:24+01:00	2022-11-07 23:10:08+01:00	764.0
4	0.000110	"False"	pytorch-training-221107-2256-008-367504cd	Completed	0.002517	2022-11-07 23:19:13+01:00	2022-11-07 23:27:11+01:00	478.0
9	0.000074	"True"	pytorch-training-221107-2256-003-7a645f34	Completed	0.002185	2022-11-07 22:57:34+01:00	2022-11-07 23:09:49+01:00	735.0
7	0.000219	"False"	pytorch-training-221107-2256-005-e96fa1c7	Completed	0.001848	2022-11-07 23:10:44+01:00	2022-11-07 23:18:42+01:00	478.0
11	0.000116	"True"	pytorch-training-221107-2256-001-15e04220	Completed	0.001526	2022-11-07 22:57:30+01:00	2022-11-07 23:09:55+01:00	745.0
8	0.000109	"False"	pytorch-training-221107-2256-004-e40591b3	Completed	0.001440	2022-11-07 23:10:40+01:00	2022-11-07 23:17:54+01:00	434.0

Models with following hyperparameters will be trained:

Model with mask
RMSE metric - 0.001526
Use mask - True
Learnign rate - 0.000116

Model without mask

RMSE metric - 0.001440
Use mask - False
Learnign rate - 0.000109

Final model architecture

Model architecture to train build with PyTorch Framework

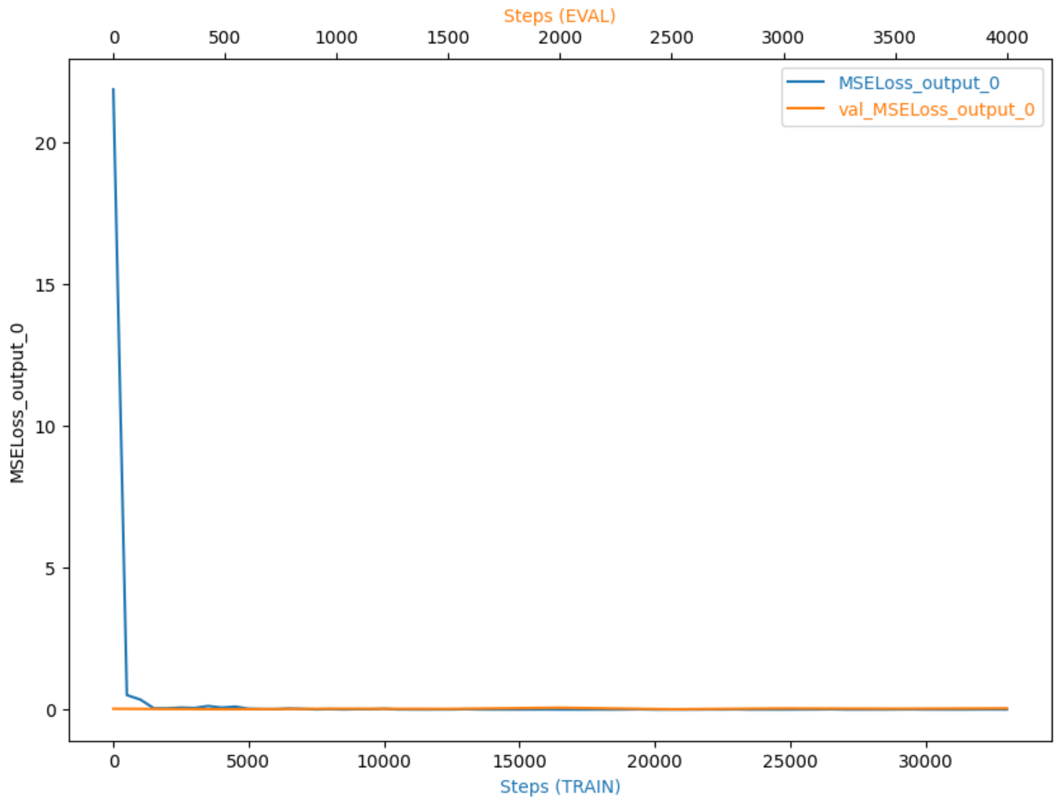
```
['MSELoss_output_0',
 'gradient/TransformerTime2Vec_decoder.bias',
 'gradient/TransformerTime2Vec_decoder.weight',
 'gradient/TransformerTime2Vec_embedding.bias',
 'gradient/TransformerTime2Vec_embedding.weight',
 'gradient/TransformerTime2Vec_time2vecEncoder.bias_linear',
 'gradient/TransformerTime2Vec_time2vecEncoder.bias_periodic',
 'gradient/TransformerTime2Vec_time2vecEncoder.weights_linear',
 'gradient/TransformerTime2Vec_time2vecEncoder.weights_periodic',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.0.linear1.bias',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.0.linear1.weight',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.0.linear2.bias',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.0.linear2.weight',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.0.norm1.bias',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.0.norm1.weight',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.0.norm2.bias',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.0.norm2.weight',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.0.self_attn.in_proj_bias',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.0.self_attn.in_proj_weight',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.0.self_attn.out_proj.bias',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.0.self_attn.out_proj.weight',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.1.linear1.bias',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.1.linear1.weight',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.1.linear2.bias',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.1.linear2.weight',
...
'gradient/TransformerTime2Vec_transformer_encoder.layers.5.norm2.weight',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.5.self_attn.in_proj_bias',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.5.self_attn.in_proj_weight',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.5.self_attn.out_proj.bias',
 'gradient/TransformerTime2Vec_transformer_encoder.layers.5.self_attn.out_proj.weight']
```

Train model with mask

Training seconds: 1495
Epochs: 1001

Train loss: 0.002568
Test loss: 0.020211

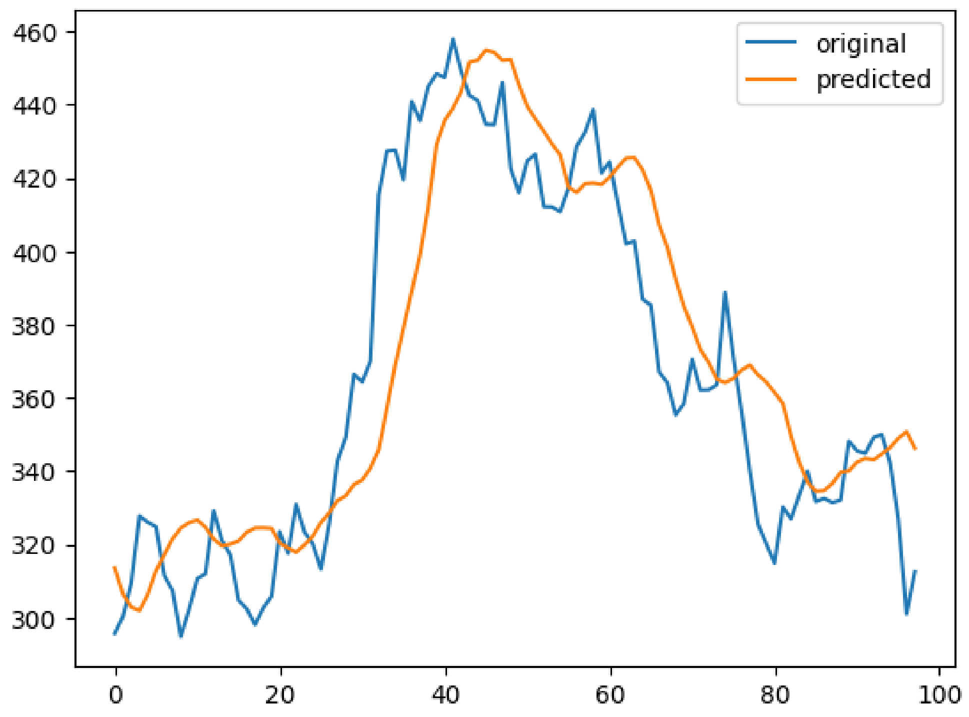
Loss function changes over a time:



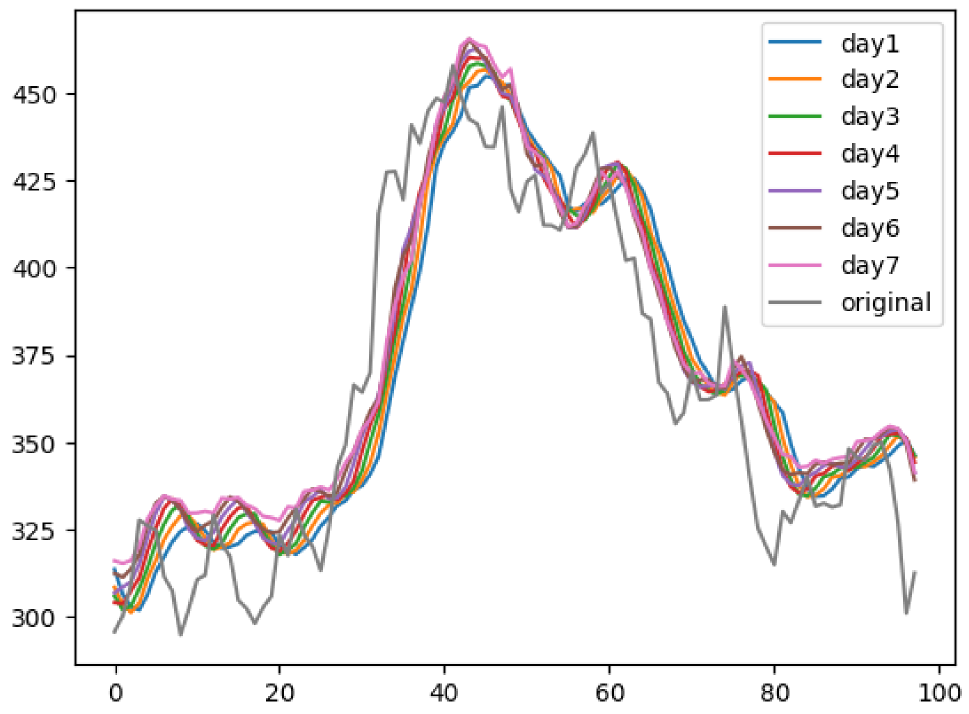
System usage statistic:

0	algo-1	Network	bytes	58685.38
1	algo-1	GPU	percentage	28
2	algo-1	CPU	percentage	96.03
3	algo-1	CPU	memory percentage	21.02
4	algo-1	GPU	memory percentage	14
5	algo-1	I/O	percentage	74.02

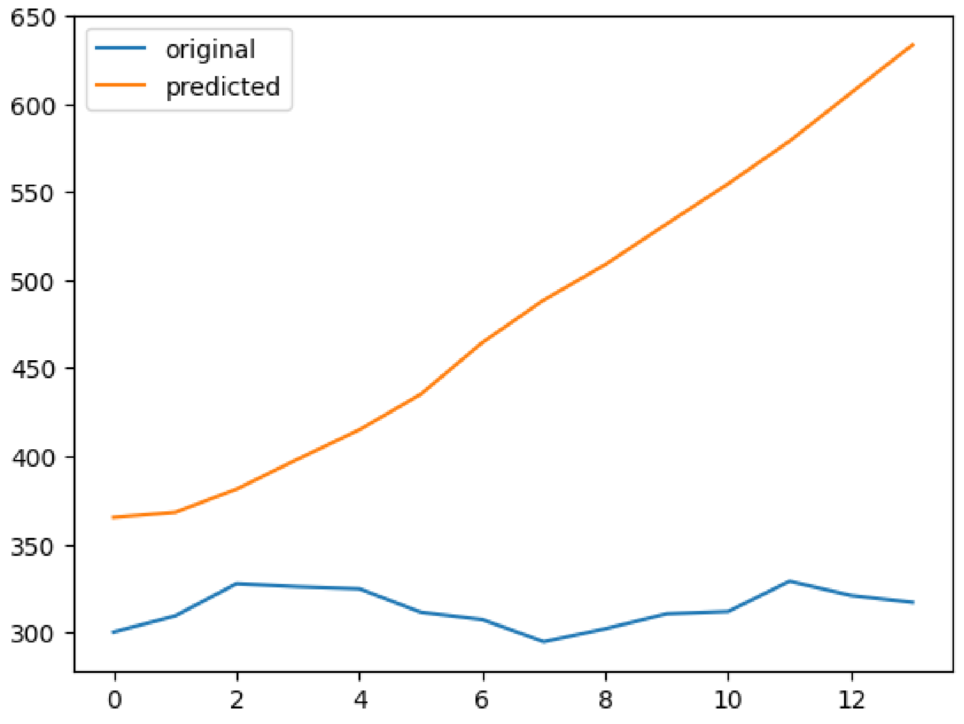
Model prediction for one day in a future:



Model prediction for several days in a future:



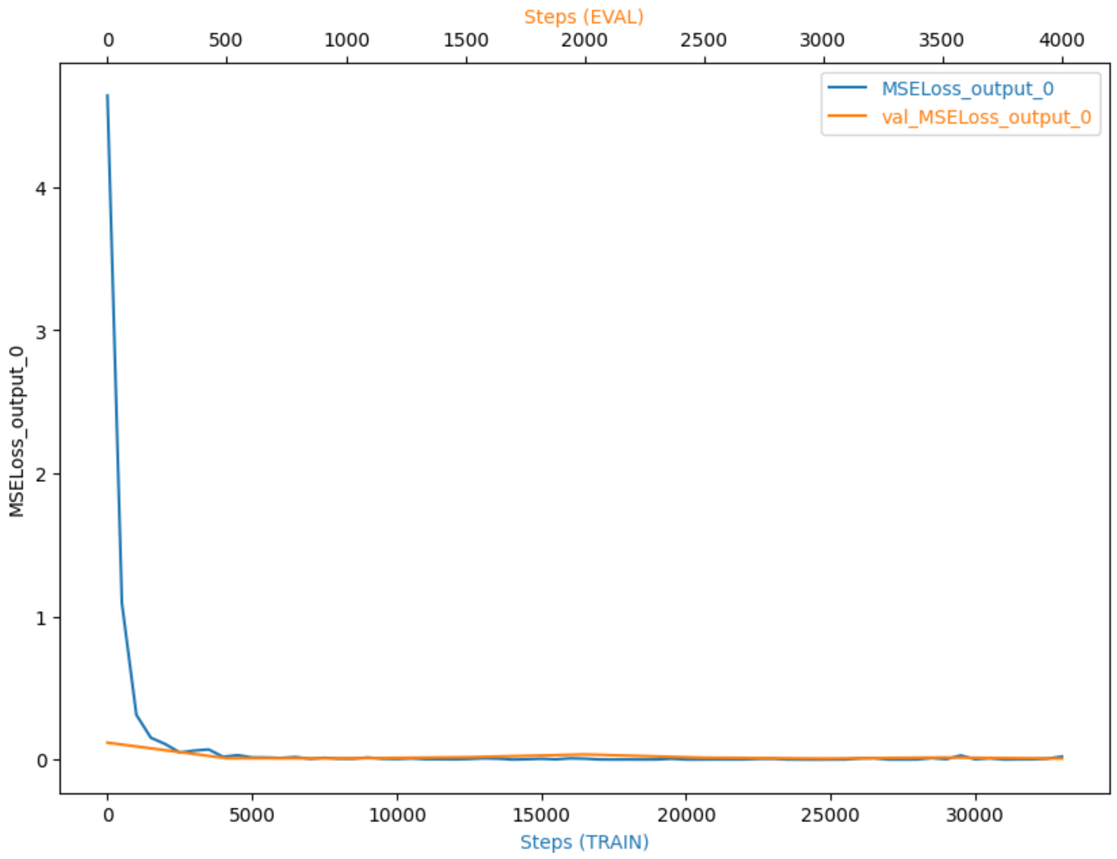
Predict 14 day in future:



Train model without mask

Training seconds: 1590
Epochs: 1001
Train loss: 0.009280
Test loss: 0.006959

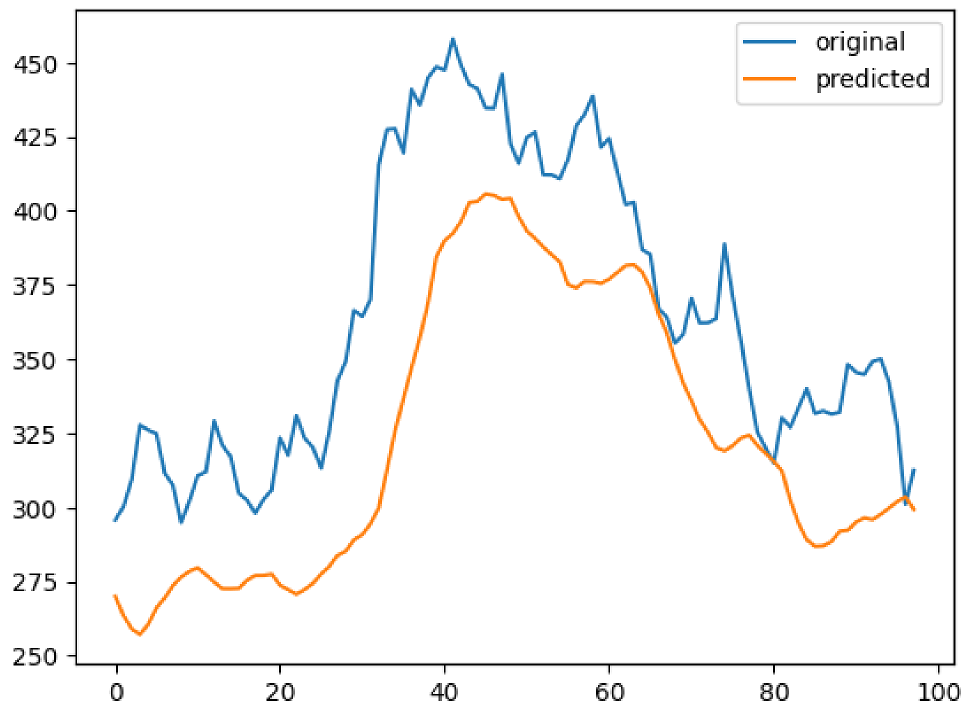
Loss function changes over a time:



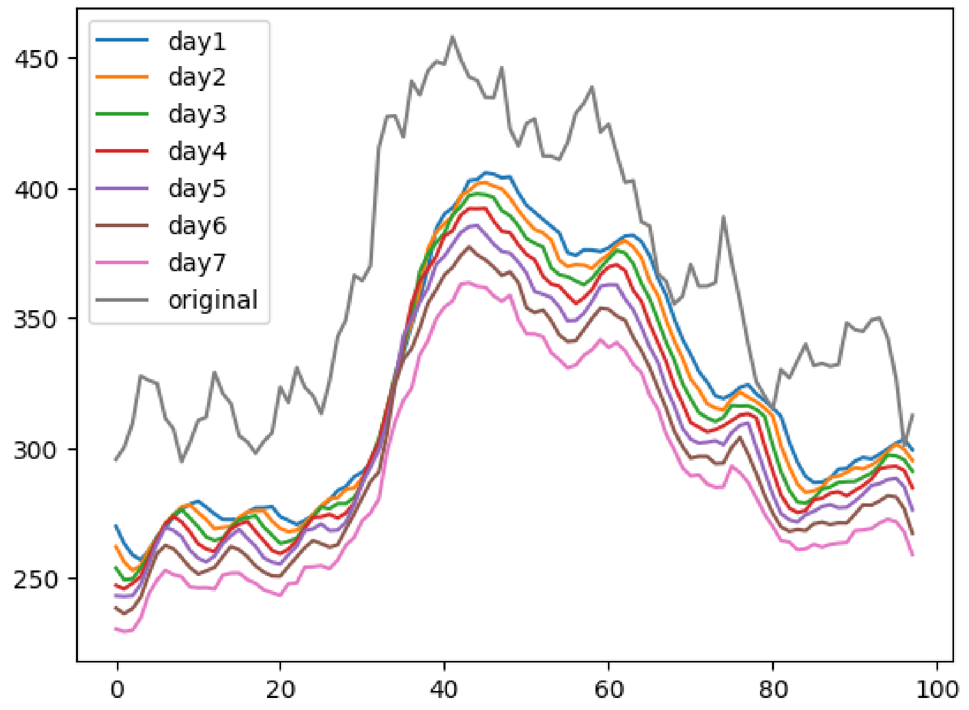
System usage statistic:

0	algo-1	Network	bytes	39364.13
1	algo-1	GPU	percentage	26
2	algo-1	CPU	percentage	99.5
3	algo-1	CPU memory	percentage	21.02
4	algo-1	GPU memory	percentage	13
5	algo-1	I/O	percentage	98.5

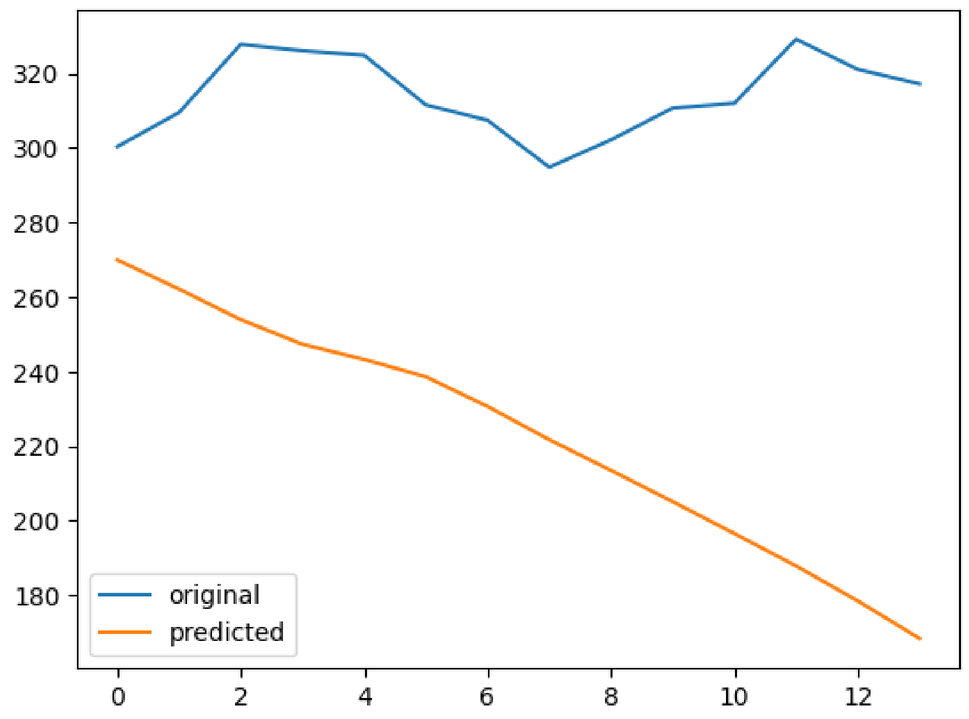
Model prediction for one day in a future:



Model prediction for several days in a future:



Predict 14 day in future:



Refinement

Final model implementation build on model validated in **architecture-validation.ipynb** notebook. As we see from the results both models perform quite well stock data. Model with mask has better performance and only this model achieve targeted mean square error 0.002.

Eventthough that model without mask win a bit model with the bask but final results shows that model with the bask is better at the end. Both model trained during 1001 epochs but has significantly different results.

Lest’s compare both models. Based on the metric collected.

Model predictions

From the models 1 day prediction chart we see that model with mask is more accurate in price prediction than model without mask. Eventhough that model without mask has more or less the same shape as original price the actual price value is shifted down a lot.

From the charts where we try to predict model value for several days in advance, we see that model with the mask has better performance since all days charts placed quite close to each other in comparison to the model without mask where days a lot more distributed from each other and then more days in a future we try to predict then bigger distance from the original chart.

But on the longer prediction period in our case 14 days both model has quite poor performance without correction of the prediction with recent data. And this is actually explain us why our prediction in reality not looks like prediction. If to look precisely on 1 day prediction charts for both models, we actually see that predicted result is always behind original and looks like moving average rather than real prediction.

Results

Model Evaluation and Validation

Final model parameters

RMSE metric - 0.002568
Use mask - True
Learnign rate - 0.000116

Prediction model architecture use Transformer architecture under the hood without encoding part of the Transformer.

Conclusion

Reflection

To use only price to predict next day price is not enough in most cases especially if daily prices are used. This is because it’s quite difficult to find price patterns across different days. Maybe this approach will work better for daily price prediction since more data are used during the day whic may create some data patterns from day to day. To improve prediction definitely some other data source have to be used like new articles, company reports etc.

Improvement

As one of the approaches to improve model prediction it’s better to convert our model from regression to classification model. For example instead of predicting real price value try to predict whether stock price goes up, down or stay neutral.
Try to use more data for prediction for example close, open prices. Try to use this model for daily price prediction use full stock candle information. Use some information resource to make a decision about stock price.

Deliverables

Write-up or Application

Model fully hosted in Sagemeker as endpoint which can be called to calculate stock prediction.

Github Repository

Source code and Jupiter notebooks can be found int this repository.