Time Series Forecasting with Deep Learning models

by Rafayel Setyan

May 19, 2022

Contents

1 Introduction **3**
- 1.1 Abstract . . . 3
- 1.2 Time series . . . 3

2 Data Pipeline **4**
- 2.1 Datasets . . . 4
- 2.2 Preprocessing . . . 4
  - 2.2.1 Windowing . . . 4
  - 2.2.2 Train-Validation-Test spliting . . . 5
  - 2.2.3 Feature Engineering . . . 5
  - 2.2.4 Augmentation . . . 8
  - 2.2.5 Metrics . . . 9

3 Proposed Models **11**
- 3.1 1D convolution . . . 11
  - 3.1.1 ResNet . . . 11
  - 3.1.2 Inception . . . 12
  - 3.1.3 Seq2Seq . . . 13
- 3.2 Recurrent Networks . . . 13
- 3.3 Attention Models . . . 14
- 3.4 State of the Art . . . 15
  - 3.4.1 NBEATS . . . 15
  - 3.4.2 Time2Vec . . . 16

4 Experimental Setup and Results **17**
- 4.1 Results . . . 17

5 Application **19**
- 5.1 Hyperparameters Tune . . . 19
- 5.2 Trading . . . 19

Chapter 1

Introduction

1.1 Abstract

In the last 2 decades Deep Learning (DL) models have achieved tremendous results in many fields. There applications for use cases are related to Image processing (classification, object detection, recognition, verification), Text processing (sentiment analysis, summarization), Audio processing (speech to text, user recognition, audio generation). In Time Series currently there are many State of The Art (SOTA) models which are used for forecasting. This research can be considered as a survey of Time Series forecasting with different DL models and with different datasets, then specifically the models will be tried to be used in financial forecasting applications such as daily trading.

1.2 Time series

We can consider Time Series data as a series of values that one or more variables take over successive time periods. As examples we can consider daily sales, energy consumption, seasonal downfall, daily exchange rate, hourly stock price etc. One of main difficulties is choosing the target variable in time series and model. Here as a target variable for Financial datasets *Close* price or *Return* are used and for other non Financial ones each of its target variables is used by its documentation. In this research I will investigate the flexibility and the accuracy of forecasting of DL models compared to traditional statistical Time Series models, on many time series datasets. As an accuracy metric, as long as our task domain is regression, MAE, MSE, MAPE, sMAPE and Huber losses are used.

The further structure of the research will be: Chapter 2 datasets, preprocessing, augmentation, feature engineering and which metrics are used, In the Chapter 3 the proposed models will be discussed, in Chapter 4 will be the results of experiments and in the Chapter 5 will be the Application results.

Chapter 2

Data Pipeline

2.1 Datasets

One of the major challenges for DL models is data. We can consider both the data quality and quantity as a challenge, because higher the complexity of the model the higher is demand for a huge dataset, otherwise the model will be overfitted in train data(we might see examples in the last section). The key idea is to give a model based on the task generic and diverse dataset. Another part is to have a human labeled high quality data, although data labeling is very costly, nowadays there are some approaches of avoiding manually labeling of data such as self-supervised learning from which as it is suggested model can learn common sense from given unlabeled data.[1] Fortunately for us there are many open source platforms from which we can get the Time Series datasets and also taking into account as long as we are doing regression we do not need any labeling. For this research below datasets are used.

Data Name	Quantity(1000's)	Description
ETTm1	69	Electricity Transformer
Electricity	140	Electricity prices
Apple	10	Apple stock price
SPY	7	SP500 stock index

2.2 Preprocessing

In Deep Learning one of the key components is the data preprocessing which includes data gathering, cleaning, preparation, feature engineering, scaling etc. So one of the main jobs of engineers is to create generic robust data pipeline which will do all processes and the input data will be ready either for training or evaluation. Below the pipeline creation is described.

2.2.1 Windowing

Time Series is a sequential which means each following value of the target variable has the relationship with the previous ones. One of the flexibilities of Neural Networks is that there are fully customizable related which input they will receive and output they should give. So for traditional statistic models

the target variable is Y 1d vector(not counting the exogenous variables yet), but for neural networks we can construct both input and output shapes however we want. For this research I used windowing mechanism with window size (W) 20 and horizon size (H) 5. E.g. neural network receives the *Close* prices of the previous 20 days and forecasts the upcoming 5 days' *Close* prices. Formally the windowing is done by this way, consider the target variable Y.

$$\begin{pmatrix} y_0, & y_1, & y_2, & \dots & y_{N-1}, & y_N \end{pmatrix}$$

With windowing W=20 and horizon H=5, we will have this X and Y matrices as an input feature matrix and the output matrix correspondingly.

$$\begin{pmatrix} x_{00}, & x_{01}, & x_{02}, & \dots & x_{018}, & x_{019} \\ x_{10}, & x_{11}, & x_{12}, & \dots & x_{118}, & x_{119} \\ x_{20}, & x_{21}, & x_{22}, & \dots & x_{218}, & x_{219} \\ & & \dots & & & \\ x_{N0}, & x_{N1}, & x_{N2}, & \dots & x_{N18}, & x_{N19} \end{pmatrix}$$

$$\begin{pmatrix} y_{00}, & \dots & y_{04} \\ y_{10}, & \dots & y_{14} \\ & \dots & \\ y_{N0}, & \dots & y_{N4} \end{pmatrix}$$

Below is the visualization of the windowing.

2.2.2 Train-Validation-Test spliting

The other key component is how to split the whole dataset. I took last 10% of raw data as a Test set and remained part I splited with 2 strategies:

- 1. **Standard** - for remained part I took the last 10% as Validation set and remaining as Train set.
- 2. **Periodic** - I splitted the remained set into equal 10 subsets and for each subset I took last 10% as Validation and remaining as Train set. The motivation for this kind splitting is that model can learn periodic patterns and we will validate it in that periods.

2.2.3 Feature Engineering

Financial sector has different external factors which can affect to the environment, it can be political or economical situation, global pandemics and wars, even the twit by one of the influencers. That

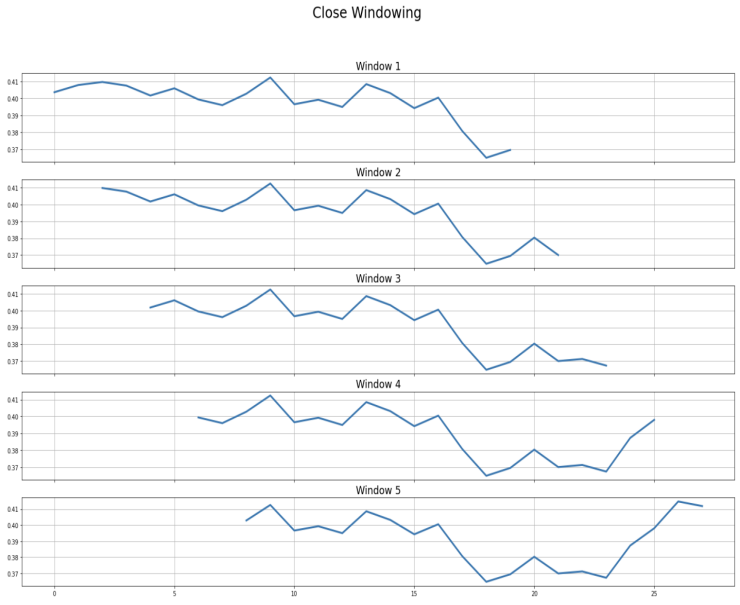

Figure 2.1: Windowed Dataset

is why analyst use a lot of external data related to the assets which are included for example in portfolio. For this research I will use only the famous financial indicators as the features combined with X matrix. Below I will list some of the used financial indicators:

- **Moving Average**

$$MA_k = \frac{1}{k} \sum_{i=n-k+1}^{n} y_i$$

- **Exponential Moving Average**

$$EMA_k = \alpha y_k + (1 - \alpha) EMA_{k-1}$$

- **Percentage Price Oscillator**

$$PPO = \frac{EMA_{12} - EMA_{26}}{EMA_{26}} * 100\%$$

- **Moving Average Convergence Divergence**

$$MACD = EMA_{12} - EMA_{26}$$

- **Triple Exponential Moving Average**

$$TEMA = (3 * EMA_1) - (3 * EMA_2) + EMA_3$$

- **Chaikin volatility**

$$ChVol_k = \frac{EMA(High - Low)_k - EMA(High - Low)_{k-10}}{EMA(High - Low)_{k-10}} * 100\%$$

- **Money Flow Index**

$$TypicalPrice_i = \frac{High_i + Close_i + Low_i}{3}$$

$$RawMoneyFlow_i = Volume_i * TypicalPrice_i$$

$$MoneyFlowRatio_i = \frac{14\ periods\ positive\ money\ flows\ addition}{14\ periods\ negative\ money\ flows\ addition}$$

$$MFI_i = 100 - \frac{100}{1 + MoneyFlowRatio_i}$$

- **Aroon High/Low indicator**

$$Aroon\ High/Low = \frac{25 - period\ since\ the\ Highest/Lowest\ price\ in\ 25\ rage}{25} * 100\%$$

Below is the figure of the input window and its corresponding some financial indicators.

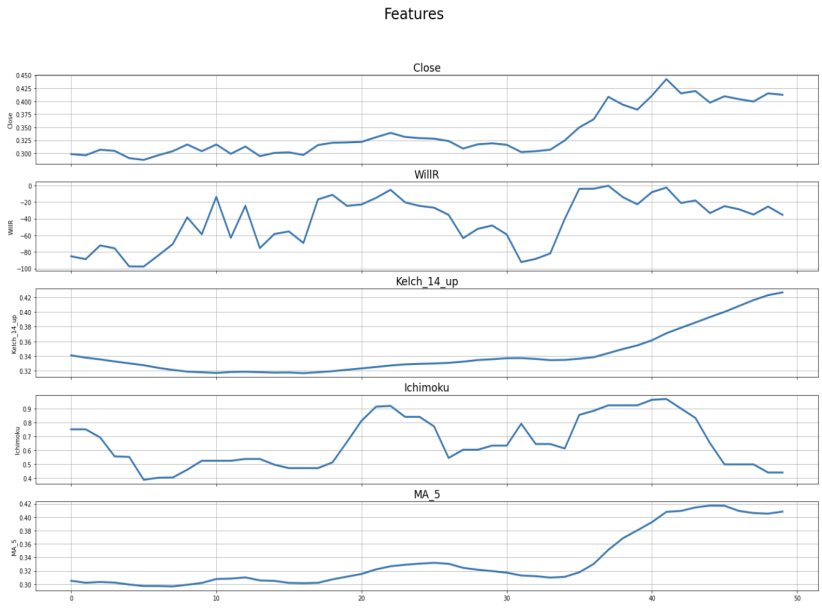

Figure 2.2: Features

2.2.4 Augmentation

Augmentation is the technic of artificially increasing the available data. For Vision there are many methods including rotation, cropping, zooming etc, for Signal Processing adding random noise, speed or volume perturbation etc. For this research I used the below techniques: [2]

- **Jittering** - this is yet the simplest one, it just adds element wise random Gaussian noise

$$\left(\begin{array}{cccc} \alpha_0 + x_0, & \alpha_1 + x_1, & \ldots & \alpha_N + x_N \end{array} \right)$$

$$where \;\; \alpha_i \sim \mathcal{N}(0, \sigma^2)$$

- **Flipping** - flipping the direction of the vector by multiplying all elements of vector with the randomly generated matrix

$$\left(\begin{array}{cccc} Rx_0, & Rx_1, & \ldots & Rx_N \end{array} \right)$$

$$where \;\; R \sim \mathcal{N}(\mu, \sigma^2) \;\; random \;\; rotation \;\; matrix.$$

- **Scaling** - scaling changes the magnitude or intensity of series which is done by elemnt wise multipying the vector with randomly generated scalar

$$\left(\begin{array}{cccc} \alpha x_0, & \alpha x_1, & \ldots & \alpha x_N \end{array} \right)$$

$$where \;\; \alpha \sim \mathcal{N}(1, \sigma^2)$$

- **Magnitude warp** - this method is time series specific method which warps the series with smoothed curve. where each α_i is got from the interpolating cubic spline from randomly generated knots.

$$\begin{pmatrix} \alpha_0 x_0, & \alpha_1 x_1, & \dots & \alpha_N x_N \end{pmatrix}$$

- **Time warp** - temporal warping is similar to changing the audio speed.
- **Permutation** - this is some advanced method, it takes any subset of the vector and performs horizontal or vertical mirroring.

Below is one example of the original window and its augmented versions.

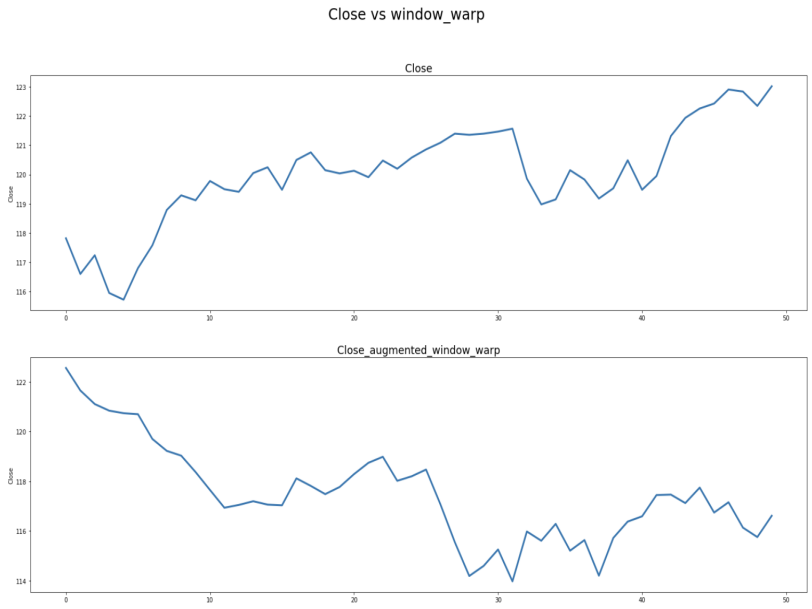

Figure 2.3: Augmented window

2.2.5 Metrics

For this research I used 5 regression metrics.

- **MAE** - Mean Absolute Error

$$MAE = \frac{1}{N}\sum_{n=1}^{N}|y_n - \hat{y_n}|$$

- **MSE** - Mean Squared Error

$$MSE = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y_n})^2$$

- **MAPE** - Mean Absolute Percentage Error

$$MAPE = \frac{1}{N}\sum_{n=1}^{N}|\frac{y_n - \hat{y_n}}{y_n}|$$

- **sMAPE** - symmetric Mean Absolute Percentage Error

$$sMAPE = \frac{2}{N}\sum_{n=1}^{N}\frac{|y_n - \hat{y_n}|}{(y_n + \hat{y_n})}$$

- **Huber** - Huber loss

$$Huber_\delta = \begin{cases} \frac{1}{2}(y-\hat{y})^2 & if\,|(y-\hat{y})| < \delta \\ \delta((y-\hat{y}) - \frac{1}{2}\delta) & otherwise \end{cases}$$

Chapter 3

Proposed Models

3.1 1D convolution

Time series is the domain which can benefit from 1D convolution, because 1D convolution is good at fitting in spatial data, in our case it is time. A convolution model can learn patterns over time. Below I represent the custom 1D convolution block types which are motivated by the State of the Art models of Computer Vision.

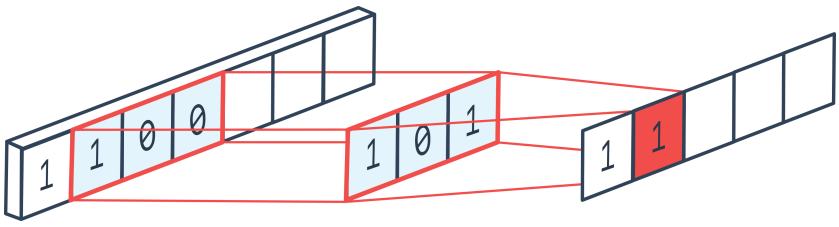

Figure 3.1: 1D convolution operation

3.1.1 ResNet

The ResNet[3] like block is 2 1D convolution layers with 3 kernel size, with skip connection at the end of block. Residual/Skip connection is element wise addition of block input and output. The main idea is for avoiding vanishing gradients while constructing very deep models(with more than 30 layers). But also empirically this trick increases the accuracy and now it is used in almost every model.

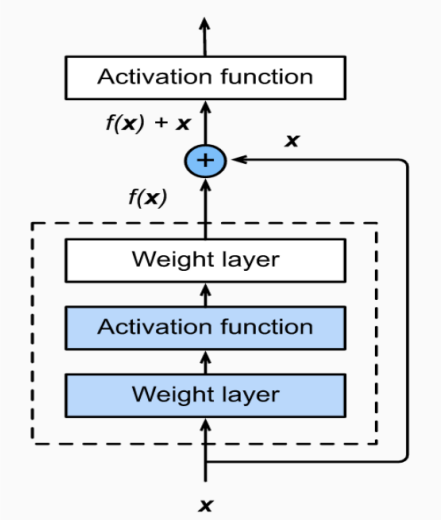

Figure 3.2: ResNet block

3.1.2 Inception

Inception[4] is a block consisting of 3 1D convolution layers each of them has correspondingly 3, 5, 7 kernel size, where at the end all spatial dimensions are concatenated together. As long as our data has size of 20, I have customized the naive original block removing max pooling layer and instead of 1x1 convolution used 7x7. For reducing the data size in the intermediate layers I used stride=2 in some of the blocks.

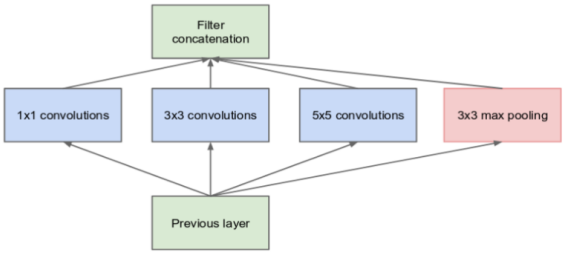

Figure 3.3: Inception naive block, when all spatial dimensions are concatenated together

3.1.3 Seq2Seq

Seq2Seq is sequence to sequence simple encoder architecture with 1D convolution with different kernel sizes and with different dilation rates, which are at the end added together. The main idea came from the WaveNet model proposed by DeepMind[5]. The core of the wave net model can be described as a stack of residual blocks that utilize dilated causal convolutions, visualized by the two diagrams. For this research the WaveNet is trained with customized configs, because the input data is less, on the original paper the input had the length of 16000. As the Paper suggests: "Stacked dilated convolutions enable networks to have very large receptive fields with just a few layers, while preserving the input resolution throughout the network as well as computational efficiency".

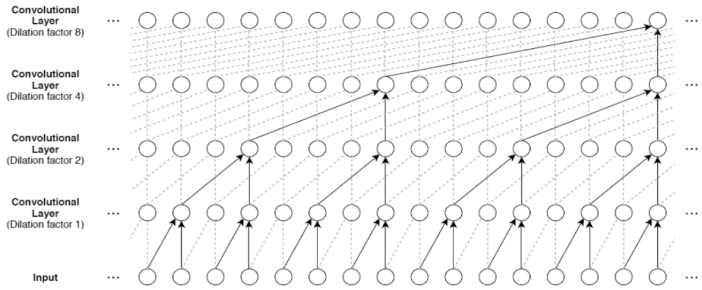

Figure 3 : Stack of dilated causal convolutional layers

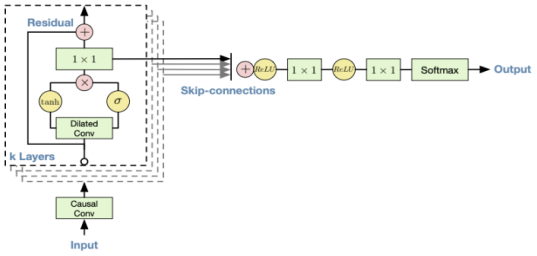

Figure 4 : Final schema of the model with all Stacks

Figure 3.4: Seq2Seq block and model

3.2 Recurrent Networks

For a long time the Recurrent Networks were considered to be the best solution for sequential data. RNNs consist of feed forward layers with the internal memory. RNN has memory capabilities. It memorizes previous data. While making a decision, it takes into consideration the current input and also what it has learned from the inputs it received previously. Output from the previous step is fed as input to the current step creating a feedback loop. For this research I have used one directional and bidirectional LSTM and GRU layers with and without stacking, also with and without 1D convolution ResNet like backbone before the Recurrent layers. Below are diagrams of LSTM[6] and GRU[7].

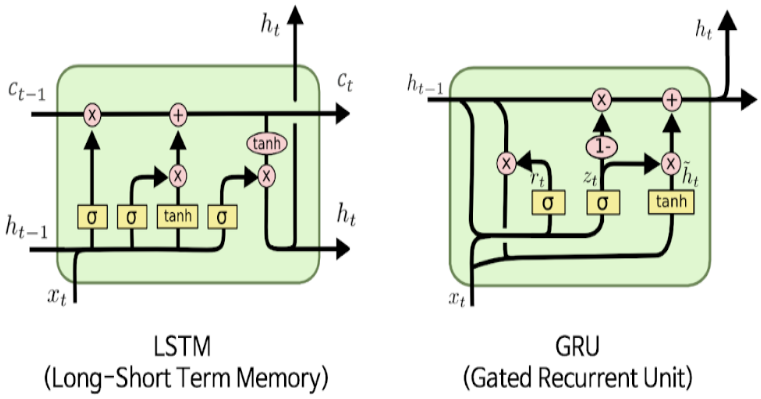

Figure 5 : LSTM and GRU layers

Figure 3.5: LSTM & GRU layers

3.3 Attention Models

After the publication of the paper "Attention is all you need"[8], the usage of this kind of models have spread in all of the domains of Deep Learning. Attention mechanism was firstly introduced for translating the text data. The core idea is to mimic cognitive attention. Formally it is using 3 weights matrices Q, K, V, which weights are learnable and attention part works as a dot product of the given each input vector with these matrices. One of the advantages of this mechanism is that the calculation is done parallelly, which is the major drawback of recurrent layers, where each sequential cell has to wait until calculations will be done for previous cells so that it can get input. In this research the Custom MultiHead Transformers are used with and without 1D convolution backbone. Below are visual diagrams of the Self-Attention mechanism.

Here in the diagram the W^Q, W^K, W^V are learnable parameters. Then the output of the Attention head will be calculated by below formula.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_K}})V \quad where\ d_K\ is\ K\ dimension$$

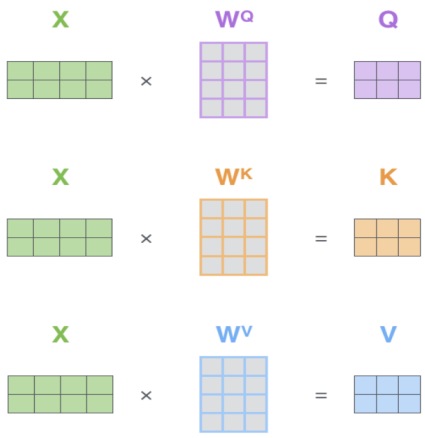

Figure 3.6: Attention mechanism, Q, K, V matrices

3.4 State of the Art

3.4.1 NBEATS

N-BEATS[9] focuses on univariate time series; it is a stacked architecture of the fully connected layers based on backcast and forecast residual blocks. It achieves state of the art accuracy on the M3, M4 and TOURISM datasets. The key methodology of constructing this model is the basic but expressive (deep) architecture which does not require any time series specific preprocessing or scaling and also model interpretability. Below is the architecture of the model.
For proper understanding the architecture of the model, below each block and stacks are described step by step.

Basic block Block input in this case is a vector with length 20, the input is given to 4 stacked fully connected layers, then at the last stacked layer the output length is the original input length and the required horizon length, in this research case it is 25 (W=20, H=5). This former is backcast and the latter is forecast, i.e. block predicts not only what comes in the horizon but also predicts the future input.

Stacks Stack consists of K basic blocks which are described above. Also it is important to note that after the first basic block for the next block the input will not be the original pure input, but will be the original input subtracted with the block backcast, so after that, for each next block input will be the output of the previous backcast and the backcast of the previous of the previous block subtracted together. The forecasts of each block are added together and the result is a Stack forecast. For interpretability, authors suggested 3 types of stacks - Basic, Trend, Seasonal. The final model prediction will be the addition of each stack's forecasts. For this research for having the fixed pattern, I used the given sequence of Stack types as a baseline : (generic, seasonal, trend, generic) and for increasing the complexity and depth of the architecture I kept this pattern by adding the same sequence multiple times.

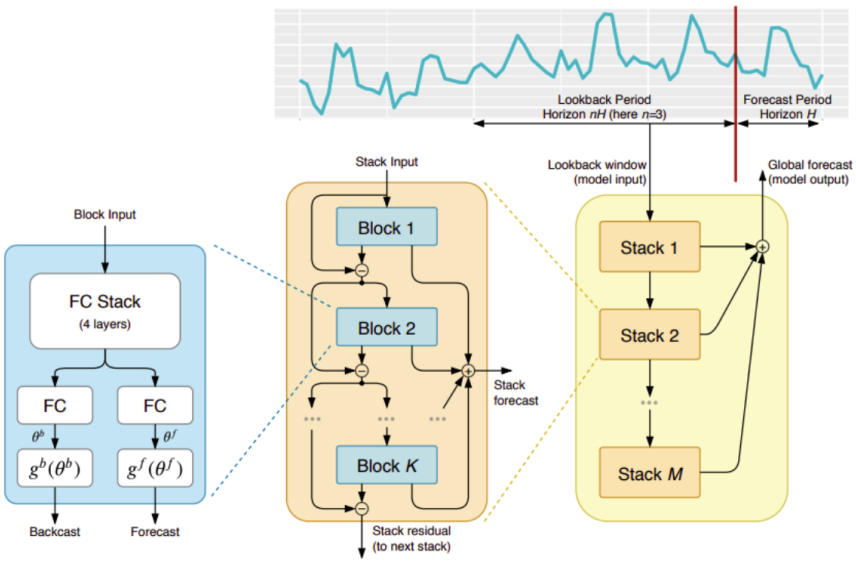

Figure 7 : N-BEATS architecture

Figure 3.7: NBEATS architecture

3.4.2 Time2Vec

Time2Vec[10] is general purpose architecture for high level time dependent series representation. This is not an End-to-End model, but a layer type from which processed data can be input to Recurrent or Self Attention layers. The core idea is avoiding manually preprocessing or feature engineering and letting the model learn the best representation for the input. The layer consists of a fully connected layer and the output is activated with sine or cosine function. In this research I used both cosine and sine activations and concatenated them.

Chapter 4

Experimental Setup and Results

4.1 Results

All models described in the above section are implemented by myself on the famous open source framework *Tensorflow2 (Keras)* and all codes for data preprocessing, models building, training and evaluation will be publicly available on Github very soon. All trainings are done on my local machine (MacBook Pro i5; 8GB RAM; CPU). The best weights for testing are saved based on the validation loss, which is calculated after each epoch. For this research I used below training configurations.

Name	Value
Batch size	32
Loss	MAE
Optimizer	Adam
Learning Rate	1e-3
Epochs	20

Also I took the big amount of test data about more than 500 samples. During the experiments I used 2 methods of the train-validation-test splitting strategies, described in the Chapter 1, but with standard splitting it achieved a better results. Below are the result of the models. (Note that the numbers are Mean Absolute Error, "-" means that the loss was so high that I did not even showed them and -*A* suffix means that the dataset is augmented)

It can be seen from the results that Neural Networks outperform traditional statistical models. Also it is notable that the augmentation helped with the overfitting of complex models such as Seq2Seq or Transformers. Below is also shown in figure the real price of S&P and some models predictions. For a next chapter when I will try trading with the models' prediction I will take Inception and NBEATS.

Model	AAPL	AAPL-A	SPY	SPY-A	ETTh1	Electricity
ARIMA	67.1	70.38	141	148.6	4.12	24.1
GARCH	38.92	66	-	-	2.1	32.1
ResNet	3.74	3.71	72.1	5.97	0.782	13.1
Inception	2.4	2.4	6.38	5.91	0.745	12.3
Seq2Seq	64	3.1	-	6.86	0.79	16.34
LSTM	3.11	4.94	33.2	9.83	0.762	13.2
GRU	5.75	2.3	40.7	8.8	0.779	12.9
Transformer	66	2.9	77	8.76	0.776	12.7
Time2Vec	49	68	169	6.16	0.85	14.5
NBEATS	2.41	2.3	6.2	5.08	0.761	11.9

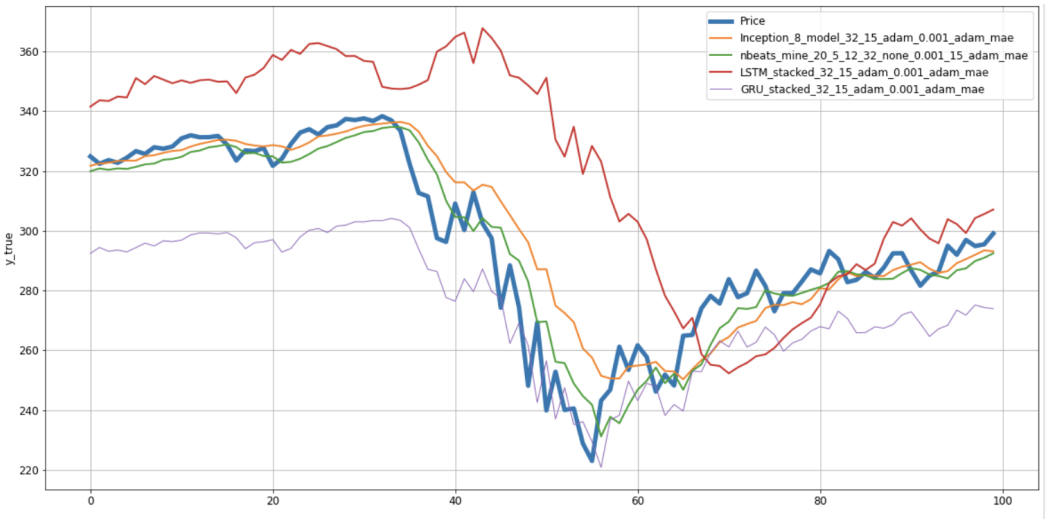

Figure 4.1: Actual vs Predictions

Chapter 5

Application

In this chapter Inception and NBEATS models are taken and I will try to implement a strategy in a short term in the Apple stock prices with the predictions of this 2 models. Before trading strategy I changed the train-validation-split proportion, taking last month as a trading/test period, from February up to March as Validation and remaining set as a train. In this case I did also an hyper parameter tuning and changed the complexity of models.

5.1 Hyperparameters Tune

For getting the best Models I did the hyper parameters tuning. I kept the window size(W) 20 and changed predicting horizon(H) to 1 for 1 day trading. During the experiments it turned out that without financial indicators models achieve the higher results. Below is the list of the hyper parameters.

- **Batch size** - 16, 32, 64
- **Loss** - MAE, MSE, MAPE, sMAPE, Huber
- **Learning Rate** - 0.001-fixed, 0.01-decay

In the table below are the best result of each model with corresponding hyper parameters which are taken for trading. (Inception-F means Inception model trained with prices and financial features)

Model	Batch size	Loss	Learning Rate	Test MAE	Test MSE	Test MAPE	Test sMAPE	Speed(ms)
Inception	64	Huber	0.001	2.38	7.91	1.43	1.44	19
Inception-F	64	Huber	0.001	2.92	9.1	1.54	1.5	21
NBEATS	64	MAPE	0.001	2.84	10.89	1.7	1.7	18

5.2 Trading

I took the 3 models shown above and tried to do basic daily trading in April daily data. From python package *yfinance* I took the 15 minutes dataset and also used daily for making predictions. The strategy is simple, if the model predicts that the price will be increased tomorrow I will buy with given *Limit Order* the stock on the next day, otherwise I will sell the stock with the same

principle, also if the prediction and actual price percentage change in absolute is bigger than 1% I will buy or sell 5 stocks. If in the next day the actual price does not cross Limit Order price I will not make any deals. Also we assume that short selling is allowed. I took the initial capital as 1000$. With NBEATS and Inception(with financial features) models, strategy made no transactions so no profit, below is the results of trading with the Inception.

Inception

- Initial Capital : 1000$
- Transactions
  - Buy 1 in 2022-04-01 for 174.81
  - Sell 5 in 2022-04-07 for 171.46
- Remained Capital in 2022-04-29 : 1682.52$
- Number of Stocks : -4
- Price of Stock : 157.65$
- Capital in the end : 1051.92$
- Profit% : 5.19%
- Buy and Hold for the same period
- S&P : -9%
- AAPL : -9.5%

Conclusion

Here in this research as a first step I showed that despite being not interpretable, demanding huge amount of data and computational resources the Deep Learning models can achieve a higher performance compared to other traditional models. In the last section I did small trading strategy and got 5.2% profit, but I tried same strategy in the Validation set which is from February up to the end of March end got -1.6%, but it also important that S&P index also had negative return -8.6%. The main motivation of this research was to show that Deep Learning models can be useful in financial sphere also.

References

- 1. Self-supervised learning: The dark matter of intelligence
- 2. An empirical survey of data augmentation for time series classification with neural networks : Brian Kenji Iwana : https://arxiv.org/pdf/2007.15951.pdf
- 3. Deep Residual Learning for Image Recognition : Kaiming He : https://arxiv.org/pdf/1512.03385.pdf
- 4. Going deeper with convolutions : Christian Szegedy : https://arxiv.org/pdf/1409.4842.pdf
- 5. WAVENET: A GENERATIVE MODEL FOR RAW AUDIO : Aaron van den Oord : https://arxiv.org/pdf/1609.03499v2.pdf
- 6. LONG SHORT-TERM MEMORY : Sepp Hochreiter : http://www.bioinf.jku.at/publications/older/2604.pdf
- 7. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling : Junyoung Chung : https://arxiv.org/pdf/1412.3555.pdf
- 8. Attention Is All You Need : Ashish Vaswani : https://arxiv.org/pdf/1706.03762.pdf
- 9. N-BEATS: NEURAL BASIS EXPANSION ANALYSIS FOR INTERPRETABLE TIME SERIES FORECASTING : Boris N. Oreshkin : https://arxiv.org/pdf/1905.10437.pdf
- 10. Time2Vec: Learning a Vector Representation of Time : Seyed Mehran Kazemi : https://arxiv.org/pdf/1907.05321.pdf