NASDAQ Forecasting Using BiLSTM and Transformer-Like Architectures

https://github.com/PeppeJerry/FinanceDoomsday

Giuseppe Murgolo
g.murgolo2@studenti.poliba.it
Polytechnic of Bari
Bari, Edoardo Orabona, 4

Abstract

The aim of this project is to predict historical price data by analyzing datasets of four companies (Amazon, Microsoft, Apple, Google) using different time-series forecasting models, namely a BiLSTM-based, a Transformer-based and a hybrid among the two.
One of the most common forecasting techniques is based on extending trends that has been shown in the past and applying those rules to the present and future. However, some important non-linear relationships may be lost by applying a simple "classic" forecasting method.
In order to grasp those rules, a Convolutional Neural Network Bi-directional long short-term memory (**CNN BiLSTM**), an **encoder-decoder Transformer** and a **Parallel CNN Encoder-BiLSTM** will be implemented, studied, and evaluated with proper metrics (Mean Absolute Error **MAE**, Symmetric Mean Absolute Percentage Error **SMAPE**, Mean Square Error **MSE**, and Root Mean Square Error **RMSE**).

Introduction

Thanks to CNN, BiLSTM and Transformers it is possible to create flexible models able to accurately predict values; one of the most notorious examples is the design proposed in the paper "**Attention Is All You Need**" which shows how it is possible to dynamically highlight relevant features in the Natural Language Processing (**NPL**) field thanks to their architecture.
This paper will focus mainly on **regression** models based on NASDAQ data[1] of the four listed companies; the data will be given as daily resume containing information such as **opening** price, **closing** price, **highest** recoded price, **lowest** recorder price, **adjusted closed** price, and **volume**. The goal is to forecast part of these data based on previous observations.
Market data may present ambiguous and complex information, along with non-linear relationships, which can make it challenging to accurately predict our data. Fortunately, powerful machine learning techniques can be employed to enhance the training and accuracy of our models. Among these techniques, we can include self-adapting capabilities to our models, which greatly improve performance compared to traditional machine learning, such as adaptive data rates and auto-regressive techniques.

[1] Easily accessible at https://finance.yahoo.com/

Summary

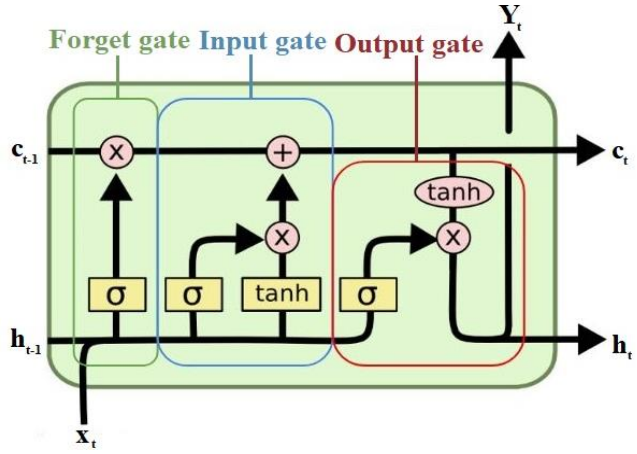
Model major blocks

Convolutional Neural Network (CNN)

A CNN consists of five important parts: **input layer**, **convolutional layer**, **pooling layer**, **fully connected layer**, and an **output layer**. The most important layers are the **convolutional** and **pooling** ones because they are able to **feature extract** the data by focusing on important aspects of the output and maintaining the important features.
Pooling is responsible of reducing the overall dimensionality which can be helpful or not depending on the specific task. It is it possible to exclude pooling techniques by using proper configurations on convolutional layers improving data representation at the cost of computational expenses.
During the development of the models it has been decided **not to use pooling** and it will be used a **zero-padding** technique that allows the model to grasp information at the edges of the sequence. Furthermore, convolutional layers will keep the original dimensionality of the sequence while increasing the channels in order to better represent the input data inside the model.

Long Short-Term Memory (LSTM)

A LSTM is a particular architecture which varies from the traditional Recurrent Neural Network (RNN) cell due to its ability to maintain long time information, which was one of the flaws of its ancestor, in order to predict important events in the future.



The cell at timestep t is presented by past information c_{t-1} and h_{t-1}, two important **streams** that keep relevant **information** across long period of times, and **gates** who regulates how the new input x_t will be handled in order to generate **new information streams** c_t and h_t.

How does a LSTM work

At the generic timestep t the cell is provided by h_{t-1} and c_{t-1} of the previous timestep $(t-1)$.
Forget gate: the forget gate f_t is a **sigmoid** that acts like a "switch" which receives the current input x_t along with the hidden state h_{t-1} in order to decide if the cell state c_{t-1} should propagate further or not in the architecture.
Input gate: A **candidate cell state** $\boldsymbol{c'_t}$ will be produced using both input x_t and hidden state h_{t-1} thanks to a $\tanh(*)$ function, and it will be combined with the cell state c^*_{t-1} depending on the state of the input gate i_t, another **sigmoid** that will act again like a switch.
Output gate: the output gate o_t is responsible of producing both the new hidden state h_t and output Y_t. This gate is a **sigmoid** which decides if the information $\tanh(c_t)$ needs to be used in order to produce a new hidden state h_t.

Equations:

Forget gate

$$f_t = \sigma\big(W_f[h_{t-1}, x_t] + b_f\big)$$

Input gate

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

Output gate

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

Previous cell state

$$c^*_{t-1} = f_t * c_{t-1}$$

New candidate cell state

$$c'_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$$

New cell state

$$c_t = f_c * c_{t-1} + i_t * c'_t$$

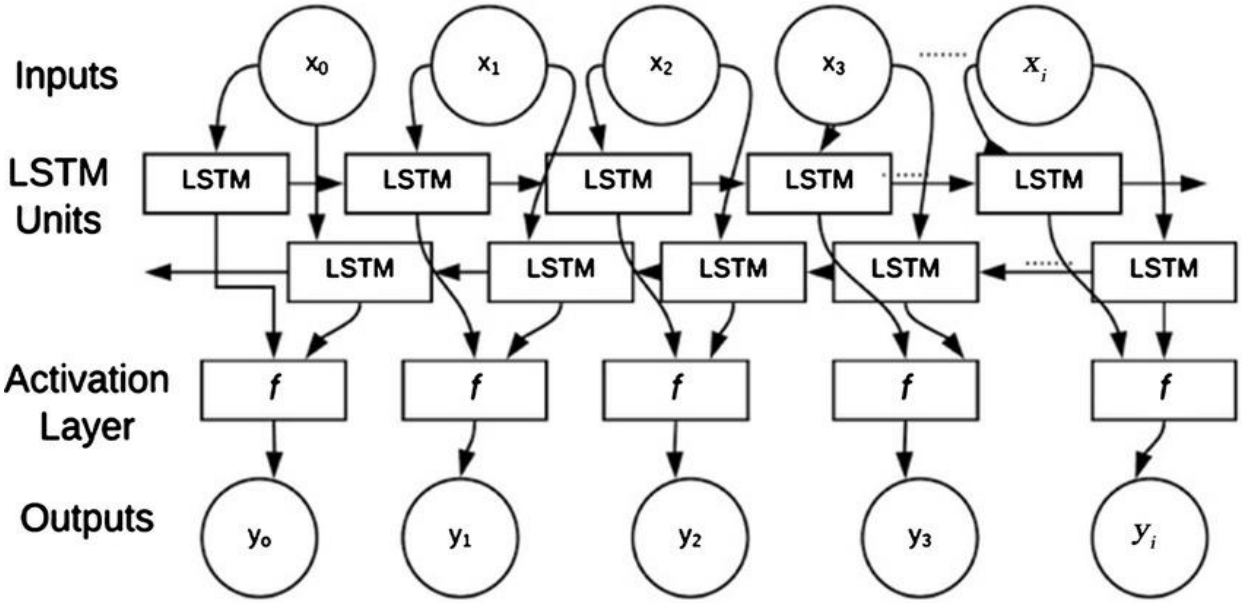
Final output

$$h_t = o_t \tanh(c_t)$$

Bidirectional LSTM (BiLSTM)

A BiLSTM is a recurrent neural network which introduces two hidden streams forward and backward. While generating the output it has access to both previous information and future information in a given timestep t improving sequential dependencies.

Note: this kind of architecture is used a lot in **NLP** due to the ability of relating past and future token. That being said, **forward** and **backward** streams can be seen simply as two separate LSTM cells that receive both the same sequence but in opposite directions so at a given timestep t while processing the input x_t, its corresponding output Y_t will be able to use both left and right dependencies (Bidirectional) improving the accuracy in the prediction.



Equations: it is possible to summarize the outputs given by forward LSTM and backward LSTM

Forward LSTM

$$\vec{h}_t = LSTM\big(x_t, \vec{h}_{t-1}\big)$$

Backward LSTM

$$\overleftarrow{h}_t = LSTM\big(x_t, \overleftarrow{h}_{t+1}\big)$$

The output states of a **BiLSTM** can be obtained by **concatenating** both forward and backward states:

$$h_t = \big[\vec{h}_t, \overleftarrow{h}_t\big]$$

Important note: due to this, the **output channel** of the BiLSTM cell will be **doubled** compared to a standard LSTM so we need to consider it while implementing our model.

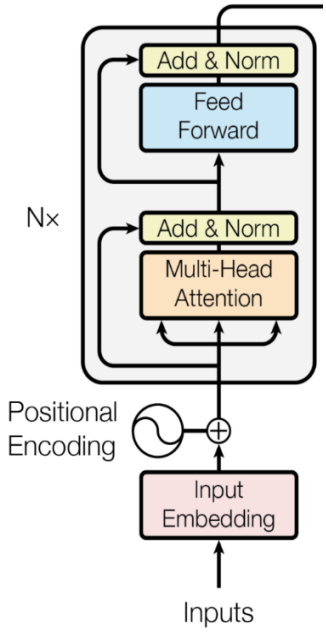
Encoder-Decoder Transformer

This architecture[2] was mainly introduced to solve the problem of Natural Language Processing (**NLP**), but it can be also applied to a larger class of problems such as **regression** which this paper is interested in. This architecture shows a great flexibility allowing to be implemented to solve a lot of problems of supervised learning and data analysis.

NLP has been a great challenge for AI and this architecture was able to challenge the problem of understanding human language allowing a machine to generate human-like text as much accurate as possible.

The **encoder-decoder Transformer** is an architecture composed by **a stack of encoders** and **a stack of decoders** (an encoder and a decoder may be seen as elemental major blocks of the overall architecture) that have different roles in the generation of the desired output.

Encoder



Input: the encoder receives a **sequence of data** that will be handled accordingly to the task problem (NLP or regression).

Input embedding: The input gets converted into **embedding vectors** which help to represent better our data.

Note: in the context of NLP, each word is usually tokenized and given in input to the architecture where a token is a most basic representation of the data that the architecture can understand.

Positional Encoding: in order to take into consideration the **order** of our sequence, the information regarding the **position** of each token will be given by adding it to the original input.

Since the encoder has not a recursive architecture, this encoding will ensure that each token will be entrusted with this information.

Even token: $PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right)$

Odd token: $PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right)$

Self-attention: this is a **technique** used to evaluate the importance of each word inside the sentence by relating it with the other words in the same sentence allowing to acquire a **global meaning**.

W_Q $(Query)$, W_K (Key), W_V $(Values)$ are matrices used in the process.

The important steps that we need to know are the ones as they follows:

First) the embedding are generated along with the matrices **query**, **key**, and **value** obtained by multiplying embedding per each of the matrices.

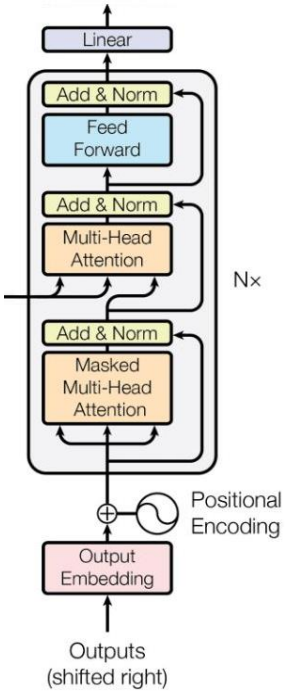
Second) a **score** is evaluated by multiplying each pair of **query** and **key** ($q_i * k_j = Score$) and it gives how much importance has query q_i with respect to k_j.

Third & forth) divide the score by the **square root** of the dimension of the embedding vectors for numerical stability and pass each score into a **softmax** which normalizes the data giving the importance of each word in the word.

[2] Architecture fully explained in the paper "*Attention Is All You Need*"

Multi-head self-attention: it is a parallel form of the **self-attention** where n **heads** are generated having different set of matrices (W_Q, W_K, W_V) behaving like a self-attention.
The process produces n projection z_i of the same input x, basically analyzing different aspects of the sequence at the same time. In order to move forward, all n projections are combined together into a single projection z using a suitable trained matrix W^O.
Residual) after this process it is possible to sum the result z with the original input x and normalize it all with a **layer normalization layer**. This step improves solving vanishing gradient situations while maintaining the original information.
Feed forward: each projection is passed down the **same feed forward network** but **separately** generating N independent streams and generating the output of a given token at step t.
Residual) it is the same process done after the **multi-head**; in this case, we are adding the input of the **Feed Forward Neural network** with its output and normalizing it all.
Encoder output: the output of the encoder is commonly called **memory** containing contextualized representations of the input by capturing relevant information.

Decoder



The overall architecture seems similar to the encoder one, but the decoder has a further step that stabilizes a connection with the encoder stack in order to process its data.
The "**multi-head attention**", also known as **encoder-decoder attention**, works differently compared to its counterpart:
The memory of the encoder stack is passed to the decoder so that while generating an output it both considers its hidden states and the encoder memory.
Decoders evaluates just the **query** by using **key** and **values** obtained by the encoders.
Masked Multi-head attention: the decoder will be provided by an additional input which is the **target sequence**.
The same steps done in the **input sequence** of the encoder stack can be done for the **target sequence** which will be given as input in the decoder stack.
The main difference is the **masked** version were all future positions are obscured to the decoder while predicting a given output at timestep t.
This is due to the fact that a decoder shouldn't be able to access information regarding the output Y_t or higher while generating an output at timestep t.
Masking process) during this process a mask will be provided to the decoder removing information above the timestep t avoiding the risk of giving importance to future information.

Preprocessing & problem definition

Problem nature & Loss function

Since the goal of this paper is to predict a **real value** of a future sample by observing a sequence of previous **real valued** observations, the problem can be classified as a **regression** problem.
This clarification is needed because the architectures that will be implemented are mainly used in **Natural Language Processing** (as previously discussed), so this is to avoid ambiguity.
The most common and practical **loss function** to use is the **Mean Square Error** (**MSE**):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i' - y_i)^2$$

Where y_i' and y_i represent the **predicted value** and the **real value** accordingly.
When it comes to loss functions, the MSE is one of the most common one regarding regression problem due to a list of factors:

1) It is a **non-negative function** with minimum value at 0, the close this function gets to it the smaller the error gets.
2) It is a **continuous function derivable** with respect to the model parameters making really easy to evaluate **gradients** to be used with **optimization algorithms**.
3) Its sensitivity to **errors** helps the model reducing as much as possible the distance between the real values and the predicted ones.

Data description

This paper has selected NASDAQ data from 13/Sep/2011 up to 13/Sep/2023 having 3020 days of resumes per company which will be used to forecast the value of $[Date, Open, High, Low, Close]$.
NASDAQ is one of the most followed markets in the USA which includes most of the notorious technological companies on the internet.
As already stated, data are presented as daily resume per company in the form of:

$$x = [Date, Open, High, Low, Close, Adj\ Close, Volume]$$

The data will be divided into three sets called **training**, **validation,** and **test** in order to efficiently train the model, tune hyperparameters and evaluate the accuracy of the model itself.

Feature removal

To maintain an acceptable capacity in the model while minimizing the amount of information removed, it has been decided to remove $(Adj\ Close)$ since this feature represents an adjusted version of $(Close)$ in proximity of **company events** such as **dividends** and **stock splits**.
Since this variable will show a linear dependency to $(Close)$ which will differ only during such events, it has been decided to remove it.
$(Volume)$ is a column representing the size (volume) of trades which is not directly related to the regression. Also, since $(Volume)$ data usually contains a significant amount of noise, it will be removed to limit overfitting problem.

Feature addiction

Since the analysis aims on accurately predict $[Open, High, Low, Close]$, it has been decided to use two commonly used features:

$$Price\ change = \frac{(Close - Price)}{Close} \quad \Big| \quad Volatility = \frac{(High - Low)}{High}$$

Price change grasp information about the variation percentage between opening price and closing price giving information about the **directionality** of the change and the **intensity**.
Volatility grasp information about how much the price has changed and giving an overall understanding of its behavior.
These variables may improve the stability because they act as a **noise smoothing** for the target variables since they observe different aspects which were not considered before.

Data normalization

This paper analyzes data from Amazon, Google, Microsoft, and Apple; therefore, data are represented on different scales generating related issues like **numerical instability**, **slow convergence**, **regularization problem**, **reduced generalization** etc.
In order to reduce such problems allowing the model to be scale free while improving convergence of the loss function, it has been decided to use **Min-Max normalization** evaluated as it follows:

$$x_i' = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

Where x_i is the original observation of a given feature and x_i' its corresponding normalized value generated by x_{max} and x_{min} which are accordingly the maximum value and minimum value recorded.
Other techniques can be used alternatively like the **Z-score normalization**, but there are advantages in using the min-max:
Range specific) Min-max allow to scale data into a given interval making them more easily readable
Note: in this paper, each company will have his own normalization in the range $[0,1]$
Data distribution) this normalization allow to **preserve relative order and distance of data point**; in context of NASDAQ data this step is very important by simply **rescaling** the range and by preserving important information in the data.

Data de-normalization

The data that needs to be predicted are also normalized so it is important to being able to denormalize it.
Note: data normalization information will be preserved allowing normalized data to be projected into a real prediction, this is an important step in order to use the model for practical uses.
The denormalization process can be evaluated as it follows:

$$Y_i = Y_i'(x_{max} - x_{min}) + x_{min}$$

Where Y_i' is the **normalized prediction** made by the model and Y_i is the actual forested value.

Input sequence and target generation

Each model during the training will be provided by a batch of sequences composed by N samples with 6 features with an overall dimensionality of $x_{batch} \in (batch\_size, N, inputDim)$ and the goal is to generate a prediction $Y'_{batch} \in (batch\_size, out\_len, outputDim)$. An ad hoc function will be responsible of taking groups of $(N + outLen)$ ordered chronologically generating the couple $\left(x^{(i)}, Y^{(i)}\right)$ that will be actively used in the learning process.

Evaluation metrics

The metrics chosen to evaluate the model are the Mean Absolute Error (**MAE**) (1), Symmetric Mean Absolute Percentage Error (**SMAPE**) (2), Mean Square Error (**MSE**) (3), and Root Mean Square Error (**RMSE**) (3)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - y_i'| \quad (1)$$

$$SMAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - y_i'|}{(|y_i| + |y_i'|)/2} * 100 \quad (2)$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2 \quad (3)$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2} \quad (4)$$

Where y_i and y_i' are respectively **real stock value** and **predicted stock value**.
MAE and **MSE** will give information about the average error with the difference that MSE will be more sensible to outliers and larger errors compared to MAE.
SMAPE in a stabler alternative to MAPE which describes as a percentage the error; it is symmetric which means that will consider both **overestimated error** and **underestimated error**.
Lastly, **RMSE** is a scaled version of MSE on the same unit of the input data.

Last notes – Generalization and normalization techniques

During the development of a neural network, prevention against **overfitting** and improving **numerical stability** are considered fundamental steps in the development of an effective and robust model.

The **overfitting** phenomenon occurs when a neural network starts "memorizing" data instead of generalizing them decreasing performances greatly. In order to avoid this problem, models will be provided with proper **dropout** techniques; it is a strategy which consists of random deactivations of each neuron by a probability $(drop\_prop)$. Every time a network perform a train iteration with dropout, it has to predict the output coping with the remaining active cells making it less dependent by a single neuron improving representation and robustness.
Note: a dropout technique will be performed on the input x with a much less probability $(drop\_propIN)$ which consists of randomly remove part of the input. This is a less common technique which should be handled carefully because it may lead to loss of important information.

Improving **numerical stability** is another important step because it aims on increasing convergence speed and the overall stability of the model. A useful technique consists in implementing "**layer normalizations**"; it consists of adding middleware layers which will be used to normalize the data. Thanks to these layers, **gradients** will be scaled according to the normalization because it will pass through these blocks limiting phenomenon of **vanishing gradients** or **exploding gradients**
First) each feature x will be normalized separately as it follows:

$$\hat{x} = \frac{x - E[x]}{\sqrt{\sigma^2 + \varepsilon}} \rightarrow \left(\sigma = E\left[x - E[x]\right]^2\right)$$

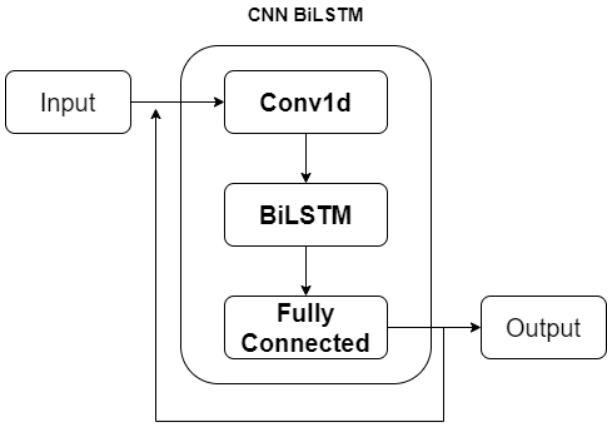
Second) each feature $\hat{x}$ will be scaled and shifted thanks to the parameters γ and β:

$$y = \gamma\hat{x} + \beta$$

These parameters are learned during the training by further improving flexibility and stability.

CNN BiLSTM model

Model structure



The first presented model will combine a better data representation given by the **convolutional layers** with the sequential context understanding capability of **Bidirectional Long Short-Term Memory** (**BiLSTM**). This model is known as **CNN BiLSTM** and combines the power of convolution with the one of recurrency in order to accurately solve a lot of sequential problems.

Convolutional 1d (Conv1d)

Since we are talking about sequences of data, convolution will act like a **sliding window** on the sequence sliding over the **time dimension**, over **all channels**.
It has been decided not to reduce the original sequence length N while the original channel size increases from $inputDim$ up to CNN_{out} which is a parameter that can set before training.
There are 6 convolutional layers in total all set up with **kernel** $K = 5$ and **stride** $S = 1$.
Since the dimension N is preserved, $padding$ needs to adapt accordingly:
$P = \frac{1}{2}(K - 1)$, this follows because **stride** has been set to 1 allowing to change freely K as long as it is an **odd number**. The model uses a $K = 5$, so $P = 2$.
Receptive sequence: since every convolutional layer uses the same parameters, knowing that we have 6 convolutional layers in total it is possible to evaluate how much of the original sequence each sample perceive from the first layer up to layer $L = 6$ receives:

$$receptive(x) = 1 + x(K - 1) \quad \rightarrow \quad receptive(L) = 25$$

BiLSTM

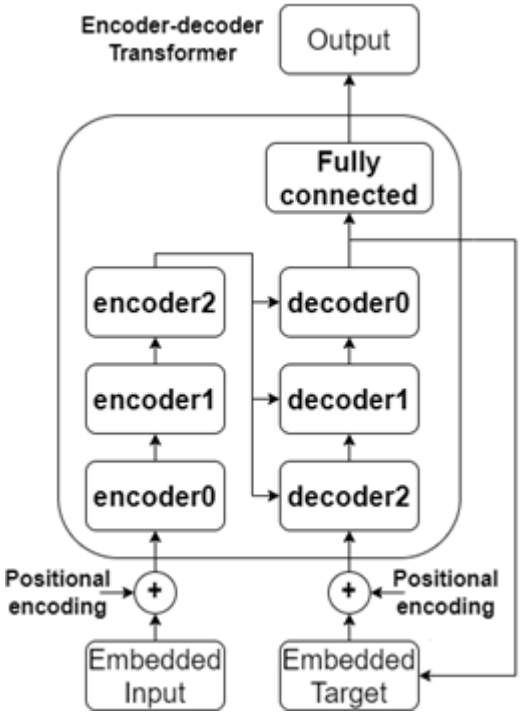
As already mentioned, BiLSTMs are a concatenation of two LSTM architecture having two different directions. The input of the cell will be given by the output of the Conv1d; therefore the cell must be able to receive an input of CNN_{out} channels. It has been decided to keep the **hidden state dimensionality** equal to the CNN_{out} which will be doubled due to its architecture.

Fully connected layer with autoregression approach

Since our model will be autoregressive, among the sample generated during the **BiLSTM** layer, only the last output will be considered in order to generate the final output Y_t.
An empty list Y will concatenate a total number of out_len samples generated by each iteration of the autoregressive cycle. During each iteration, the output of the recurrent cell will pass through a **fully connected layer** generating the new sample of the sequence.
Note: the fully connected layer should be able to reduce the input channel dimensionality of $2 * CNN_{out}$ down to $outputDim = 4$ channels.

Encoder-decoder transformer model

Model structure



The second model has revolutionized the field of machine learning with its outstanding capability of handling sequences of data in a parallel way.

The architecture is mainly composed by a **stack of encoder** and a **stack of decoders** which have a very important role.

Encoders have a particular importance in acquiring the input and thanks to their **self-attention mechanism** are able to grasp underling relationships and dependencies within the samples in a sequence.

Decoders, on the other hand, have the roles of generating sequences based on the **encoder** result (its **memory**).

Thanks to a **conditioned self-attention** and an **autoregressive decoding**, it is able to generate the output sequence result of complex relationships in the data.

Encoder & Decoder

Since the input and output sequence are real values, a tokenization is not needed, the input may be given to the network as it is.

It has been decided to fix the number of heads in both encoder and decoder to the same number $nhead$. The dimension of the embedding vector d_{model} can remain equal to $inputDim$ (Number of features per observation x) or it may be increased using an **embedding layer** which consists of multiple layers of fully connected layers alternating with ReLU increasing the overall dimensionality.

Note: increasing the dimensionality from $inputDim$ to d_{model} may lead to some advantages. In the first place, it increases the **capacity** of the model allowing more complex representations. Furthermore, while deciding the parameter d_{model}, $nhead$ will be bounded by it since it must be a divisor of d_{model}. To conclude, by changing d_{model} we allows the possibility of a better accuracy given by a better representation and a better configuration of the parameters.

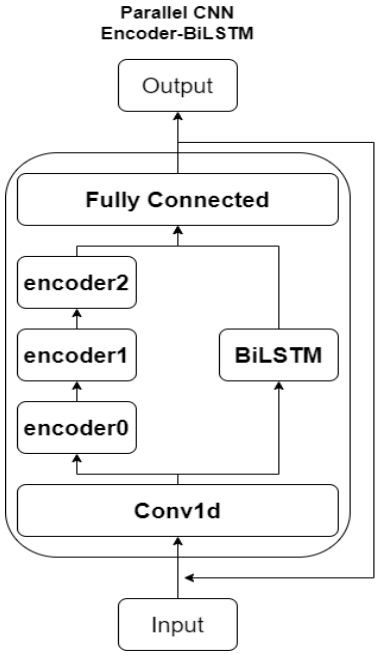
Fully connected layer with autoregression approach

Since the decoder will be autoregressive generating the target samples, at the generic timestep t it will receive a sequence long $(t+1)$ samples masking the element at index (t) due to the masked attention layer avoiding the decoder to "cheat" by knowing future positions.

The autoregressive cycle will end when out_len samples have been generated as output.

Parallel CNN encoder-BiLSTM model

Model structure



The last model is a combination of the two already presented models. The idea of this architecture is to combine the advantage of three already discussed components.
Conv1d: the convolution over 1D is a great tool to increase the **receptive sequence** of each sample while allowing a better representation of the data.
Encoder stack: useful to grasp hierarchical relationship and to improve discovering more complex representations of the samples.
BiLSTM: due to the bidirectionality of the cell, it is an excellent way of discovering dependencies both forward and backward in the sequence. Furthermore, LSTM are notorious for their ability of keeping long-term dependencies.
Concatenation: information generated by both **encoder** and **BiLSTM** will be combined allowing the model to use diversified data to generate an accurate prediction
Fully connected layer: the concatenated input will pass through a fully connected layer reshaping the dimensionality and it will produce the final output that.
Autoregression: this model will receive the output produced at timestep $(t-1)$ to produce a new output at timestep t. The process will end when a total of out_len samples have been produced.

Parameters configuration & last notes

Conv1d: it has been decided to keep the same configuration of the **CNN BiLSTM** having 6 convolutional layers with $(Kernel, Stride, Padding) = (5,1,2)$ with a receptive sequence of 25.
The output batch will have dimensionality $z0 \in (Batch, sequence\_len, CNN_{out})$
Encoder stack & positional encoding: it has been decided to **exclude positional encoding** due to the presence of the **BiLSTM layer** which will handle dynamically temporal dependencies in the data.
The d_{model} of the encoder will be the same as CNN_{out}; alternatively, it is possible to include an **embedding layer** in order to dynamically change the value of d_{model} and by extension $nhead$.
Note: an embedding layer may not be necessary if CNN_{out} is carefully chosen.
The output batch will have dimensionality $z1 \in (Batch, sequence\_len, d_{model})$
BiLSTM: the layer must be able to take a batch with CNN_{out} channels and it has been decided to keep it as output dimensionality, but keep in mind that it will be doubled due the bidirectionality.
The output batch will have dimensionality $z2 \in (Batch, sequence\_len, 2 * CNN_{out})$
Concatenation & fully connected layer: the outputs of the two architectures will be concatenated with respect to the channel axis generating an overall dimensionality of $(d_{model} + 2 * CNN_{out})$.
The **fully connected layer** must reduce $(d_{model} + 2 * CNN_{out})$ down to $outputDim = 4$.
Autoregressive approach) the last sample produced will be passed down as input to the architecture.
The process will end when out_len samples have been produced.

Training and model evaluation

Training process

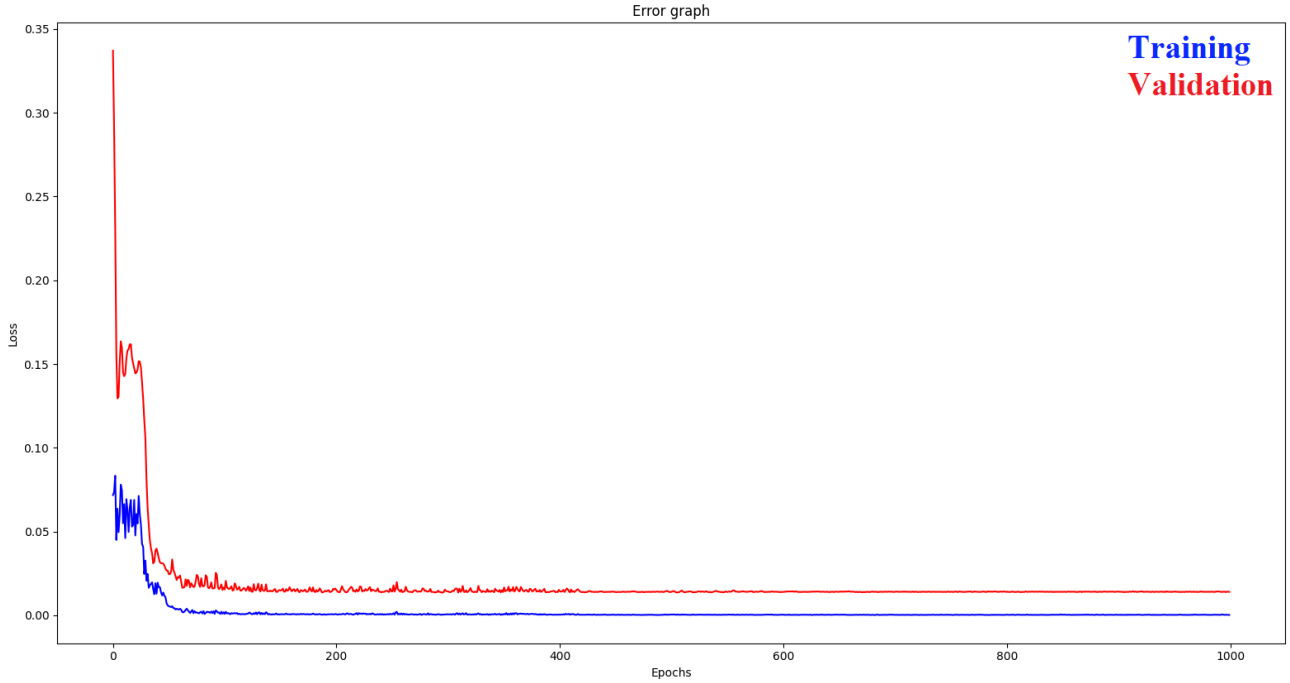
The already normalized samples will be divided into three categories (**Training**, **Evaluation**, **Test**); training data will contain samples inside the range (13/Sep/2011 to 12/Sep/2021), evaluation samples inside the range (13/Sep/2021 to 12/Sep/2022), and the remaining samples up to (12/Sep/2023) will be given to the test set; these three sets will be used to respectively to **train** the model by tuning its weights, **tune** hyperparameters such as **learning rate**, **normalization**, **dropout probability** etc., and **evaluation** of the model using data that have been obscured from the first two processes.
Due to limited hardware capacity, a **holdout** will be used instead of a **cross-validation**.

More notes

Since the **encoder-decoder Transformer** presented a **serious overfitting problem**, a further normalization technique has been added called **data augmentation**; this technique consists of adding a **gaussian noise** to the original data proportional to the **sigma value** ("$self.noise$" inside the code).
The application consists of creating a $noise = torch(x.size()) * self.noise$ which will be added to the data $x = x + noise$.
Decreasing Learning Rate) during the training process, the learning rate lr will decrease if both **training set** and **validation set** don't improve after $threshold$ iteration.
Note: if $threshold$ is incorrectly chosen, it will result in slowing the convergence.
Too many decreasing) if the model does not improve even by **decreasing** the learning rate, it will activate a counter $decreased$ which will stop the training process if after 4 consecutive decreased learning rate the model hasn't improved yet.

CNN BiLSTM

The best set of parameters found after the hyperparameters tuning phase are $lr = 0.005$, $lmd = 0.01$ (lmd stands for **lambda**), $drop\_prop = 0.5$, and $drop\_propIN = 0.01$.
The losses of the best set of parameters has been stored; the final losses generated by the model are approximately $loss_{train} \cong 1.58 * 10^{-4}$ and $loss_{valid} \cong 1.357 * 10^{-2}$ stabilizing around the 160th iteration.



Evaluation metrics

The metric evaluated are summarized in the table below:

	MAE	**SMAPE** (%)	**MSE**	**RMSE**
Open	0.061	25.12	0.0063	0.079
Close	0.067	25.48	0.0077	0.086
Low	0.065	25.94	0.0071	0.084
High	0.063	25.34	0.0068	0.083

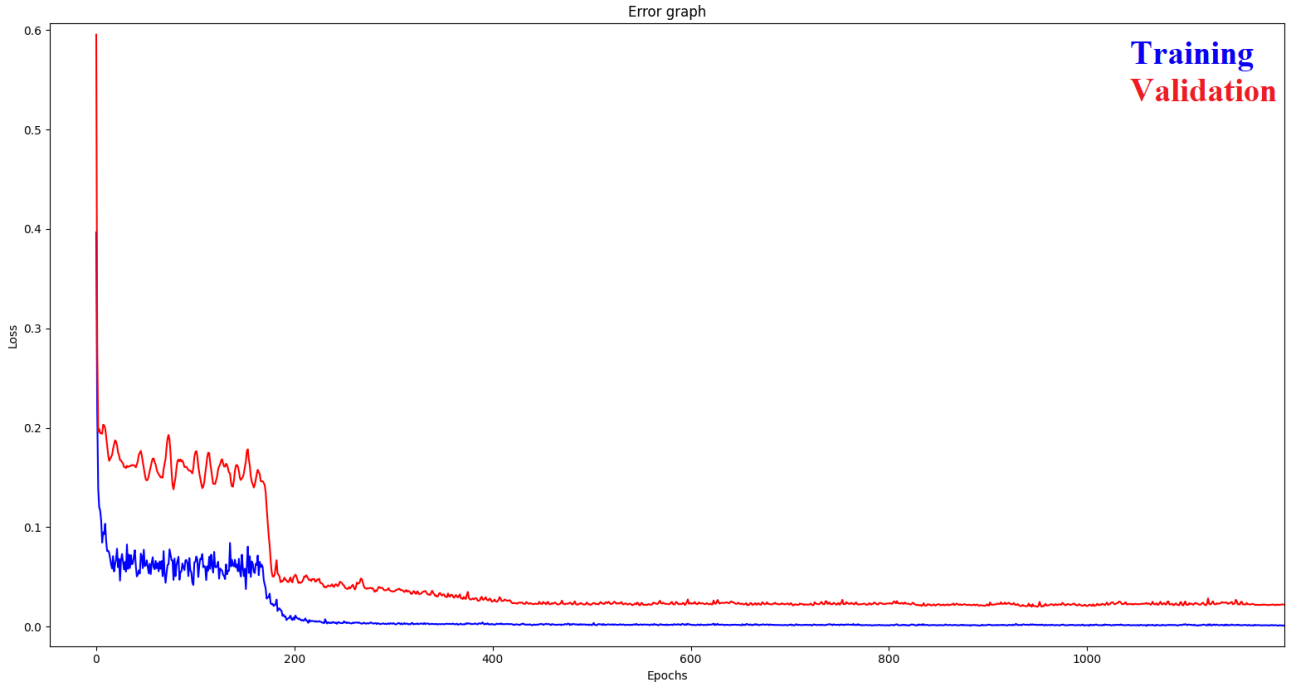
The results shows the best result recorded for the variable **Open** and the worth for the variable **Close**. Overall performance of the model can be summarized by an average of all the listed features.

MAE	**SMAPE** (%)	**MSE**	**RMSE**
0.064	25.47	0.00696	0.083

Encoder-Decoder Transformer

The best set of parameters found after the hyperparameters tuning phase are $lr = 0.01$, $lmd = 0.001$, $drop\_prop = 0.2$, $drop\_propIN = 0$, and $self.noise = 0.001$.

The losses of the best set of parameters has been stored; the final losses generated by the model are approximately $loss_{train} \cong 7.74 * 10^{-4}$ and $loss_{valid} \cong 1.997 * 10^{-2}$ stabilizing around the 500th iteration.



Note: during the training phase, the model seems to have found an initial plateau which was later surpassed.

Evaluation metrics

The metric evaluated are summarized in the table below:

	MAE	**SMAPE** (%)	**MSE**	**RMSE**
Open	0.0733	26.35	0.0085	0.092
Close	0.0812	26.89	0.0103	0.101
Low	0.0798	26.48	0.0099	0.099
High	0.0743	26.14	0.0088	0.094

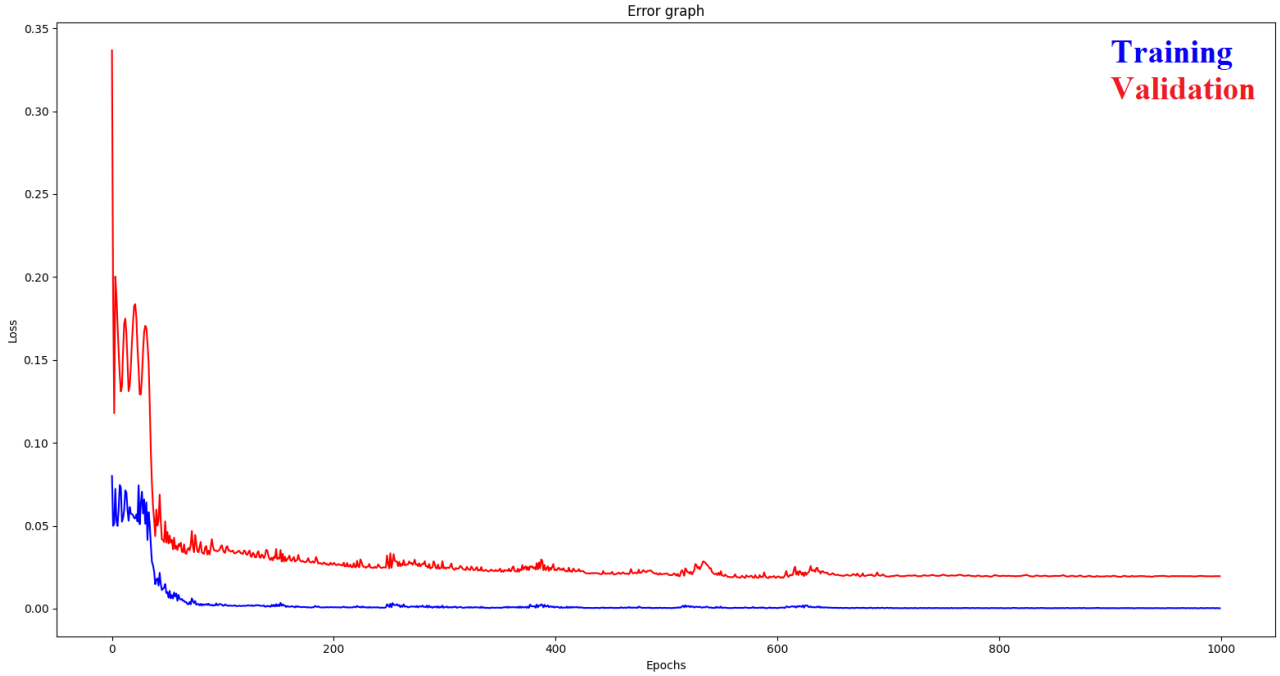
The results shows the best result recorded for the variable **Open** and the worth for the variable **Close**. Overall performance of the model can be summarized by an average of all the listed features.

MAE	**SMAPE** (%)	**MSE**	**RMSE**
0.0773	26.71	0.0094	0.097

Parallel CNN Encoder-BiLSTM

The best set of parameters found after the hyperparameters tuning phase are $lr = 0.005$, $lmd = 0.001$, $drop\_prop = 0.2$, $drop\_propIN = 0$ and $self.noise = 0$.

The losses of the best set of parameters has been stored; the final losses generated by the model approximately $loss_{train} \cong 1.35 * 10^{-4}$ and $loss_{valid} \cong 1.69 * 10^{-2}$ stabilizing around the 700th iteration.



Evaluation metrics

The metric evaluated are summarized in the table below:

	MAE	**SMAPE** (%)	**MSE**	**RMSE**
Open	0.0746	29.01	0.0086	0.093
Close	0.0815	29.27	0.0103	0.102
Low	0.0797	29.75	0.0098	0.099
High	0.0758	28.93	0.0092	0.096

The results shows the best result recorded for the variable **Open** and the worth for the variable **Close**. Overall performance of the model can be summarized by an average of all the listed features.

MAE	**SMAPE** (%)	**MSE**	**RMSE**
0.079	29.23	0.0095	0.097

Model comparison

	MAE	SMAPE (%)	MSE	RMSE
CNN BiLSTM	0.0640	25.47	0.0070	0.083
Open	0.0610	25.12	0.0063	0.079
Close	0.0670	25.48	0.0077	0.086
Low	0.0650	25.94	0.0071	0.084
High	0.0630	25.34	0.0068	0.083
Transformer	0.0773	26.71	0.0094	0.097
Open	0.0733	26.35	0.0085	0.092
Close	0.0812	26.89	0.0103	0.101
Low	0.0798	26.48	0.0099	0.099
High	0.0743	26.14	0.0088	0.094
Parallel	0.0790	29.23	0.0095	0.097
Open	0.0746	29.01	0.0086	0.093
Close	0.0815	29.27	0.0103	0.102
Low	0.0797	29.75	0.0098	0.099
High	0.0758	28.93	0.0092	0.096

The best model seems to be the **CNN BiLSTM** with the lowest metrics recorded.
Some common patterns can be recognizable such as **Open** being the best feature to predict while **Close** being the worst one maybe due to lack of relations to further improve its prediction.
The **hybrid model** performed worse than the other two models having similar **MSE** and **RMSE** to the **transformer**; what distinguishes it by the other models is having a high **MAE** and **SMAPE**; basically, the **parallel** model generates more error on the average prediction while the **transformer** will less commonly produce errors, but they will be higher than the other two architectures.
The **CNN BiLSTM** has the best results of them all:
MAE) a low value of MAE indicates a more precise model; this metric will not weight much outliers.
SMAPE) strong metric against division by 0 (compared to MAPE) which indicates intuitively the overall error both left and right with respect to the real value. This model has a lower range of error around the prediction.
RMSE and **MSE**) a lower **RMSE** and **MSE** indicate that the model doesn't generate large errors otherwise, they would have increased this metric significantly.

Transformer and Parallel CNN encoder-BiLSTM analysis

The two architectures distinguish themselves in terms of metrics giving some hint on what can be improved:
Transformer-MSE) since this metric its higher it may suggest a particular kind of **overfitting** due to lack of learned pattern inside the model which leads to huge error when a sequence with an underling patter is inserted into the model. In order to improve the architecture it is advisable to **reduce the complexity** of the model, **limiting the capacity**, or most likely **increase the amount of data**.
Parallel-SMAPE) the high margin of error around the prediction may be a sign of lacking **generalization** or **data representation**. In order to improve the model it is advisable to accurately change the model's architecture to allow it to efficiently grasp important relationships.

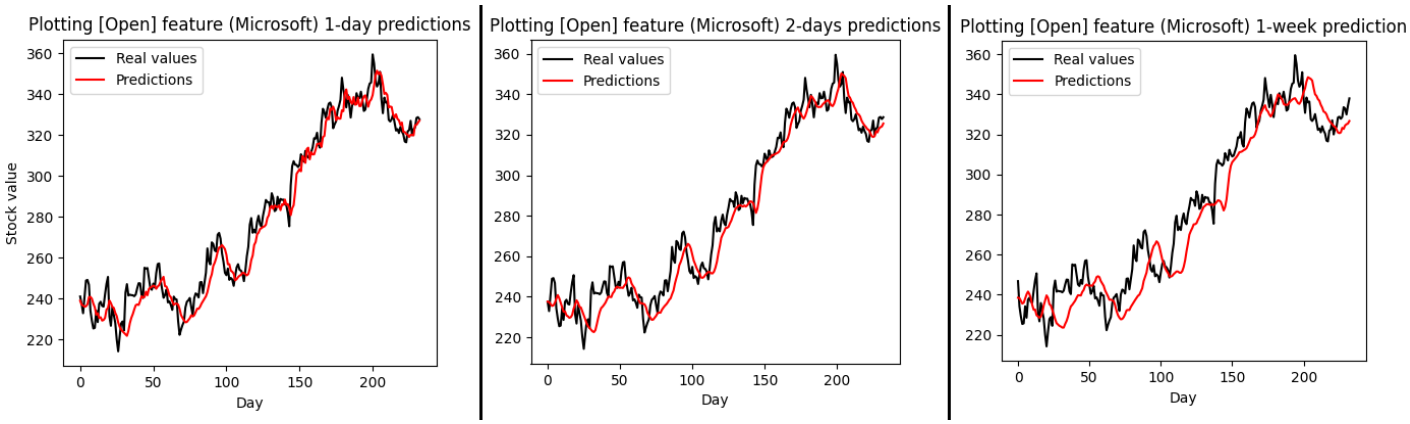
Practical uses and conclusion

Practical use

It is possible to plot each variable of the predicted data by **de-normalizing** their prediction using information previously stored during the **normalization process**, by doing so the model is allowed to make a prediction of the future which may assist the user during the decision-making.

Related problems

These models have the ability to take an input sequence N giving as output a sequence out_len.
By giving more sample it may be possible to improve predictions both short and long terms, but it may result in the opposite effect by giving to many samples in a given sequence without allowing the model to grasp important relationships.
Long forecasting) intuitively, the more a model gets pushed into forecasting the future the less accurate the predictions become.



These graphs show how the same samples forecasted at different range of times can differ; the **1-day prediction** (left graph) has an overall accurate prediction compared to their corresponding values. On the contrary, the **1-week prediction** (Right graph) shows a larger gap between real value and forecasted one.
Note: these predictions were made by using the **CNN BiLSTM** since it has been stated to be the best among all created models.

Future developments

Normalization

The min-max normalization has shown that it is important to normalize the input data, but this doesn't ensure that this particular normalization was the best one in this scenario.
Further developments may be conducted by using different kind of normalization, comparing the results, and finding a more suitable normalization for the problem.

Generalization (Overfitting)

Even though a lot of techniques and tricks have been implemented in order to reduce overfitting, models have shown signs of it even balancing all the normalization techniques during the **hyperparameters tuning phase**. Future developments may include more techniques that may reduce overfitting even more.

Hyperparameters tuning (Cross-validation)

Due to hardware limitations, it has not possible to implement a **cross-validation** which would have generated better results. Future developments may substitute the implemented **holdout** with a **cross-validation** using a wider range of hyperparameters.

Network structure

Even by considering the best model (**CNN BiLSTM**), it can be assumed that different configurations of layers may result in better result, especially for the **transformer** and the **parallel CNN encoder-BiLSTM**. Future developments may focus on finding a trade-off between complexity, capacity, and model structure in order to further improve every of the presented models.

Analyzed data

Even though the paper was focused on forecasting NASDAQ data of four listed companies, it is possible that by including data generated by **other companies** may improve on the overall generalization of the problem. By increasing the amount of data it is possible to create more complex networks and it is also a great tool to prevent overfitting.
Future developments highly suggest the usage of data of different companies.

Conclusion

This paper has shown strong evidence that deep learning models can be applied to NASDAQ data in order to generate an accurate prediction. The results suggested that a careful data analysis with a proper training of a deep neural network can capture complex implicit relationships within the data that can be exploited in the stock market. The ability of these networks to learn complex non-linear relations makes them tools of great values which may generate interest among investors depending on the accuracy.
Even though these models can understand multiple complex scenarios it doesn't mean that they are the best way of investing money; multiple aspects that has not been considered in this paper may significantly influence the accuracy of the predictions. Nonetheless, they remain great tools that, given a reasonable amount of time, may improve in ways that people can't even imagine.