VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

PROJECT

INTRODUCTION TO MACHINE LEARNING

FINAL PROJECT

Authors: **NGUYEN NGOC TU – 520K0231**
LE HUYNH HUYEN TRANG – 520C0156
Class**:** **20K50301**
Admission Course**:** **24**
Supervisor: **Prof. LE ANH CUONG**

HO CHI MINH CITY, 2022

VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

PROJECT

INTRODUCTION TO MACHINE LEARNING

FINAL PROJECT

Authors: **NGUYEN NGOC TU – 520K0231**
LE HUYNH HUYEN TRANG – 520C0156
Class**:** **20K50301**
Admission Course**:** **24**
Supervisor: **Prof. LE ANH CUONG**

HO CHI MINH CITY, 2022

ACKNOWLEDGEMENTS

We are really grateful because we managed to complete our report assignment about Introduction To Machine Learning within the time given. We sincerely thank our lecturer, Mr. Le Anh Cuong for the guidance and encouragement in finishing this assignment and also teaching us in this course.

PROJECT COMPLETED AT TON DUC THANG UNIVERSITY

We hereby declare that this is our own project and is under the guidance of Mr. Le Anh Cuong. The research contents and results in this topic are honest and have not been published in any publication before. The data in the tables for analysis, comments and evaluation are collected by the author himself from different sources, clearly stated in the reference section.

In addition, the project also uses a number of comments, assessments as well as data of other authors, other agencies and organizations, with citations and source annotations.

If we find any fraud, we will take full responsibility for the content of our project. Ton Duc Thang University is not related to copyright and copyright violations caused by us (if any).

Ho Chi Minh City, 16th October , 2022

Authors

(signature and full name)

Nguyen Ngoc Tu

Le Huynh Huyen Trang

LECTURER'S ASSESSMENT

_____

_____

_____

_____

_____

_____

_____

_____

Ho Chi Minh, date

(sign)

ABSTRACT

CONTENT

LIST OF ABBREVIATION

LIST OF FIGURE

LIST OF TABLE

PROBLEM 1 AND 2

I. Describe dataset:

Uniqlo (FastRetailing) Stock Price Prediction: https://www.kaggle.com/datasets/daiearth22/uniqlo-fastretailing-stock-price-prediction

In this project, we will use a financial data referred to as "Uniqlo (FastRetailing) Stock Price Prediction"

Stock market data can be interesting to analyze and as a further incentive, strong predictive models can have large financial payoffs. The amount of financial data on the web is seemingly endless. A large and well structured dataset on a wide array of companies can be hard to come by.

All the files have the following columns:

- Date - in format: yy-mm-dd
- Open - price of the stock at market open
- High - Highest price reached in the day
- Low - Lowest price reached in the day
- Close - Closing price
- Volume - Number of shares traded
- Stock Trading - Amount of money of volume in the day

II. Preprocessing

1. Loading Initial Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error as mse
```

2. Importing dataset

```
dataset = pd.read_csv('uniqlo_stock_trainset.csv')
dataset.head(3)
```

	Date	Open	High	Low	Close	Volume	Stock Trading
0	12/30/2016	42120	42330	41700	41830	610000	2.562803e+10
1	12/29/2016	43000	43220	42540	42660	448400	1.918823e+10
2	12/28/2016	43940	43970	43270	43270	339900	1.478067e+10

3. Change date column to right format

```
import datetime

# input_format = '%m/%d/%Y' # input format
```

```
# date_format = '%d/%m/%Y'
#          dataset.Date          =          [datetime.datetime.strptime(d,
input_format).strftime(date_format) for d in dataset.Date]
dataset.Date = pd.to_datetime( dataset.Date )
```

4. **Sort the dataset increasing by date => the latest value will locate at the end**

```
dataset = dataset.sort_values(by='Date').reset_index(drop=True)
```
✓ 0.6s

```
dataset.head(5)
```
✓ 0.1s

	Date	Open	High	Low	Close	Volume	Stock Trading
0	2012-01-04	14050	14050	13700	13720	559100	7.719804e+09
1	2012-01-05	13720	13840	13600	13800	511500	7.030811e+09
2	2012-01-06	13990	14030	13790	13850	765500	1.063561e+10
3	2012-01-10	13890	14390	13860	14390	952300	1.353341e+10
4	2012-01-11	14360	14750	14280	14590	1043400	1.519199e+10

```
max_open = description.Open[-1]
dataset.dtypes
```
✓ 0.1s

```
Date             datetime64[ns]
Open                      int64
High                      int64
Low                       int64
Close                     int64
Volume                    int64
Stock Trading           float64
dtype: object
```

5. Normalization

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(dataset.iloc[:,1:].values)  # do not include Date column
```

✓ 0.6s

```
data_shape = data_scaled.shape
print(data_scaled[:5])
print("shape", data_shape)
```

✓ 0.6s

```
[[0.00689944 0.00436318 0.00212134 0.         0.08753282 0.02641958]
 [0.         0.         0.         0.00165941 0.07761244 0.02157021]
 [0.00564499 0.00394764 0.00403055 0.00269654 0.13054896 0.04694202]
 [0.00355425 0.01142738 0.00551549 0.01389753 0.16948022 0.06733777]
 [0.01338072 0.01890713 0.01442512 0.01804605 0.18846651 0.07901139]]
shape (1226, 6)
```

6. Train Test Split

```
from sklearn.model_selection import train_test_split
```

✓ 0.1s

```
X = []
y = []
dates = []
n_step = 100 # predict
for i in range(n_step, data_shape[0]): # choose only ''OPEN''feature
    X.append(data_scaled[i-n_step:i, 0])
    y.append(data_scaled[i, 0])
    dates.append(dataset.Date[i])
X, y = np.array(X), np.array(y)

print(X.shape)
print(y.shape)
```

✓ 0.1s

```
(1126, 100)
(1126,)
```

```
split_rate = 0.9  # first 90% of time interval is used for training
split_size = (int)(X.shape[0] * split_rate)
X_train,  X_test = X[:split_size,:], X[split_size:,:]
y_train, y_test = y[:split_size], y[split_size:]
train_date, test_date = dates[:split_size], dates[split_size:]
print("train shape", X_train.shape)
print("test shape", X_test.shape)
```

✓ 0.7s

```
train shape (1013, 100)
test shape (113, 100)
```

```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
print(X_train.shape)
```

✓ 0.7s

```
(1013, 100, 1)
```

```
# input: the scaled value (predictions of model), an original max value of the scaled value
# output: origin state of the scaled value
def restore_ground_val(scaled_p, origin_max):
    return np.array([p*origin_max for p in scaled_p])
```

✓ 0.7s

PROBLEM 3

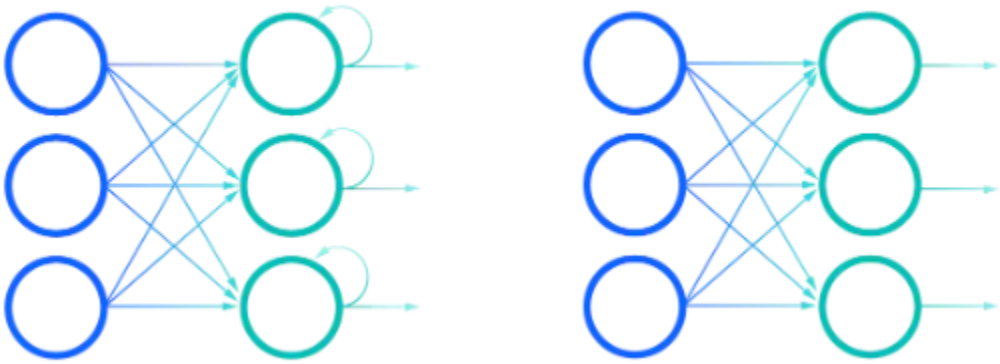
I. Introduction of two models

1. Recurrent Neural Network (RNN)

1.1.What are recurrent neural networks?

A recurrent neural network (RNN) is a type of artificial neural network that works with time series or sequential data. These deep learning algorithms are commonly used for ordinal or temporal problems like language translation, natural language processing (nlp), speech recognition, and image captioning; they are included in popular applications like Siri, voice search, and Google Translate. Recurrent neural networks, like feedforward and convolutional neural networks (CNNs), learn from training data. They are distinguished by their "memory," which allows them to influence the current input and output by using information from previous inputs. While traditional deep neural networks assume that inputs and outputs are independent of one another, the output of recurrent neural networks is dependent on the sequence's prior elements.While future events would be useful in predicting the outcome of a given sequence, unidirectional recurrent neural networks cannot account for them in their predictions.

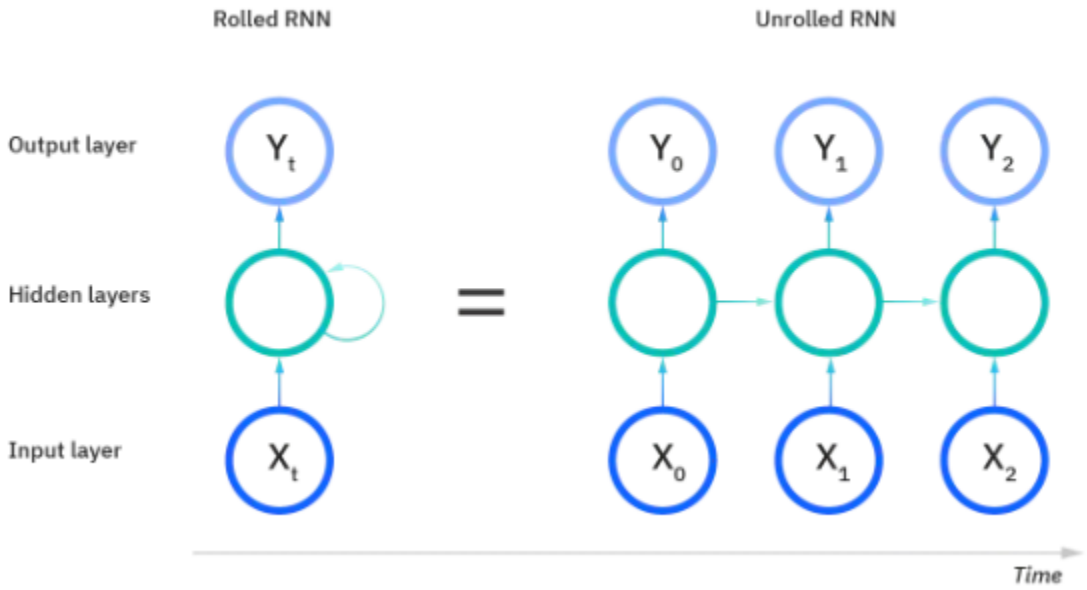
1.2.Recurrent Neural Network vs. Feedforward Neural Network



Comparison of Recurrent Neural Networks (on the left) and Feedforward Neural Networks (on the right)

- Let us use an idiom, such as "feeling under the weather," which is commonly used when someone is sick, to help us understand RNNs. The idiom must be expressed in that specific order for it to make sense. As a result, recurrent networks must account for the position of each word in the idiom before predicting the next word in the sequence.
- Looking at the image below, the RNN's "rolled" visual represents the entire neural network, or rather the entire predicted phrase, such as "feeling under the weather." The "unrolled" visual represents the neural network's individual layers, or time steps. Each layer corresponds to a single word in that phrase, for example, "weather." In the third timestep, prior inputs such as "feeling" and

"under" would be represented as a hidden state to predict the output in the sequence, "the."



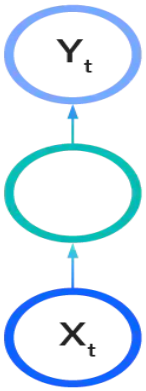
- Another distinguishing feature of recurrent networks is that they share parameters across all network layers. While feedforward networks have different weights for each node, recurrent neural networks share the same weight parameter across all layers. However, these weights are still adjusted in the backpropagation and gradient descent processes to facilitate reinforcement learning.
- To determine the gradients, recurrent neural networks use the backpropagation through time (BPTT) algorithm, which is slightly different from traditional backpropagation because it is specific to sequence data. BPTT works on the same principles as traditional backpropagation, in which the model trains itself by calculating errors from its output layer to its input layer. These calculations allow us to appropriately adjust and fit the model's parameters. BPTT differs from the traditional approach in that it sums errors at each time step, whereas feedforward networks do not have to sum errors because parameters are not shared across layers.
- RNNs frequently encounter two issues during this process: exploding gradients and vanishing gradients. The size of the gradient, which is the slope of the loss function along the error curve, defines these issues. When the gradient is too small, it continues to shrink, updating the weight parameters until they become insignificant—that is, 0—and then stops. When this happens, the algorithm stops learning. When the gradient is too large, it explodes, resulting in an unstable model. In this case, the model weights will become too large and will be represented as NaN. One approach to addressing these issues is to reduce the

number of hidden layers within the neural network, thereby removing some of the complexity in the RNN model.

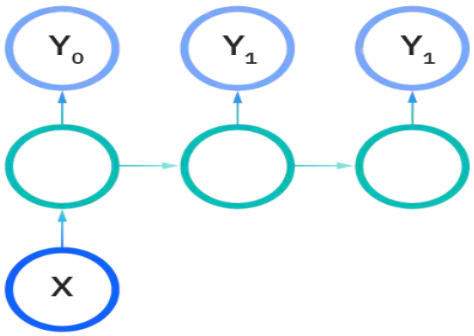
1.3.Types of recurrent neural networks

- Feedforward networks map one input to one output, and while we've shown recurrent neural networks in this manner in the diagrams above, they don't actually have this constraint. Instead, the length of their inputs and outputs can vary, and different types of RNNs are used for different applications such as music generation, sentiment classification, and machine translation. The following diagrams are commonly used to represent various types of RNNs:

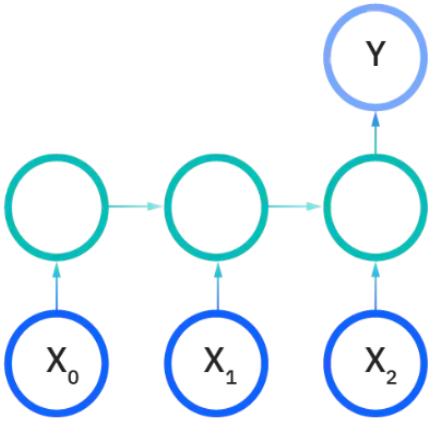
One-to-one:



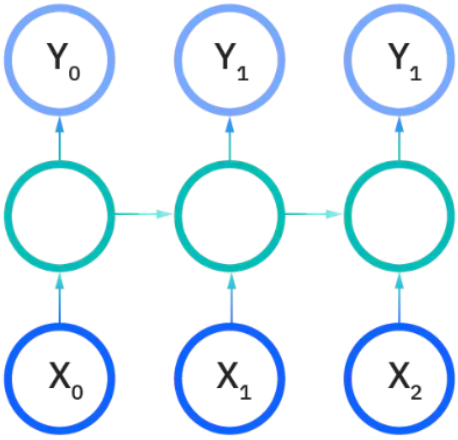
One-to-many:



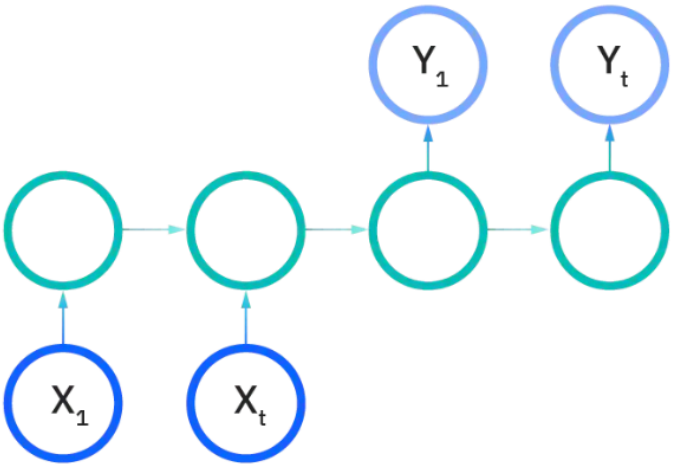
Many-to-one:



Many-to-many:



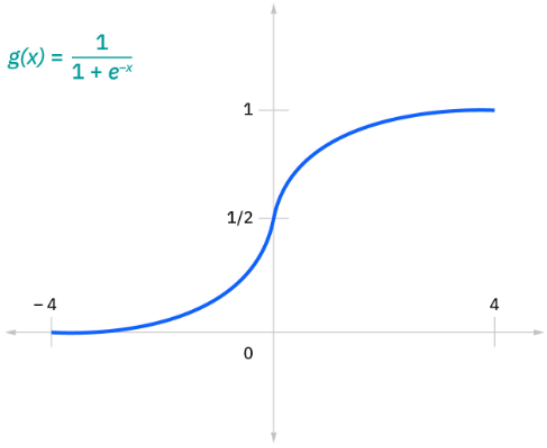
Many-to-many:



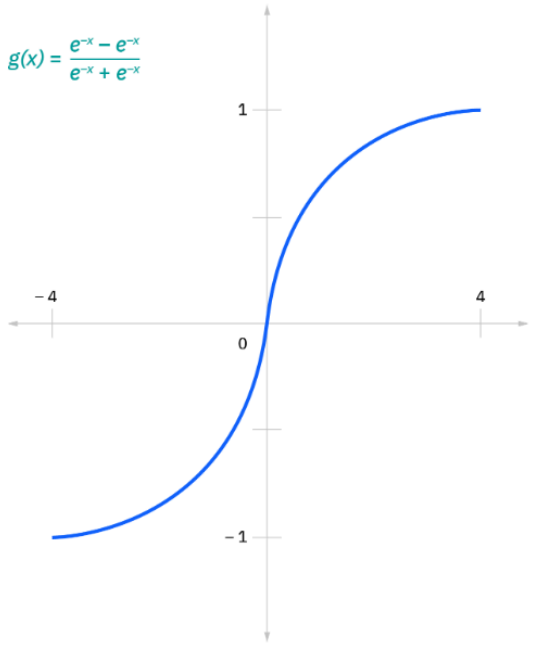
1.4.Common activation functions

As discussed in the Learn article on Neural Networks, an activation function determines whether a neuron should be activated. The nonlinear functions typically convert the output of a given neuron to a value between 0 and 1 or -1 and 1. Some of the most commonly used functions are defined as follows:

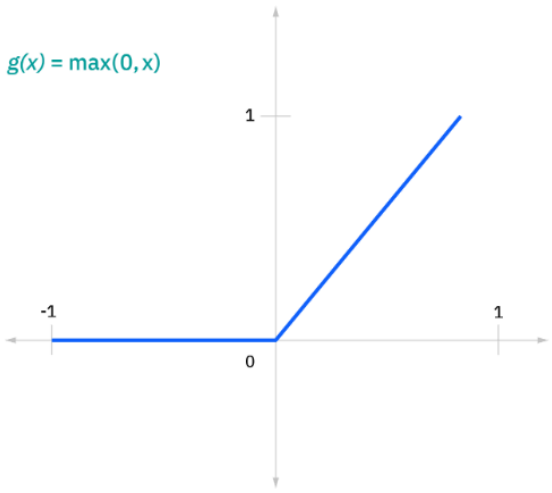
- **Sigmoid**: This is represented with the formula g(x) = 1/(1 + e^-x).



- **Tanh**: This is represented with the formula g(x) = (e^-x - e^-x)/(e^-x + e^-x).



- **Relu**: This is represented with the formula g(x) = max(0 , x)



1.5.Variant RNN architectures

- **Bidirectional recurrent neural networks (BRNN):** are a type of RNN network architecture. While unidirectional RNNs can only predict the current state based on previous inputs, bidirectional RNNs incorporate future data to improve accuracy. If we return to the example of “feeling under the weather” earlier in this article, the model can better predict that the second word in that phrase is “under” if it knew that the last word in the sequence is “weather.”

- **Long short-term memory (LSTM):** Sepp Hochreiter and Juergen Schmidhuber introduced this popular RNN architecture as a solution to the vanishing gradient problem. They address the issue of long-term dependencies in their paper (PDF, 388 KB) (link resides outside IBM). That is, if the previous state influencing the current prediction is not recent, the RNN model may be unable to predict the current state accurately. As an example, suppose we wanted to predict the italicized words in the sentence below: "Alice is allergic to nuts." She is allergic to peanut butter." The context of a nut allergy can assist us in anticipating that the food that cannot be consumed contains nuts.However, if that context was a few sentences earlier, it would be difficult, if not impossible, for the RNN to connect the dots. To address this, LSTMs have "cells" in the neural network's hidden layers that have three gates: an input gate, an output gate, and a forget gate. These gates regulate the flow of information required to predict the network's output. For example, if gender pronouns, such as "she," were used repeatedly in previous sentences, you could exclude them from the cell state.
- **Gated recurrent units (GRUs):** Similar to LSTMs, this RNN variant works to address the short-term memory problem of RNN models. It uses hidden states to regulate information rather than "cell states," and instead of three gates, it has two—a reset gate and an update gate. The reset and update gates, like the gates in LSTMs, control how much and which information is retained.

1.6.Recurrent neural networks and IBM Cloud

- IBM has been a pioneer in the development of AI technologies and neural networks for decades, as evidenced by the development and evolution of IBM Watson. Watson is now a trusted solution for businesses looking to apply advanced natural language processing and deep learning techniques to their systems while adhering to a tried-and-true tiered approach to AI adoption and implementation.
- IBM Watson Machine Learning, for example, supports popular Python libraries used in recurrent neural networks, such as TensorFlow, Keras, and PyTorch. Your enterprise can seamlessly bring your open-source AI projects into production while deploying and running your models on any cloud by utilizing tools such as IBM Watson Studio and Watson Machine Learning.

1.7.Modeling

```
    no_unit = 100
    # Modeling
    RNN = Sequential(layers=
        [
                        SimpleRNN(units=no_unit,  return_sequences=True,
input_shape = (X_train.shape[1], 1)),
             SimpleRNN(units=no_unit, return_sequences=True),
```

```
            SimpleRNN(units=no_unit, return_sequences=False),
            Dense(units = 1)
        ]
    )

    # Compiling
    RNN.compile(optimizer='adam', loss = 'mse', metrics=["mae"])
    # RNN.summary()
```

1.8.Fit train data

```
no_epoch = 10
batch_size = 32
validation_split = 0.2

history_rnn = RNN.fit(X_train, y_train, validation_split=validation_split, epochs = no_epoch, batch_size = batch_size, verbose=0)
```

2. MultiLayer Perceptron (MLP)

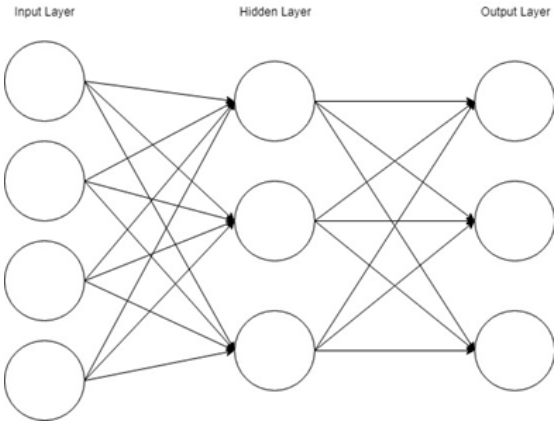
2.1.What are multilayer perceptrons?

The perceptron is very useful for classifying data sets that are linearly separable. They encounter serious limitations with data sets that do not conform to this pattern as discovered with the XOR problem. The XOR problem shows that for any classification of four points that there exists a set that is not linearly separable.

The MultiLayer Perceptron (MLPs) breaks this restriction and classifies datasets which are not linearly separable. They do this by using a more robust and complex architecture to learn regression and classification models for difficult datasets.

2.2. How does a multilayer perceptron work?

The Perceptron is made up of two fully connected layers: input and output. MLPs have the same input and output layers but may have multiple hidden layers in between, as shown below.



The algorithm for the MLP is as follows:

1. The MLP, like the perceptron, pushes inputs forward by taking the dot product of the input and the weights that exist between the input layer and the hidden layer (WH). At the hidden layer, this dot product produces a value. We do not, however, advance this value as we would with a perceptron.
2. MLPs utilize activation functions at each of their calculated layers. There are many activation functions to discuss: rectified linear units (ReLU), sigmoid function, tanh. Push the calculated output at the current layer through any of these activation functions.
3. Once the calculated output at the hidden layer has been pushed through the activation function, push it to the next layer in the MLP by taking the dot product with the corresponding weights.
4. Repeat steps 2 and 3 until you reach the output layer.
5. The calculations will be used at the output layer for either a backpropagation algorithm that corresponds to the activation function that was chosen for the MLP (in the case of training) or a decision will be made based on the output.

➔ *MLPs serve as the foundation for all neural networks and have significantly increased computer power when applied to classification and regression problems. Because of the multilayer perceptron, computers are no longer limited by XOR cases and can learn rich and complex models.*

2.3.Modeling

```
MLP = Sequential(layers=
    [
        Dense(units=128, activation='relu', input_dim = n_step),
            BatchNormalization(),
        Dense(units=64, activation='relu'),
            BatchNormalization(),
        Dense(units=32, activation='relu'),
            BatchNormalization(),
            Dropout(rate=0.3),
        Dense(units = 1)
```

```
        ]
    )

    # Compiling
    MLP.compile(optimizer='adam', loss = 'mse', metrics=["mae"])
```

2.5.Fit train data

```
no_epoch = 100
batch_size = 32
validation_split = 0.2

history_mlp = MLP.fit(x=X_train,y=y_train,
    validation_split=validation_split,
    epochs = no_epoch,
    batch_size = batch_size,
    verbose = 0
)
```

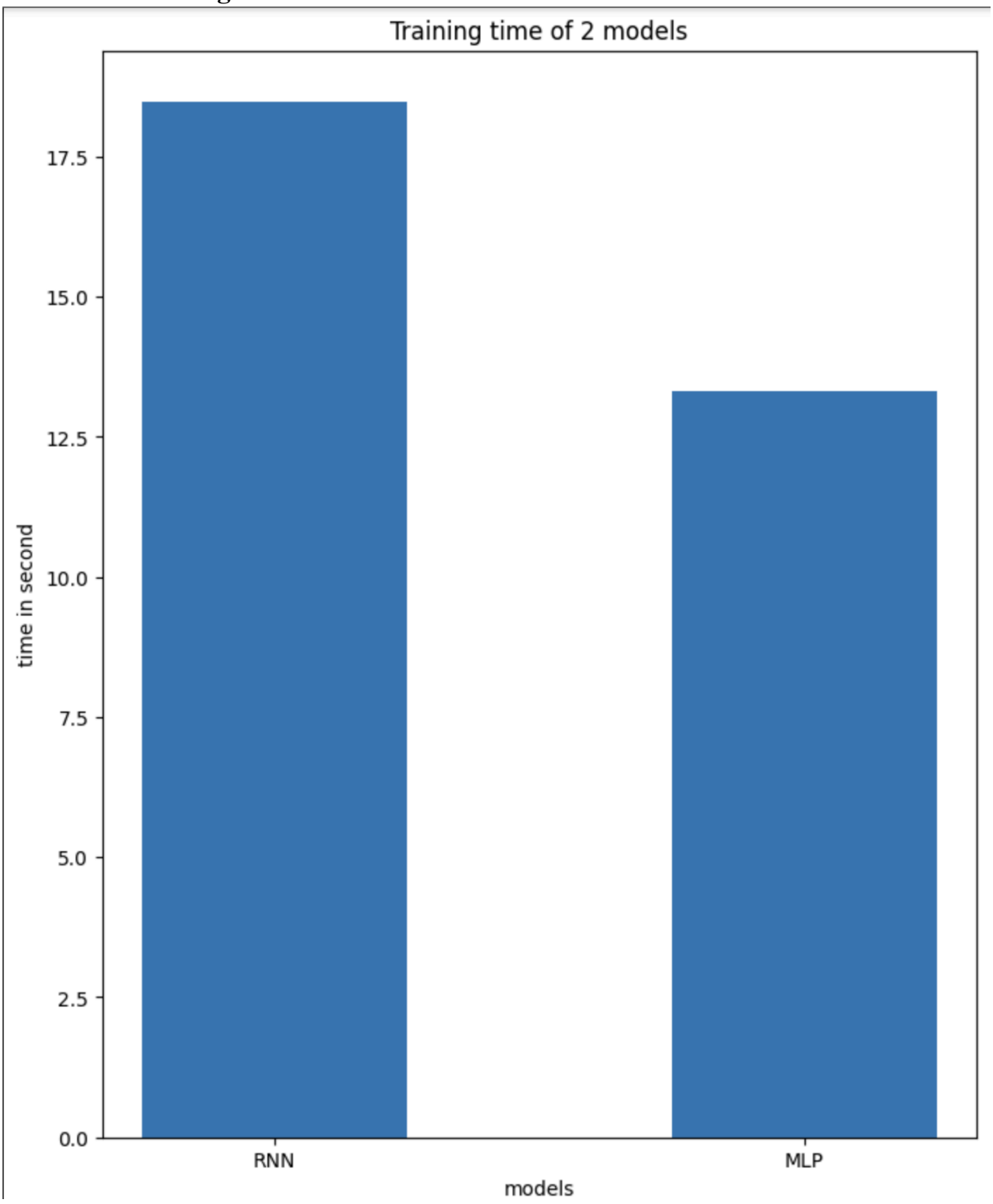
```
MLP.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 128)               12928

 batch_normalization (BatchN  (None, 128)              512
 ormalization)

 dense_2 (Dense)             (None, 64)                8256

 batch_normalization_1 (Batc  (None, 64)               256
 hNormalization)

 dense_3 (Dense)             (None, 32)                2080

 batch_normalization_2 (Batc  (None, 32)               128
 hNormalization)

 dropout (Dropout)           (None, 32)                0

 dense_4 (Dense)             (None, 1)                 33

=================================================================
Total params: 24,193
Trainable params: 23,745
Non-trainable params: 448
```

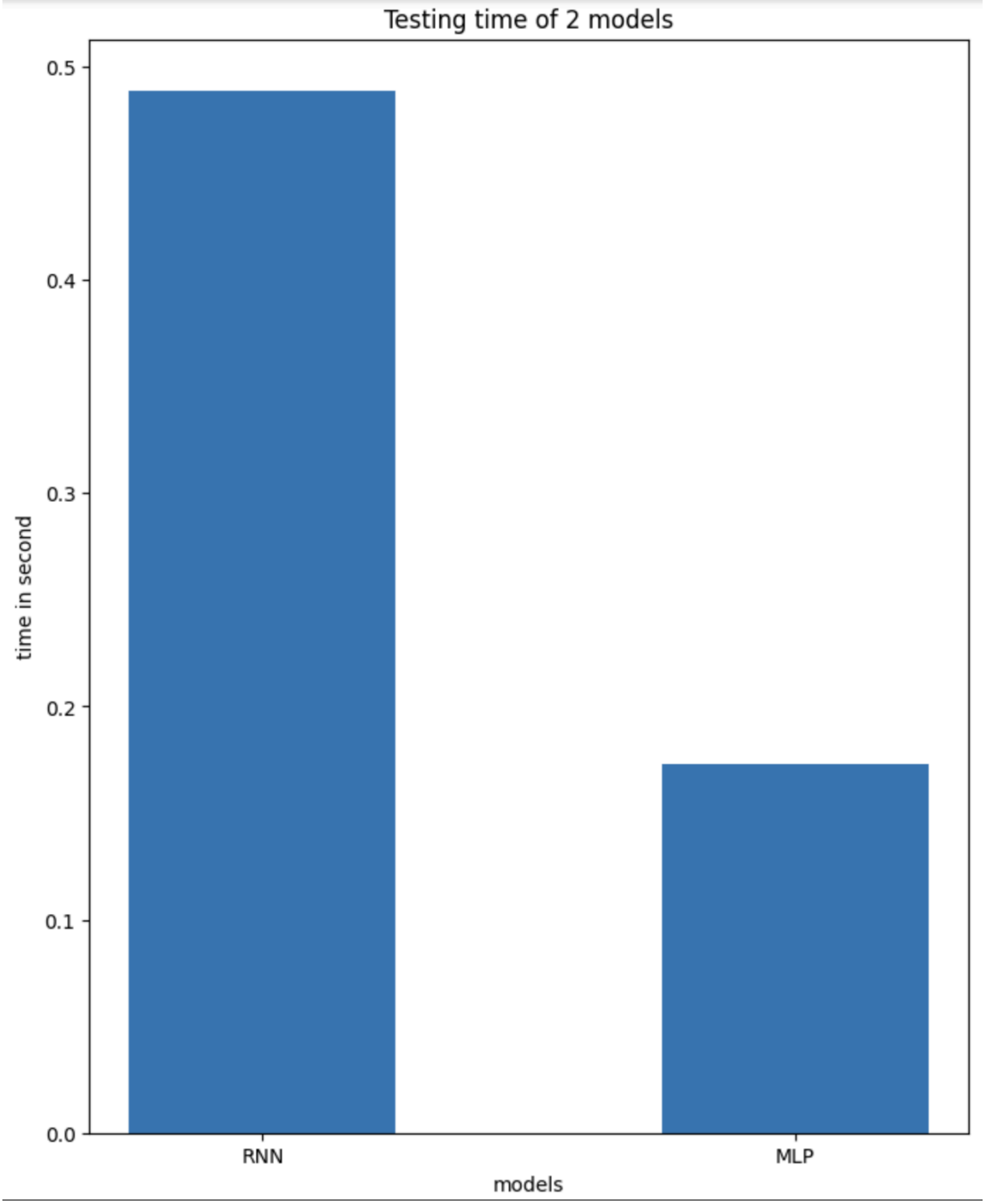
II. Comparison and evaluations of 2 models:

1. Training time and testing time of 2 models

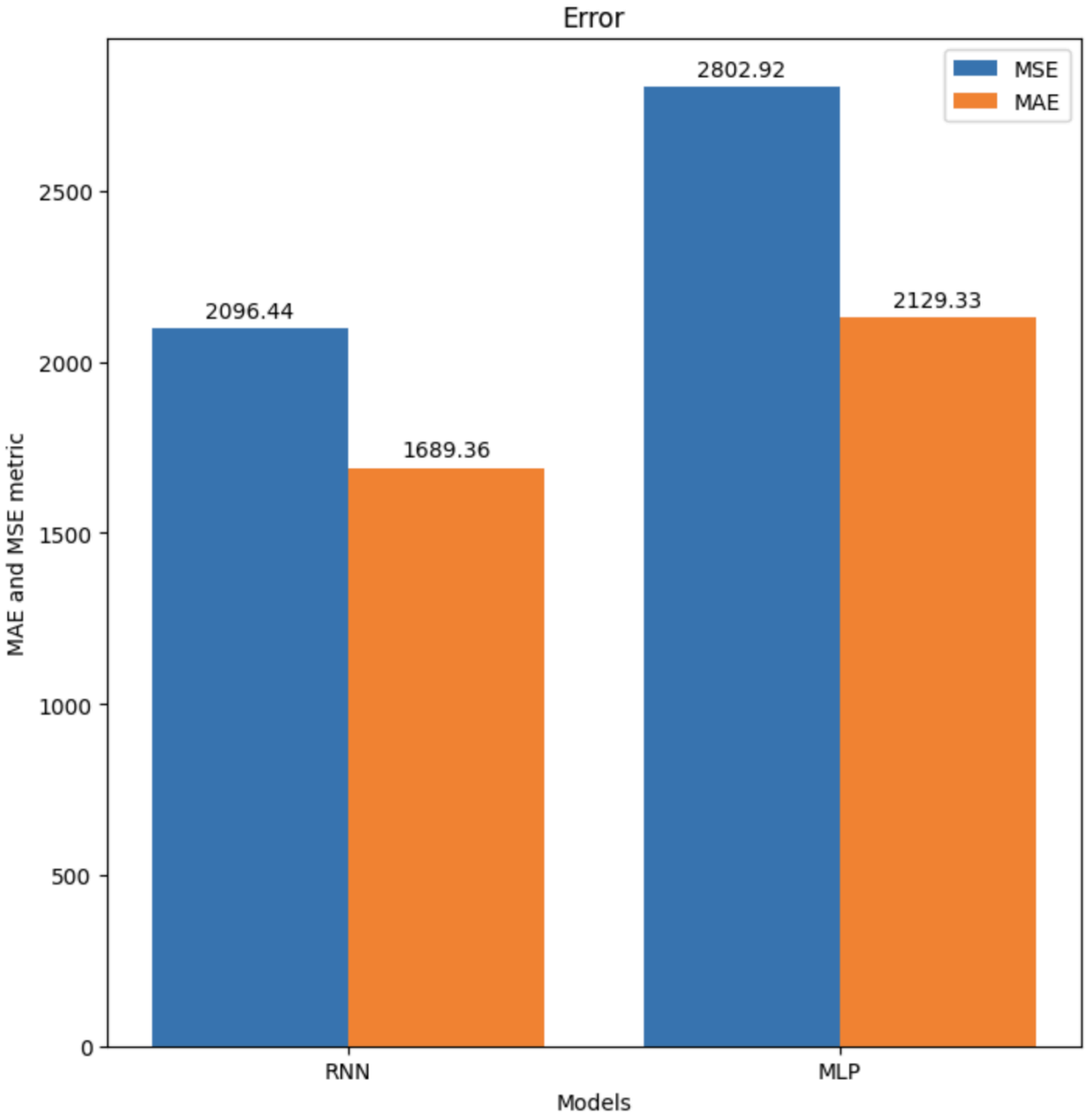
1.1. Training time:



1.2. Testing time:



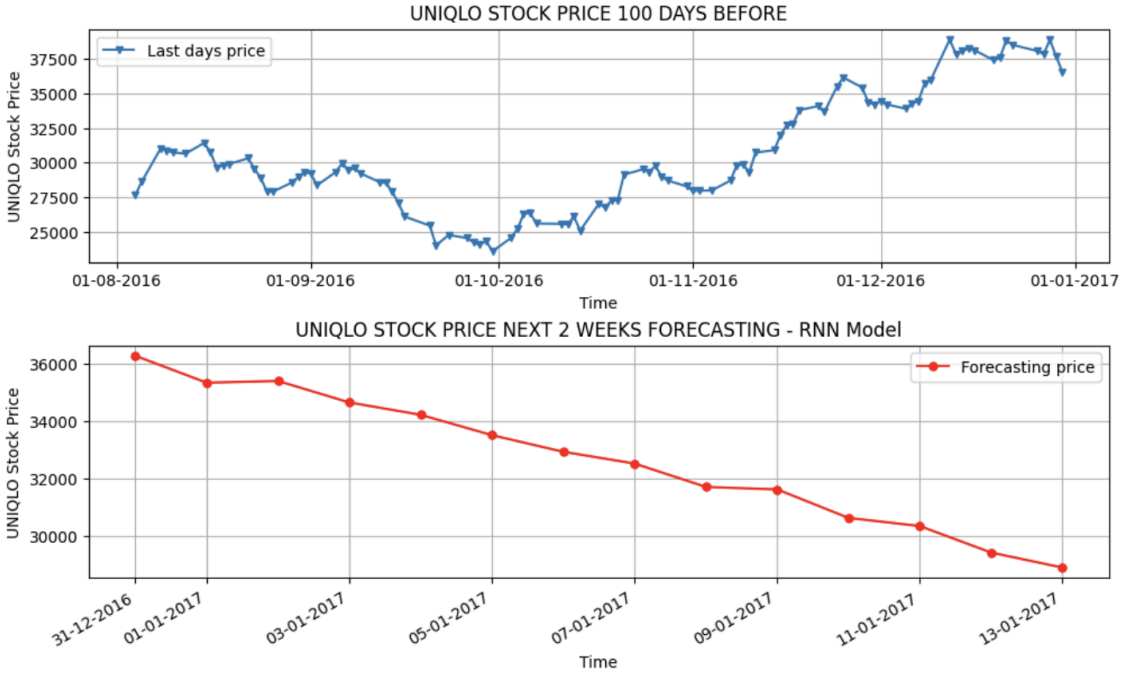
2. MSE and MAE of 2 models



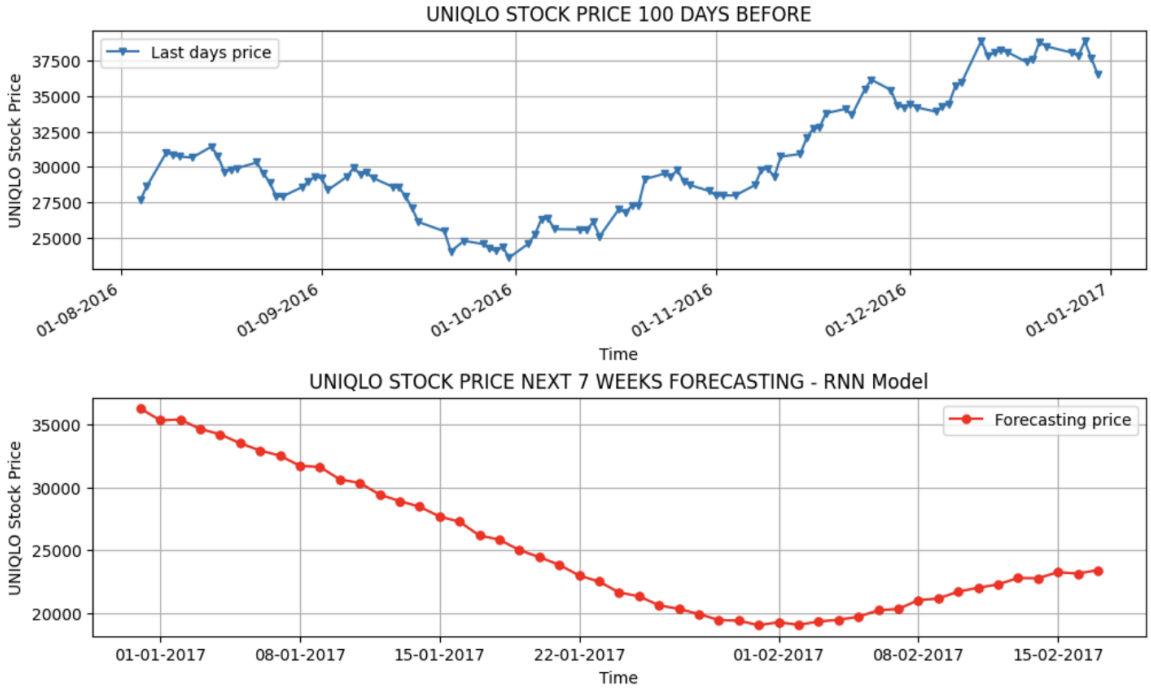
III. Forecasting

1. RNN model

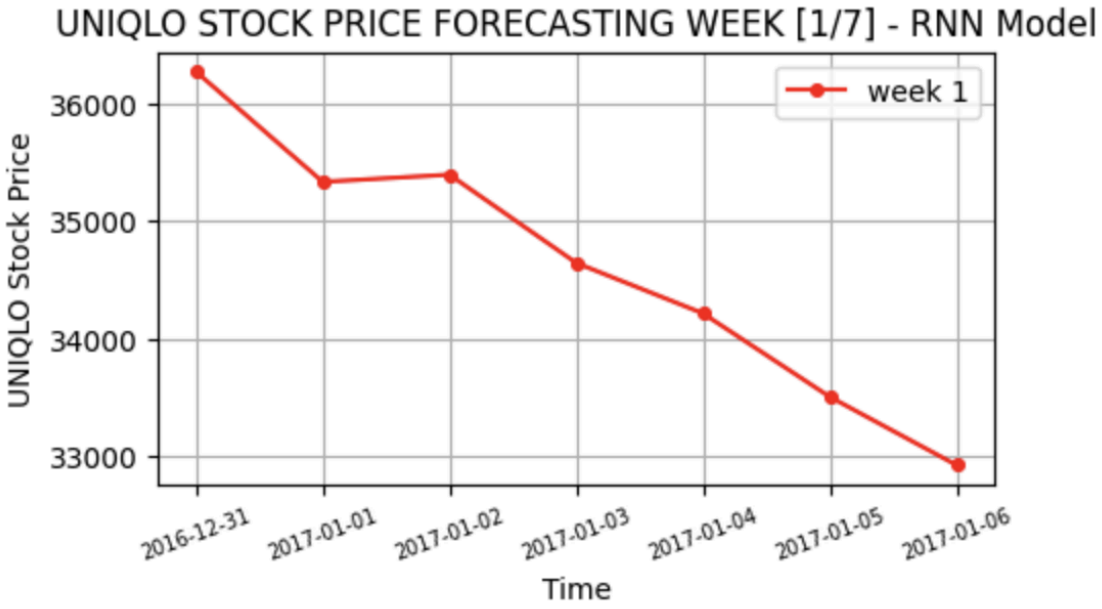
1.1. Next 2 weeks forecasting:

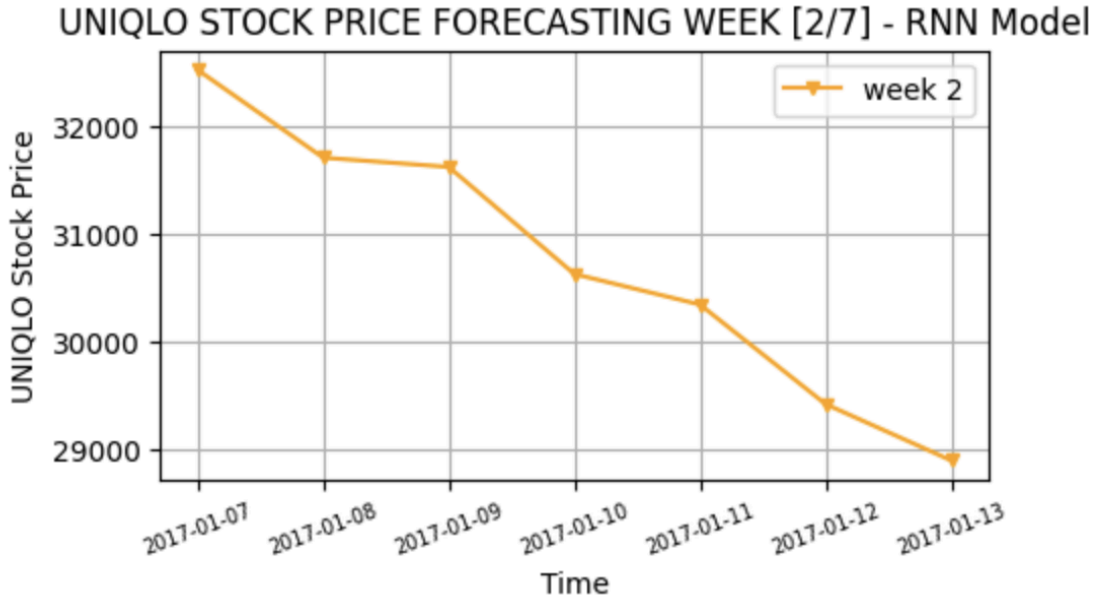


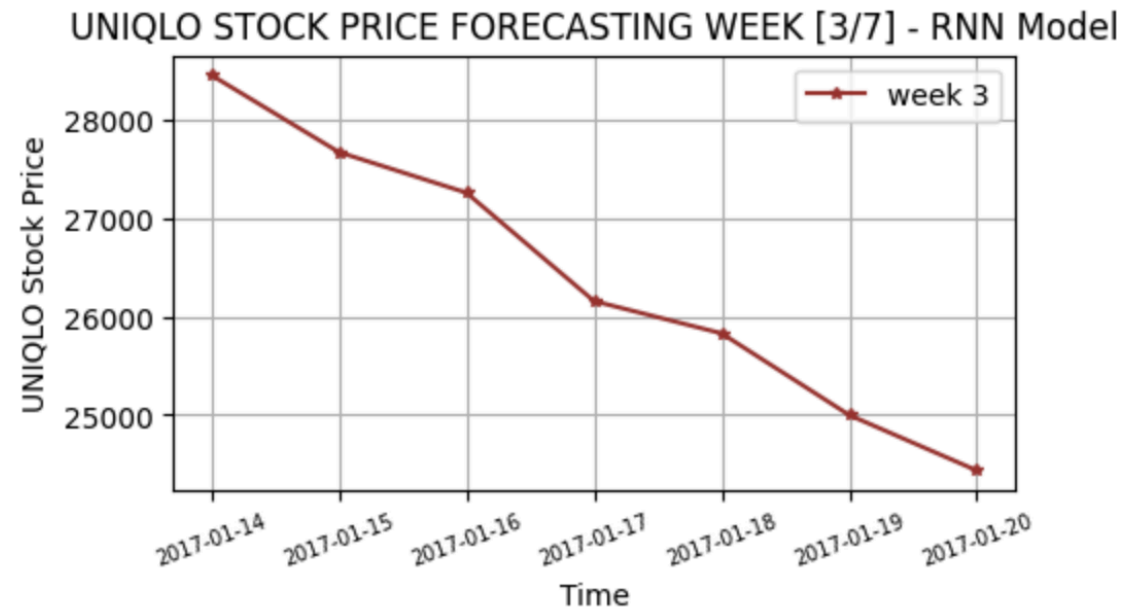
1.2. Next 7 weeks forecasting:

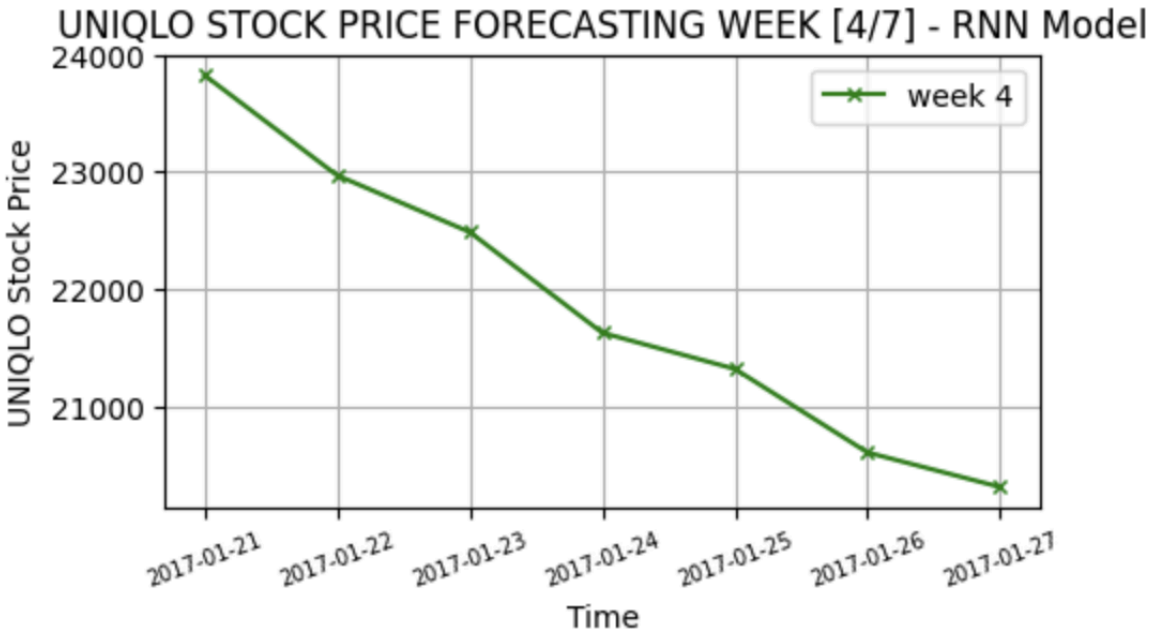


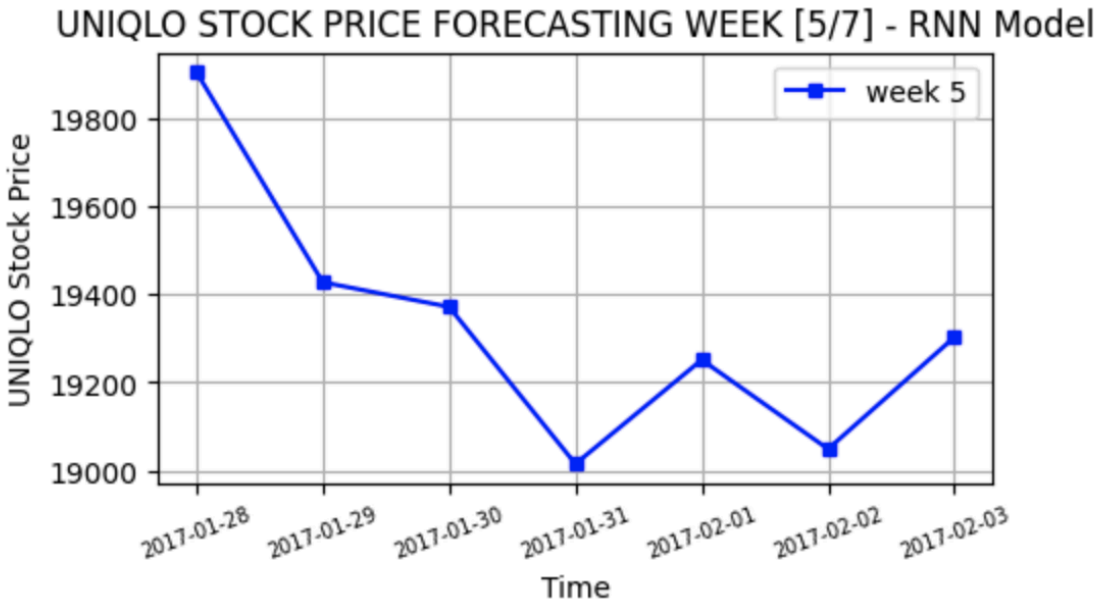

UNIQLO STOCK PRICE NEXT 7 WEEKS FORECASTING - RNN Model
UNIQLO Stock Price
35000
32500
30000
27500
25000
22500
20000
2017-01-01
2017-01-08
2017-01-15
2017-01-22
2017-02-01
2017-02-08
2017-02-15
Time
week 1
week 2
week 3
week 4
week 5
week 6
week 7

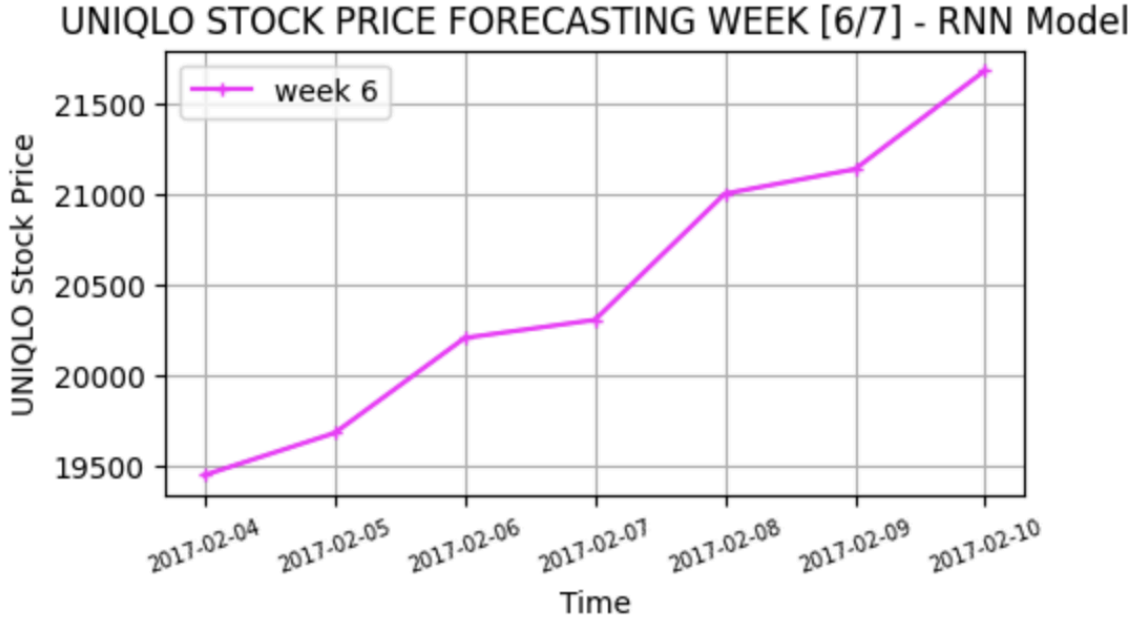

UNIQLO STOCK PRICE FORECASTING WEEK [1/7] - RNN Model
UNIQLO Stock Price
36000
35000
34000
33000
2016-12-31
2017-01-01
2017-01-02
2017-01-03
2017-01-04
2017-01-05
2017-01-06
Time
week 1

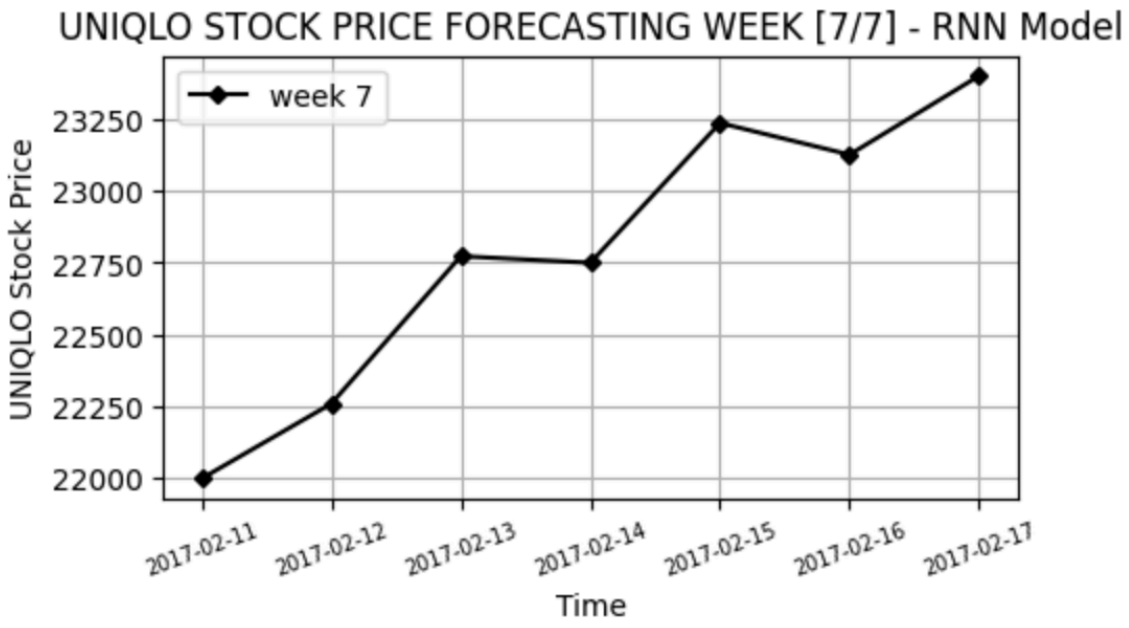

UNIQLO STOCK PRICE FORECASTING WEEK [2/7] - RNN Model
UNIQLO Stock Price
32000
31000
30000
29000
2017-01-07
2017-01-08
2017-01-09
2017-01-10
2017-01-11
2017-01-12
2017-01-13
Time
week 2


UNIQLO STOCK PRICE FORECASTING WEEK [3/7] - RNN Model
week 3
UNIQLO Stock Price
28000
27000
26000
25000
2017-01-14
2017-01-15
2017-01-16
2017-01-17
2017-01-18
2017-01-19
2017-01-20
Time


UNIQLO STOCK PRICE FORECASTING WEEK [4/7] - RNN Model
week 4
UNIQLO Stock Price
24000
23000
22000
21000
2017-01-21
2017-01-22
2017-01-23
2017-01-24
2017-01-25
2017-01-26
2017-01-27
Time

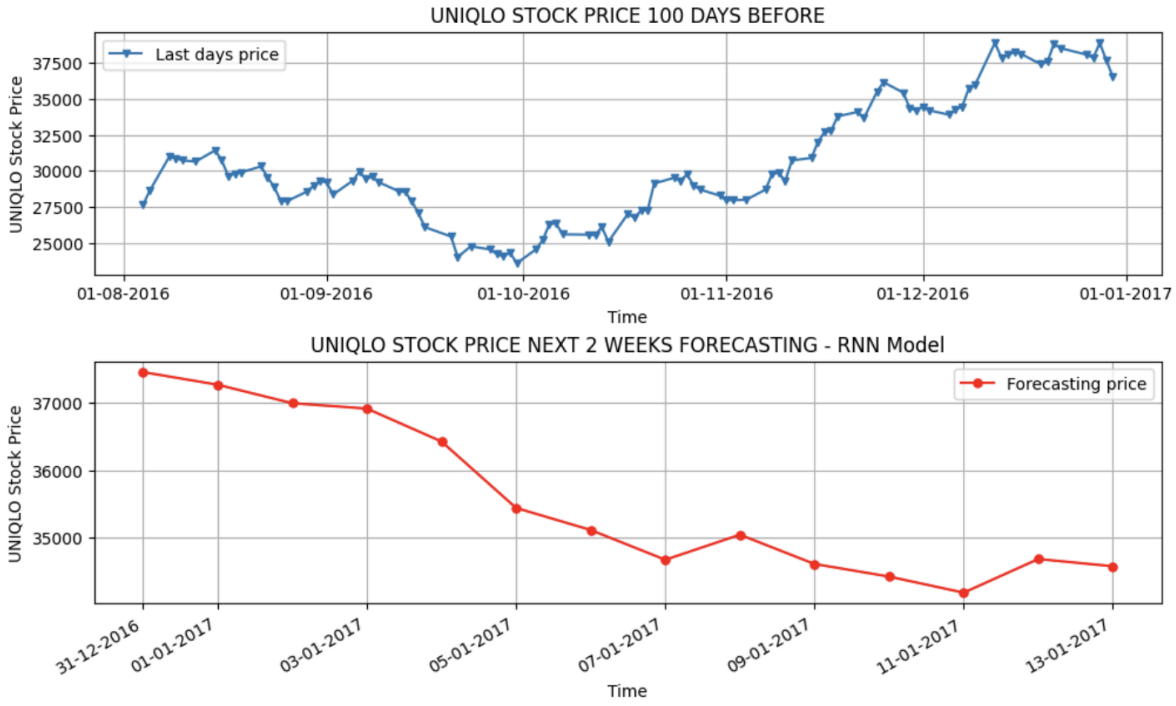

UNIQLO STOCK PRICE FORECASTING WEEK [5/7] - RNN Model
week 5
UNIQLO Stock Price
19800
19600
19400
19200
19000
2017-01-28
2017-01-29
2017-01-30
2017-01-31
2017-02-01
2017-02-02
2017-02-03
Time


UNIQLO STOCK PRICE FORECASTING WEEK [6/7] - RNN Model
week 6
UNIQLO Stock Price
21500
21000
20500
20000
19500
2017-02-04
2017-02-05
2017-02-06
2017-02-07
2017-02-08
2017-02-09
2017-02-10
Time



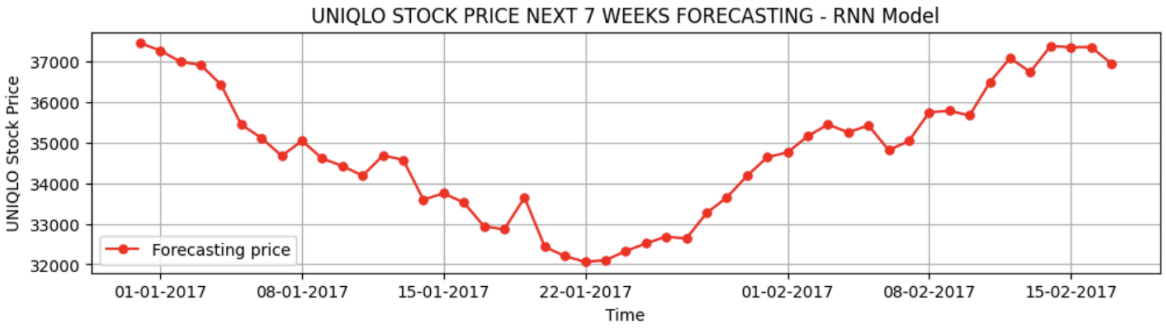
2. MLP model

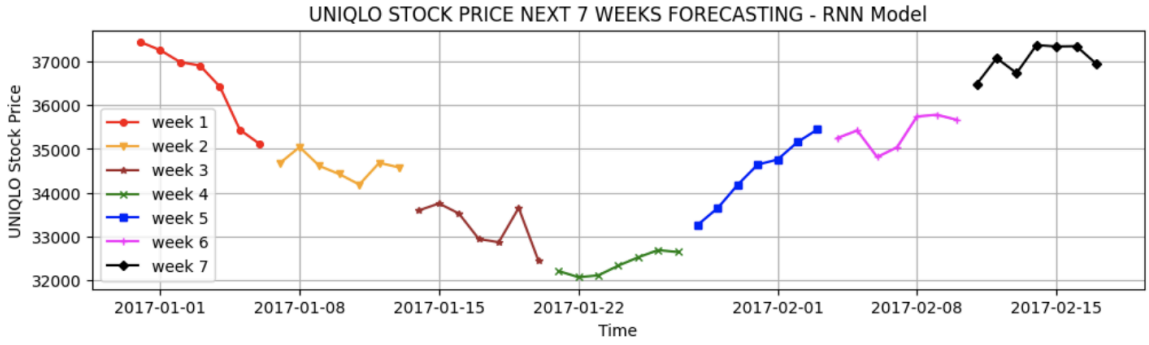
1.1. Next 2 weeks forecasting:

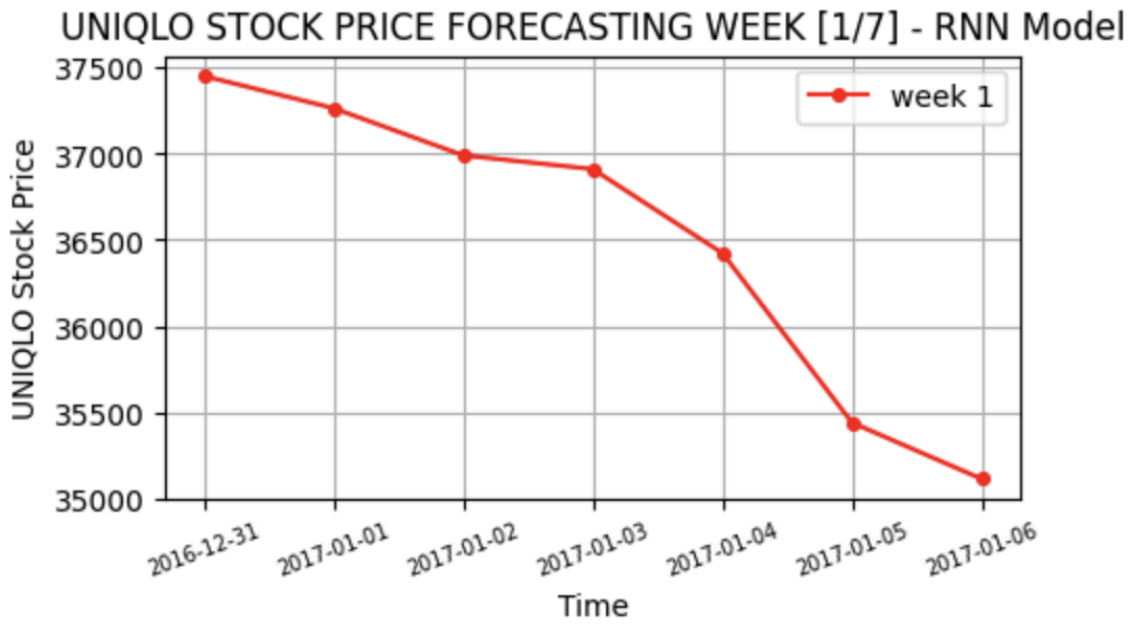


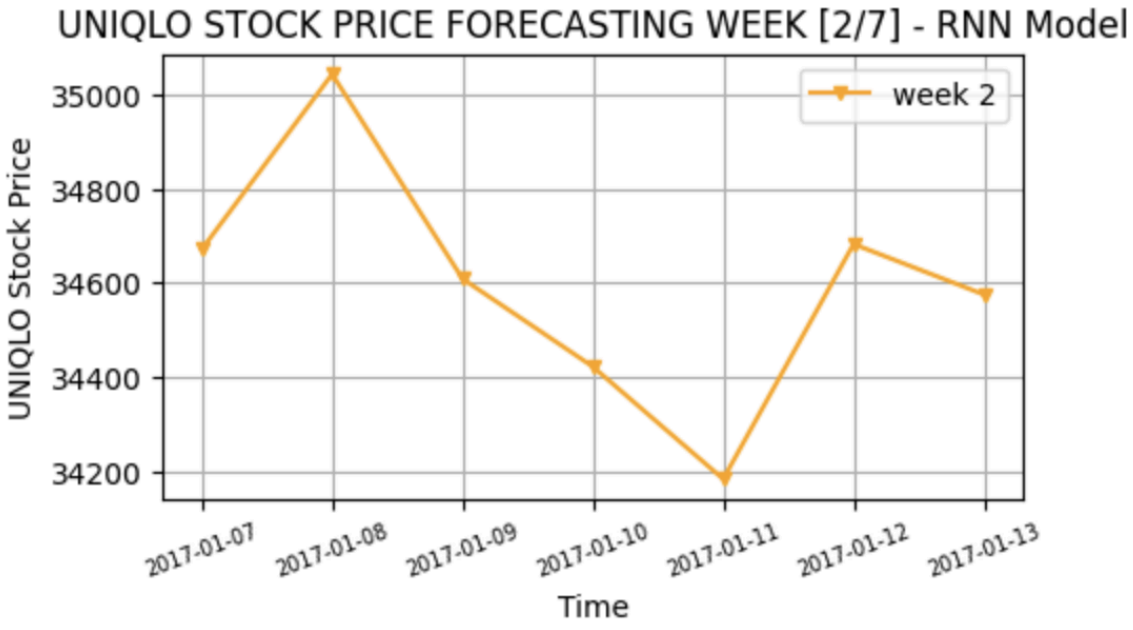
1.2. Next 7 weeks forecasting:

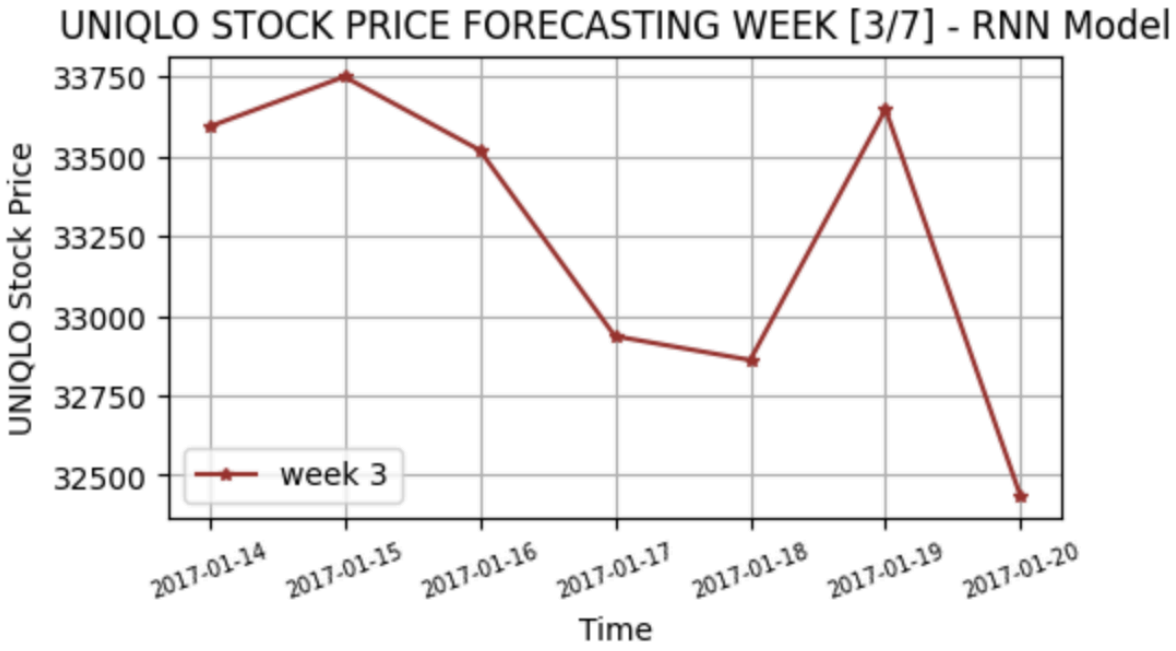

UNIQLO STOCK PRICE 100 DAYS BEFORE
Last days price
UNIQLO Stock Price
37500
35000
32500
30000
27500
25000
01-08-2016
01-09-2016
01-10-2016
01-11-2016
01-12-2016
01-01-2017
Time

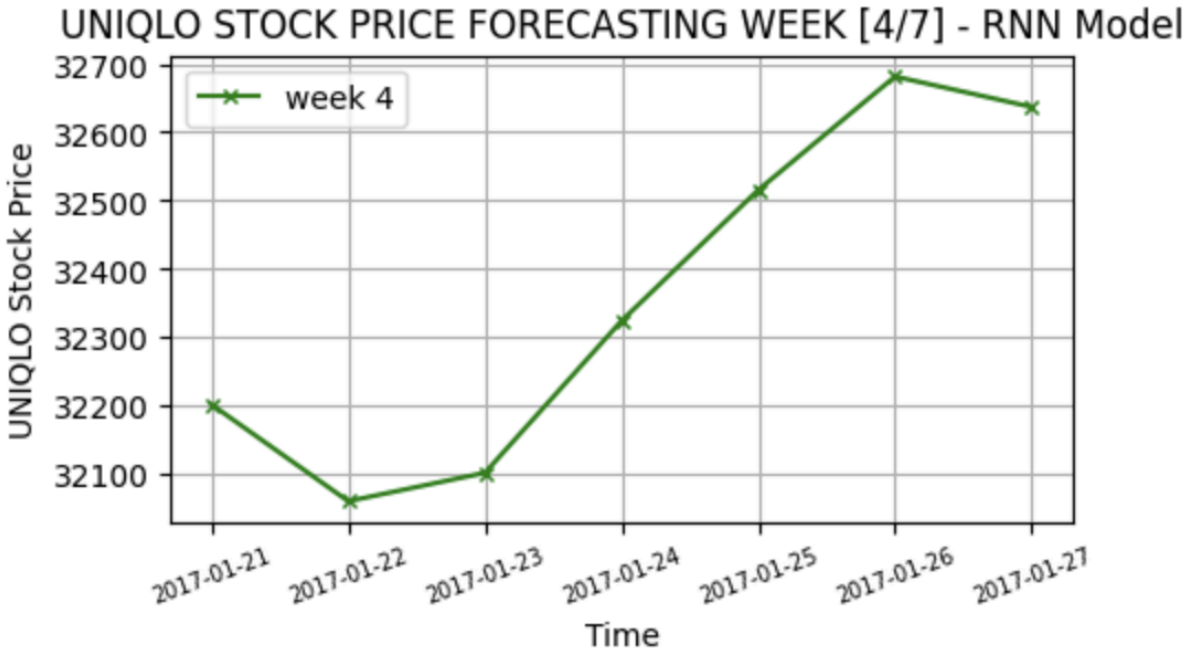

UNIQLO STOCK PRICE NEXT 7 WEEKS FORECASTING - RNN Model
Forecasting price
UNIQLO Stock Price
37000
36000
35000
34000
33000
32000
01-01-2017
08-01-2017
15-01-2017
22-01-2017
01-02-2017
08-02-2017
15-02-2017
Time

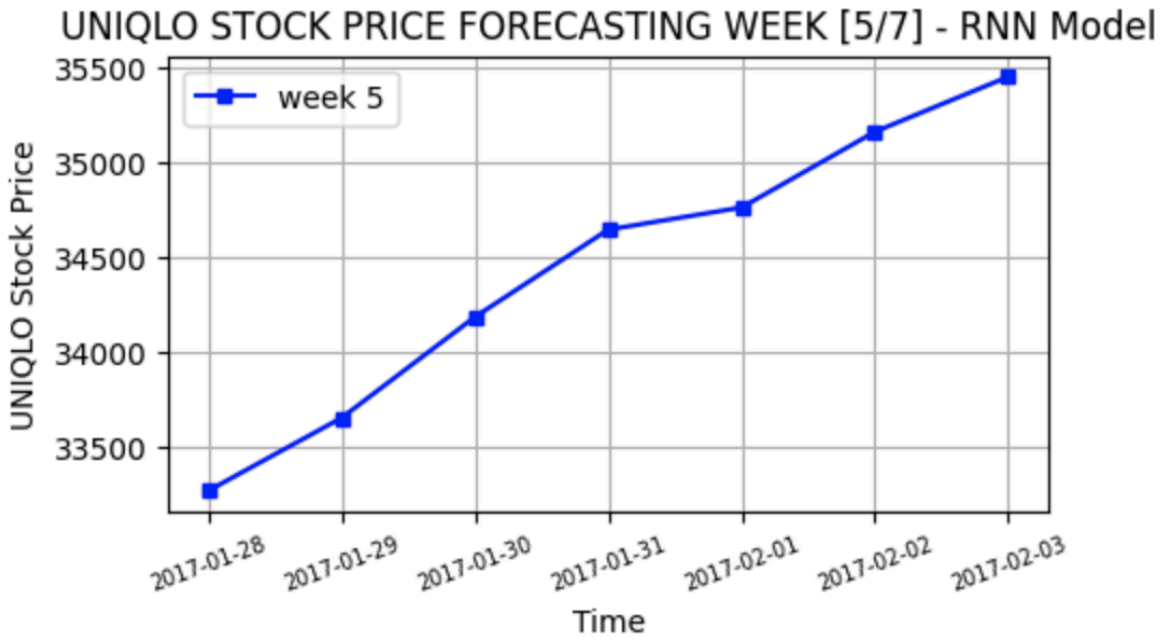

UNIQLO STOCK PRICE NEXT 7 WEEKS FORECASTING - RNN Model
week 1
week 2
week 3
week 4
week 5
week 6
week 7
UNIQLO Stock Price
37000
36000
35000
34000
33000
32000
2017-01-01
2017-01-08
2017-01-15
2017-01-22
2017-02-01
2017-02-08
2017-02-15
Time

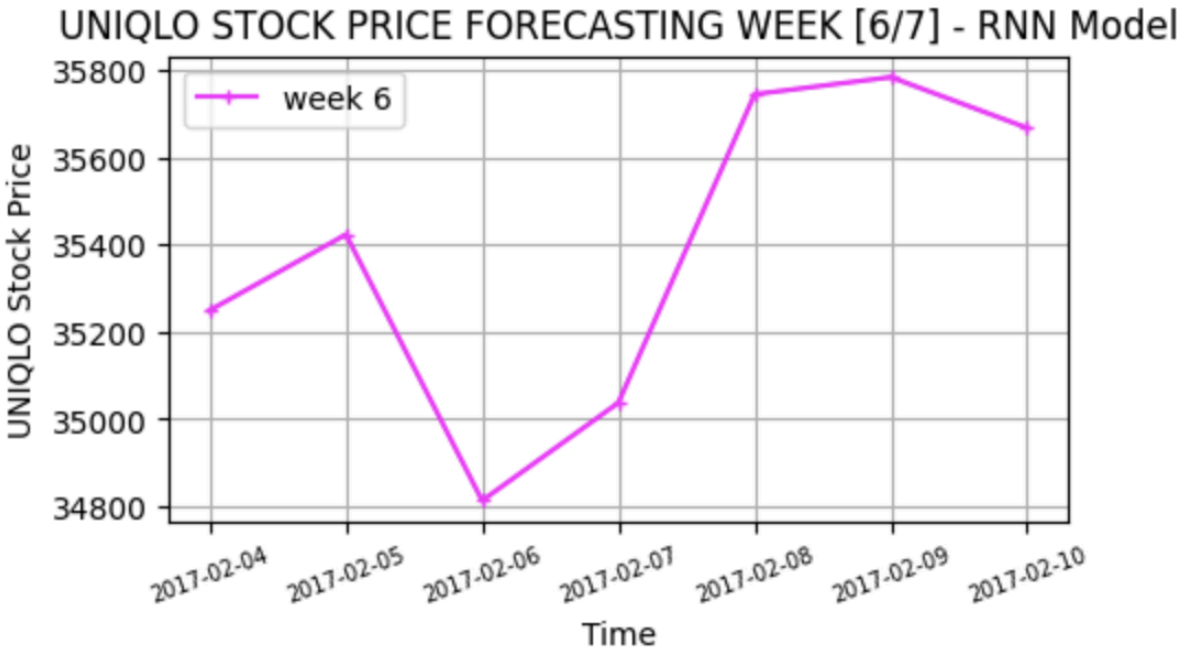

UNIQLO STOCK PRICE FORECASTING WEEK [1/7] - RNN Model
week 1
UNIQLO Stock Price
37500
37000
36500
36000
35500
35000
2016-12-31
2017-01-01
2017-01-02
2017-01-03
2017-01-04
2017-01-05
2017-01-06
Time

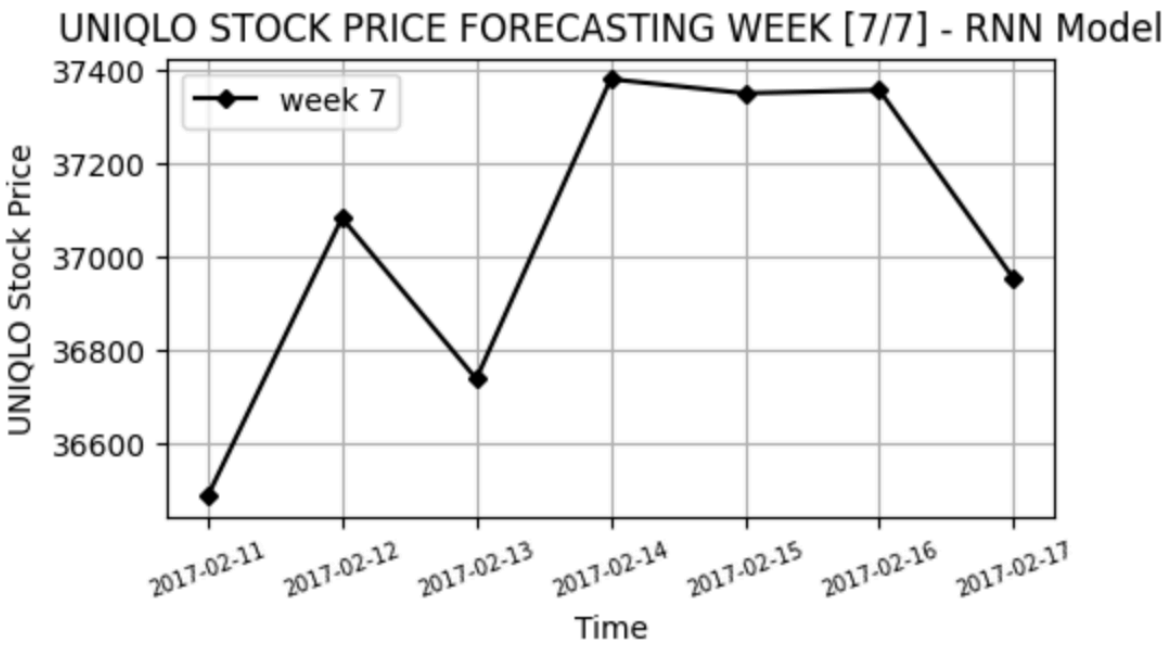

UNIQLO STOCK PRICE FORECASTING WEEK [2/7] - RNN Model
week 2
UNIQLO Stock Price
35000
34800
34600
34400
34200
2017-01-07
2017-01-08
2017-01-09
2017-01-10
2017-01-11
2017-01-12
2017-01-13
Time


UNIQLO STOCK PRICE FORECASTING WEEK [3/7] - RNN Model
UNIQLO Stock Price
33750
33500
33250
33000
32750
32500
week 3
2017-01-14
2017-01-15
2017-01-16
2017-01-17
2017-01-18
2017-01-19
2017-01-20
Time


UNIQLO STOCK PRICE FORECASTING WEEK [4/7] - RNN Model
UNIQLO Stock Price
32700
32600
32500
32400
32300
32200
32100
week 4
2017-01-21
2017-01-22
2017-01-23
2017-01-24
2017-01-25
2017-01-26
2017-01-27
Time


UNIQLO STOCK PRICE FORECASTING WEEK [5/7] - RNN Model
week 5
UNIQLO Stock Price
35500
35000
34500
34000
33500
2017-01-28
2017-01-29
2017-01-30
2017-01-31
2017-02-01
2017-02-02
2017-02-03
Time


UNIQLO STOCK PRICE FORECASTING WEEK [6/7] - RNN Model
week 6
UNIQLO Stock Price
35800
35600
35400
35200
35000
34800
2017-02-04
2017-02-05
2017-02-06
2017-02-07
2017-02-08
2017-02-09
2017-02-10
Time


UNIQLO STOCK PRICE FORECASTING WEEK [7/7] - RNN Model
week 7
UNIQLO Stock Price
37400
37200
37000
36800
36600
2017-02-11
2017-02-12
2017-02-13
2017-02-14
2017-02-15
2017-02-16
2017-02-17
Time

PROBLEM 6

I. Long Short-Term Memory (LSTM)

1. What Does Long Short-Term Memory (LSTM) Mean?

A recurrent neural network structure includes long short-term memory (LSTM) units or blocks. Recurrent neural networks are designed to use specific types of artificial memory processes that can assist these artificial intelligence programs in better imitating human thought.

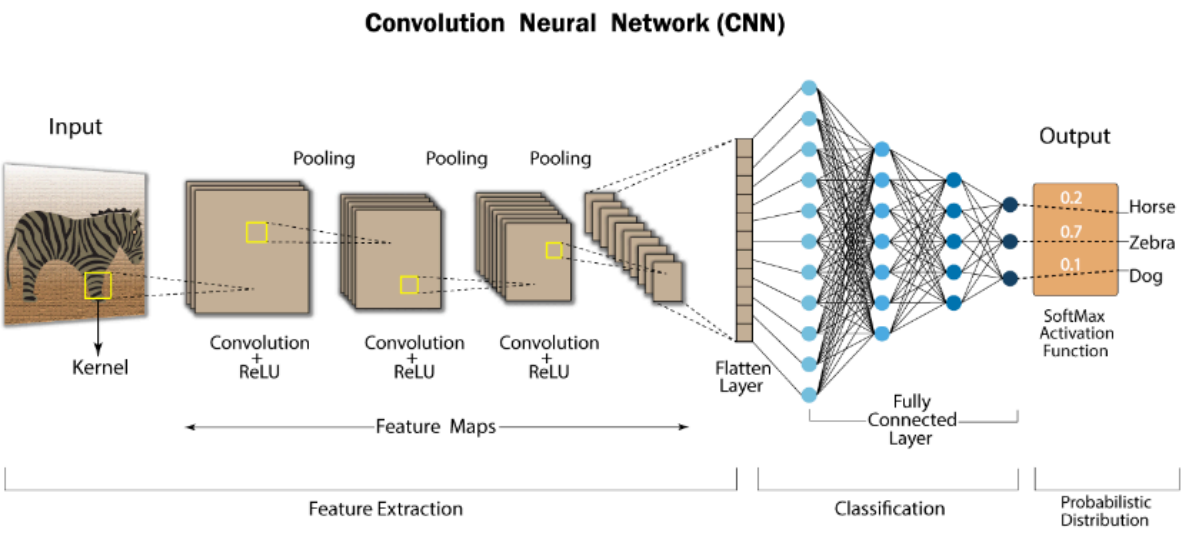
2. Explains Long Short-Term Memory (LSTM)

- Long short-term memory blocks are used by the recurrent neural network to provide context for how the program receives inputs and generates outputs. The long short-term memory block is a complex unit that includes weighted inputs, activation functions, inputs from previous blocks, and eventual outputs.
- Because the program uses a structure based on short-term memory processes to create longer-term memory, the unit is known as a long short-term memory block. These systems are frequently used in natural language processing, for example. Long short-term memory blocks are used by the recurrent neural network to evaluate a specific word or phoneme in the context of others in a string, where memory can be useful in sorting and categorizing these types of inputs. In general, LSTM is a well-known and widely used concept in pioneering recurrent neural networks.

II. Convolutional Neural Network (CNN)

1. What is Convolutional Neural Network?

CNNs, also known as Convolutional Neural Networks, are a type of feed-forward artificial neural network whose connectivity structure is inspired by the organization of the animal visual cortex. Certain areas of the visual field are sensitive to small clusters of cells in the visual cortex. Individual neuronal cells in the brain only respond or fire when certain edge orientations are present. When shown vertical edges, some neurons fire, while others fire when shown horizontal or diagonal edges. Convolutional neural networks are artificial neural networks that are used in deep learning to evaluate visual data. These networks are capable of handling a wide variety of tasks that involve images, sounds, texts, videos, and other media. In the late 1990s, Bell Labs Professor Yann LeCunn developed the first successful convolution networks.



- Convolutional Neural Networks (CNNs) can learn complicated objects and patterns because they have an input layer, an output layer, numerous hidden layers, and millions of parameters. It subsamples the given input using convolution and pooling processes before applying an activation function, where all of them are hidden layers that are partially connected, with the fully connected layer at the end resulting in the output layer. The size of the output image is similar to the size of the input image.
- The process of combining two functions to produce the output of the other function is known as convolution. The input image is convoluted by the use of filters in CNNs, yielding a Feature map. Filters are weights and biases that are generated at random in the network. CNN uses the same weights and biases for all neurons rather than having individual weights and biases for each neuron. Many filters can be created, each capturing a different aspect of the input. Filters are also known as kernels.

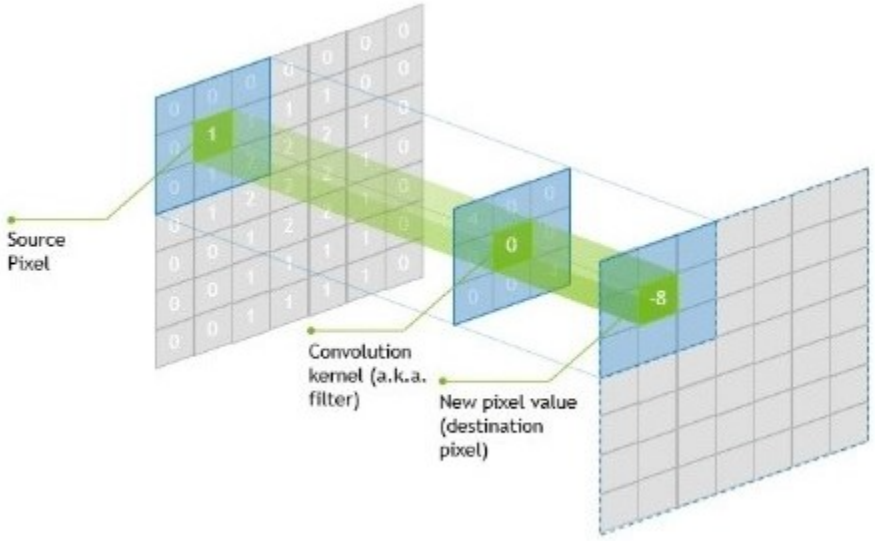
2. Convolutional Layer

- Convolutional layers are the primary building blocks in convolutional neural networks. This layer frequently contains input vectors like an image, filters like a feature detector, and output vectors like a feature map. After passing through a convolutional layer, the image is abstracted to a feature map, also known as an activation map.

Feature Map = Input Image x Feature Detector

- Convolutional layers convolve the input and pass the output to the next layer. In the visual cortex, this is analogous to a neuron's response to a single stimulus. Each convolutional neural processes data only for the receptive field to which it is assigned.
- In mathematics, a convolution is a grouping function. In CNNs, convolution occurs when two matrices (rectangular arrays of numbers arranged in columns and rows) are combined to form a third matrix.

- These convolutions are used to filter input data and find information in the convolutional layers of a CNN.



- The center element of the kernel is placed above the source pixel. The source pixel is then replaced with a weighted sum of its neighboring pixels.
- CNNs employ two principles: parameter sharing and local connectivity. Parameter sharing occurs when all neurons in a feature map share weights. The term "local connection" refers to the idea that each neuron is connected to only a portion of the input image (as opposed to a neural network in which all neurons are fully connected). This reduces the number of system parameters and speeds up the calculation.

3. Padding and Stride

- Padding and stride both influence how the convolution procedure is performed. Padding and stride can be used to increase or decrease the dimensions of input/output vectors (height and width). Padding is a term used in convolutional neural networks to describe how many pixels are added to an image during the CNN kernel processing. If the padding in a CNN is set to zero, for example, the value added to each pixel will be zero. If the zero padding is set to one, the image will have a one-pixel border with a pixel value of zero.

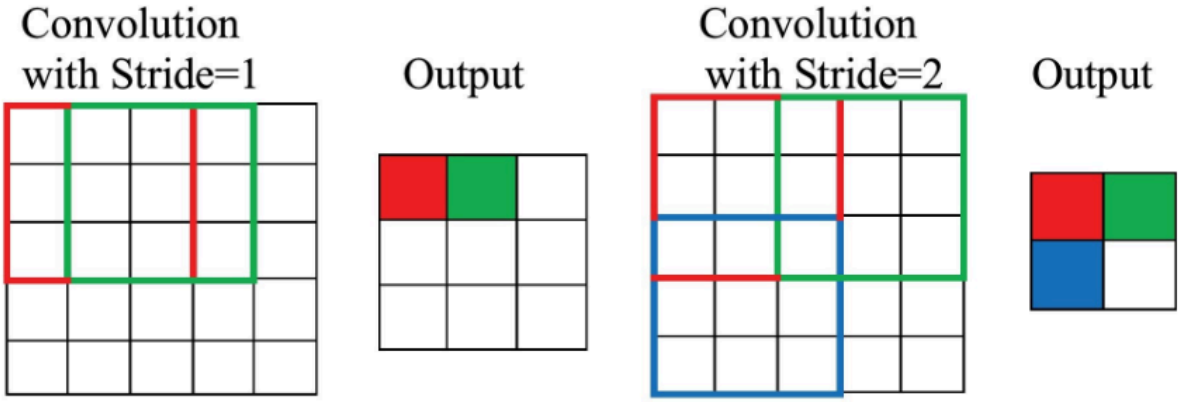
0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114	328	-26	470	158
53	266	-61	-30	344
403	116	-47	295	244
108	-135	256	-128	344
314	346	279	153	421

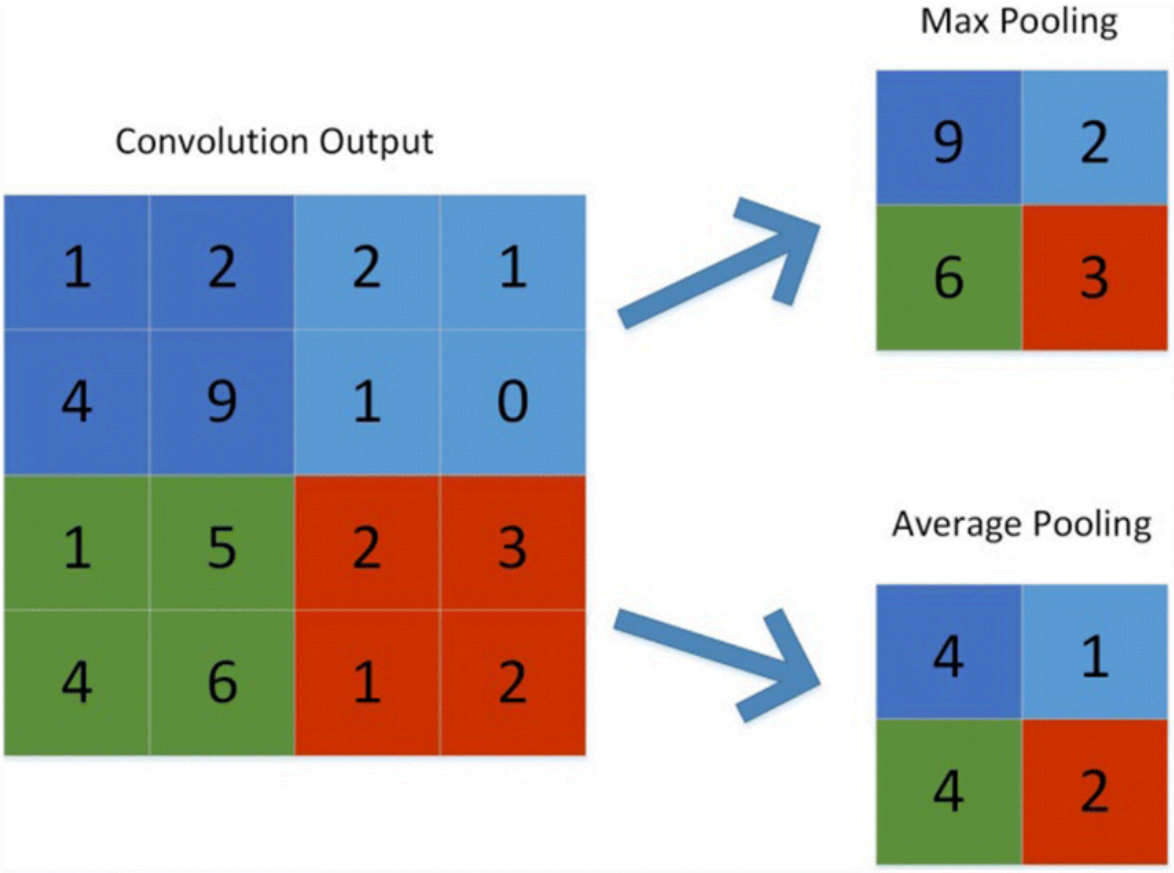
- Padding works by expanding a convolutional neural network's processing region. The kernel is a neural network filter that scans each pixel in a picture and converts the data into a smaller or larger format. Padding is added to the image frame to help the kernel process the image by giving the kernel more room to cover the image. Padding a CNN-processed image allows for more accurate image analysis.



- The stride of the filter determines how many pixels shift as it convolves over the input matrix. When stride is set to 1, the filter moves across one pixel at a time, while stride 2 moves across two pixels at a time. The output is proportional to the stride value, and vice versa.

4. Pooling

- Its purpose is to gradually shrink the representation's spatial size to reduce the number of parameters and computations in the network. The pooling layer treats each feature map separately.



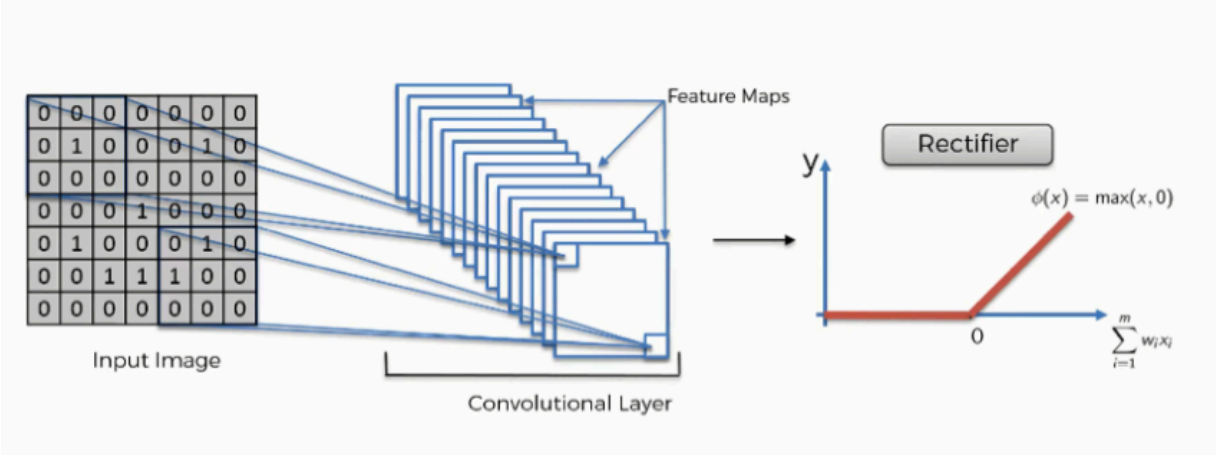
The following are some methods for pooling:

- **Max-pooling:**It selects the most important element from the feature map. The significant features of the feature map are stored in the resulting max-pooled layer. It is the most widely used method because it yields the best results.
- **Average pooling:** It entails calculating the average for each region of the feature map.

Pooling reduces the spatial dimension of the representation gradually, reducing the number of parameters and computations in the network and preventing overfitting. If no pooling is used, the output.

5. **ReLU**

- The rectified linear activation function, or ReLU for short, is a piecewise linear function that outputs the input directly if the input is positive; otherwise, it outputs zero. It has become the default activation function for many types of neural networks because it is faster to train and generally produces better results.



- There is a Fully connected layer of neurons at the end of CNN. Neurons in a fully connected layer, like those in a conventional Neural Network, have full connections to all activations in the previous layer and function similarly. Following training, the fully connected layer's feature vector is used to classify images into distinct categories. Every activation unit in the next layer is coupled to all of this layer's inputs. Because all of the parameters are occupied in the fully-connected layer, overfitting occurs. A variety of strategies, including dropout, can be used too.
- Soft-max is an activation layer that is typically applied to the network's last layer, which serves as a classifier. This layer is responsible for categorizing provided input into distinct types. A network's non-normalized output is mapped to a probability distribution using the softmax function.

REFERENCES

RNN: https://www.ibm.com/cloud/learn/recurrent-neural-networks

https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron

MLP:https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron