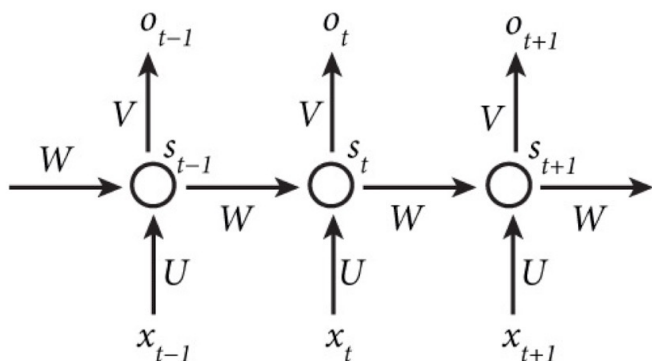


Recurrent Neural Network (RNN)

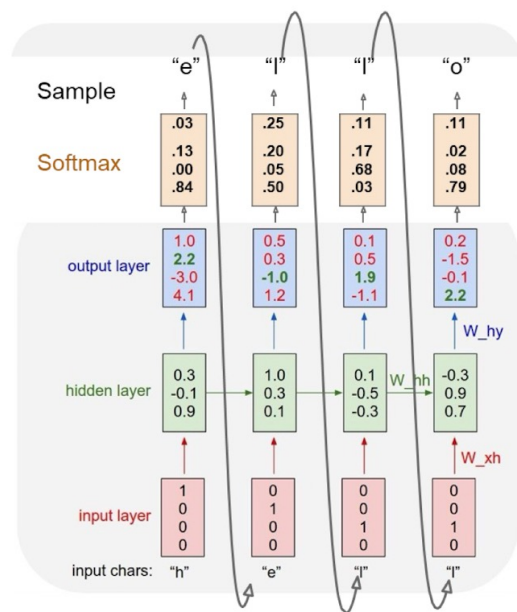
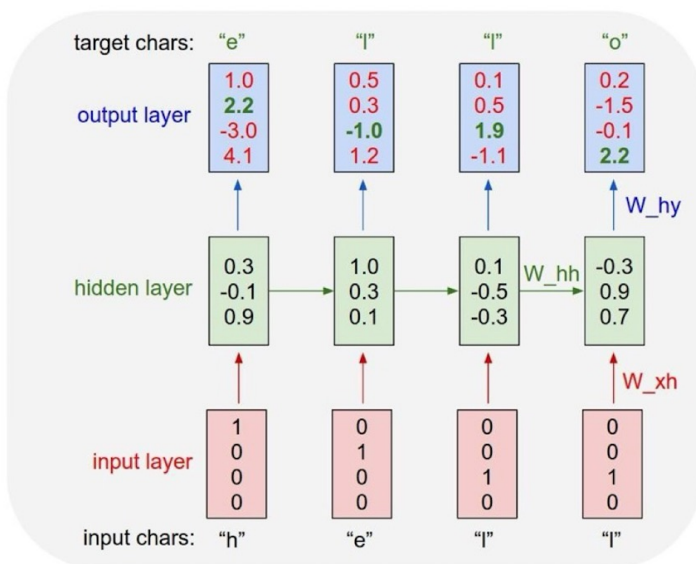
2-2

RNN 기본구조



- Parameters: W, V, U
- Inputs: x_t
- Hidden State: $s_t = f(Ux_t + Ws_{t-1})$
- Outputs: $o_t = \text{softmax}(Vs_t)$

RNN 예시: Character-level Language Model



- RNN은 순차적인(sequential) 데이터를 활용하는 머신러닝 기법이다
- 순차적으로 의존적인 여러 NN layer를 활용해 Input 값과 가중치 parameter를 기반으로 예측하고 싶은 Output 값을 산출한다
- 지도학습을 통해 Input과 Output이 주어졌을 때 평균오차를 최소화하는 가중치 parameter를 학습시키고 학습된 parameter를 테스트하는 과정을 반복
- RNN의 활용 예시로 [h, e, l, o] 가 주어졌을 때 "hello"를 출력하는 Language Model의 일종이 있다

Recurrent Neural Network (RNN)

Implementation: Python (keras, sklearn)

```
StockPrediction_RNN.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
import yfinance as yf

tf.random.set_seed(777) # for reproducibility

[42] # Load data: 3mo
xy = yf.download('AAPL', start = '2023-09-01', end = '2023-11-23')
[*****100%*****] 1 of 1 completed

[53] # Load data: 1yr
xy = yf.download('AAPL', start = '2022-01-01', end = '2023-11-23')
[*****100%*****] 1 of 1 completed

[] # Load data: 5yr
xy = yf.download('AAPL', start = '2019-01-01', end = '2023-11-23')

[54] # using Open, High, Low, Close, Volume
xy = xy[['Open', 'High', 'Low', 'Volume', 'Close']]
# visualizing data
xy.head(10)

Date      Open      High      Low      Volume      Close
2022-01-03  177.830002  182.880005  177.710007  104487900   182.009995
2022-01-04  182.630005  182.940002  179.119995  99310400   179.699997
2022-01-05  179.610001  180.169998  174.639999  94537400   174.919998
2022-01-06  172.699997  175.300003  171.639999  96904000   172.000000
2022-01-07  172.889999  174.139999  171.029999  86709100   172.169998
2022-01-10  169.080002  172.500000  168.169998  106765000  172.190002
2022-01-11  172.320007  175.179993  170.820007  76138300   175.080002
2022-01-12  176.119995  177.179993  174.820007  74805200   175.529999
2022-01-13  176.779999  176.619995  171.789993  84505800   172.190002
2022-01-14  171.339996  173.779999  171.089996  80440800   173.070007

[55] # RNN parameters
input_dim = 5
hidden_size = 10
output_dim = 1
seq_length = 7
learning_rate = 0.01
iterations = 100

[56] # xy to numpy
xy = xy.to_numpy()

# Use scikit-learn's MinMaxScaler for normalization
scaler = MinMaxScaler(feature_range=(0, 1))
xy = scaler.fit_transform(xy)

x = xy
y = xy[:, -1:] # just Close col
```

```
StockPrediction_RNN.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

[57] # Build a dataset
dataX = []
dataY = []
for i in range(0, len(y) - seq_length):
    _x = x[i:i + seq_length]
    _y = y[i + seq_length] # Next close price
    dataX.append(_x)
    dataY.append(_y)

[58] # Split dataset to train and test
train_size = int(len(dataY) * 0.7)
test_size = len(dataY) - train_size
trainX, testX = np.array(dataX[0:train_size]), np.array(dataX[train_size:len(dataX)])
trainY, testY = np.array(dataY[0:train_size]), np.array(dataY[train_size:len(dataY)])

[59] # Create LSTM model
model = Sequential()
model.add(LSTM(hidden_size, input_shape=(seq_length, input_dim)))
model.add(Dense(output_dim))

[60] model.compile(loss='mean_squared_error', optimizer=tf.keras.optimizers.Adam(learning_rate))

[61] # Train
model.fit(trainX, trainY, epochs=iterations, verbose=1)

11/11 [*****] - 0s 12ms/step - loss: 0.0020
Epoch 73/100
11/11 [*****] - 0s 9ms/step - loss: 0.0023
Epoch 74/100
11/11 [*****] - 0s 13ms/step - loss: 0.0020
Epoch 75/100
11/11 [*****] - 0s 11ms/step - loss: 0.0021
Epoch 76/100
11/11 [*****] - 0s 9ms/step - loss: 0.0020
Epoch 77/100
11/11 [*****] - 0s 8ms/step - loss: 0.0020
Epoch 78/100
11/11 [*****] - 0s 8ms/step - loss: 0.0020
Epoch 79/100
11/11 [*****] - 0s 9ms/step - loss: 0.0021
Epoch 80/100
11/11 [*****] - 0s 13ms/step - loss: 0.0020
Epoch 81/100
11/11 [*****] - 0s 11ms/step - loss: 0.0020
Epoch 82/100
11/11 [*****] - 0s 12ms/step - loss: 0.0021
Epoch 83/100
11/11 [*****] - 0s 9ms/step - loss: 0.0020
Epoch 84/100
11/11 [*****] - 0s 9ms/step - loss: 0.0020
Epoch 85/100
11/11 [*****] - 0s 9ms/step - loss: 0.0021
Epoch 86/100
11/11 [*****] - 0s 10ms/step - loss: 0.0020
Epoch 87/100
11/11 [*****] - 0s 12ms/step - loss: 0.0021
Epoch 88/100
11/11 [*****] - 0s 13ms/step - loss: 0.0021
Epoch 89/100
11/11 [*****] - 0s 18ms/step - loss: 0.0020
Epoch 90/100
11/11 [*****] - 0s 15ms/step - loss: 0.0020
Epoch 91/100
11/11 [*****] - 0s 17ms/step - loss: 0.0020
Epoch 92/100
11/11 [*****] - 0s 20ms/step - loss: 0.0020
Epoch 93/100
11/11 [*****] - 0s 18ms/step - loss: 0.0020
Epoch 94/100
11/11 [*****] - 0s 11ms/step - loss: 0.0021
Epoch 95/100
11/11 [*****] - 0s 10ms/step - loss: 0.0020
Epoch 96/100
11/11 [*****] - 0s 13ms/step - loss: 0.0020
```

```
StockPrediction_RNN.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

testY = np.array(testY)

# Calculate the absolute differences
absolute_differences = np.abs(results - testY)

# Compute the mean of these differences
mean_absolute_error = np.mean(absolute_differences)
print("Mean Absolute Error:", mean_absolute_error)

import matplotlib.pyplot as plt
plt.figure(figsize=(19, 6))
plt.plot(results, label='Predicted')
plt.plot(testY, label='Ground Truth')
plt.legend()
plt.title('Comparison of Predictions and Ground Truth')
plt.xlabel('Sample Index')
plt.ylabel('Value')
plt.show()

Predicted: [0.7455273] GT: [0.75178583]
Predicted: [0.7503839] GT: [0.72988536]
Predicted: [0.72089997] GT: [0.71146576]
Predicted: [0.717084] GT: [0.70514682]
Predicted: [0.7112583] GT: [0.67082671]
Predicted: [0.66913366] GT: [0.67170661]
Predicted: [0.67415551] GT: [0.67786532]
Predicted: [0.67518854] GT: [0.64518723]
Predicted: [0.6483115] GT: [0.58616831]
Predicted: [0.58633816] GT: [0.60478797]
Predicted: [0.6252728] GT: [0.63370727]
Predicted: [0.6433812] GT: [0.64048729]
Predicted: [0.63494337] GT: [0.68528636]
Predicted: [0.68318737] GT: [0.7358543]
Predicted: [0.7357137] GT: [0.7228855]
Predicted: [0.7098693] GT: [0.75892481]
Predicted: [0.7569587] GT: [0.70518424]
Predicted: [0.7836788] GT: [0.81016383]
Predicted: [0.7968986] GT: [0.80344483]
Predicted: [0.7804723] GT: [0.85930277]
Predicted: [0.8421828] GT: [0.83698335]
Predicted: [0.8166228] GT: [0.8738626]
Predicted: [0.85688295] GT: [0.88184233]
Predicted: [0.86788524] GT: [0.98564282]
Predicted: [0.88316613] GT: [0.98536197]
Predicted: [0.8771848] GT: [0.9388814]
Predicted: [0.98688784] GT: [0.91866166]
Predicted: [0.88566613] GT: [0.92884145]
Mean Absolute Error: 0.02650213578958414

Comparison of Predictions and Ground Truth
Value
Sample Index
— Predicted
— Ground Truth
```

Source: AIC2120, YIG, colab link: https://drive.google.com/file/d/108BnEhPXHNqVEARk_H_Bz2tnQeg1W3P0/view?usp=sharing

Result analysis: Region effect

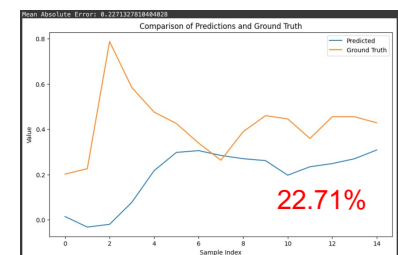
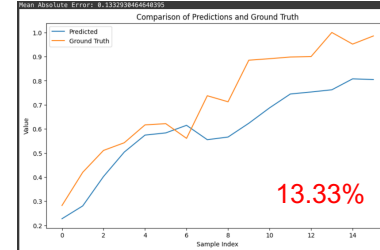
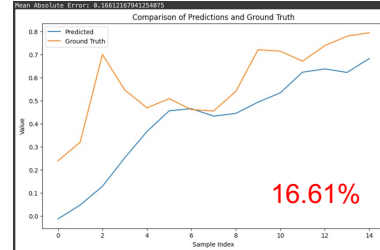
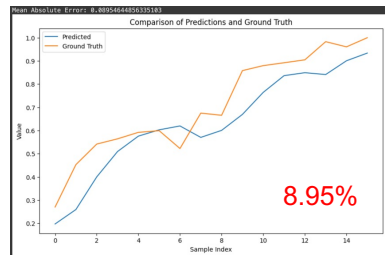
S&P500 (SPY)

KOSPI 200
(KODEX 200)

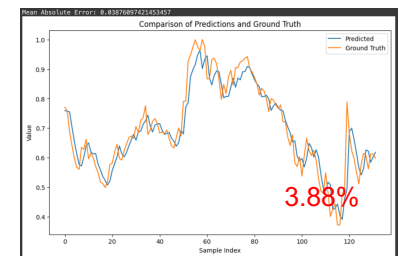
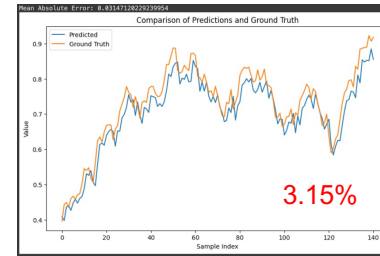
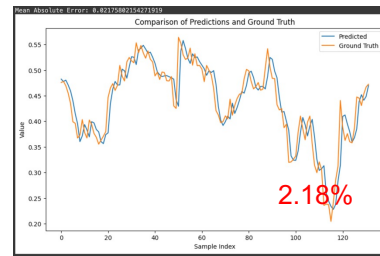
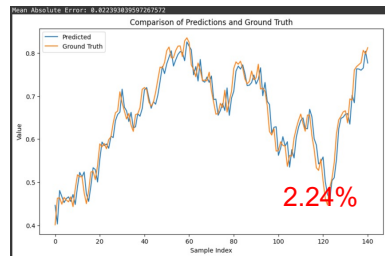
NASDAQ 100 (QQQ)

KOSDAQ 150
(TIGER 코스닥 150)

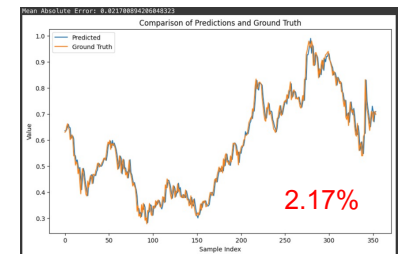
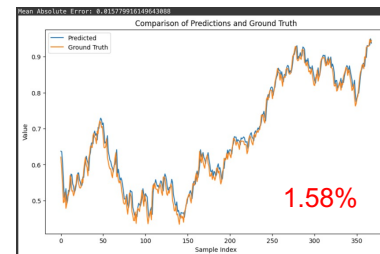
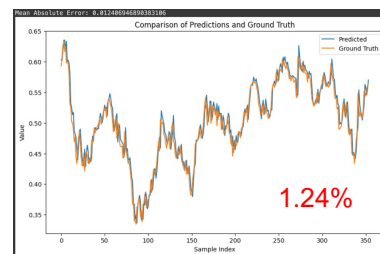
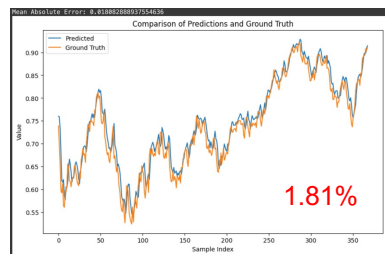
3mo



1yr



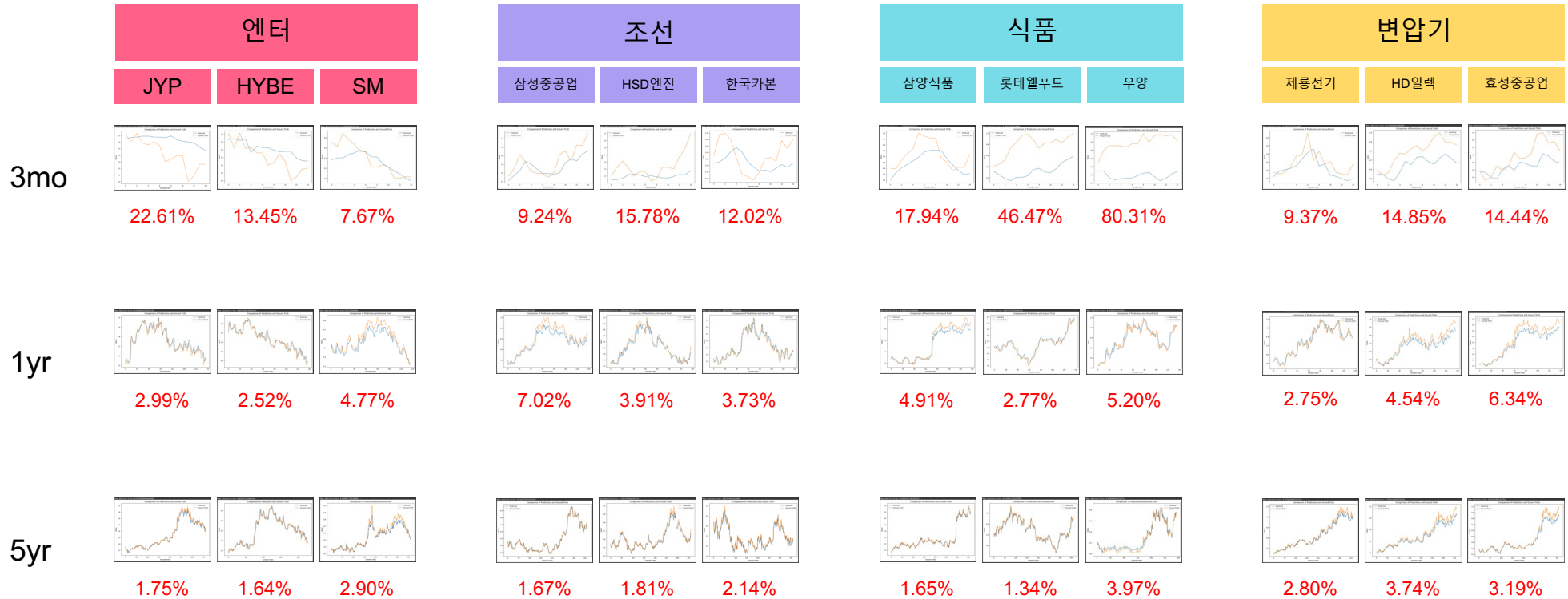
5yr



- RNN 기법을 활용한 주가 예측의 정확도가 지역의 영향을 받는지 비교를 위해 미국과 한국 주식시장을 비교
- 1년치보다 많은 데이터를 사용할 시 지역의 영향은 미미하지만 단기 데이터를 사용할 경우, 한국시장에 대한 정확도가 상대적으로 많이 감소

Source: YIG

Result analysis: Industry effect



- RNN 기법을 활용한 주가 예측의 정확도가 산업의 영향을 받는지 비교를 위해 엔터, 조선, 식품, 변압기 산업을 비교
- 같은 산업 내 정확도 수치 간 유의한 연관성 부족
- 변동성이 더 심한 개별주식 데이터를 다루기에 충분히 오차를 줄이기 위해 5년치 이상의 데이터 필요
- 결론: 결국 온전히 수치적인 접근이기에 지역, 산업 등을 떠나 모델 features와 데이터 기간에 따라 정확도 변화