13. 발전된 RNN 모델

2강. 시계열 데이터를 예측하는 GRU 구현

학습목표

- 시계열 데이터를 예측하는 GRU를 구현할 수 있다.

학습내용

- 시계열 데이터를 예측하는 GRU 구현

1. 시계열 데이터를 예측하는 GRU 구현

(1) 시계열 데이터를 예측하는 LSTM 구현

- Gradient Vanishing 문제를 해결하여 Long-Term Dependency 학습 가능한 LSTM 구현

① LSTM 구현에 필요한 라이브러리 임포트

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense, Dropout, GRU
from keras.optimizers import SGD
import math

plt.style.use('fivethirtyeight')
```

② 시각화와 최적화 등 도움이 되는 기능들 구현

```
def plot_predictions(test,predicted):
    plt.plot(test, color='red',label='Real IBM Stock Price')
    plt.plot(predicted, color='blue',label='Predicted IBM Stock Price')
    plt.title('IBM Stock Price Prediction')
    plt.xlabel('Time')
    plt.ylabel('IBM Stock Price')
    plt.legend()
    plt.show()

def return_rmse(test,predicted):
    rmse = math.sqrt(mean_squared_error(test, predicted))
    print("The root mean squared error is {}.".format(rmse))
```

③ 데이터 불러오기

- 예측에 사용할 시계열 데이터 주식 시장 데이터 csv 파일 로딩

```
dataset = pd.read_csv('IBM_2006-01-01_to_2018-01-01.csv', index_col='Date', parse_dates=['Date'])
dataset.head()
```

- 데이터셋 출력 결과

Date	Open	High	Low	Close	Volume	Name
2006-01-03	82.45	82.55	80.81	82.06	11715200	IBM
2006-01-04	82.20	82.50	81.33	81.95	9840600	IBM
2006-01-05	81.40	82.90	81.00	82.50	7213500	IBM
2006-01-06	83.95	85.03	83.41	84.95	8197400	IBM
2006-01-09	84.10	84.25	83.38	83.73	6858200	IBM

④ 결측 데이터 확인 및 보완

```
training_set = dataset[:'2016'].iloc[:,1:2].values
test_set = dataset['2017':].iloc[:,1:2].values
```

⑤ 가격에 대해 '높음' 속성을 선택 구현

```
dataset["High"][:'2016'].plot(figsize=(16,4),legend=True)
dataset["High"]['2017':].plot(figsize=(16,4),legend=True)
plt.legend(['Training set (Before 2017)','Test set (2017 and beyond)'])
plt.title('IBM stock price')
plt.show()
```

⑥ 훈련 데이터 확장

```
sc = MinMaxScaler(feature_range=(0,1))
training_set_scaled = sc.fit_transform(training_set)
```

⑦ LSTM은 장기 메모리 상태를 저장하기 때문에 60개의 시간단계와 1개의 출력이 있는 데이터 구조를 생성

- 학습 데이터의 각 요소에 대해 60개의 이전 학습 데이터 요소가 있음

```
X_train = []
y_train = []
for i in range(60,2769):
    X_train.append(training_set_scaled[i-60:i,0])
    y_train.append(training_set_scaled[i,0])
X_train, y_train = np.array(X_train), np.array(y_train)
```

⑧ 효율적인 모델링을위한 X_train 재구성

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

⑨ LSTM 구조 구현

```
regressor = Sequential()
# 드롭 아웃 정규화를 사용하는 첫 번째 LSTM 계층
regressor.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],1)))
regressor.add(Dropout(0.2))
# 두 번째 LSTM 계층
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# 세 번째 LSTM 계층
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# 네 번째 LSTM 계층
regressor.add(LSTM(units=50))
regressor.add(Dropout(0.2))
# 출력 계층
regressor.add(Dense(units=1))

regressor.compile(optimizer='rmsprop',loss='mean_squared_error')
regressor.fit(X_train,y_train,epochs=50,batch_size=32)
```

- 학습 시 출력된 학습 진행 상황을 나타내는 결과

```
Epoch 1/50
2709/2709 [==============================] - 50s 18ms/step - loss: 0.0292
Epoch 2/50
2709/2709 [==============================] - 41s 15ms/step - loss: 0.0117
Epoch 3/50
2709/2709 [==============================] - 40s 15ms/step - loss: 0.0091
Epoch 4/50
2709/2709 [==============================] - 40s 15ms/step - loss: 0.0075
Epoch 5/50
2709/2709 [==============================] - 40s 15ms/step - loss: 0.0063
Epoch 6/50
2709/2709 [==============================] - 45s 17ms/step - loss: 0.0060
Epoch 7/50
2709/2709 [==============================] - 44s 16ms/step - loss: 0.0054
Epoch 8/50
2709/2709 [==============================] - 41s 15ms/step - loss: 0.0045

...

Epoch 44/50
2709/2709 [==============================] - 39s 14ms/step - loss: 0.0016
Epoch 45/50
2709/2709 [==============================] - 41s 15ms/step - loss: 0.0015
Epoch 46/50
2709/2709 [==============================] - 47s 17ms/step - loss: 0.0015
Epoch 47/50
2709/2709 [==============================] - 41s 15ms/step - loss: 0.0015
Epoch 48/50
2709/2709 [==============================] - 41s 15ms/step - loss: 0.0014
Epoch 49/50
2709/2709 [==============================] - 40s 15ms/step - loss: 0.0015
Epoch 50/50
2709/2709 [==============================] - 40s 15ms/step - loss: 0.0015
```

⑩ 훈련 데이터와 비슷한 방법으로 테스트 데이터를 준비

- 처리를 위한 '높음' 속성 데이터

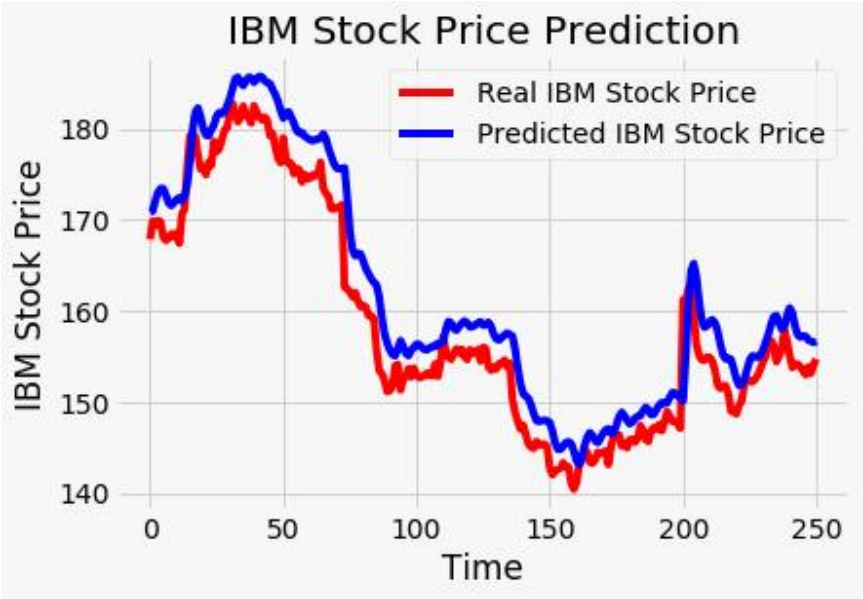
```
dataset_total = pd.concat((dataset["High"][:'2016'],dataset["High"]['2017':]),axis=0)
inputs = dataset_total[len(dataset_total)-len(test_set) - 60:].values
inputs = inputs.reshape(-1,1)
inputs  = sc.transform(inputs)
```

⑪ X_test 준비 및 가격 예측

```
X_test = []
for i in range(60,311):
    X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

⑫ LSTM 결과 시각화

```
1 plot_predictions(test_set,predicted_stock_price)
```



⑬ 모델 평가

```
1 return_rmse(test_set,predicted_stock_price)
```

```
The root mean squared error is 3.9373376584241067.
```

(2) 시계열 데이터를 예측하는 GRU 구현

- Gate mechanism이 적용된 RNN의 일종으로 LSTM에서 영감을 받은 GRU 구현

① GRU 구현에 필요한 라이브러리 임포트

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense, Dropout, GRU
from keras.optimizers import SGD
import math

plt.style.use('fivethirtyeight')
```

② 시각화와 최적화 등 도움이 되는 기능들 구현

```
def plot_predictions(test,predicted):
    plt.plot(test, color='red',label='Real IBM Stock Price')
    plt.plot(predicted, color='blue',label='Predicted IBM Stock Price')
    plt.title('IBM Stock Price Prediction')
    plt.xlabel('Time')
    plt.ylabel('IBM Stock Price')
    plt.legend()
    plt.show()

def return_rmse(test,predicted):
    rmse = math.sqrt(mean_squared_error(test, predicted))
    print("The root mean squared error is {}.".format(rmse))
```

③ 데이터 불러오기

- 예측에 사용할 시계열 데이터 주식 시장 데이터 csv 파일 로딩

```
dataset = pd.read_csv('IBM_2006-01-01_to_2018-01-01.csv', index_col='Date', parse_dates=['Date'])
dataset.head()
```

- 데이터셋 출력 결과

Date	Open	High	Low	Close	Volume	Name
2006-01-03	82.45	82.55	80.81	82.06	11715200	IBM
2006-01-04	82.20	82.50	81.33	81.95	9840600	IBM
2006-01-05	81.40	82.90	81.00	82.50	7213500	IBM
2006-01-06	83.95	85.03	83.41	84.95	8197400	IBM
2006-01-09	84.10	84.25	83.38	83.73	6858200	IBM

④ 결측 데이터 확인 및 보완

```
training_set = dataset['2016'].iloc[:,1:2].values
test_set = dataset['2017':].iloc[:,1:2].values
```

```
[[ 82.55]
 [ 82.5 ]
 [ 82.9 ]
 ...
 [167.74]
 [166.99]
 [166.7 ]]
```

⑤ 가격에 대해 '높음' 속성을 선택 구현

```
dataset["High"][:'2016'].plot(figsize=(16,4),legend=True)
dataset["High"]['2017':].plot(figsize=(16,4),legend=True)
plt.legend(['Training set (Before 2017)','Test set (2017 and beyond)'])
plt.title('IBM stock price')
plt.show()
```

⑥ 훈련 데이터 확장

```
sc = MinMaxScaler(feature_range=(0,1))
training_set_scaled = sc.fit_transform(training_set)
test_set_scaled = sc.fit_transform(test_set)
print(training_set_scaled.shape)
print(test_set_scaled.shape)
```

```
(2769, 1)
(251, 1)
```

⑦ LSTM은 장기 메모리 상태를 저장하기 때문에 60개의 시간단계와 1개의 출력이 있는 데이터 구조를 생성

- 학습 데이터의 각 요소에 대해 60개의 이전 학습 데이터 요소가 있음

```
X_train = []
y_train = []
for i in range(60,2769):
    X_train.append(training_set_scaled[i-60:i,0])
    y_train.append(training_set_scaled[i,0])
X_train, y_train = np.array(X_train), np.array(y_train)

X_test = []
y_test = []
for i in range(60,251):
    X_test.append(test_set_scaled[i-60:i,0])
    y_test.append(test_set_scaled[i,0])
X_test, y_test = np.array(X_test), np.array(y_test)
```

⑧ 효율적인 모델링을위한 X_train 재구성

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

```
(2709, 60)
```

```
(2709, 60, 1)
```

⑨ GRU 구조 구현

```
regressorGRU = Sequential()
# 드롭 아웃 정규화를 사용하는 첫 번째 GRU 계층
regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],1), activation='tanh'))
regressorGRU.add(Dropout(0.2))
# 두 번째 GRU 계층
regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],1), activation='tanh'))
regressorGRU.add(Dropout(0.2))
# 세 번째 GRU 계층
regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],1), activation='tanh'))
regressorGRU.add(Dropout(0.2))
# 네 번째 GRU 계층
regressorGRU.add(GRU(units=50, activation='tanh'))
regressorGRU.add(Dropout(0.2))
# 출력 계층
regressorGRU.add(Dense(units=1))

regressorGRU.summary()

regressorGRU.compile(optimizer=SGD(lr=0.01, decay=1e-7, momentum=0.9, nesterov=False),loss='mean_squared_error')
history = regressorGRU.fit(X_train,y_train,epochs=10,batch_size=150, validation_data=(X_test, y_test))
```

- GRU 모델 summary 출력 결과

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_1 (GRU)                  (None, 60, 50)            7800
_________________________________________________________________
dropout_1 (Dropout)          (None, 60, 50)            0
_________________________________________________________________
gru_2 (GRU)                  (None, 60, 50)            15150
_________________________________________________________________
dropout_2 (Dropout)          (None, 60, 50)            0
_________________________________________________________________
gru_3 (GRU)                  (None, 60, 50)            15150
_________________________________________________________________
dropout_3 (Dropout)          (None, 60, 50)            0
_________________________________________________________________
gru_4 (GRU)                  (None, 50)                15150
_________________________________________________________________
dropout_4 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 51
=================================================================
Total params: 53,301
Trainable params: 53,301
Non-trainable params: 0
_________________________________________________________________
```

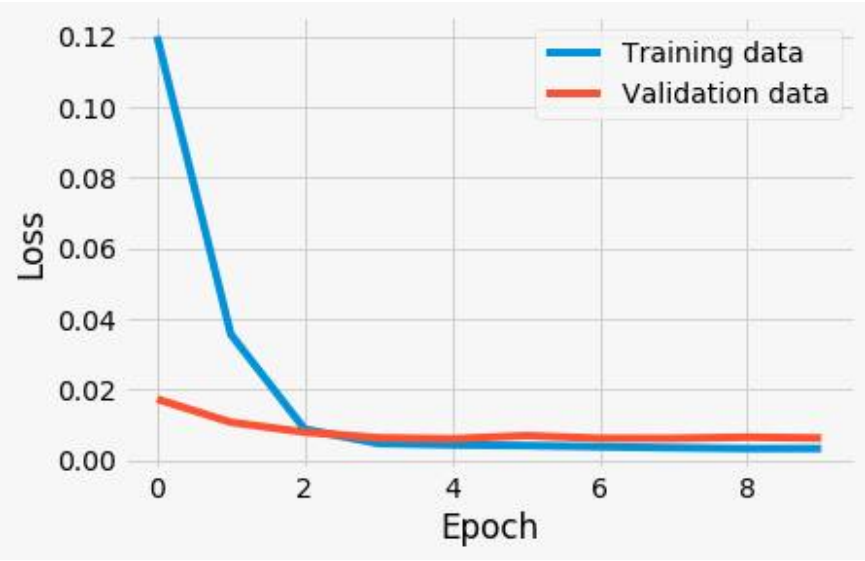
- 학습 시 출력된 학습 진행 상황을 나타내는 결과

```
Train on 2709 samples, validate on 191 samples
Epoch 1/10
2709/2709 [==============================] - 4s 1ms/step - loss: 0.1199 - val_loss: 0.0173
Epoch 2/10
2709/2709 [==============================] - 3s 1ms/step - loss: 0.0356 - val_loss: 0.0107
Epoch 3/10
2709/2709 [==============================] - 3s 1ms/step - loss: 0.0088 - val_loss: 0.0079
Epoch 4/10
2709/2709 [==============================] - 3s 965us/step - loss: 0.0047 - val_loss: 0.0063
Epoch 5/10
2709/2709 [==============================] - 3s 925us/step - loss: 0.0043 - val_loss: 0.0060
Epoch 6/10
2709/2709 [==============================] - 3s 960us/step - loss: 0.0041 - val_loss: 0.0069
Epoch 7/10
2709/2709 [==============================] - 2s 919us/step - loss: 0.0038 - val_loss: 0.0062
Epoch 8/10
2709/2709 [==============================] - 3s 951us/step - loss: 0.0035 - val_loss: 0.0061
Epoch 9/10
2709/2709 [==============================] - 3s 969us/step - loss: 0.0033 - val_loss: 0.0065
Epoch 10/10
2709/2709 [==============================] - 3s 948us/step - loss: 0.0033 - val_loss: 0.0062
```

⑩ GRU 모델 학습 과정에서 손실 학습 곡선 시각화

```python
if not isinstance(history, dict):
    history = history.history

plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training data', 'Validation data'], loc=0)
plt.show()
```



⑪ 훈련 데이터와 비슷한 방법으로 테스트 데이터를 준비

- 처리를 위한 '높음' 속성 데이터

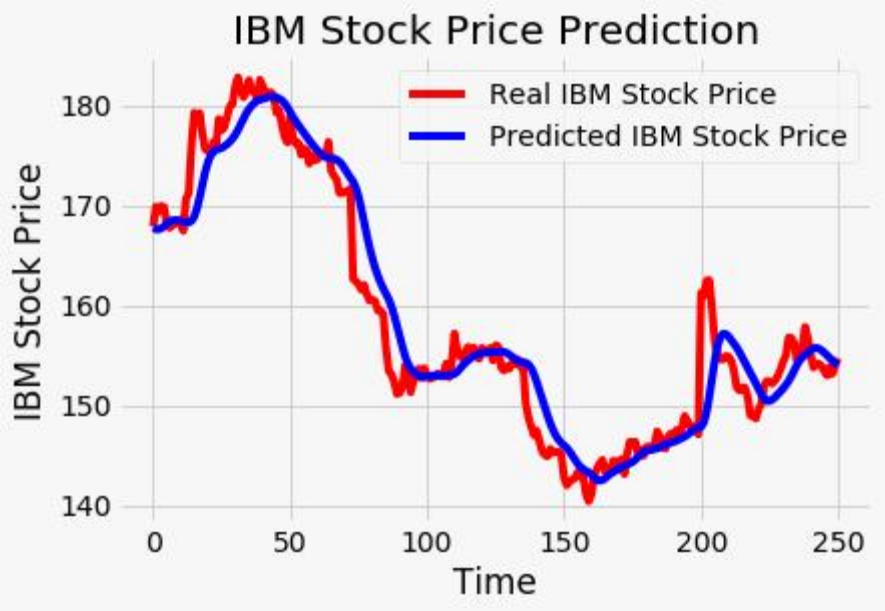
```
dataset_total = pd.concat((dataset["High"][:'2016'],dataset["High"]['2017':]),axis=0)
inputs = dataset_total[len(dataset_total)-len(test_set) - 60:].values
inputs = inputs.reshape(-1,1)
inputs  = sc.transform(inputs)
```

⑫ X_test 준비 및 가격 예측

```
X_test = []
for i in range(60,311):
    X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
GRU_predicted_stock_price = regressorGRU.predict(X_test)
GRU_predicted_stock_price = sc.inverse_transform(GRU_predicted_stock_price)
```

⑬ LSTM 결과 시각화

```
plot_predictions(test_set,GRU_predicted_stock_price)
```



⑭ 모델 평가

```
return_rmse(test_set,GRU_predicted_stock_price)
```

```
The root mean squared error is 3.2956514743855503.
```

평가하기

1. 과거의 데이터를 초기화 시키는 것을 목적하는 게이트를 고르시오.

① Forget Gate

② Input Gate

③ Reset Gate

④ Update Gate

- 정답 : ③번

해설 : GRU의 Reset Gate는 과거의 데이터를 Reset(초기화) 시키는 것을 목적으로 하는 게이트이다.

2. 가져가야 할 데이터 양과 잊어버려야 할 데이터의 양을 결정하는 게이트를 고르시오.

① Forget Gate

② Input Gate

③ Reset Gate

④ Update Gate

- 정답 : ④번

해설 : GRU의 Update Gate는 과거와 현재의 정보 업데이트 비율을 결정하는 게이트이다.

학습정리

1. 시계열 데이터를 예측하는 GRU 구현

- Reset Gate와 Update Gate가 존재하는 GRU 구현