به نام خدا

تمرین دوازدهم یادگیری عمیق

غزل زمانی‌نژاد

۹۷۵۲۲۱۶۶

1. داده آموزش و آزمون را مطابق روزهای خواسته شده دانلود میکنیم.

```
import yfinance as yf
import datetime
import pandas as pd

start_train = datetime.datetime(2014,1,1)
end_train = datetime.datetime(2021,9,1)

# type: data frame
train_data = yf.download(tickers='BTC-USD', start=start_train, end=end_train)
```

```
[*********************100%***********************]  1 of 1 completed
```

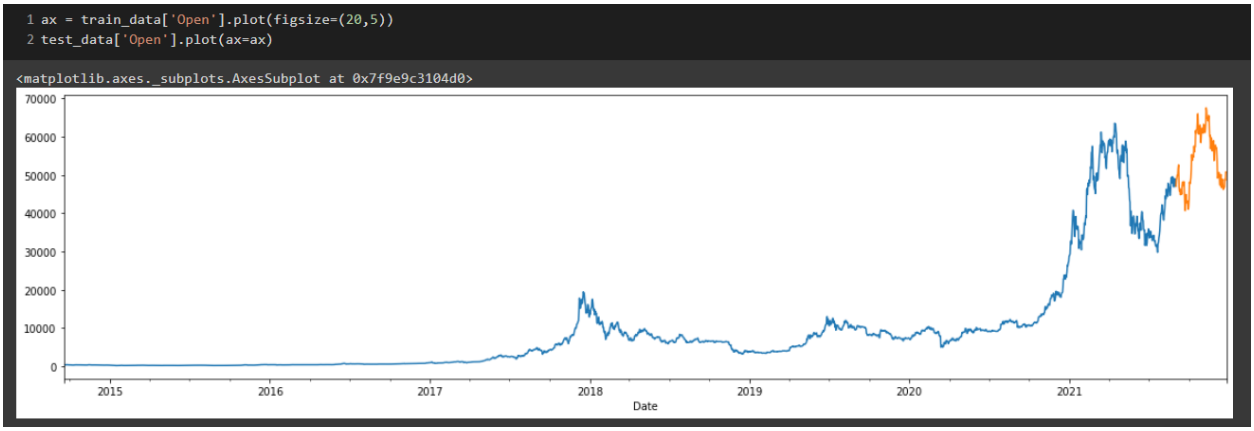
```
test_data = yf.download(tickers='BTC-USD', start=end_train)
```

```
[*********************100%***********************]  1 of 1 completed
```

سپس نمودارشان را رسم میکنیم.

```
ax = train_data['Open'].plot(figsize=(20,5))
test_data['Open'].plot(ax=ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9e9c3104d0>
```



سپس داده ها را با استفاده از MinMaxScaler از کتابخانه sklearn نرمالایز میکنیم.

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np

# normalize train and test data with sklearn.preprocessing.MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(train_data['Open'].values.reshape(-1, 1))
train_normalized = scaler.transform(train_data['Open'].values.reshape(-1, 1))
train_normalized = np.squeeze(train_normalized)

scaler2 = MinMaxScaler()
scaler2.fit(test_data['Open'].values.reshape(-1, 1))
test_normalized = scaler2.transform(test_data['Open'].values.reshape(-1, 1))
test_normalized = np.squeeze(test_normalized)

print("train sequence shape", train_normalized.shape)
print("test sequence shape", test_normalized.shape)
print(train_normalized[2400])
print(test_normalized[50])
```

```
train sequence shape (2542,)
test sequence shape (116,)
0.9426374953289349
0.9424117074486829
```

در سلولی بعدی مقداری را برای past در نظر میگیریم (در اینجا 60). سپس باید دنباله زمانی را در یک حلقه تشکیل بدهیم. برای این کار روی داده ها iterate میکنیم. هر 60 داده را به عنوان input و 61مین را به عنوان برچسب در نظر میگیریم. همین کار را تا جایی که طول input کمتر از 60 نشود تکرار میکنیم.

سپس دو لیست را به numpy array تبدیل میکنیم و یک بعد به آن اضافه میکنیم.

```
past = 60
x_train = []
y_train = []

# prepocess data
for i in range(train_normalized.size-past):
  x = train_normalized[i:i+past]
  y = train_normalized[i+past]
  x_train.append(x)
  y_train.append(y)

x_train = np.array(x_train)
x_train = np.expand_dims(x_train, axis=-1)
print(x_train.shape)

y_train = np.array(y_train)
print(y_train.shape)
```

```
(2482, 60, 1)
(2482,)
```

همین کار را برای داده آزمون انجام میدهیم.

```
x_test = []
y_test = []

for i in range(test_normalized.size-past):
  x = test_normalized[i:i+past]
  y = test_normalized[i+past]
  x_test.append(x)
  y_test.append(y)

x_test = np.array(x_test)
x_test = np.expand_dims(x_test, axis=-1)
print(x_test.shape)

y_test = np.array(y_test)
print(y_test.shape)
```

```
(56, 60, 1)
(56,)
```

سپس مدل خواسته شده را تشکیل میدهیم. در این مدل، مطابق شکل برای لایه بازگشتی اول تا سوم باید دنباله برگردانیم ولی برای لایه LSTM آخر یک خروجی برمی گردانیم.

```
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(past, 1)))
model.add(Dropout(0.2))
model.add(LSTM(50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(50))
model.add(Dense(1))
```

مدل را در 100 ایپوک و با batch size = 32 آموزش میدهیم.

```
model.compile(loss='mean_squared_error', optimizer='adam')
history = model.fit(x_train, y_train, epochs=epoch_size, batch_size=batch_size)
```

نتیجه ایپوک های پایانی به صورت زیر است:

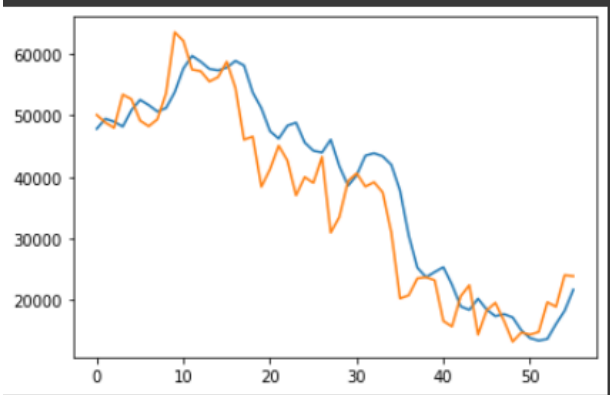
```
Epoch 95/100
78/78 [==============================] - 5s 62ms/step - loss: 2.4356e-04
Epoch 96/100
78/78 [==============================] - 5s 62ms/step - loss: 2.4032e-04
Epoch 97/100
78/78 [==============================] - 5s 62ms/step - loss: 2.6001e-04
Epoch 98/100
78/78 [==============================] - 5s 62ms/step - loss: 2.3535e-04
Epoch 99/100
78/78 [==============================] - 5s 62ms/step - loss: 2.7303e-04
Epoch 100/100
78/78 [==============================] - 5s 62ms/step - loss: 2.4439e-04
```

برای اینکه ببینیم پیش بینی مدل از داده آزمون تا چه میزان دقیق بوده، آن را در یک صفحه مختصات به همراه مقادیر اصلی رسم میکنیم. برای این کار در ابتدا از متد inverse_transform استفاده میکنیم تا مقادیر را از حالت نرمالایز به حالت اصلی شان بازگردانیم. سپس هر دو آرایه را با استفاده از matplotlib رسم میکنیم.

```python
import matplotlib.pyplot as plt

# predict test data to check the performance of model
pred_norm = model.predict(x_test)
# print(pred_norm.shape)

# inverse of normalize
# to find true values
test_pred_val = np.squeeze(scaler.inverse_transform(pred_norm))
y_test_val = np.squeeze(scaler.inverse_transform([y_test]))
# plot true labels of test data and prediction
plt.plot(test_pred_val)
plt.plot(y_test_val)
plt.show()
```



در اینجا نمودار آبی مربوط به پیش بینی مدل و نمودار نارنجی مقادیر اصلی است. پیش بینی مدل خیلی دقیق نبوده اما به صورت حدودی توانسته قیمت ها را پیش بینی کند. ممکن است در صورت آموزش مدل در ایپوک های بیشتر نتیجه ذره ای بهتر شود.

در گام آخر قیمت بیت کوین را برای 3 ماه آینده پیش بینی می کنیم. برای این کار از 60 داده نهایی استفاده میکنیم تا 1 روز بعد را پیش بینی کنیم. مقدار پیش بینی شده را به یک لیست اضافه میکنیم. سپس در یک حلقه نخستین المان لیست را پاک میکنیم و آخرین مقدار پیش بینی شده را به داده اضافه میکنیم و در یک حلقه این کار را تکرار میکنیم.

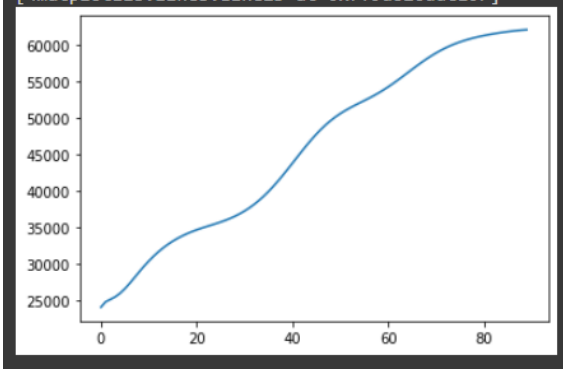
```
# predict next 3 months
future = 3 * 30
data = list(test_normalized[-past:])

future_preds = []
for i in range(future):
  data_batch = np.expand_dims(data, 0)
  prediction = model.predict(data_batch)[0][0]
  future_preds.append(prediction)
  # remove first elemenet from data and append new prediction for next loop
  data.pop(0)
  data.append(prediction)
```

نتایج موجود در لیست future_preds را از مقادیر نرمالایز به مقادیر اصلی تبدیل میکنیم و در نمودار رسم میکنیم.

```
future_preds = np.array(future_preds)
inv_pred = scaler.inverse_transform(future_preds.reshape(-1, 1))
plt.plot(np.squeeze(inv_pred))
```

```
[<matplotlib.lines.Line2D at 0x7f9d826ddc10>]
```



بررسی مقدار پارامتر past: اگر مقدار آن زیاد باشد، مدل برای پیش بینی روزهای بیشتری را در نظر می گیرد. اعدادی که پیش بینی میکند نسبت به هم بسیار متفاوت نیستند و در یک بازه قرار میگیرند. اگر آنها را رسم کنیم نوسان نمودار آن زیاد نیست. اما اگر این مقدار بسیار زیاد باشد مدل پیش بینی خوبی نخواهد داشت. اگر مقدار آن کم باشد مدل تنها به داده های اخیر برای پیش بینی تکیه می کند و پیش بینی آن به این مقادیر نزدیک خواهد بود.

در کل یک tradeoff در اینجا وجود دارد. و باید این پارامتر را به گونه ای تنظیم کنیم که نه خیلی زیاد و نه خیلی کم باشد.

2. ابتدا فایل را میخوانیم.

سپس دو دیکشنری میسازیم. در دیکشنری اول کلید کاراکترهای موجود در کلمات و مقادیر ایندکس است. دیکشنری دوم برعکس دیکشنری اول است. متد covert یک رشته به عنوان ورودی دریافت میکند و در یک حلقه هر یک از کاراکترهای آن را به one hot انکود میکند. نتیجه را به صورت numpy array برمیگرداند.

```
with open('enc-dec.txt') as f:
  lines = f.read().splitlines()
```

```
import numpy as np

word_len = 10
alphabet = " abcdefghijklmnopqrstuvwxyz"
alpha_index = dict((c,i) for i,c in enumerate(alphabet))
# print(alpha_index)
index_alpha = dict((i,c) for i,c in enumerate(alphabet))
# print(index_alpha)

def convert(data):
  one_hot = []
  for d in data:
    vec = np.zeros((len(alphabet)))
    vec[alpha_index[d]] = 1
    one_hot.append(vec)
    # print(vec)
  return np.array(one_hot)

x = convert("sehlckaohz")
print(x.shape)
```

```
(10, 27)
```

در سلول بعدی در یک حلقه تک تک خطوط فایل را پردازش می کنیم. در هر خط، کلمه دیکود شده را از کلمه اصلی جدا میکنیم و در صورت نیاز کاراکتر space اضافه میکنیم تا طول آنها 10 شود. و آنها را به لیست های مربوطه اضافه میکنیم. سپس به numpy array تبدیل میکنیم.

```
x = []
y = []

for line in lines:
  splited = line.split('\t')
  enc = splited[0]
  while len(enc) < 10:
    enc += ' '

  dec = splited[1]
  while len(dec) < 10:
    dec += ' '

  enc_onehot = convert(enc)
  dec_onehot = convert(dec)

  x.append(enc_onehot)
  y.append(dec_onehot)
```

```
x = np.array(x)
y = np.array(y)
print(x.shape)
print(y.shape)
```

```
(152273, 10, 27)
(152273, 10, 27)
```

بعد با استفاده از متد train_test_split **از کتابخانه** sklearn **داده ها را** shuffle **میکنیم و 10 درصد از داده ها را به مجموعه تست اختصاص میدهیم.**

```
from sklearn.model_selection import train_test_split

# shuffle data and divide into train and test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=4)
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(137045, 10, 27)
(137045, 10, 27)
(15228, 10, 27)
(15228, 10, 27)
```

برای اینکه بتوانیم چند مدل را امتحان کنیم، یک متد میسازیم که تعداد نورون های مخفی و همچنین تعداد لایه های recurrent **را به عنوان ورودی دریافت میکند و مدل ساخته شده را به عنوان خروجی برمی گرداند.**

```
BATCH_SIZE = 128
```

```
from keras.models import Sequential
from keras.layers import Dense, GRU, Dropout, RepeatVector

def build_model(layers, hidden):
  model = Sequential()
  model.add(GRU(hidden, input_shape=(word_len, len(alphabet))))

  # As the decoder RNN's input, repeatedly provide with the last output of
  # RNN for each time step. Repeat 'word_len' times
  model.add(RepeatVector(word_len))

  # The decoder RNN could be multiple layers stacked or a single layer.
  for _ in range(layers):
      model.add(GRU(hidden, return_sequences=True))

  model.add(Dense(len(alphabet), activation='softmax'))

  return model
```

به کمک این تابع 4 مدل گوناگون تولید کردیم و یکی از مدل ها را در تعداد ایپوک های مختلف آموزش دادیم. مدل چهارم بهترین دقت تست را بدست آورد. این مدل دارای 4 لایه بازگشتی است و هریک از این لایه ها 256 یونیت مخفی دارند.

```
# 4 GRU layers + 256 hidden units
model4 = build_model(3, 256)
model4.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model4.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_10 (GRU)                (None, 256)               218880

 repeat_vector_2 (RepeatVect  (None, 10, 256)          0
 or)

 gru_11 (GRU)                (None, 10, 256)           394752

 gru_12 (GRU)                (None, 10, 256)           394752

 gru_13 (GRU)                (None, 10, 256)           394752

 dense_2 (Dense)             (None, 10, 27)            6939

=================================================================
Total params: 1,410,075
Trainable params: 1,410,075
Non-trainable params: 0
_________________________________________________________________
```

این مدل در ابتدا ورودی را دریافت میکند و از یک لایه GRU عبور میدهد. در این لایه return sequence صحیح نیست یعنی تنها یک خروجی برمی گرداند. به همین دلیل آن را با لایه RepeatVector به تعدادی که

میخواهیم کپی میکنیم و به عنوان ورودی های لایه GRU بعدی قرار میدهیم. پس از آن 2 لایه GRU دیگر هم داریم. در نهایت از یک لایه Dense عبور میدهیم.

سپس مدل را با batch size = 128 و در 15 ایپوک آموزش میدهیم.

```
history4 = model4.fit(x_train, y_train,
                      batch_size=BATCH_SIZE,
                      epochs=15,
                      validation_data=(x_test, y_test))
```

نتیجه ایپوک های پایانی به صورت زیر است:

```
Epoch 12/15
1071/1071 [==============================] - 28s 26ms/step - loss: 0.1684 - accuracy: 0.9477 - val_loss: 0.1948 - val_accuracy: 0.9390
Epoch 13/15
1071/1071 [==============================] - 27s 26ms/step - loss: 0.1583 - accuracy: 0.9509 - val_loss: 0.1712 - val_accuracy: 0.9474
Epoch 14/15
1071/1071 [==============================] - 27s 25ms/step - loss: 0.1541 - accuracy: 0.9521 - val_loss: 0.1631 - val_accuracy: 0.9497
Epoch 15/15
1071/1071 [==============================] - 27s 25ms/step - loss: 0.1477 - accuracy: 0.9540 - val_loss: 0.1603 - val_accuracy: 0.9510
```

مدل توانسته دقت خوبی روی داده های تست بدست بیاورد.

اکنون نوبت به رمزگشایی جمله داده شده است. ابتدا بر روی آن پردازش هایی مشابه قبل انجام میدهیم تا برای شبکه قابل فهم باشد.

```
# model4 has the best val accuracy
# so use it to decode the password
coded = "onmltsrqpoihgrezcba lknrvjihgfueiizltflk"

one_hot_coded = []
# prepocess the coded string
for i in range(0, len(coded), word_len):
  word = coded[i:i+word_len]
  one_hot = convert(word)
  one_hot_coded.append(one_hot)

one_hot_coded = np.array(one_hot_coded)
print(one_hot_coded.shape)
```

```
(4, 10, 27)
```

سپس آن را به مدل میدهیم تا پیش بینی کند. و بعد با استفاده از دیکشنری index_alpha که تعریف کرده بودیم، ایندکس ها را به کاراکتر الفبا تبدیل میکنیم. در نهایت جمله کامل را چاپ میکنیم.

```
decoded = model4.predict(one_hot_coded)
print(decoded.shape)
```

```
(4, 10, 27)
```

```
max_prob = np.argmax(decoded, axis=-1)
print(max_prob.shape)
sentence = []
for w in max_prob:
  word = ""
  for ch in w:
    word += index_alpha[ch]
  sentence.append(word)
```

```
(4, 10)
```

```
print(sentence)
```

```
['    i     ', '   love   ', 's deep    ', 'plearning ']
```