فاطمه توکلی_۴۰۰۱۳۱۰۱۶

۱-ساختار حافظه کوتاه مدت بلند شامل سه گیت : input,forget,recall/output و همچنین یک cell stateمی باشد و ساختار GRU شامل دو گیت: reset, update می‌باشد و شکل کلی آن به صورت زیر می‌باشد:

گیت فراموشی(Forget Gate)

این گیت تصمیم می‌گیرد کدام اطلاعات حفظ و کدام فراموش شود. اطلاعات ورودی گام جدید به‌همراه اطلاعات حالت نهان (Hidden State)گام قبلی به این گیت وارد می‌شوند و از تابع Sigmoid عبور می‌کنند. خروجی این تابع میان ۰ تا ۱ است.

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

گیت ورودی(Input Gate)

این گیت برای به‌روزرسانی مقادیر موجود در cell state تعبیه شده است. اطلاعات ورودی گام جدید، به‌همراه اطلاعات hidden state گام قبلی، به این گیت وارد می‌شوند و از تابع Sigmoid عبور می‌کنند تا این تابع تصمیم بگیرد کدام دور انداخته و کدوم به‌روزرسانی شوند. هم‌چنین اطلاعات ورودی گام جدید، به‌همراه اطلاعات hidden state گام قبلی، به تابع Tanh وارد می‌شوند تا مقادیرشان میان -۱ تا ۱ قرار بگیرد. درنهایت خروجی تابع Sigmoid و Tanh با هم ضرب می‌شوند تا تابع Sigmoid تصمیم بگیرد چه مقادیری از خروجی تابع Tanhباید حفظ شوند.

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

گیت خروجی(Output Gate)

این گیت درنهایت تصمیم می‌گیرد که hidden state بعدی چه مقداری باشد. اول اطلاعات ورودی گام جدید به‌همراه اطلاعات hidden stateگام قبلی یه تابع Sigmoid وارد می‌شوند. مقدار به روز شده cell state به تابع Tanh وارد می‌شود. خروجی این دو تابع با هم ضرب می‌شود تا تصمیم گرفته شود hidden state چه اطلاعاتی را با خودش به گام بعدی ببرد. درنهایت cell state جدید و hidden stateجدید به گام زمانی بعدی منتقل می‌شوند.

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

$$h_t = o_t * \tanh\left(C_t\right)$$

Cell state:

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

گیت به‌روزرسانی(Update Gate)

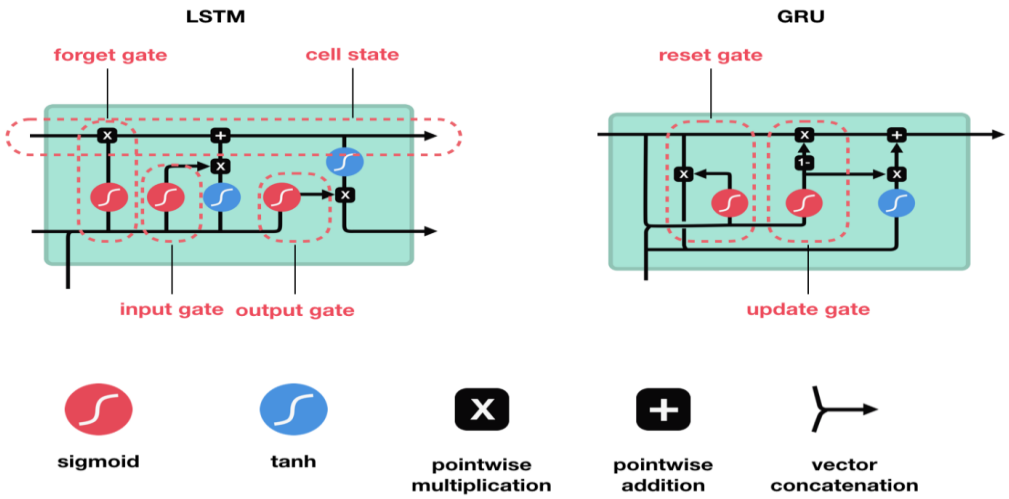
این گیت دقیقاً مانند دو گیت فراموشی (Forget Gate) و ورودی (Input Gate) در شبکه LSTM عمل می‌کند. این گیت تصمیم می‌گیرد چه مقدار از اطلاعاتی که در گام‌های قبلی داشتیم، نگه داریم. در این گیت مقدار ورودی جدید (x_t) به‌همراه مقدار حالت نهان گام قبلی (h_{t-1}) در وزن متناظر خود ضرب و سپس با هم جمع می‌شوند و به یک تابع sigmoid وارد می‌شوند تا خروجی میان بازه ۰ تا ۱ قرار بگیرد. در زمان آموزش شبکه این وزن‌ها هر بار به‌روزرسانی می‌شوند.

$$z_t = \sigma(W^{(z)} x_t + U^{(z)} h_{t-1})$$

گیت تنظیم مجدد(Reset Gate)

این گیت تصمیم می‌گیرد چه مقدار از اطلاعات گذشته، یعنی اطلاعات گام‌های قبلی، فراموش شود. در اینجا هم مقدار ورودی جدید(x_t) ، به‌همراه مقدار حالت نهان گام قبلی(h_{t-1}) ، در وزن متناظر خود ضرب و سپس با هم جمع می‌شوند و به یک تابع sigmoid وارد می‌شوند تا خروجی بین بازه ۰ تا ۱ قرار بگیرد. تفاوتی که با گیت به‌روزرسانی دارد این است که وزن‌هایی که مقدار ورودی و حالت نهان گام قبلی در آن ضرب می‌شوند متفاوت است و این یعنی بردارهای خروجی در اینجا با بردار خروجی که در گیت به‌روزرسانی داریم متفاوت می‌باشد.

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$



تفاوت اصلی بین GRU و LSTM این است که کیف GRU دارای دو گیت reset و update است. در حالی که LSTM دارای سه گیت است که input، output، forget هستند. GRU پیچیدگی کمتری نسبت به LSTM دارد زیرا تعداد گیت های کمتری دارد. اگر مجموعه داده کوچک است، GRU ترجیح داده می شود در غیر این صورت LSTM برای مجموعه داده بزرگتر ترجیح داده می شود.GRU هیچ حافظه داخلی ندارد، و مانند LSTM گیت خروجی ندارد.

۲- کتابخانه‌های مورد نیاز :

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
import glob
from sklearn.preprocessing import MinMaxScaler
from tensorflow import keras
import numpy
import math
from keras.models import Sequential
from keras.layers import Dense, LSTM,Dropout,GRU
from sklearn.metrics import mean_squared_error
from keras.engine.input_layer import Input
from sklearn.metrics import confusion_matrix
```

در ابتدا تمامی دیتاست را میخوانیم و فایل های خالی را حذف می‌کنیم و سپس بر روی باقیمانده آنها بازه سال اخیر را اعمال می‌کنیم و ستون‌های شامل حروف را حذف می‌کنیم و در نهایت ادغام می‌کنیم :

```
dataset =pd.DataFrame(columns=['<DTYYYYMMDD>'])
path = "/content/*.*"
for file in glob.glob(path):
  print(file)
  data = pd.read_csv(file, delimiter=',', encoding='utf-16_le')
  if not data.empty:

    # Select a range of data
    data = pd.DataFrame(data[(data['<DTYYYYMMDD>'] > 20210523)], columns=data.columns).reset_index(drop=True)

    #delete string columns
    drop_attrs = [0, 10, 11]
    data.drop(data.columns[drop_attrs], axis=1, inplace=True)

    #merge
    dataset = pd.merge(dataset, data, left_on='<DTYYYYMMDD>', right_on='<DTYYYYMMDD>', how="right", sort=False)
```

سپس مجموعه داده ساخته شده را نرمالایز می‌کنیم و چک میکنیم ایا مقدار null داریم یا نه و سپس ایندکس آن را در می‌اوریم و حذف میکنیم:

```
dataset.shape
```

```
(238, 145)
```

```
ds = MinMaxScaler().fit_transform(dataset)
ds = pd.DataFrame(ds)
ds.isnull().values.any()
```

```
True
```

```
rows_with_nan = []
for index, row in ds.iterrows():
    is_nan_series = row.isnull()
    if is_nan_series.any():
        rows_with_nan.append(index)
rows_with_nan
```

```
[201]
```

```
ds = ds.drop([201],axis=0)
```

```
ds.shape
```

```
(237, 145)
```

برای لیبل دیتاست از فایلی که شامل شاخص کل است استفاده می‌کنیم و مقدار ستون close هر سطر را منهای سطر قبلی کرده و در صورتی که بزرگتر صفر بوده مقدار ۱ به عنوان لیبل ان سطر می‌دهیم و در صورتی که کوچکتر از صفر بود مقدار ۰ به عنوان لیبل داده میشود:

```
def label (data):
  label =[]
  for i in range(1,len(data)) :
    var = data[i] - data[i-1]
    if var < 0:
        label.append(0)
    else:
        label.append(1)
  return np.array(label)
```

```
path = '/content/IRX6XTPI0009.csv'
dt = pd.read_csv(path, delimiter=',', encoding='utf-16_le')
dt = pd.DataFrame(dt[(dt['<DTYYYYMMDD>'] > 20210522)], columns=dt.columns).reset_index(drop=True)
labels = label(dt['<CLOSE>'])
```

سپس برای اینکه سری زمانی برای مجموعه داده اماده شده بسازیم به صورت زیر عمل می‌کنیم و دیتا و برچسب ها را دسته بندی می‌کنیم:

```
1  def batch_ds (dt,lbl, batchsize):
2    data_time=[]
3    lables=label(lbl)
4    label_time=[]
5    for i in range(batchsize, len(dt)):
6      data_time.append(dt[i - batchsize : i ])
7      label_time.append(lables[i-1])
8    return np.array(data_time), np.array(label_time)
```

```
1  data_timestep, label_timestep = batch_ds(ds,labels,batchsize=10)
```

```
1  data_timestep.shape
```

```
(227, 10, 145)
```

در نهایت داده را به سه دسته تقسیم می‌کنیم:

```
X_train, X_test, y_train, y_test = train_test_split(data_timestep, label_timestep, test_size=0.3, random_state=22)
X_test , X_val, y_test, y_val =train_test_split(X_test,y_test,test_size=0.33,random_state=22)
```

LSTM MODEL:

با استفاده از مدل sequentialکراس مدل ترتیبی ایجاد کرده و از tf.keras.layers.LSTM که قبلا import شده است به صورت اماده استفاده می‌کنیم و در نهایت یک لایه dense برای رسیدن به یک خروجی(۰یا ۱) اضافه می‌کنیم:

```
model = Sequential()
model.add(Input(shape = (10,145)))
model.add(LSTM(4))
model.add(Dense(1, activation ='relu'))
```

```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(X_train, y_train, epochs=5)
```

دقت به دست امده برروی مجموعه داده اموزش و سپس ارزیابی مدل:

```
Epoch 5/5
5/5 [==============================] - 0s 7ms/step - loss: 0.1969 - acc: 0.7342
<keras.callbacks.History at 0x7f21fc20ebd0>
```

```
model.evaluate(X_test,y_test)
```

```
2/2 [==============================] - 0s 9ms/step - loss: 0.1344 - acc: 0.8696
[0.13442111015319824, 0.8695651888847351]
```

ماتریس در هم ریختگی:

```
y_pred = model.predict(X_val)
```

```
confusion_matrix(y_val, y_pred.round())
```

```
array([[ 0,  4],
       [ 0, 19]])
```

از نتیجه دریافت میشود که شاخص ما صعودی است

نتیجه استفاده از ابعاد مختلف لایه نهان: نتیجه گرفته می‌شود که اگر ابعاد لایه مخفی بیشتر از حدی شود شبکه توانایی یادگیری خودش را از دست می‌دهد و نسبت به تغییرات اموزش کمتری دارد.

```
model = Sequential()
model.add(Input(shape = (10,145)))
model.add(LSTM(10))
model.add(Dense(1, activation ='relu'))
```

```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
5/5 [==============================] - 3s 8ms/step - loss: 0.7342 - acc: 0.2658
Epoch 2/5
5/5 [==============================] - 0s 10ms/step - loss: 0.7342 - acc: 0.2658
Epoch 3/5
5/5 [==============================] - 0s 8ms/step - loss: 0.7342 - acc: 0.2658
Epoch 4/5
5/5 [==============================] - 0s 8ms/step - loss: 0.7342 - acc: 0.2658
Epoch 5/5
5/5 [==============================] - 0s 9ms/step - loss: 0.7342 - acc: 0.2658
<keras.callbacks.History at 0x7f21ebf5ca10>
```

```
model = Sequential()
model.add(Input(shape = (10,145)))
model.add(LSTM(5))
model.add(Dense(1, activation ='sigmoid'))
```

```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
5/5 [==============================] - 2s 8ms/step - loss: 0.2569 - acc: 0.4494
Epoch 2/5
5/5 [==============================] - 0s 8ms/step - loss: 0.2198 - acc: 0.6899
Epoch 3/5
5/5 [==============================] - 0s 7ms/step - loss: 0.2053 - acc: 0.7342
Epoch 4/5
5/5 [==============================] - 0s 7ms/step - loss: 0.2010 - acc: 0.7342
Epoch 5/5
5/5 [==============================] - 0s 7ms/step - loss: 0.1986 - acc: 0.7342
<keras.callbacks.History at 0x7f21e2b22d90>
```

```
model = Sequential()
model.add(Input(shape = (10,145)))
model.add(LSTM(15))
model.add(Dense(1, activation ='sigmoid'))
```

```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
5/5 [==============================] - 2s 8ms/step - loss: 0.2009 - acc: 0.7342
Epoch 2/5
5/5 [==============================] - 0s 6ms/step - loss: 0.1990 - acc: 0.7342
Epoch 3/5
5/5 [==============================] - 0s 7ms/step - loss: 0.1975 - acc: 0.7342
Epoch 4/5
5/5 [==============================] - 0s 7ms/step - loss: 0.1966 - acc: 0.7342
Epoch 5/5
5/5 [==============================] - 0s 7ms/step - loss: 0.1959 - acc: 0.7342
<keras.callbacks.History at 0x7f21e24a0690>
```

GRU MODEL:

```
model = Sequential()
model.add(Input(shape = (10,145)))
model.add(GRU(4))
model.add(Dense(1, activation ='sigmoid'))
```

```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(data_timestep, label_timestep, epochs=5)
```

دقت مدل بر روی مجموعه آموزش و ارزیابی آن:

```
Epoch 5/5
8/8 [==============================] - 0s 7ms/step - loss: 0.2052 - acc: 0.7709
<keras.callbacks.History at 0x7f21ded88190>
```

```
model.evaluate(X_test,y_test)
```

```
2/2 [==============================] - 0s 14ms/step - loss: 0.1257 - acc: 0.8696
[0.12568818032741547, 0.8695651888847351]
```

ماتریس در هم ریختگی:

```
y_pred = model.predict(X_val)
```

```
confusion_matrix(y_val, y_pred.round())
```

```
array([[ 0,  4],
       [ 0, 19]])
```

ابعاد مختلف {۵و۱۵و۲۵}را امتحان کرده و تنها تفاوت در سرعت همگرایی می باشد و هرچه بیشتر باشد سرعت همگرایی نیز بیشتر است:

```
model = Sequential()
model.add(Input(shape = (10,145)))
model.add(GRU(5))
model.add(Dense(1, activation ='sigmoid'))
```

```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(data_timestep, label_timestep, epochs=5)
```

```
Epoch 1/5
8/8 [==============================] - 2s 7ms/step - loss: 0.2712 - acc: 0.4670
Epoch 2/5
8/8 [==============================] - 0s 7ms/step - loss: 0.1894 - acc: 0.7709
Epoch 3/5
8/8 [==============================] - 0s 7ms/step - loss: 0.1799 - acc: 0.7709
Epoch 4/5
8/8 [==============================] - 0s 6ms/step - loss: 0.1813 - acc: 0.7709
Epoch 5/5
8/8 [==============================] - 0s 6ms/step - loss: 0.1816 - acc: 0.7709
<keras.callbacks.History at 0x7f21e8192b50>
```

```
model = Sequential()
model.add(Input(shape = (10,145)))
model.add(GRU(10))
model.add(Dense(1, activation ='sigmoid'))
```

```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(data_timestep, label_timestep, epochs=5)
```

```
Epoch 1/5
8/8 [==============================] - 2s 9ms/step - loss: 0.2297 - acc: 0.6123
Epoch 2/5
8/8 [==============================] - 0s 7ms/step - loss: 0.1857 - acc: 0.7709
Epoch 3/5
8/8 [==============================] - 0s 7ms/step - loss: 0.1809 - acc: 0.7709
Epoch 4/5
8/8 [==============================] - 0s 8ms/step - loss: 0.1809 - acc: 0.7709
Epoch 5/5
8/8 [==============================] - 0s 7ms/step - loss: 0.1805 - acc: 0.7709
<keras.callbacks.History at 0x7f21ddff4590>
```

```
model = Sequential()
model.add(Input(shape = (10,145)))
model.add(GRU(25))
model.add(Dense(1, activation ='sigmoid'))
```

```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(data_timestep, label_timestep, epochs=5)
```

```
Epoch 1/5
8/8 [==============================] - 2s 8ms/step - loss: 0.1857 - acc: 0.7665
Epoch 2/5
8/8 [==============================] - 0s 8ms/step - loss: 0.1812 - acc: 0.7709
Epoch 3/5
8/8 [==============================] - 0s 8ms/step - loss: 0.1842 - acc: 0.7709
Epoch 4/5
8/8 [==============================] - 0s 9ms/step - loss: 0.1844 - acc: 0.7709
Epoch 5/5
8/8 [==============================] - 0s 8ms/step - loss: 0.1795 - acc: 0.7709
<keras.callbacks.History at 0x7f21db49d950>
```

در این مسئله بهترین دقت مدل GRU 77% و بهترین دقت مدلLSTM 73% می باشد, علت آن کوچک بودن مجموعه داده می‌باشد

۳- با استفاده از ۲ لایه LSTM از Keras مانند قبل عمل می‌کنیم و خروجی اولی را به عنوان ورودی به لایه بعدی میدهیم:

```
model = Sequential()
model.add(LSTM(16, input_shape=(10,145), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(8, input_shape=(10,16), return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(4,activation='relu'))
model.add(Dense(1,activation='relu'))
```

نتیجه مدل بر روی داده آموزش

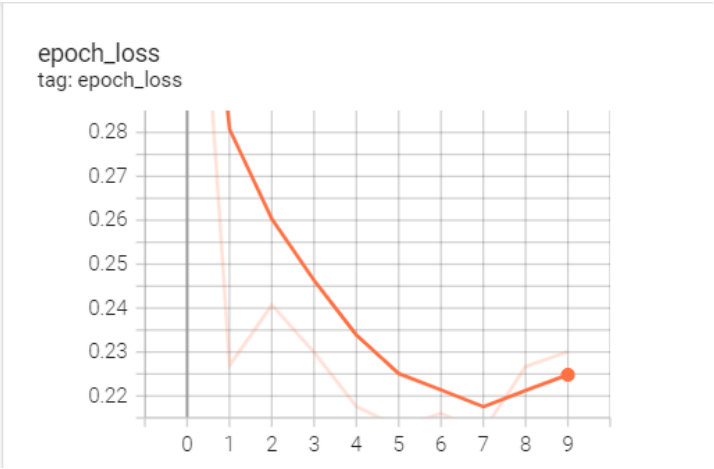
```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(data_timestep, label_timestep, epochs=10,callbacks=[tensorboard_callback])
```

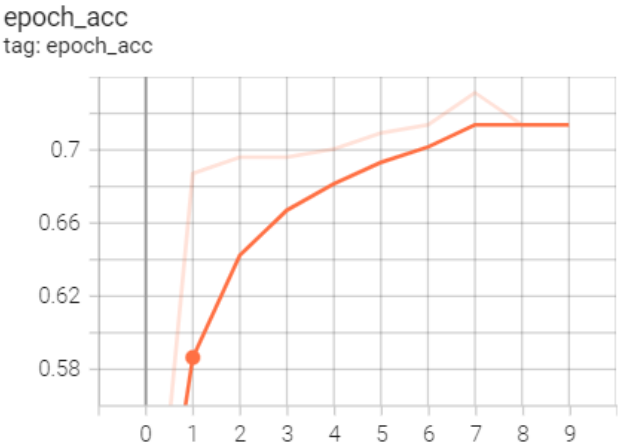
```
Epoch 1/10
8/8 [==============================] - 6s 19ms/step - loss: 0.4761 - acc: 0.2996
Epoch 2/10
8/8 [==============================] - 0s 20ms/step - loss: 0.2164 - acc: 0.7093
Epoch 3/10
8/8 [==============================] - 0s 18ms/step - loss: 0.2162 - acc: 0.7445
Epoch 4/10
8/8 [==============================] - 0s 19ms/step - loss: 0.2081 - acc: 0.7445
Epoch 5/10
8/8 [==============================] - 0s 23ms/step - loss: 0.1991 - acc: 0.7533
Epoch 6/10
8/8 [==============================] - 0s 19ms/step - loss: 0.2048 - acc: 0.7048
Epoch 7/10
8/8 [==============================] - 0s 19ms/step - loss: 0.2013 - acc: 0.7533
Epoch 8/10
8/8 [==============================] - 0s 18ms/step - loss: 0.1919 - acc: 0.7357
Epoch 9/10
8/8 [==============================] - 0s 18ms/step - loss: 0.1917 - acc: 0.7621
Epoch 10/10
8/8 [==============================] - 0s 18ms/step - loss: 0.1962 - acc: 0.7533
<keras.callbacks.History at 0x7f21d23c0390>
```

نتیجه ارزیابی مدل:

```
model.evaluate(X_test,y_test)
```

```
2/2 [==============================] - 2s 12ms/step - loss: 0.1230 - acc: 0.8696
[0.12303298711776733, 0.8695651888847351]
```





نتیجه سه ابعاد متفاوت:

با ابعاد کوچکتر نمیتوان نتیجه مطلوبی دریافت کرد

```
model = Sequential()
model.add(LSTM(8, input_shape=(10,145), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(4, input_shape=(10,8), return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(2,activation='relu'))
model.add(Dense(1,activation='relu'))
```

```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(data_timestep, label_timestep, epochs=10,callbacks=[tensorboard_callback])
```

```
Epoch 1/10
8/8 [==============================] - 4s 18ms/step - loss: 0.7709 - acc: 0.2291
Epoch 2/10
8/8 [==============================] - 0s 18ms/step - loss: 0.7709 - acc: 0.2291
Epoch 3/10
8/8 [==============================] - 0s 20ms/step - loss: 0.7709 - acc: 0.2291
Epoch 4/10
8/8 [==============================] - 0s 18ms/step - loss: 0.7709 - acc: 0.2291
Epoch 5/10
8/8 [==============================] - 0s 18ms/step - loss: 0.7709 - acc: 0.2291
Epoch 6/10
8/8 [==============================] - 0s 19ms/step - loss: 0.7709 - acc: 0.2291
Epoch 7/10
8/8 [==============================] - 0s 19ms/step - loss: 0.7709 - acc: 0.2291
Epoch 8/10
8/8 [==============================] - 0s 18ms/step - loss: 0.7709 - acc: 0.2291
Epoch 9/10
8/8 [==============================] - 0s 18ms/step - loss: 0.7709 - acc: 0.2291
Epoch 10/10
8/8 [==============================] - 0s 17ms/step - loss: 0.7709 - acc: 0.2291
<keras.callbacks.History at 0x7f21c5938890>
```

```
model = Sequential()
model.add(LSTM(16, input_shape=(10,145), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(8, input_shape=(10,16), return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(4,activation='relu'))
model.add(Dense(1,activation='relu'))
```

```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(data_timestep, label_timestep, epochs=10,callbacks=[tensorboard_callback])
```

```
Epoch 1/10
8/8 [==============================] - 4s 19ms/step - loss: 0.5731 - acc: 0.2423
Epoch 2/10
8/8 [==============================] - 0s 21ms/step - loss: 0.2945 - acc: 0.5110
Epoch 3/10
8/8 [==============================] - 0s 20ms/step - loss: 0.2324 - acc: 0.6432
Epoch 4/10
8/8 [==============================] - 0s 19ms/step - loss: 0.2174 - acc: 0.7269
Epoch 5/10
8/8 [==============================] - 0s 19ms/step - loss: 0.2328 - acc: 0.7181
Epoch 6/10
8/8 [==============================] - 0s 20ms/step - loss: 0.2114 - acc: 0.7225
Epoch 7/10
8/8 [==============================] - 0s 19ms/step - loss: 0.2109 - acc: 0.7181
Epoch 8/10
8/8 [==============================] - 0s 23ms/step - loss: 0.2162 - acc: 0.6872
Epoch 9/10
8/8 [==============================] - 0s 20ms/step - loss: 0.1986 - acc: 0.6960
Epoch 10/10
8/8 [==============================] - 0s 20ms/step - loss: 0.2078 - acc: 0.7313
<keras.callbacks.History at 0x7f21c3daaed0>
```

```
model = Sequential()
model.add(LSTM(64, input_shape=(10,145), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(32, input_shape=(10,64), return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(8,activation='relu'))
model.add(Dense(1,activation='relu'))
```

```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(data_timestep, label_timestep, epochs=10,callbacks=[tensorboard_callback])
```

```
Epoch 1/10
8/8 [==============================] - 4s 24ms/step - loss: 0.4035 - acc: 0.4229
Epoch 2/10
8/8 [==============================] - 0s 23ms/step - loss: 0.2688 - acc: 0.7577
Epoch 3/10
8/8 [==============================] - 0s 26ms/step - loss: 0.2018 - acc: 0.7357
Epoch 4/10
8/8 [==============================] - 0s 23ms/step - loss: 0.1933 - acc: 0.7577
Epoch 5/10
8/8 [==============================] - 0s 26ms/step - loss: 0.2035 - acc: 0.7621
Epoch 6/10
8/8 [==============================] - 0s 24ms/step - loss: 0.1931 - acc: 0.7533
Epoch 7/10
8/8 [==============================] - 0s 23ms/step - loss: 0.2082 - acc: 0.7401
Epoch 8/10
8/8 [==============================] - 0s 26ms/step - loss: 0.1941 - acc: 0.7665
Epoch 9/10
8/8 [==============================] - 0s 24ms/step - loss: 0.1930 - acc: 0.7445
Epoch 10/10
8/8 [==============================] - 0s 24ms/step - loss: 0.1966 - acc: 0.7489
<keras.callbacks.History at 0x7f21c2bcba90>
```

استفاده از دو لایه GRU پشت سر هم :

```
model = Sequential()
model.add(GRU(16, input_shape=(10,145), return_sequences=True))
model.add(Dropout(0.2))
model.add(GRU(8, input_shape=(10,16), return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(4,activation='relu'))
model.add(Dense(1,activation='relu'))
```

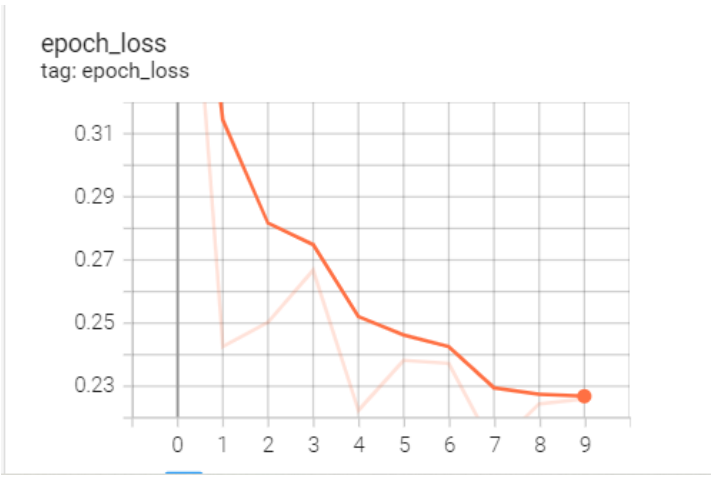
```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(data_timestep, label_timestep, epochs=10,callbacks=[tensorboard_callback])
```

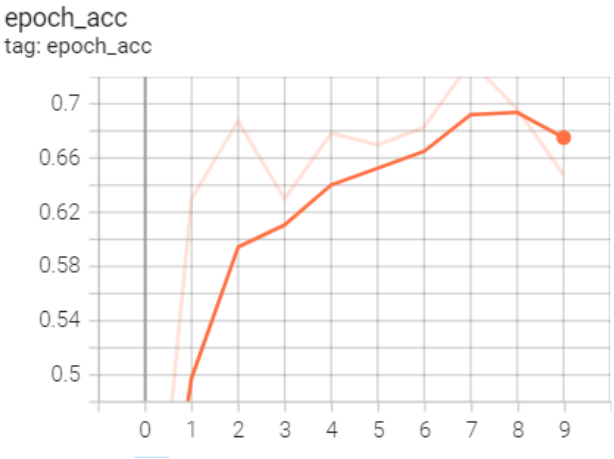
نتیجه مدل بر روی داده آموزش و ارزیابی ان:

```
Epoch 1/10
8/8 [==============================] - 4s 20ms/step - loss: 0.5471 - acc: 0.3260
Epoch 2/10
8/8 [==============================] - 0s 21ms/step - loss: 0.2971 - acc: 0.6828
Epoch 3/10
8/8 [==============================] - 0s 19ms/step - loss: 0.2927 - acc: 0.6784
Epoch 4/10
8/8 [==============================] - 0s 20ms/step - loss: 0.2715 - acc: 0.5771
Epoch 5/10
8/8 [==============================] - 0s 21ms/step - loss: 0.2657 - acc: 0.6079
Epoch 6/10
8/8 [==============================] - 0s 20ms/step - loss: 0.2533 - acc: 0.6828
Epoch 7/10
8/8 [==============================] - 0s 19ms/step - loss: 0.2559 - acc: 0.6167
Epoch 8/10
8/8 [==============================] - 0s 21ms/step - loss: 0.2495 - acc: 0.6388
Epoch 9/10
8/8 [==============================] - 0s 21ms/step - loss: 0.2552 - acc: 0.7093
Epoch 10/10
8/8 [==============================] - 0s 20ms/step - loss: 0.2399 - acc: 0.7137
<keras.callbacks.History at 0x7f21cfbdf790>
```

```
model.evaluate(X_test,y_test)
```

```
2/2 [==============================] - 1s 9ms/step - loss: 0.2440 - acc: 0.5217
[0.24404744803905487, 0.52173912525177]
```





سه ابعاد مختلف: همانطور که مشاهده میشود حتی با ابعاد کم نیز نتیجه مطلوبی حاصل میشود درحالیکه وقتی ابعاد زیاد میشود دقت افت پیدا میکند

```
model = Sequential()
model.add(GRU(16, input_shape=(10,145), return_sequences=True))
model.add(Dropout(0.2))
model.add(GRU(8, input_shape=(10,16), return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(4,activation='relu'))
model.add(Dense(1,activation='relu'))
```

```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(data_timestep, label_timestep, epochs=10,callbacks=[tensorboard_callback])
```

```
Epoch 1/10
8/8 [==============================] - 9s 20ms/step - loss: 0.4146 - acc: 0.5419
Epoch 2/10
8/8 [==============================] - 0s 21ms/step - loss: 0.2941 - acc: 0.5066
Epoch 3/10
8/8 [==============================] - 0s 19ms/step - loss: 0.2764 - acc: 0.5463
Epoch 4/10
8/8 [==============================] - 0s 20ms/step - loss: 0.2739 - acc: 0.6608
Epoch 5/10
8/8 [==============================] - 0s 19ms/step - loss: 0.2472 - acc: 0.6344
Epoch 6/10
8/8 [==============================] - 0s 20ms/step - loss: 0.2430 - acc: 0.6388
Epoch 7/10
8/8 [==============================] - 0s 19ms/step - loss: 0.2372 - acc: 0.6828
Epoch 8/10
8/8 [==============================] - 0s 21ms/step - loss: 0.2179 - acc: 0.7137
Epoch 9/10
8/8 [==============================] - 0s 21ms/step - loss: 0.2304 - acc: 0.6696
Epoch 10/10
8/8 [==============================] - 0s 23ms/step - loss: 0.2265 - acc: 0.7004
<keras.callbacks.History at 0x7f21c0211d10>
```

```
model = Sequential()
model.add(GRU(32, input_shape=(10,145), return_sequences=True))
model.add(Dropout(0.2))
model.add(GRU(8, input_shape=(10,32), return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(4,activation='relu'))
model.add(Dense(1,activation='relu'))
```

```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(data_timestep, label_timestep, epochs=10,callbacks=[tensorboard_callback])
```

```
Epoch 1/10
8/8 [==============================] - 5s 20ms/step - loss: 0.3538 - acc: 0.6035
Epoch 2/10
8/8 [==============================] - 0s 23ms/step - loss: 0.2627 - acc: 0.6388
Epoch 3/10
8/8 [==============================] - 0s 24ms/step - loss: 0.2622 - acc: 0.6520
Epoch 4/10
8/8 [==============================] - 0s 22ms/step - loss: 0.2333 - acc: 0.6696
Epoch 5/10
8/8 [==============================] - 0s 27ms/step - loss: 0.2469 - acc: 0.6167
Epoch 6/10
8/8 [==============================] - 0s 27ms/step - loss: 0.2324 - acc: 0.6872
Epoch 7/10
8/8 [==============================] - 0s 21ms/step - loss: 0.2240 - acc: 0.7225
Epoch 8/10
8/8 [==============================] - 0s 20ms/step - loss: 0.2070 - acc: 0.6784
Epoch 9/10
8/8 [==============================] - 0s 22ms/step - loss: 0.2232 - acc: 0.7269
Epoch 10/10
8/8 [==============================] - 0s 24ms/step - loss: 0.2039 - acc: 0.7269
<keras.callbacks.History at 0x7f21c09abb90>
```

```
model = Sequential()
model.add(GRU(128, input_shape=(10,145), return_sequences=True))
model.add(Dropout(0.2))
model.add(GRU(8, input_shape=(10,128), return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(16,activation='relu'))
model.add(Dense(1,activation='relu'))
```

```
model.compile(loss='mean_squared_error', optimizer='adam',metrics=["acc"])
model.fit(data_timestep, label_timestep, epochs=10,callbacks=[tensorboard_callback])
```

```
Epoch 1/10
8/8 [==============================] - 4s 30ms/step - loss: 0.3901 - acc: 0.4405
Epoch 2/10
8/8 [==============================] - 0s 29ms/step - loss: 0.2241 - acc: 0.6696
Epoch 3/10
8/8 [==============================] - 0s 31ms/step - loss: 0.2367 - acc: 0.6740
Epoch 4/10
8/8 [==============================] - 0s 28ms/step - loss: 0.2192 - acc: 0.6784
Epoch 5/10
8/8 [==============================] - 0s 30ms/step - loss: 0.2124 - acc: 0.7004
Epoch 6/10
8/8 [==============================] - 0s 27ms/step - loss: 0.2189 - acc: 0.6740
Epoch 7/10
8/8 [==============================] - 0s 30ms/step - loss: 0.2136 - acc: 0.6608
Epoch 8/10
8/8 [==============================] - 0s 28ms/step - loss: 0.2192 - acc: 0.6872
Epoch 9/10
8/8 [==============================] - 0s 31ms/step - loss: 0.2091 - acc: 0.6916
Epoch 10/10
8/8 [==============================] - 0s 29ms/step - loss: 0.2155 - acc: 0.6696
<keras.callbacks.History at 0x7f21bf814e90>
```

همانطور که مشاهده میشود مدل LSTM با افزایش ابعاد دقت بهتری پیدا می‌کند ولی GRU در ابعاد پایین نتیجه مناسبی می دهد و بعد از مرحله‌ایی دقت افت پیدا می‌کند.