به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

شبکه های عصبی

مینی پروژه 2

نام و نام خانوادگی:

یلدا فروتن(810196265)

محمد دهقان روزی(810197243)

پاییز 1398

فهرست

چکیده

در این قسمت به صورت چکیده هدف از این سری تمرین برای سوال اول آشنایی با سری های زمانی و کاربرد شبکه های LSTM و GRU و RNN در بازارهای سهام گوگل و اپل بود.

در سوال دوم نیز با استفاده از شبکه های فوق به تولید متن و محتوی پرداخته می شود.

سوال 1

(1

برای این بخش کد q1_a1.py زده شده است که صرفا نمودار را نمایش می دهد:

کد به صورت زیر می باشد:

ابتدا کتابخانه های لازم لود می شوند و تابع موردنظر برای لود داده ها نوشته می شود.

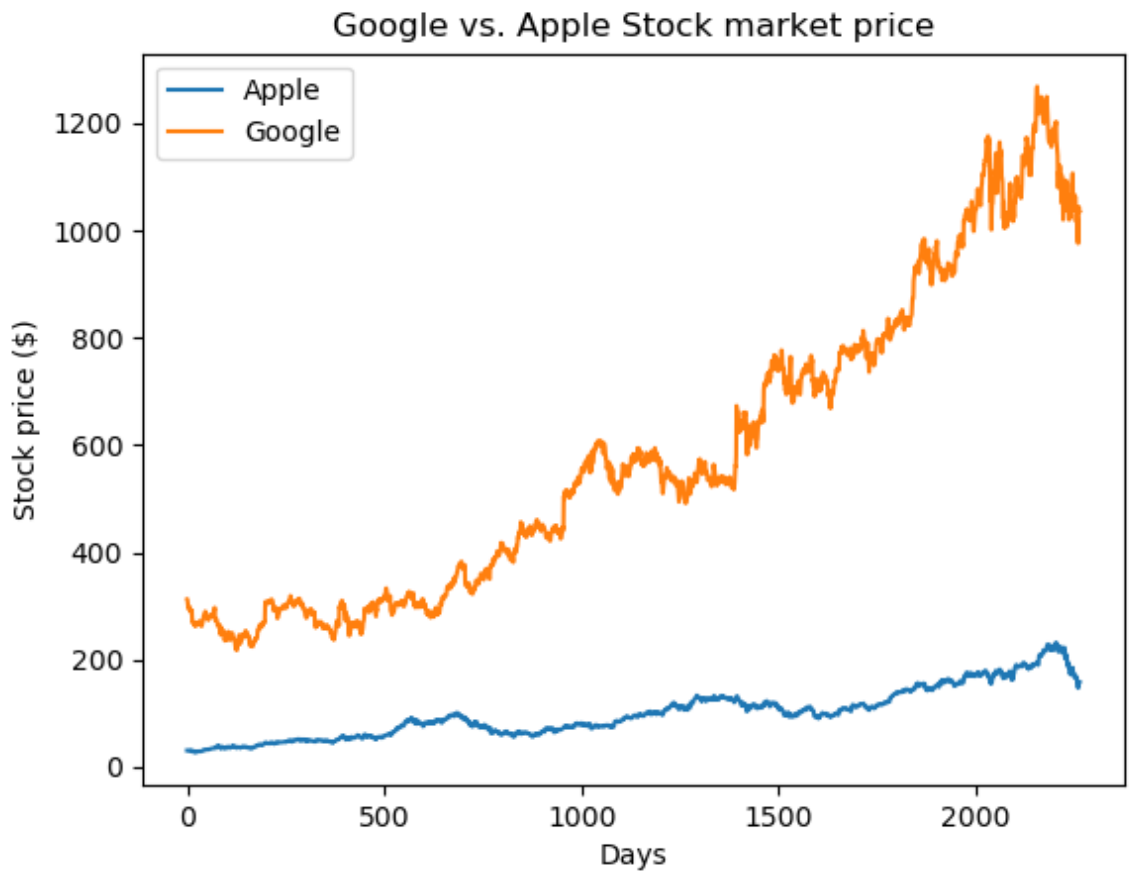
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import datetime
import pandas_datareader.data as web
import os.path


def data_loader():
    company_name = 'AAPL'
    a = os.path.isfile('apple.csv')
    if a is False:
        start = datetime.datetime(2010, 1, 1)
        end = datetime.datetime(2019, 1, 1)
        df_aapl1 = web.DataReader(company_name, 'yahoo', start, end)
        df_aapl1.to_csv(r'apple.csv', sep='\t', encoding='utf-8', header='true')
        df_aapl1 = pd.read_csv('apple.csv', sep='\t')
    else:
        df_aapl1 = pd.read_csv('apple.csv', sep='\t')
    # print(df_aapl)

    company_name = 'GOOG'
    a = os.path.isfile('google.csv')
    if a is False:
        start = datetime.datetime(2010, 1, 1)
```

در تابع فوق در صورتی که داده قبلا دانلود شده باشد، از داده دانلود شده استفاده می کند و در غیر این صورت به صورت آنلاین داده ها را دانلود کرده و می خواند.

سپس مقادیر به صورت زیر نمایش داده می شوند:

```
''' Load data '''
df_apple, df_google = data_loader()

''' Merge values '''
df = pd.merge(df_apple, df_google, how='inner', left_index=True, right_index=True)

''' Plot '''
data_apple = df['Close_x'].to_numpy()
data_google = df['Close_y'].to_numpy()

plt.plot(np.arange(df_apple.shape[0]), data_apple, label='Apple')
plt.plot(np.arange(df_apple.shape[0]), data_google, label='Google')
plt.xlabel('Days')
plt.ylabel('Stock price ($)')
plt.title('Google vs. Apple Stock market price')
plt.legend()

plt.show()
```

خروجی نیز به صورت زیر است:



همانطور که مشاهده می شود، قیمت سهام با روز رسم شده است که برای شرکت گوگل رشد چشمگیری داشته است.

(2

برای این قسمت کد q1_lstm.py زده شده است.

در ابتدا کد را توضیح می دهیم:

```
''' Load data '''
df_apple, df_google = data_loader()

print(df_apple)
print(df_google)

''' If filling values are necessary '''
# df_apple['Date'] = pd.to_datetime(df_apple['Date'])
# df_apple = df_apple.set_index('Date').asfreq('24h', method='bfill')
# df_apple['Date_col'] = df_apple.index
#
# df_google['Date'] = pd.to_datetime(df_google['Date'])
# df_google = df_google.set_index('Date').asfreq('24h', method='bfill')
# df_google['Date_col'] = df_google.index

''' Merge values '''
df = pd.merge(df_apple, df_google, how='inner', left_index=True, right_index=True)
print(df)
```

در کد فوق داده های دو شرکت به صورت دیتافریم ذخیره شده اند. برای بخش بعد (filling values)، برای تست آنکه به ازای مقادیر خالی چه عددی قرار داده شود، می توانستیم داده های روز قبل را برای روز بعد ذخیره کنیم. که اینکار در این قسمت انجام نشد و فرض شد که داده ها به همان صورت می باشند. سپس داده های شرکت در دیتا فریم merge شده اند.

در بخش بعد داریم:

```
''' data and target '''
data = df[['High_x', 'Low_x', 'Open_x', 'Volume_x', 'Adj Close_x',
           'High_y', 'Low_y', 'Open_y', 'Volume_y', 'Adj Close_y', 'Close_x', 'Close_y']].to_numpy()
print(data.shape)

'''Scale data '''
from sklearn import preprocessing

# MinMax
scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(data)
print(data)

# Normalize
# data = preprocessing.normalize(data)
# print(data.shape)
# print(data.std(axis=0))
```

در کد فوق، داده ها خوانده شده و نرمالایز می شوند.

```
''' Generate time series data '''
period = 31
data_list = data_for_training(data, period)
print(data_list.shape)
target_list = np.zeros((len(data_list), 2))
for i in range(len(data_list)):
    value = data_list[i][30][10]
    value1 = data_list[i][30][11]
    target_list[i][0] = value
    target_list[i][1] = value1

data_list = data_list[:, :30, :]
print(data_list.shape)
print(target_list.shape)
print(target_list)

''' Split data '''
my_test_size = 0.1
ratio_train = int((1 - my_test_size) * len(data_list))
x_train = data_list[:ratio_train]
x_test = data_list[ratio_train:]
y_train = target_list[:ratio_train]
y_test = target_list[ratio_train:]
```

در کد فوق داده های کل با یک window با پریود 31 روز ایجاد می شود. برای ایجاد این پنجره 31 روزه نیز تابع data_for_trainig نوشته شده است. ایجاد می شوند و در بخش split داده های 10 درصد از روزهای انتهایی به عنوان داده تست جدا می گردند.

سپس مدل ساخته و آموزش داده می شود. پارامتر ها نیز در تصویر و کد مشخص می باشند:

```
''' Build model '''
model = Sequential()
model.add(LSTM(50, batch_input_shape=(None, 30, 12), return_sequences=True))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(2, activation='relu'))
model.compile(loss='mse', optimizer='adam', metrics=['mae'])
model.summary()

''' Training '''
history = model.fit(x_train, y_train, epochs=10, validation_split=0.2, verbose=1, batch_size=30)

```

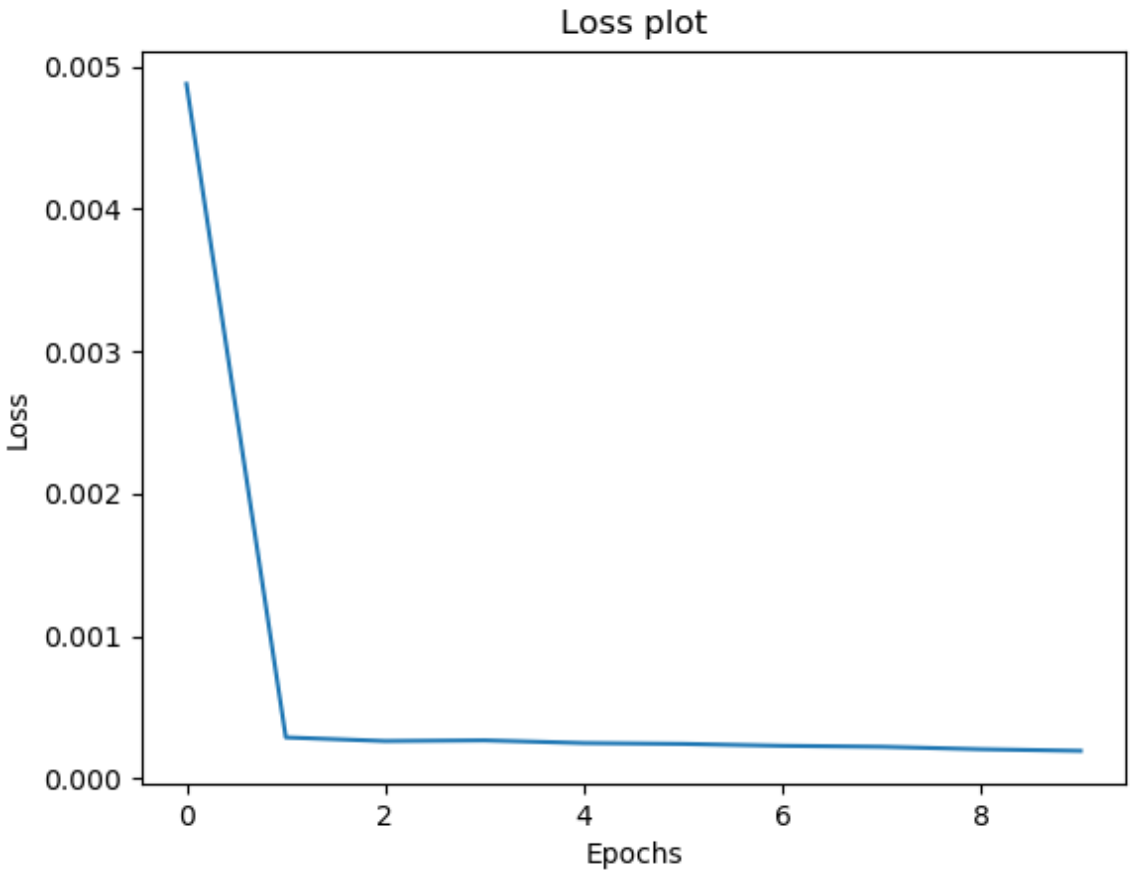
در کد فوق از دو لایه LSTM استفاده شده است (می توانست حتی تک لایه هم باشد) که هر کدام دارای 50 یونیت می باشند. در لایه آخر نیز برای خروجی دو شرکت یک لایه dense با دو نورون قرار داده شده است. تابع loss نیز mse می باشد.

سپس نتایج Visualize می گردند:

```
''' Results '''
results_train = model.evaluate(x_train, y_train)
results_test = model.evaluate(x_test, y_test)

print('train loss: {}, train acc: {}'.format(results_train[0], results_train[1]))
print('test loss: {}, test acc: {}'.format(results_test[0], results_test[1]))

plt.plot(history.history['loss'])
plt.title('Loss plot')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()

results_test = model.predict(x_test)
arr_y_test = output_inversed(scaler, y_test)
arr_results_test = output_inversed(scaler, results_test)
plt.plot(np.arange(len(results_test)), arr_y_test[:, 0], label='True label Apple')
plt.plot(np.arange(len(results_test)), arr_y_test[:, 1], label='True label Google')
plt.plot(np.arange(len(results_test)), arr_results_test[:, 0], label='predicted Apple')
plt.plot(np.arange(len(results_test)), arr_results_test[:, 1], label='predicted Google')
plt.legend()

plt.xlabel('Time in test (Day)')
plt.ylabel('Output value ($)')
plt.title('Predicted vs. True label')
plt.show()
```

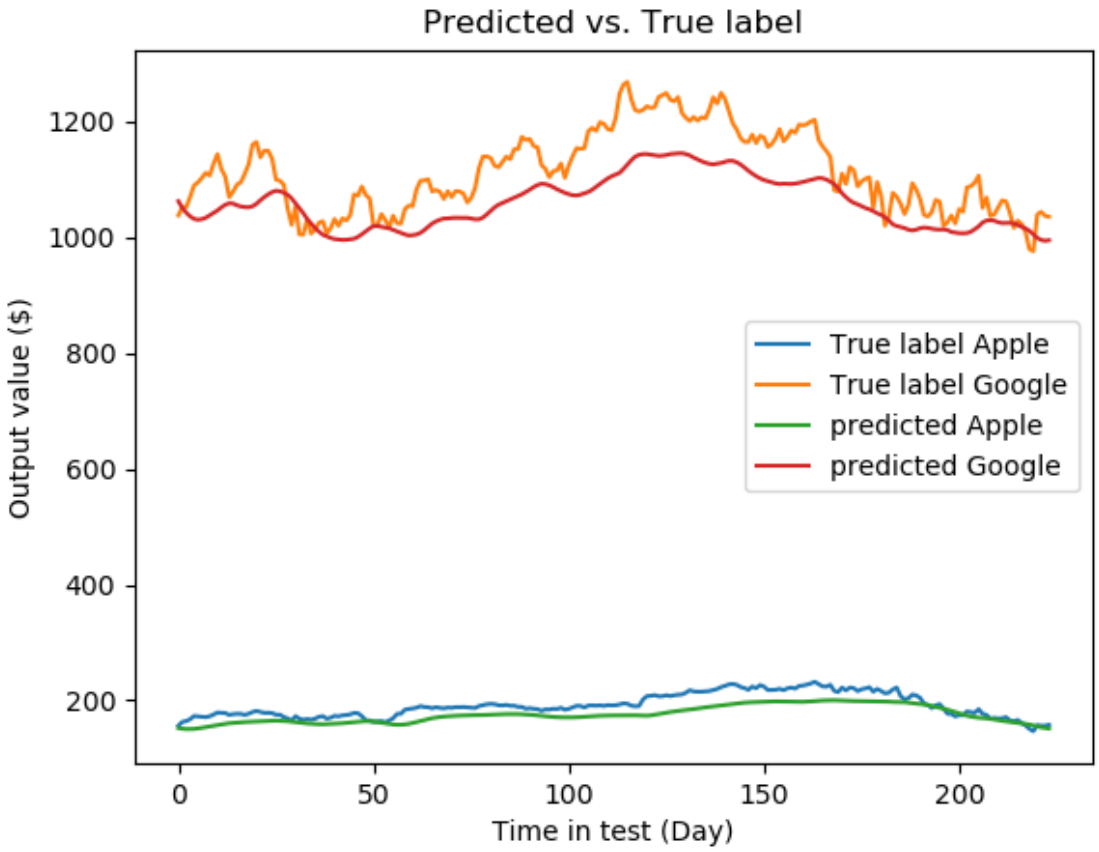
خروجی نیز به صورت زیر می باشد:

نمودار مقدار Loss:



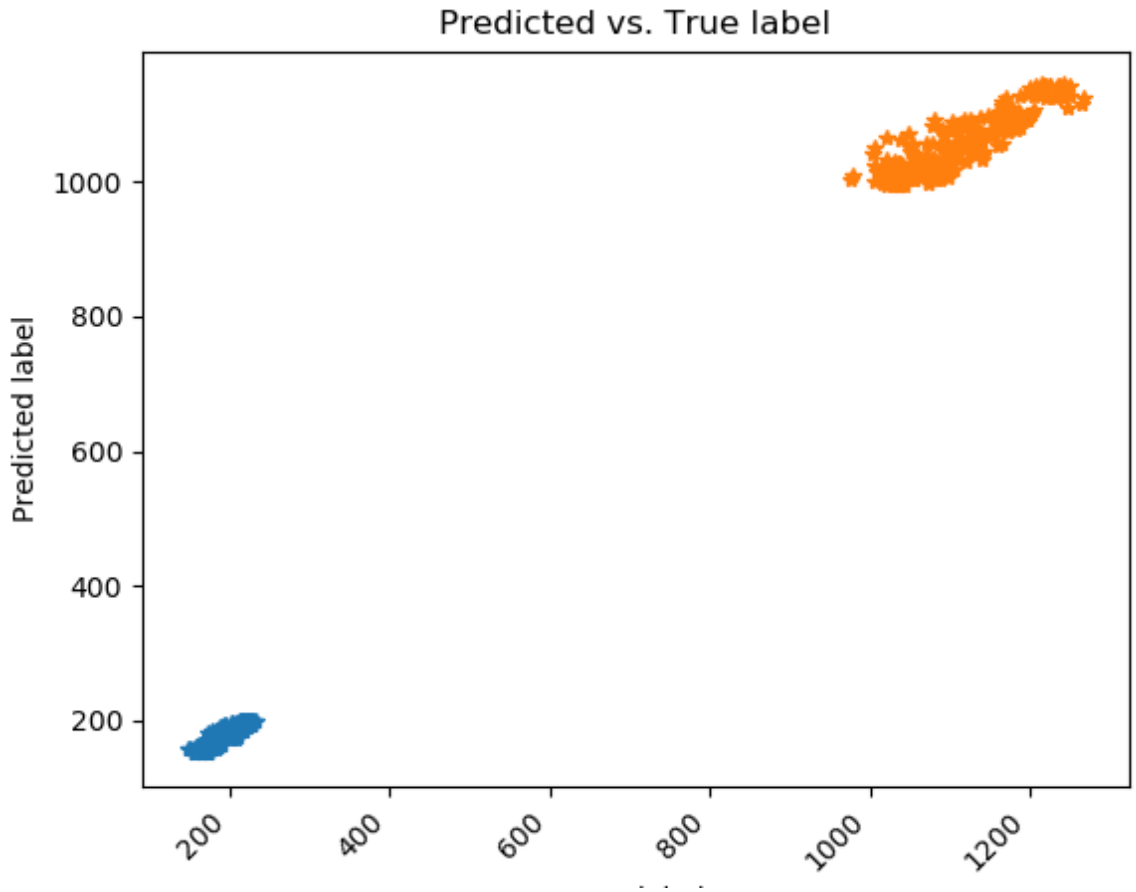
همانطور که مشاهده می شود، مقدار loss در همان epoch های اول به سرعت کاهش می یابد.

نمودار مقدار پیشبینی شده و مقدار واقعی برای ده درصد روزهای آخر:



همانطور که مشاهده می شود، خروجی نمودار از لحاظ trend شبیه می باشد.

در انتها نیز برای بررسی دقیقتر موضوع، نمودار داده های پیشبینی شده و مقدار واقعی را رسم کردیم که نتیجه به صورت زیر می باشد:



محور x مقدار True label و محور y مقدار پیشبینی شده می باشد، همانطور که مشاهده می شود، نمودار هر دو شرکت گوگل و اپل نزدیک خط y=x می باشند و پیشبینی مناسبی صورت گرفته است.

3)

برای این بخش با سه شبکه مختلف ران گرفته شد: کد q1_gru.py برای این بخش زده شده است که شبکه GRU می باشد:

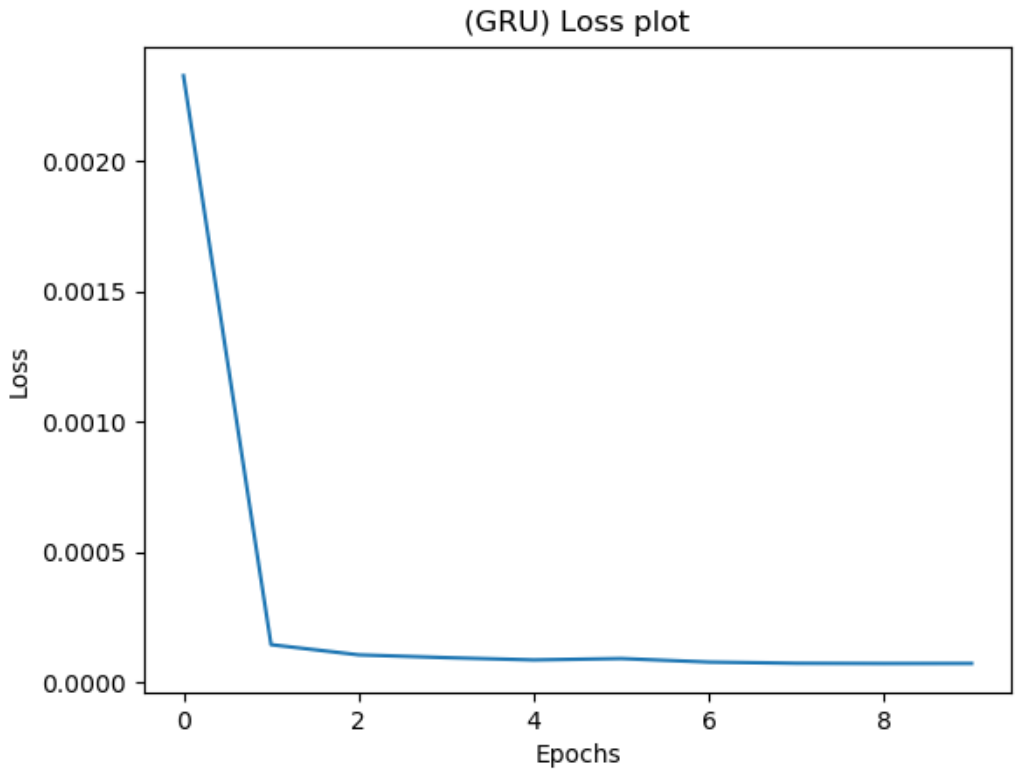
برای LSTM در قسمت قبل توضیح داده شد. برای GRU داریم:

```
''' Build model '''
model = Sequential()
model.add(GRU(50, batch_input_shape=(None, 30, 12)))
# model.add(GRU(50, batch_input_shape=(2234, 30, 12)))
model.add(Dense(2, activation='relu'))
model.compile(loss='mse', optimizer='adam', metrics=['mae'])
model.summary()
```

کد موردنظر برای این بخش به صورت فوق می باشد. با درنظر گرفتن 50 یونیت و تابع فعالساز relu و loss که mse است مدل را ساخته و کامپایل می کنیم.

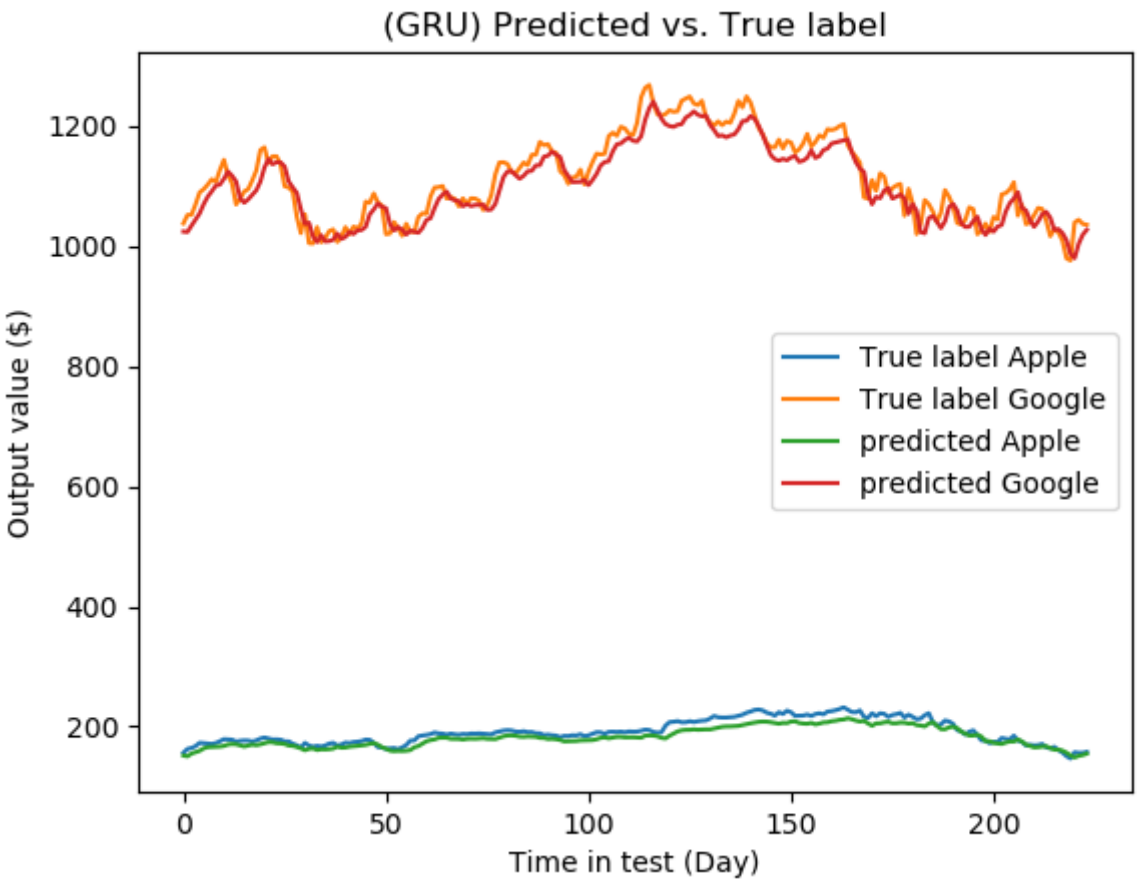
نتایج به دست آمده به صورت زیر می باشند:

برای Loss داریم:



همانطور که مشخص است، Loss الگوریتم GRU کمتر از LSTM می باشد و بهینه تر است.

برای پیشبینی ها داریم:



همانطور که مشخص است، پیشبینی GRU بهتر از LSTM عمل کرده است.

حال کد فوق را برای RNN اجرا می کنیم.

برای RNN کد q1_rnn.py زده شده است.

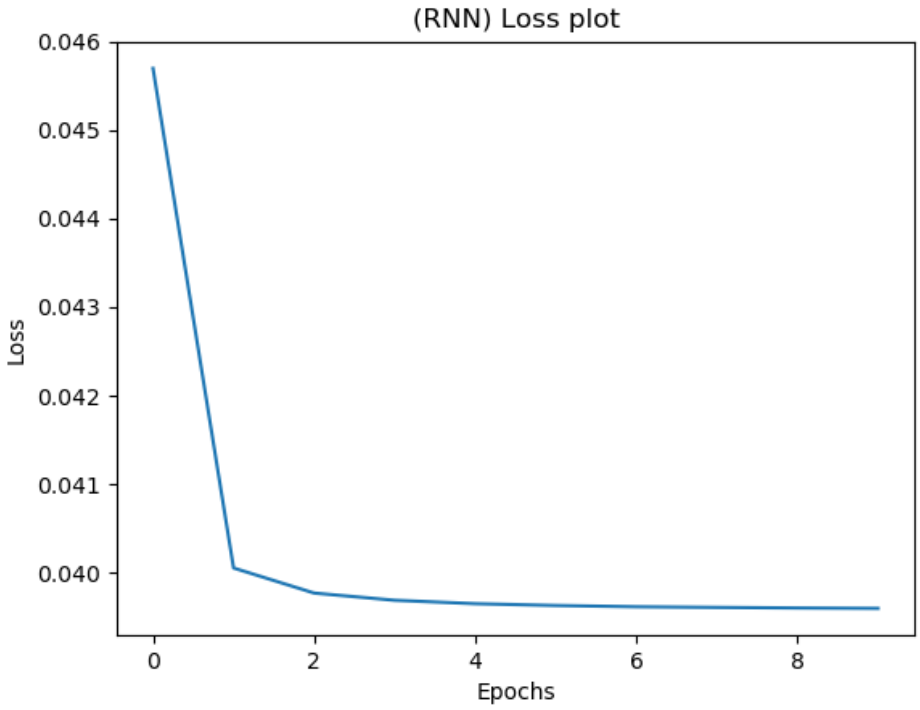
برای مدل RNN داریم:

```
125
126     ''' Build model '''
127     model = Sequential()
128     model.add(SimpleRNN(units=100, input_shape=x_train.shape[1:], activation="relu", recurrent_dropout=0.0))
129     # model.add(GRU(50, batch_input_shape=(2234, 30, 12)))
130     model.add(Dense(2, activation='relu'))
131     model.compile(loss='mse', optimizer='adam', metrics=['mae'])
132     model.summary()
```

در کد فوق با استفاده از تابع فعال ساز relu تعداد 100 یونیت را درنظر گرفته و در انتها به منظور پیشبینی خروجی دو شرکت یک لایه dense با دو نورون قرار می دهیم.

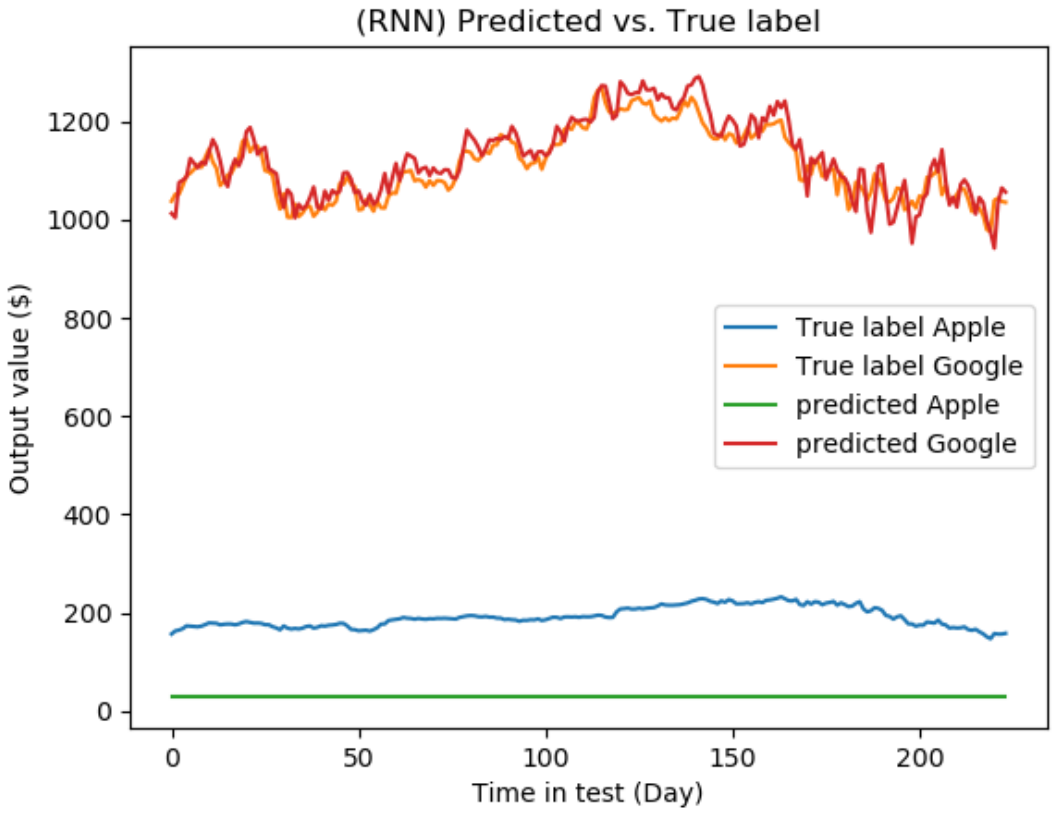
خروجی نیز به صورت زیر می باشد:

مقدار LOSS:



همانطور که مشاهده می شود، مقدار Loss الگوریتم RNN از هر دو الگوریتم دیگر بدتر می باشد و بیشترین Loss را در مقایسه با الگوریتم های فوق دارد.

مقدار پیشبینی شده:



همانطور که مشخص است، الگوریتم RNN برای گوگل بسیار با دقت پیشبینی کرده است ولی برای سهام apple نتیجه خوبی را ارائه نکرده است.

نکته: به نظر می رسد که اگر برای هر شرکت به صورت جداگانه الگوریتم را اجرا کنیم، نتایج دو شرکت بهتر می بود.

ولی چون سوال خواسته بود که در شرایط برابر الگوریتم ها مقایسه شوند، بنابراین این، برای الگوریتم RNN به صورت جدا حساب نشد.

5)

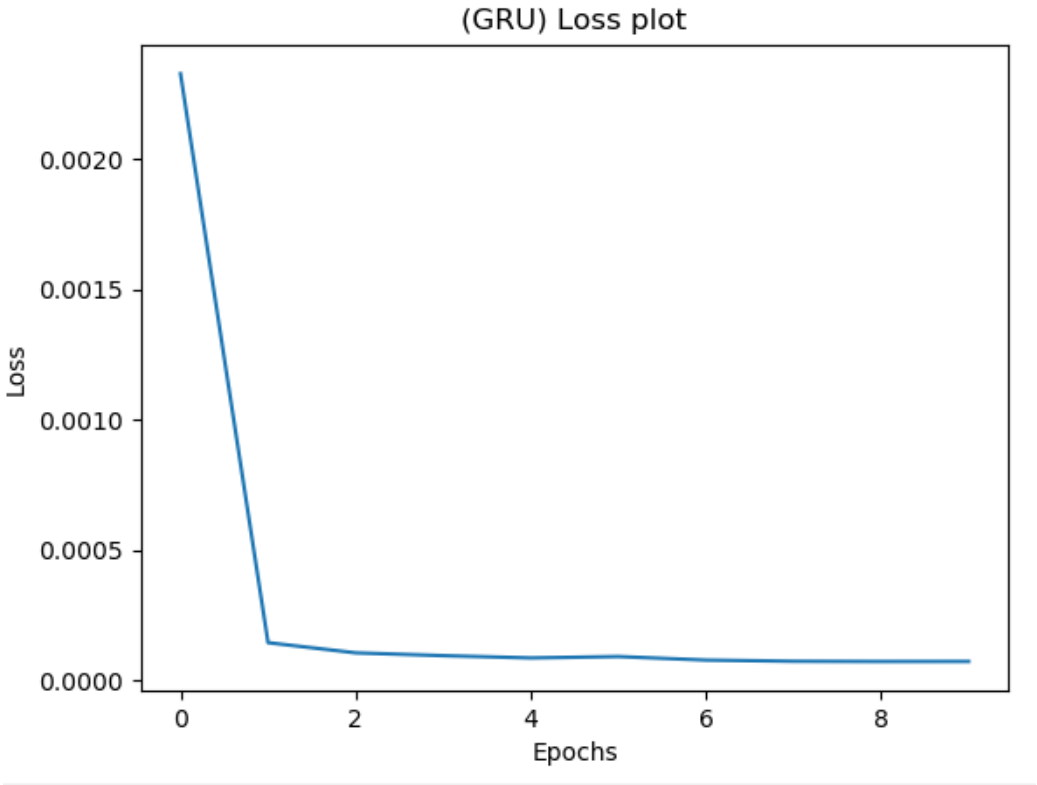
باتوجه به نتایج سه الگوریتم، به نظر می رسد الگوریتم GRU بهترین نتیجه را برای هر دو شرکت ارائه کرده است.

6)

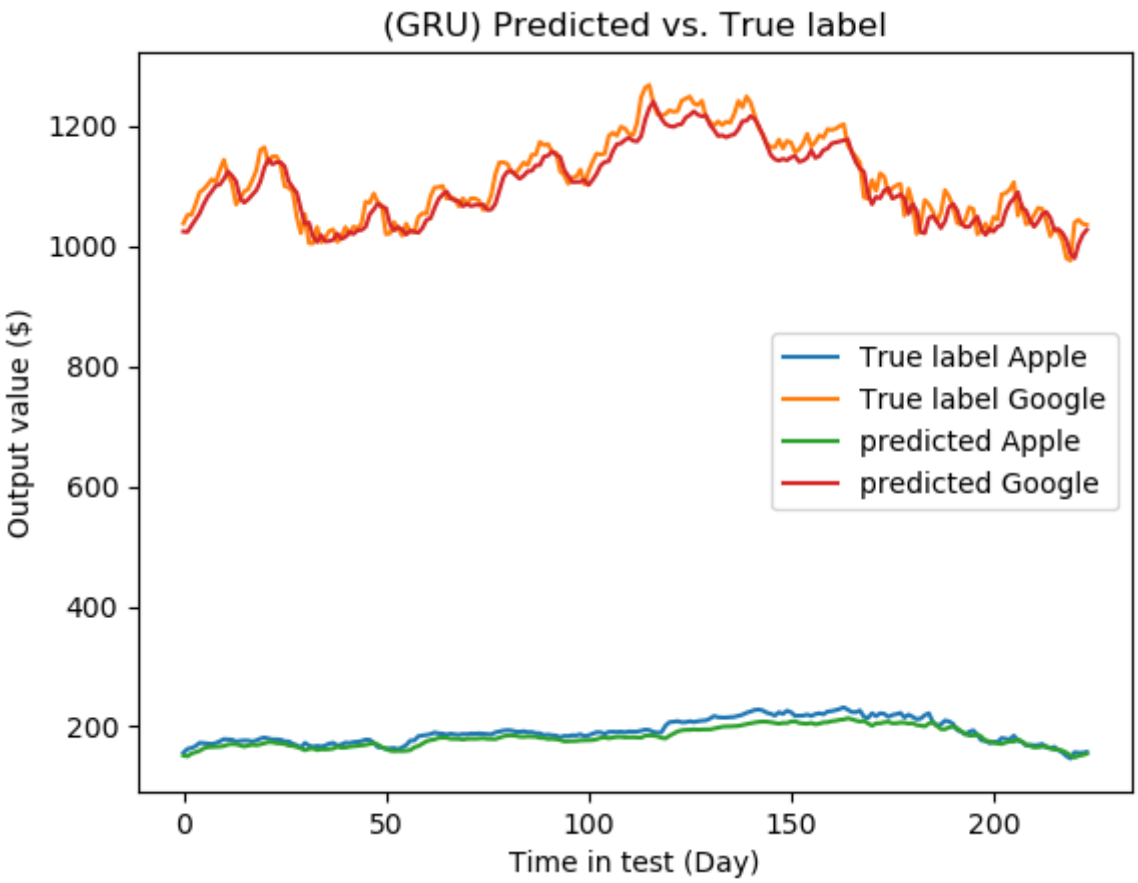
عملکرد شبکه GRU در کد q1_gru.py به صورت زیر می باشد:

برای حالت MSE داریم:

برای Loss داریم:

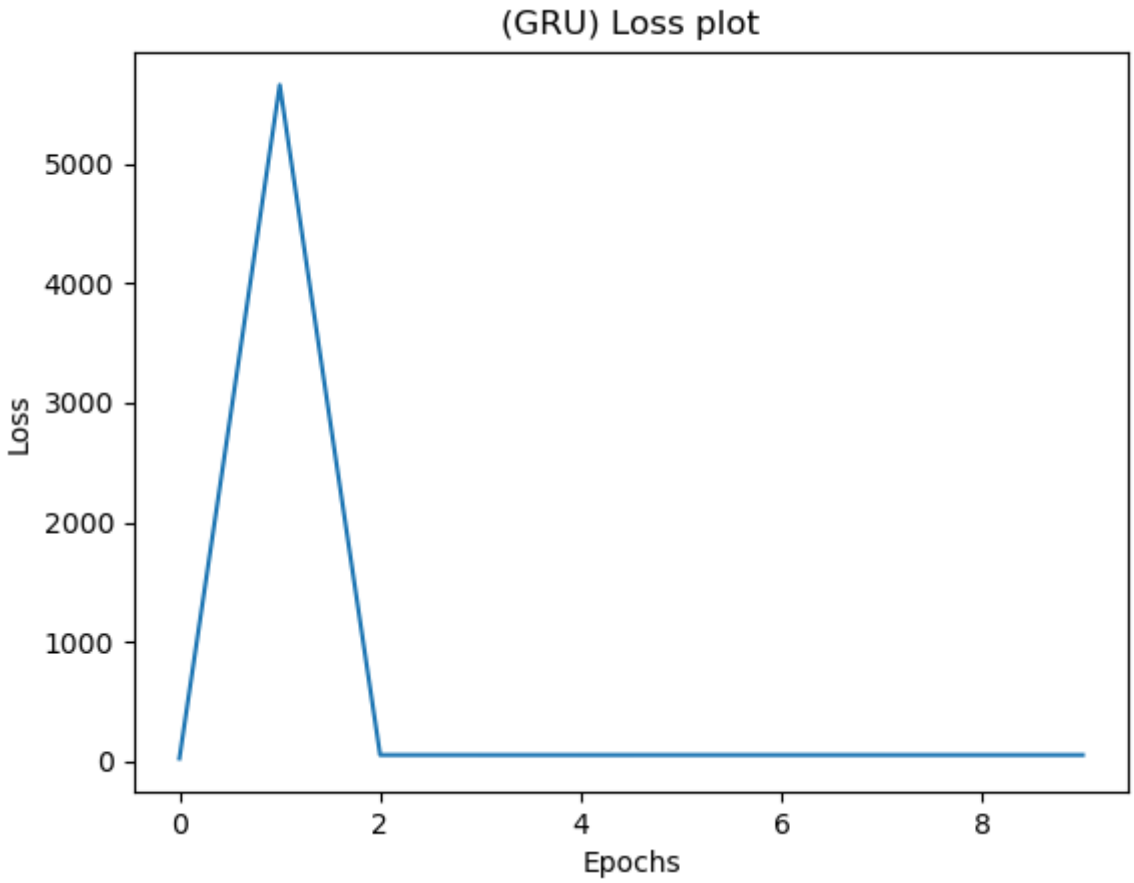


برای مقدار پیشبینی شده نیز داریم:

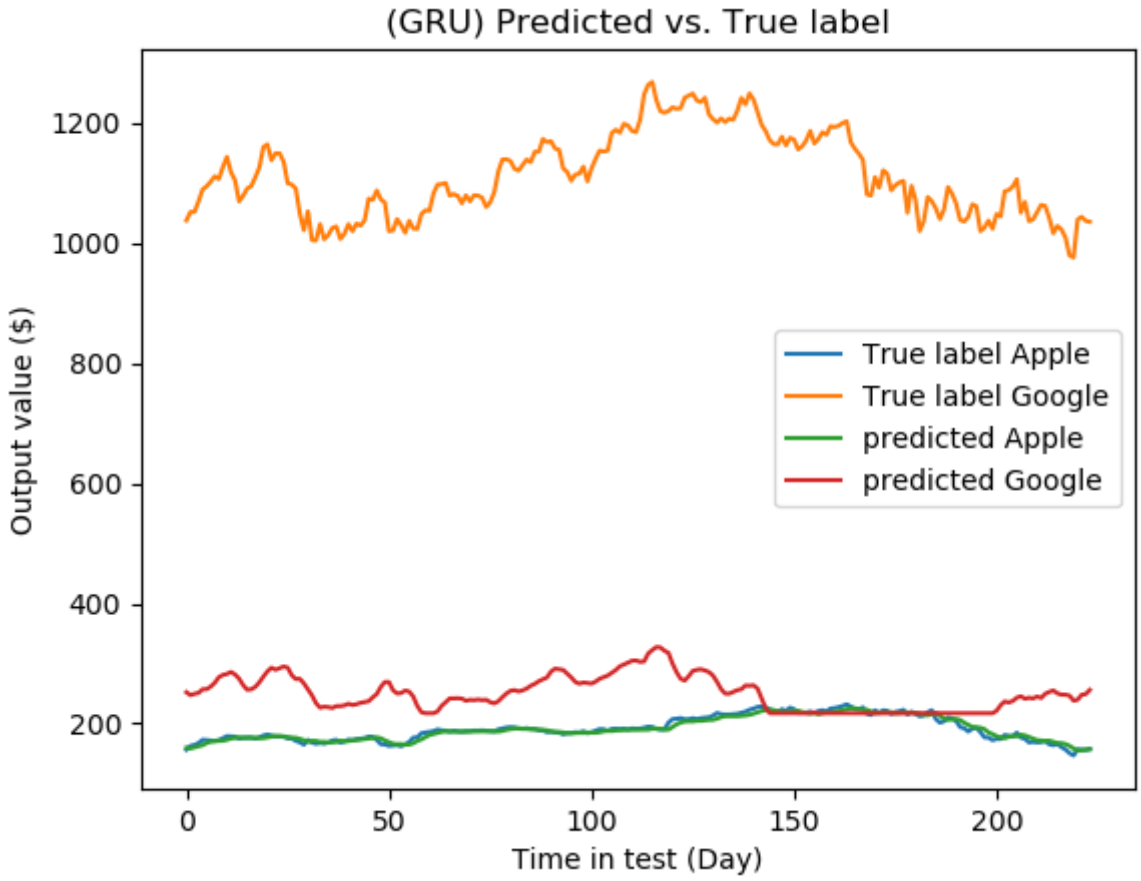


برای حالت MAPE:

برای Loss:



برای مقدار پیشبینی شده:



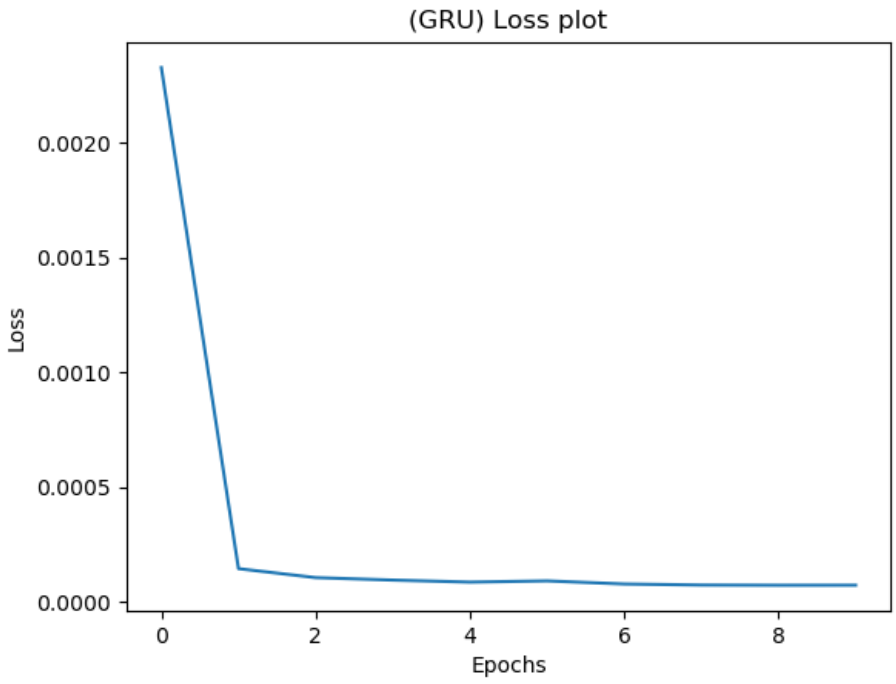
مقایسه: همانطور که مشاهده می شود، اگر تابع loss را MAPE قرار دهیم، برای یکی از شرکت ها اصلا جواب مناسبی را نمی دهد. و تابع MSE عملکرد بسیار بهتری دارد.

7)

برای اوپتیمایز های مختلف داریم:

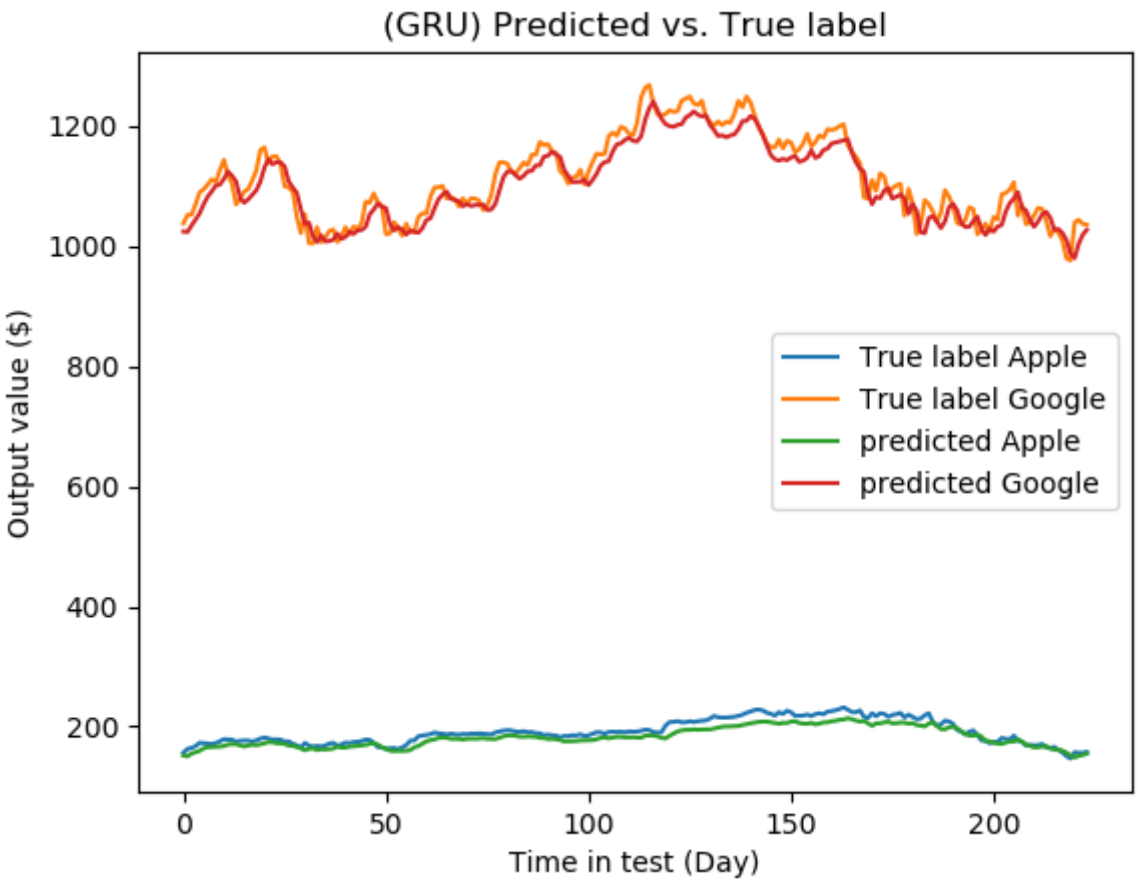
برای adam:

مقدار loss:



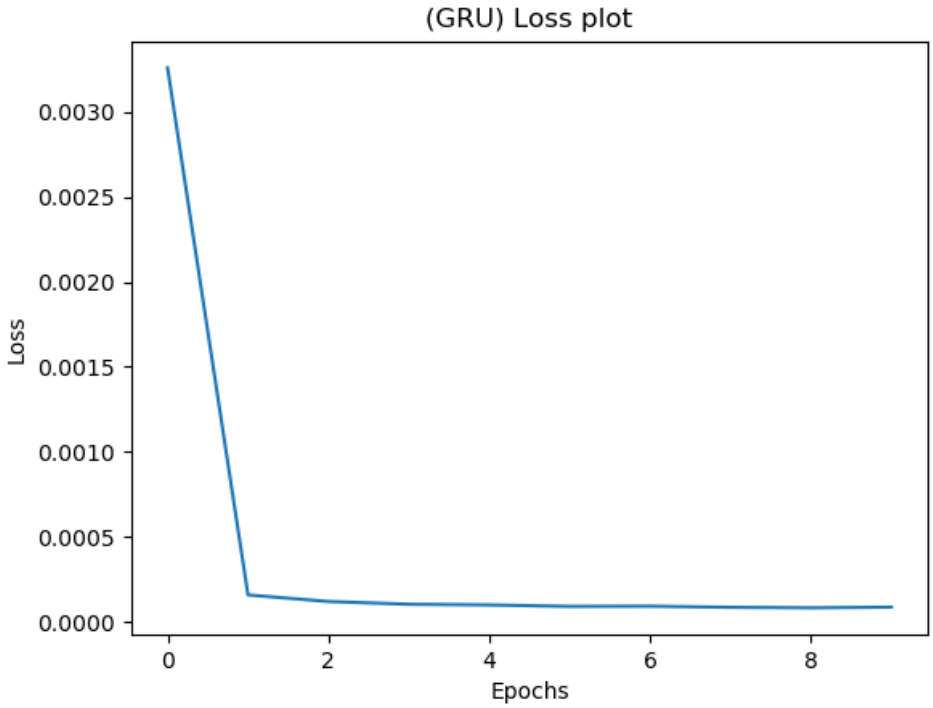
مقدار loss بعد از 10 ایپاک برای داده های تست: 0.0018967110331037215

مقداری پیشبینی شده:



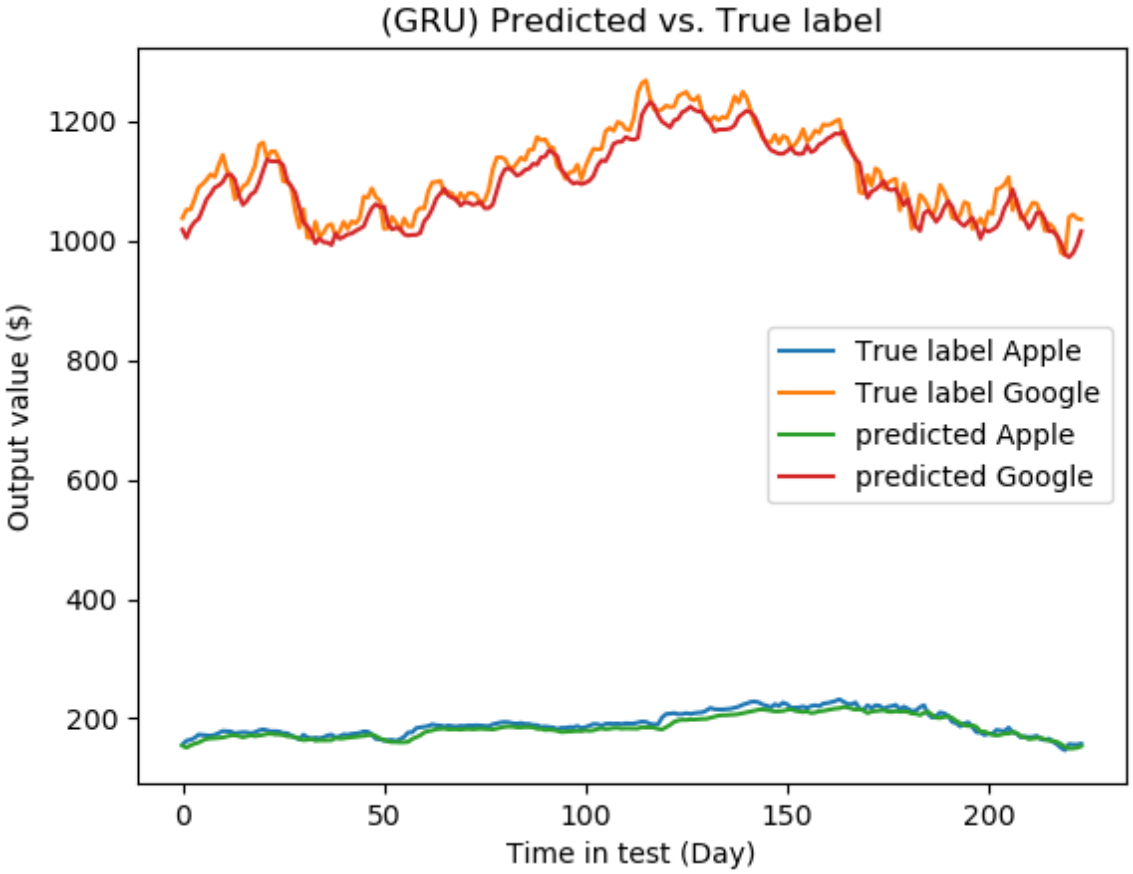
برای adagrad:

مقدار loss:



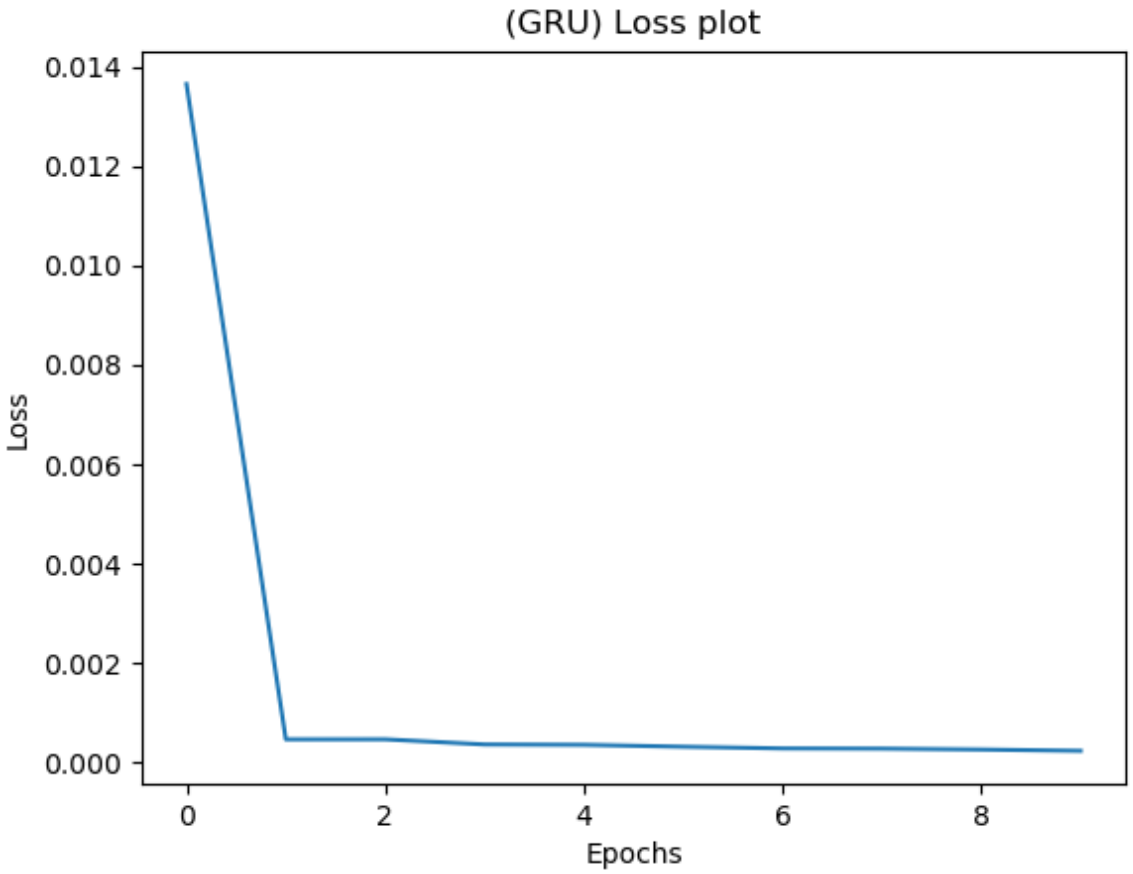
مقدار loss بعد از 10 ایپاک برای داده های تست: 0.005928242645625558

مقداری پیشبینی شده:



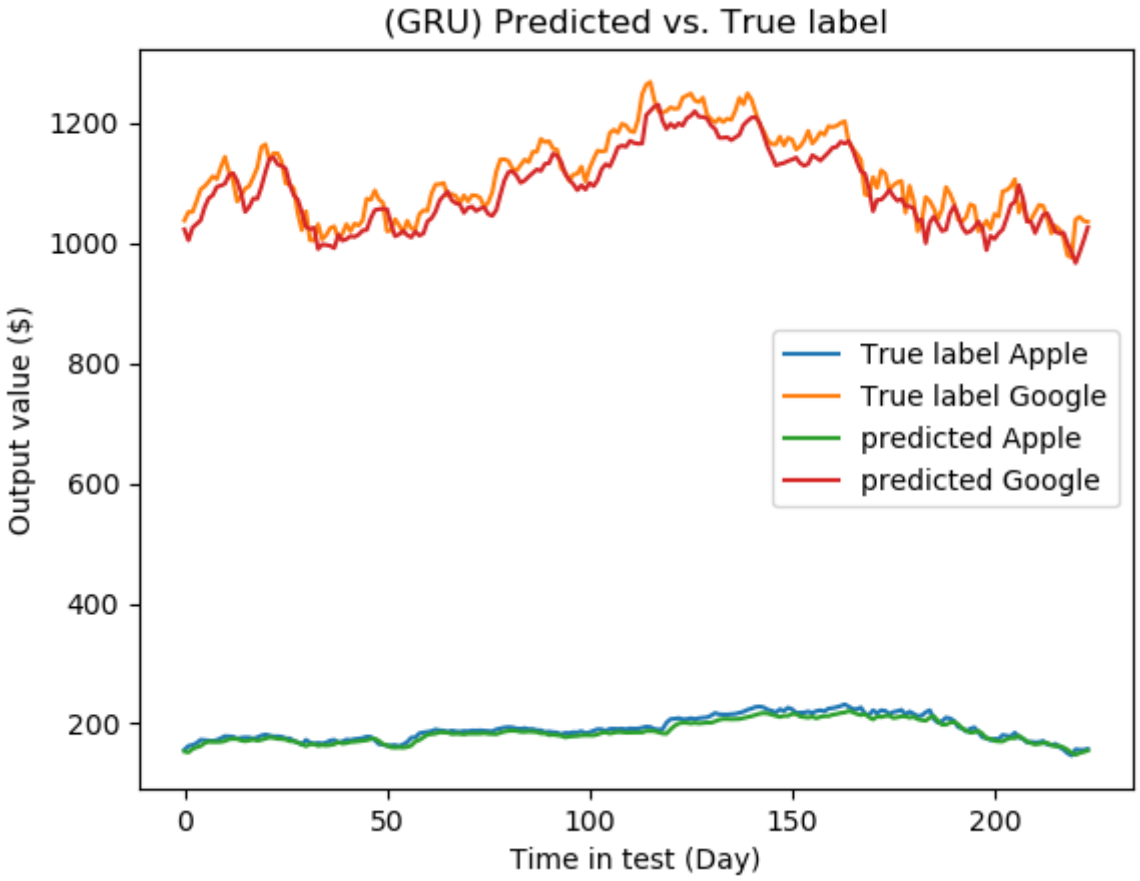
برای RMSProp:

مقدار loss:



مقدار loss بعد از 10 ایپاک برای داده های تست: 0.00464295845345727

مقداری پیشبینی شده:



مقایسه: همانطور که مشاهده می شود، الگوریتم adagrad برای داده های تست دارای loss با مقدار 0.005 می باشد و بیشترین loss را دارد. این درحالی است که الگوریتم RMSprop مقدار loss 0.004 می باشد. و مقدار loss برای adam نیز برابر 0.001 می باشد. **بنابراین بهترین الگوریتم برای بهینه سازی باتوجه بخ مقادیر loss الگوریتم adam می باشد.**

8)

تاثیر dropout:

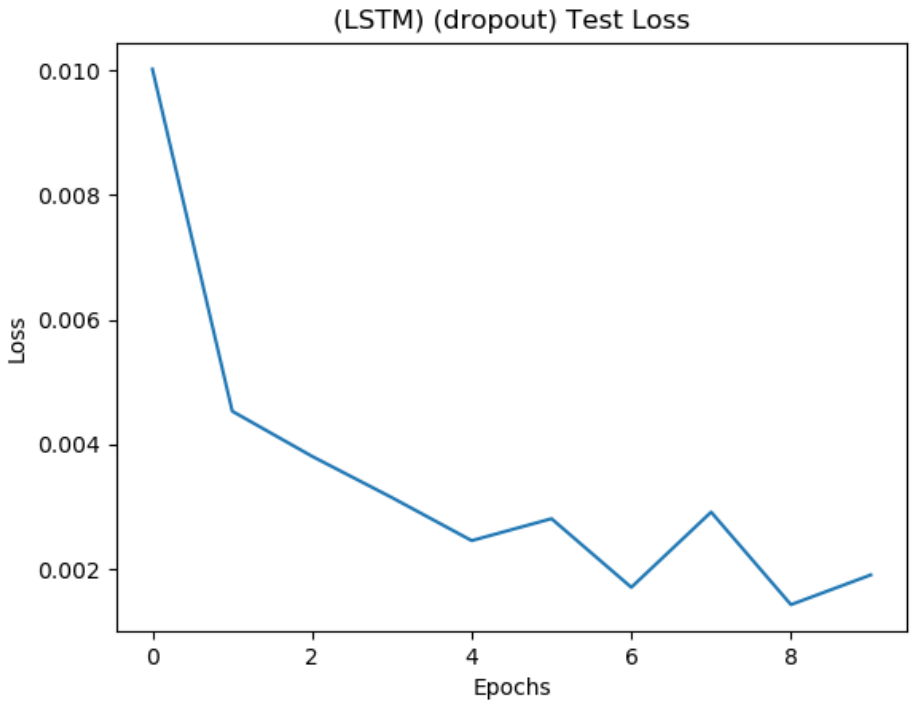
باتوجه به سه کد **q1_lstm.py** و **q1_gru.py** و **q1_rnn.py** به این کد ها بخش drop out اضافه می کنیم و نتیجه را بررسی می کنیم.

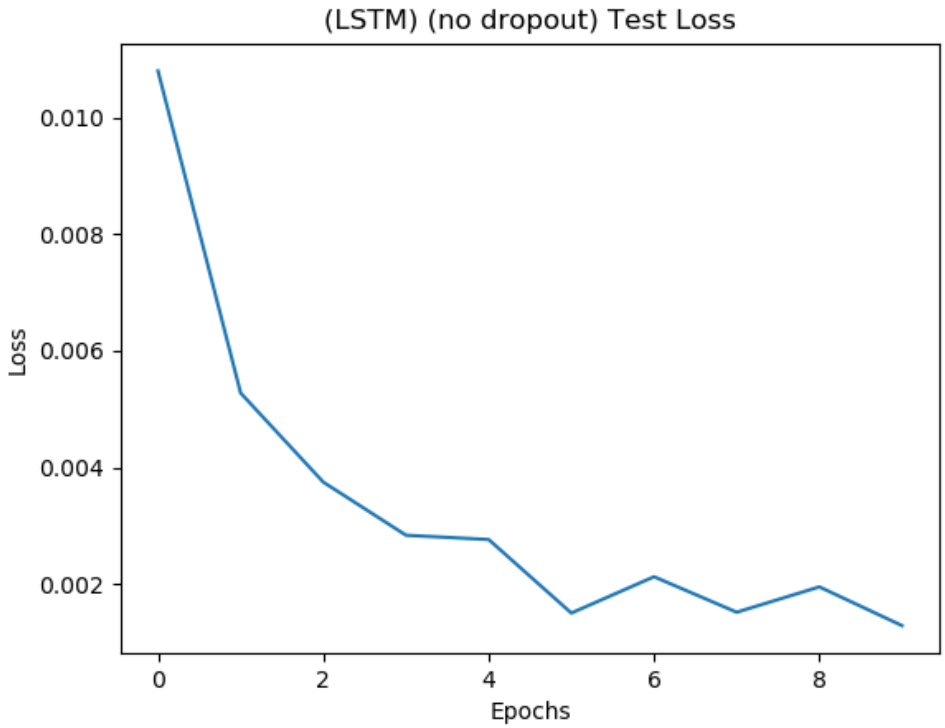
1. **تاثیر drop out بر روی LSTM:**

Network	Loss (with dropout)		Loss (no dropout)	
	train	**test**	**train**	**test**
LSTM	0.0005287	0.0095427	0.00053548	0.009753
GRU	0.0055547	0.026142	0.0001573	0.00131339
RNN	0.068784	0.36567	8.471008e-05	0.00048

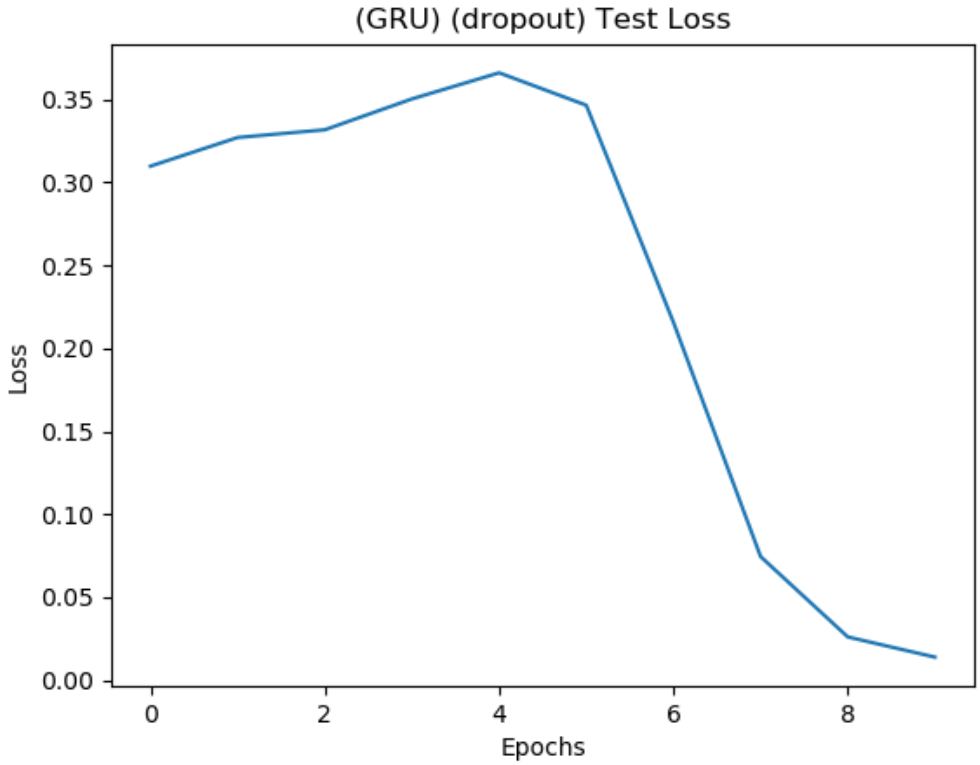
برای نمودارها نیز داریم:

برای LSTM:

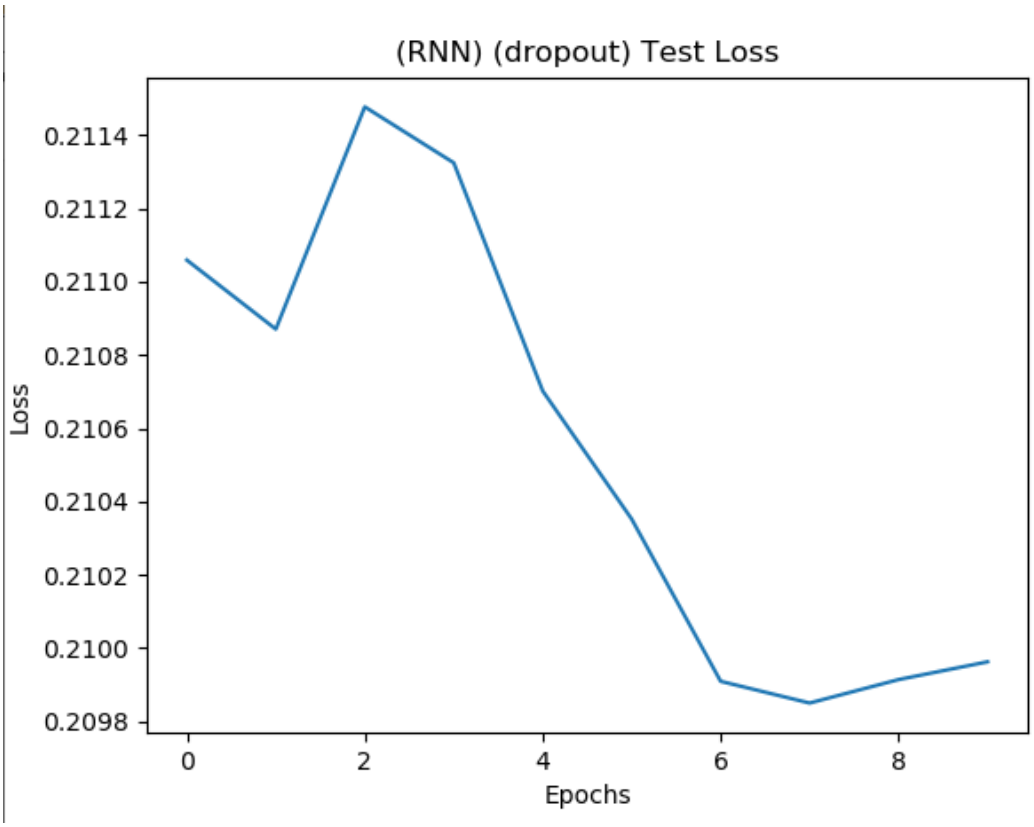


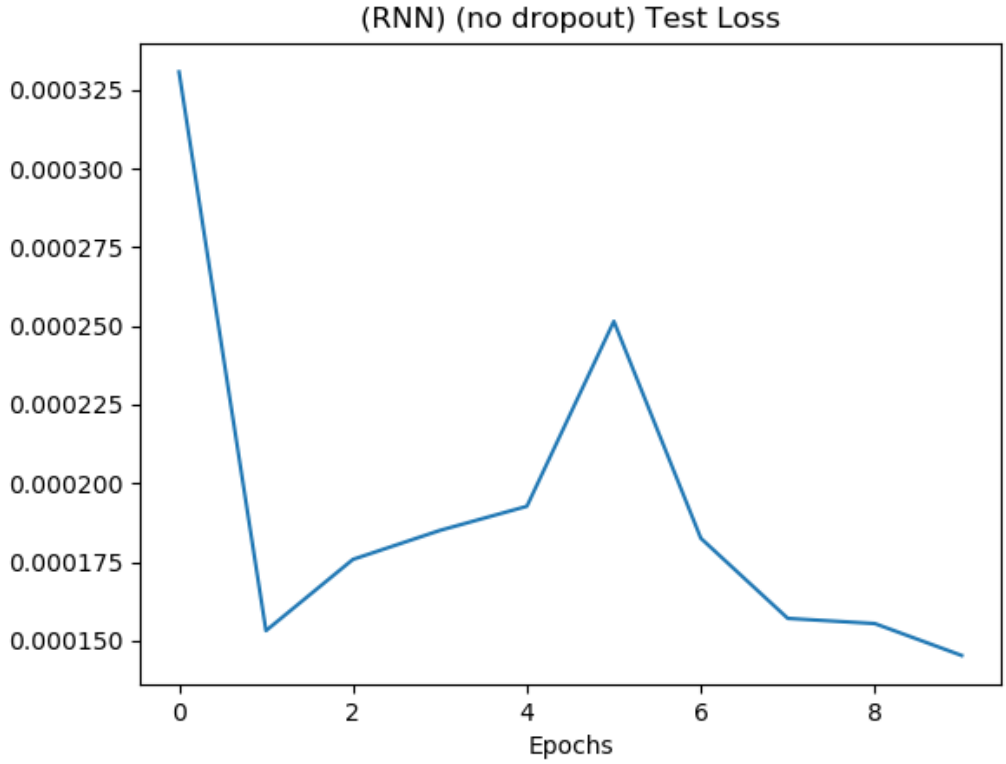


برای GRU داریم:



برای RNN نیز داریم:





تحلیل بخش 8:

در شبکه های recurrent همانطور که مشاهده شد، در تمامی شبکه ها با درنظر گرفتن droupout دقت مدل پایین می آید. همانطور که مشخص است، train loss و test loss هر دو در مواردی که dropout داریم افزایش یافته اند. پس بهتر است در این شبکه ها، recurrent dropout درنظر نگیریم.

سوال 2

سوال ۲ – طراحی شبکه Recurrent برای تولید متن

یکی از کاربردهای شبکه‌های عصبی recurrent، تولید متن است. بدین‌صورت که شبکه وابستگی بین نمونه‌های ورودی را فرا گرفته و در دامنه آنها یک توالی جدید تولید می‌کنند. در ادامه با استفاده از ماژول‌های

LSTM، GRU و RNN و مجموعه دیالوگ‌های کتاب شکسپیر، شبکه‌ای در راستای یادگیری توالی کرکترهای کتاب ساخته می‌شود. همچنین این شبکه توانایی تولید یک خروجی implicit را خواهد داشت.

در ابتدا لازم است مجموعه مدنظر در حافظه ذخیره گردد. سپس در راستای آماده‌سازی دیتا برای اعمال به شبکه recurrent، کرکترها باید به اعداد integer تبدیل گردند. بنابراین لازم است یک مجموعه از کرکترهای موجود در کتاب ساخته شود و به هر کرکتر یک عدد اختصاص داده شود.

```
('characters are :',
 array(['\n', ' ', '!', '$', '&', "'", ',', '-', '.', '3', ':', ';', '?',
        'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
        'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',
        'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
        'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'],
       dtype='<U1'))
```

شکل – کرکترهای غیرتکراری موجود در دیتاست

بنابراین تعداد کل کرکترهای موجود (تکراری) در دیتاست و تعداد کرکترهای مستقل در ادامه آمده است. لازم به ذکر است تعداد کرکترهای غیرتکراری بیش‌تر از ۲۶ حرف زبان انگلیسی می‌باشد. درنهایت هر کرکتر به عدد متناظر اختصاص داده شد. تعداد کل کرکترها در متن، 1115394 و تعداد کرکترهای غیرتکراری 65 است. حال لازم است تا دیتا ورودی و تارگت برای اعمال به شبکه ساخته شود.

در این قسمت یک تابع نوشته شده است که با استفاده از آن دیتاست به صورت داده‌های ورودی و تارگت درمی‌آید تا بتوان به آنها را شبکه اعمال کرد. برای این منظور از دیتاست که به صورت یک بردار 1115394 درایه است، استفاده می‌شود. یک پنجره با ابعاد ۲۰ درنظر گرفته شده که بر روی دیتاست sweep خواهد شد. این سوییپ میتواند با استرایدی مخالف یک نیز حرکت کند. در اینجا ۴ درنظر گرفته شده است. از طرفی تارگت نیز هم طول با ورودی یعنی ۲۰ درنظر گرفته شده است. به طور کلی این تابع ۲۰ کرکتر اول را به عنوان ورودی برداشته و کرکتر ۲ تا ۲۱ را به عنوان تارگت درنظر می‌گیرد. حال کرکتر

۴ تا ۲۴ را به عنوان ورودی و کرکتر ۵ تا ۲۵ را به عنوان تارگت می‌گیرد. درنهایت ابعاد ورودی و تارگت به صورت (278839, 20) می‌شوند.

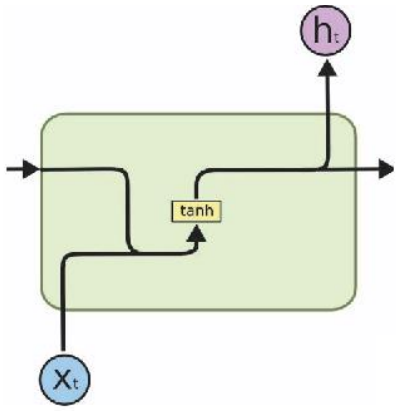
گفته شد که ابعاد ورودی و تارگت شبکه، 278839 است. حال برای اعمال داده‌ها به صورت batch، لازم است تا داده‌های ورودی مضربی از سایز batch باشند. ازآنجایی که طول داده‌های ورودی عدد اول شده است، داده آخر آن حذف شده تا طول ورودی مضربی از batch_size شود. در نهایت سایز داده‌های ورودی به 278838 تقلیل یافت و عوامل آن محاسبه گردید که به صورت [2,3,3,7,2213] درآمد. با ضرب ۴ عامل اول آن عدد ۱۲۶ حاصل می‌شود که به عنوان Batch_size در این مساله مورداستفاده قرار گرفته است.

۱. مقایسه سه ماژول RNN، LSTM و GRU

در طراحی شبکه‌های عصبی با هدف تداعی کردن یک پترن، نیاز به حافظه است. حال پترن ورودی می‌تواند به صورت explicit یا implicit باشد. منظور از ورودی implicit آن است که ورودی‌ها توسط یک اردر زمانی یا مکانی به شبکه داده می‌شوند. درغیراینصورت ورودی‌ها explicit است. طراحی یک شبکه عصبی برای ورودی‌های explicit ساده‌تر از طراحی شبکه برای داده‌های implicit است و تنها به حافظه استاتیکی نیاز دارند. در مسائل واقعی و پرچالش‌تر ابعاد داده‌های ورودی ثابت نبوده و وجود نویز غیرقابل انکار است. دراین‌گونه مسائل که ورودی‌ها implicit هستند، از شبکه‌های recurrent استفاده می‌شود. در شبکه‌های recurrent پس از طی کردن مسیر feedforward در راستای ساخت خروجی، آنرا به ورودی اعمال می‌کنند. در اینحالت خروجی همواره از exogenous input بهره می‌برد تا تداعی را به‌درستی انجام دهد. قاعده یادگیری در شبکه‌های RNN، gradient updating rule است. در ادامه سه نمونه از شبکه‌های recurrent بررسی خواهد شد.

- **سلول RNN**

در سلول RNN ورودی (input) در لحظه کنونی با خروجی (hidden state) لحظه قبل ترکیب شده و پس از عبور از تابع فعال‌ساز tanh، hidden state لحظه کنونی یا همان حافظه را می‌سازند. تابع tanh برای کنترل فلو اطلاعاتی در شبکه استفاده می‌شود. در ادامه یک سلول RNN آمده است.



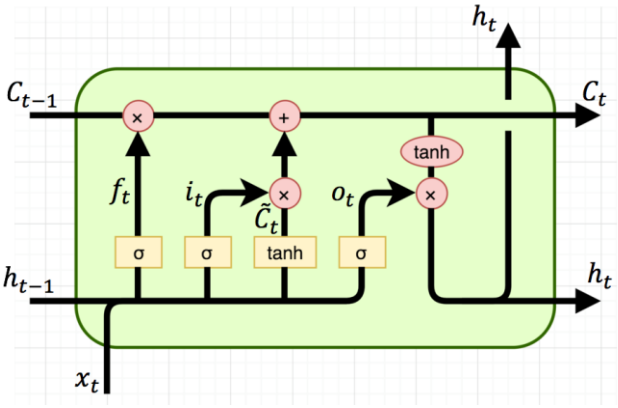
شکل – سلول RNN

پیشتر گفته شد که در شبکه‌های RNN از قاعده یادگیری گرادیان استفاده می‌کنند. بنابراین با افزایش طول ورودی‌های implicit، مسیرهای برگشتی طولانی‌تر شده و محاسبات سخت خواهد شد. همچنین ازآنجایی‌که از توابع فعال‌ساز با مشتق زیر یک استفاده می‌شود، در ذنجیره‌های طولانی، مولفه خطای برگشتی برای وزن‌هایی که در زمان‌های دور هستند، کوچک می‌شود و واکشی اطلاعات از لحظه کنونی برای به‌روزرسانی وزن‌های متاثر از داده‌های قدیمی ضعیف خواهد بود. برای حل این مشکل می‌توان از توابع فعال‌سازی همچون ReLU بهره برد اما این تابع نیز برای مقادیر نامثبت، مشکل‌ساز می‌شود و باید به دنبال ساز و کارهایی غیر از توابع فعال‌ساز رفت. درواقع شبکه‌های RNN به دلیل سادگی، برای کاربردهایی که نیاز به short depency دارند، استفاده می‌شوند.

- **سلول LSTM**

برای حل مشکل بازیابی اطلاعات برای داده‌های با فاصله طولانی و برقراری long dependency استفاده از ماژول LSTM پیشنهاد می‌شود. LSTM اجازه می‌دهد داده‌ها از زمان‌های دور ذخیره گردند. به عنوان مثال برای پردازش یک پاراگراف در راستای تولید متن، RNN کلاسیک اطلاعات مهمی که در

ابتدای متن هستند را درنظر نمی‌گیرد. حال سلول‌هایی همانند LSTM باعث می‌شوند مشکل short-term memory حل شود؛ زیرا با وجود مکانیزم‌های داخلی (گیت‌ها) فلو اطلاعاتی را کنترل می‌کنند. در ادامه یک سلول LSTM آمده است.



شکل – سلول LSTM

سلول LSTM همانند RNN، فلو اطلاعاتی را کنترل می‌کند و در مسیر forward انتشار می‌دهند با این تفاوت که در سلول LSTM عملیات متفاوتی انجام می‌شود. این عملیات به LSTM اجازه می‌دهند تا اطلاعات را حفظ و یا پاک کنند. هسته مرکزی LSTM درواقع cell state و گیت‌های آن است که منجر می‌شود اطلاعات مفید (مهم نیست برای چه مدت پیش هستند) در حافظه محفوظ بمانند. از طرفی گیت‌ها ممکن است به ذنجیره، اطلاعاتی بیافزایند و یا از حذف کنند. درواقع این گیت‌ها هستند که یاد می‌گیرند اطلاعاتی مفید بوده و یا باید فراموش شود. گیت‌ها دارای تابع فعال‌ساز سیگموید هستند. تفاوت sigmoid و tanh در آن است که sigmoid خروجی را بین ۰ و ۱ می‌برد. بنابراین اگر اطلاعاتی باید فراموش گردد در صفر ضرب شده و حذف می‌گردد. در LSTM سه گیت مختلف وجود دارد که فلو اطلاعاتی را کنترل می‌کنند. گیت فراموشی، گیت ورودی و گیت خروجی.

گیت فراموشی: اطلاعات ورودی کنونی و خروجی (hidden state) قبل ترکیب شده و به sigmoid اعمال می‌شوند و خروجی آن مقداری بین صفر تا یک دارد.

گیت ورودی: کاربرد این گیت در راستای update کردن cell state است. اطلاعات input کنونی و hidden state قبلی ترکیب شده و به sigmoid اعمال می‌شوند. حال sigmoid تصمیم می‌گیرد که چه اطلاعاتی باید update شوند. همچنین ترکیب input کنونی و hidden state قبلی وارد یک tanh شده و خروجی sigmoid و tanh یا یکدیگر ضرب می‌شوند. Sigmoid تصمیم می‌گیرد که چه اطلاعاتی مهم بوده و باید حفظ شوند.

Cell state: حال با استفاده از خروجی گیت فراموشی و گیت ورودی، اطلاعات لازم برای محاسبه cell state ساخته شده است. درواقع گیت فراموشی تصمیم می‌گیرد که اطلاعات ساخته شده در گیت ورودی مهم بوده و یا نه و cell state جدید ساخته می‌شود.

گیت خروجی: این گیت تصمیم می‌گیرد که hidden state بعدی چه باید باشد. درواقع hidden state دارای اطلاعاتی از ورودی‌های قبلی است و برای پیشبینی نیز استفاده می‌شود. عملکرد این گیت بدین صورت است که hidden state قبلی و ورودی کنونی به تابع sigmoid داده شده و cell state جدید به تابع tanh اعمال می‌شود. حال خروجی tanh و sigmoid ضرب می‌شوند تا اطلاعاتی که hidden state باید داشته باشد مشخص گردد. درواقع خروجی سلول، hidden state است.

به طور کلی گیت فراموشی تصمیم می‌گیرد که چه اطلاعاتی از حال‌های قبلی باید حفظ شود. گیت ورودی تصمیم می‌گیرد چه اطلاعاتی از ورودی جدید با حالت‌های قبل مرتبط بوده است. گیت خروجی نیز hidden state بعدی را مشخص می‌کند.

درواقع سلول LSTM در مقایسه با RNN، از درجه آزادی بیشترب برخوردار است و امکان ترکیب ورودی‌ها با داده‌های بیشتری وجود داشته که منجر به کنترل بهتر خروجی‌ها نیز می‌شود. بنابراین سلول LSTM کنترل بهتر پارامترها و درنتیجه نتایج بهتر را به ارمغان می‌آورد اما هزینه آن پیچیدگی و عملیات بیشتر است.

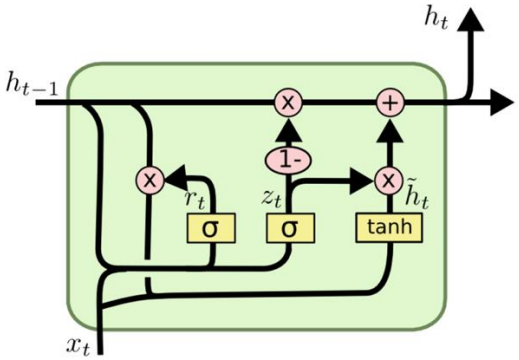
- **سلول GRU**

سلول GRU نسل جدیدی از شبکه‌های عصبی recurrent است و شباهت زیادی با LSTM دارد. در GRU، cell state حذف شده و از hidden state برای انتقال اطلاعات استفاده می‌کند. همچنین گیت‌های آن update و reset هستند (یک گیت کمتر از LSTM).

گیت update: این گیت شبیه به گیت فراموشی و گیت ورودی LSTM عمل می‌کند و تصمیم می‌گیرد که چه اطلاعاتی حذف و یا اضافه شود.

گیت reset: گیت ریست نیز تصمیم می‌گیرد که چه اطلاعات گذشته‌ای باید حذف شوند.

درواقع سلول GRU، محاسبات کمتری داشته که منجر می‌شود در مقایسه با LSTM دارای سرعت بیشتری باشد. البته هر یک از GRU و LSTM بسته به کاربدر ممکن است بهتر از دیگری باشد. در ادامه یک سلول GRU قابل مشاهده است.



شکل – سلول GRU

۲. توابع هزینه و اپتیمایزر

- **Mean Squared Error**

مجموع مربعات خطا، تابع پیش‌فرض مسائل Regression است. این خطا به‌صورت میانگین مجذور اختلاف بین خروجی‌های پیش‌بینی شده و خروجی Target محاسبه می‌گردد. MSE جدای از علامت Predict و Target، همواره دارای مقدار مثبت است و بهترین مقدار خطا برای آن صفر خواهد بود. استفاده از مجذور خطا نمایانگر آن است که اشتباهات بزرگ‌تر منجر به خطاهای بیش‌تر می‌شود.

$$MSE\ Loss: J(y) = \frac{1}{n}\sum_{i=1}^{n}(t_i - h_i)^2$$

- **Categorical Cross Entropy**

منظور از Softmax Cross Entropy یا Categorical Cross Entropy، تابع فعال‌ساز Softmax و تابع هزینه Cross entropy است. درواقع تابع Softmax، یک حالت Soft از تابع ماکزیمم است که احتمال ماکزیمم و نزدیک به آن را می‌دهد. ورودی تابع Softmax یک بردار N بعدی و خروجی آن یک بردار بین ۰ تا ۱ است که در ادامه آمده است.

$$P_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e_k^a}$$

ازآنجایی‌که تابع Softmax یک توزیع احتمالی به‌عنوان خروجی می‌دهد، در لایه خروجی شبکه‌های عصبی از آن استفاده می‌گردد. در هنگام Backpropagation لازم است مشتق و یا گرادیان Softmax محاسبه گردد که در ادامه آمده است.

$$\frac{\partial p_i}{\partial a_j} = \frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e_k^a}}{\partial a_j} = \cdots = p_i(\delta_{ij} - p_j)$$

تابع هزینه Cross Entropy فاصله بین توزیع واقعی مدل و توزیعی که شبکه به دست آورده را مشخص می‌کند که در ادامه رابطه آن آمده است.

$$H(y,p) = -\sum_i y_i \log(p_i)$$

درنهایت از مشتق تابع فعال‌ساز Softmax برای محاسبه مشتق تابع خطا Cross entropy استفاده می‌شود.

درواقع خروجی شبکه بسته به تعداد کرکتری که قرار است پیشبینی شود، تعدادی بردار ۶۵ درایه‌ای است. هر درایه نشان دهنده احتمال یکی از کرکترها است که در آخر بیشترین احتمال برنده می‌شود. بنابراین استفاده از تابع هزینه Categorical Cross Entropy به MSE ارجعیت دارد.

- **RMSProp**

بهینه‌ساز Root Mean Square Prppagation منجر به کاهش نوسانات می شود. همچنین نیازی به تنظیم دستی نرخ یادگیری ندارد بلکه به صورت اتوماتیک آن را تنظیم می‌کند. در روش RMSProp، به‌روزرسانی پارامترها به صورت زیر است:

$$For\ each\ Parameter\ w^j$$
$$(j\ subscript\ dropped\ for\ clarity)$$
$$\nu_t = \rho\nu_{t-1} + (1-\rho) * g_t^2$$
$$\Delta\omega_t = -\frac{\eta}{\sqrt{\nu_t + \epsilon}} * g_t$$
$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

درواقع برای هر پارامتر، میانگین نمایی مجذور گرادیان آن محاسبه می‌شود. استفاده از مجذور گرادیان منجر می‌شود، وزن پارامترهای پایانی بیشتر از قبلی‌ها به‌روزرسانی شود. سپس در معادله دوم، میزان step توسط میانگین نمایی محاسبه می‌گردد. به عنوان مثال اگر میانگین w1 بزرگتر از میانگین w2 باشد، step یادگیری برای w1 کوچکتر از w2 خواهد بود و منجر به یافتن مینیمم‌ها می‌شود. بنابراین هنکامی که تابع هزینه به نقاط مینیمم نزدیک می‌شود، RMSProp از قدم‌های کوچکتر استفاده می‌کند.

- **Adam**

در روش Adam(Adaptive Moment Estimation)، همانند RMSProp از نرخ یادگیری متفاوت برای آپدیت کردن هر پارامتر استفاده می‌شود. همچنین علاوه بر Learning Rate، از ترم Momentum متفاوت نیز استفاده می‌شود.

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$$

$$\widehat{v_t} = \frac{v_t}{1-\beta_2^t}$$

بنابراین الگوریتم Adam از رابطه زیر برای آپدیت کردن پارامترها استفاده می‌کند.

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\widehat{v_t}} + \varepsilon} . \widehat{m_t}$$

مقادیر رایج برای β_1، β_2 و ε به ترتیب ۰,۹، ۰,۹۹۹ و 10^{-8} است. درواقع روش Adam منجر به همگرایی سریع‌تر نسبت به سایر الگوریتم‌های بهینه‌سازی می‌شود. همچنین با مشکلاتی همچون همگرایی نرخ یادگیری به صفر و کاهش سرعت همگرایی تابع خطا، مواجه نمی‌شود.

به‌طورکلی الگوریتم Adam بهتر از الگوریتم‌های Adaptive دیگر همانند RMSProp عمل می‌کند. حال اگر داده‌های ورودی به‌اصطلاح sparse باشند، روش‌هایی مانند SGD و Momentum ضعیف عمل می‌کنند و باید از روش‌های Adaptive استفاده کرد. برای دستیابی به همگرایی سریع‌تر در مدل‌های عمیق و پیچیده، الگوریتم Adam بهتر از سایرین عمل می‌کند.

۳. مشاهده دادگان آموزش و هدف

در این بخش تعداد کل کرکترهای موجود در کتاب یعنی 1115394 بر قسمت‌های مختلف تقسیم می‌شود و برای ساخت دادگان هدف نیز، از shifting متفاوت استفاده می‌شود. در ادامه کد مربوط به این قسمت آمده است.

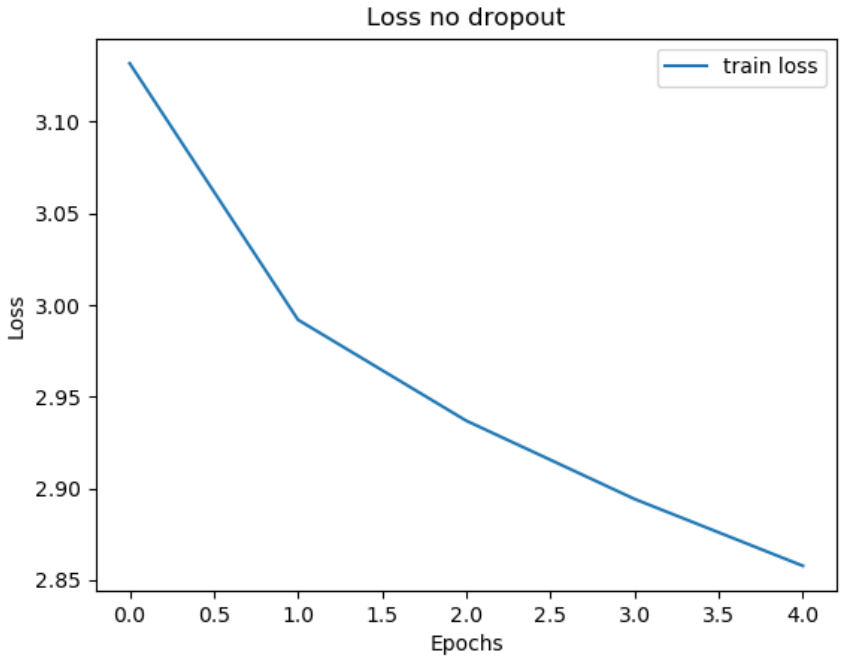
```
period = 20
shifting = 4

examples = len(text)//(period+1)

char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)
sequences = char_dataset.batch(period+1, drop_remainder=True)

def split_input_target(chunk):
    input_  = chunk[:-shifting]
    target_ = chunk[shifting: ]
    return input_, target_

dataset = sequences.map(split_input_target)

for input_example, target_example in  dataset.take(1):
  print (repr(''.join(idx2char[input_example.numpy()])))
  print (repr(''.join(idx2char[target_example.numpy()])))
```

```
'First Citizen:\nBe'
't Citizen:\nBefore'
```

period = 20

shifting = 4

در این حالت برای loss داریم:



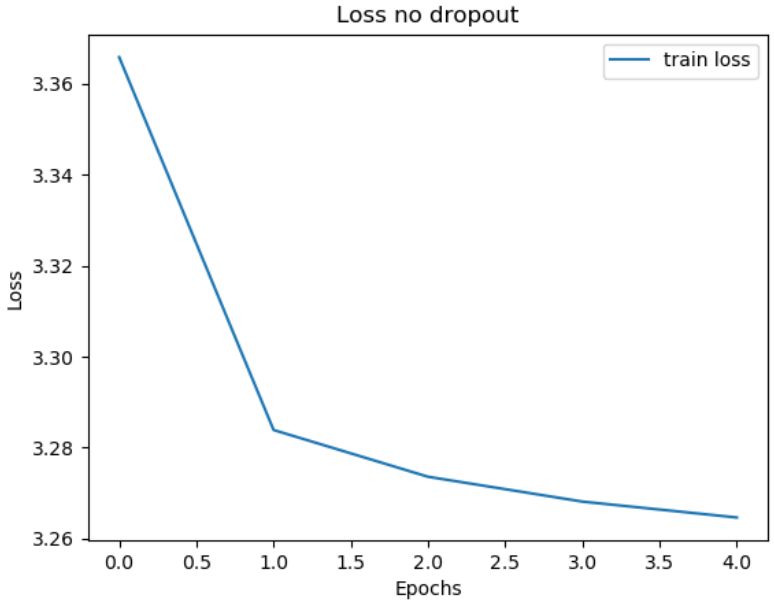
شکل – نمودار خطا برای تعداد کرکتر ۲۰ و ۴ کرکتر شیفت

عدد نهایی آن نیز 3.85 می باشد.

period = 50

shifting = 12

در این حالت برای loss داریم:



شکل – نمودار خطا برای تعداد کرکتر ۵۰ و ۱۲ کرکتر شیفت

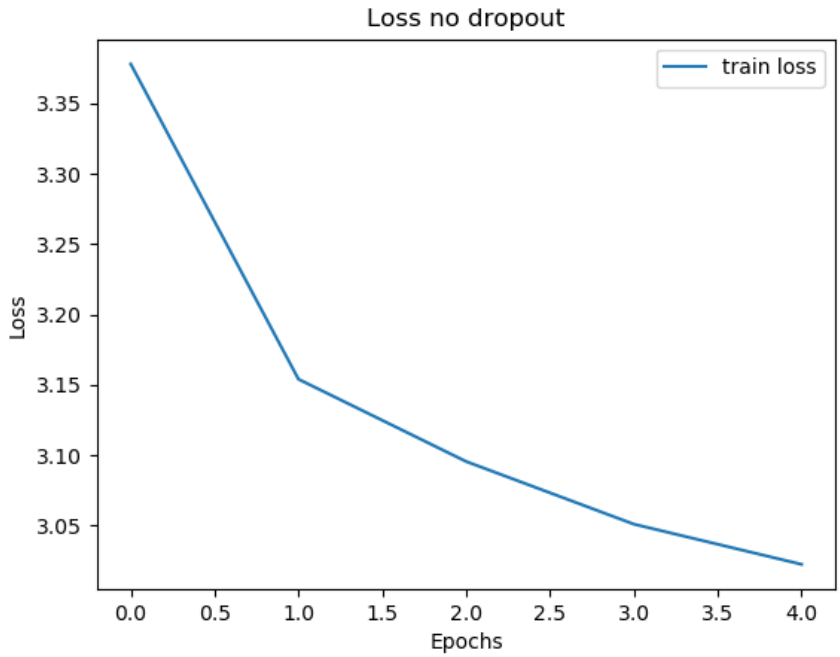
عدد نهایی آن نیز 3.26 می باشد.

```
'First Citizen:\nBefore we proceed any fu'
'n:\nBefore we proceed any further, hear '
```

period = 100

shifting = 5

در این حالت برای loss داریم:



شکل – نمودار خطا برای تعداد کرکتر ۱۰۰ و ۵ کرکتر شیفت

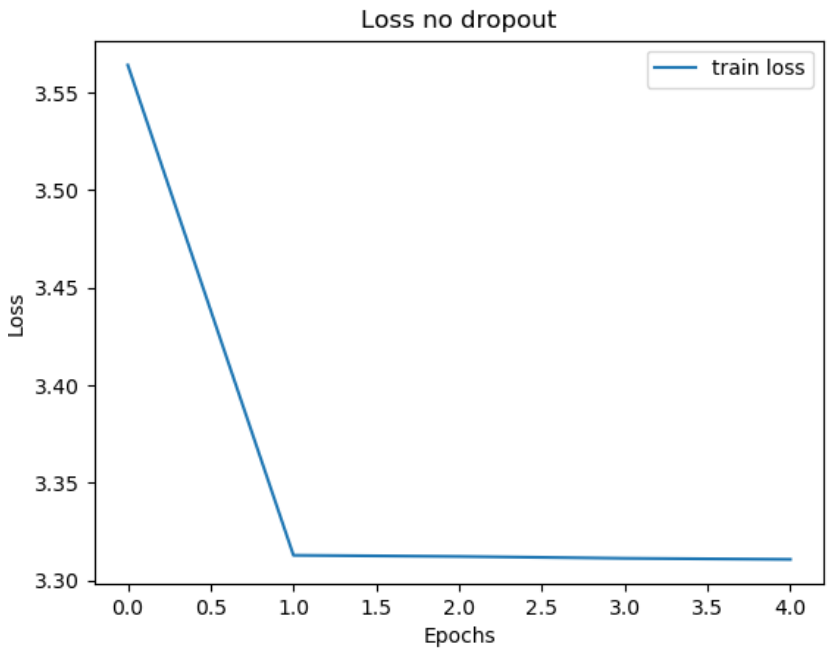
عدد نهایی آن نیز 3.02 می باشد.

```
'First Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:'
' Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\nYou '
```

period = 200

shifting = 25

در این حالت برای loss داریم:



شکل – نمودار خطا برای تعداد کرکتر ۲۰۰ و ۲۵ کرکتر شیفت

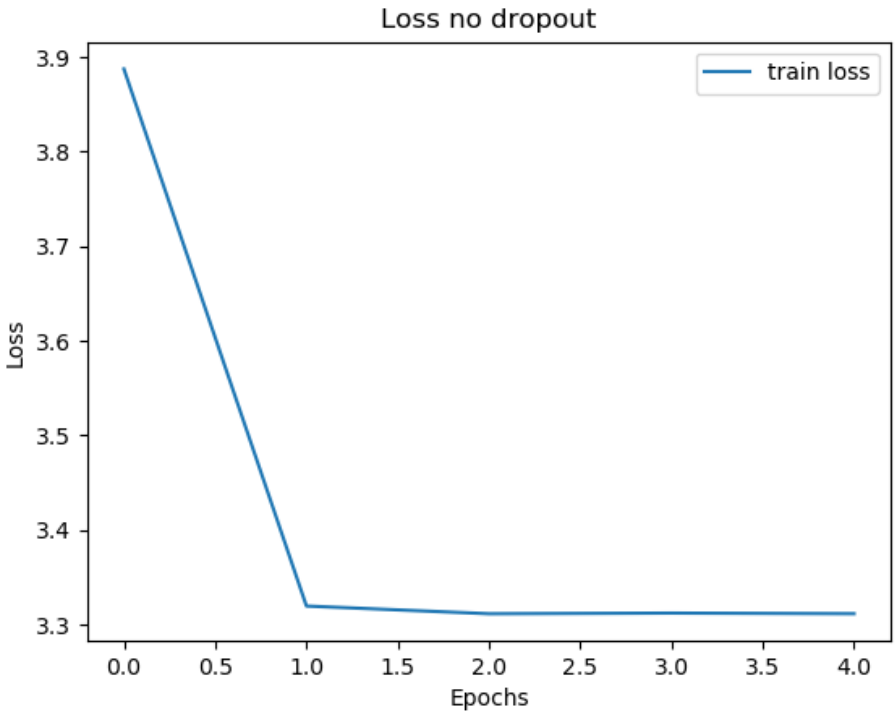
عدد نهایی آن نیز 3.31 می باشد.

```
'First Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\nYou are all resolved rather to die than to famish?\n\nAll:\nResolved. resolved.\n\nF'
'proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\nYou are all resolved rather to die than to famish?\n\nAll:\nResolved. resolved.\n\nFirst Citizen:\nFirst, you '
```

period = 500

shifting = 100

در این حالت برای loss داریم:



شکل – نمودار خطا برای تعداد کرکتر ۵۰۰ و ۱۰۰ کرکتر شیفت

عدد نهایی آن نیز 3.31 می باشد.

```
"First Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSp
eak, speak.\n\nFirst Citizen:\nYou are all resolved rather to die than to
famish?\n\nAll:\nResolved. resolved.\n\nFirst Citizen:\nFirst, you know Ca
ius Marcius is chief enemy to the people.\n\nAll:\nWe know't, we know't.\n
\nFirst Citizen:\nLet us kill him, and we'll have corn at our own price.\n
Is't a verdict?\n\nAll:\nNo more talking on't; let it b"
```

```
" are all resolved rather to die than to famish?\n\nAll:\nResolved. resolv
ed.\n\nFirst Citizen:\nFirst, you know Caius Marcius is chief enemy to the
 people.\n\nAll:\nWe know't, we know't.\n\nFirst Citizen:\nLet us kill him
, and we'll have corn at our own price.\nIs't a verdict?\n\nAll:\nNo more
talking on't; let it be done: away, away!\n\nSecond Citizen:\nOne word, go
od citizens.\n\nFirst Citizen:\nWe are accounted poor "
```

۴. پارامترهای مدل GRU, LSTM, RNN

برای GRU داریم:

Batch_size = 64

vocab_size = 65

embedding_dim = 256

gru_units = 1024

از یک لایه GRU استفاده شده است. قبل از این لایه نیز عمل Embeding صورت گرفته است.

بعد از لایه GRU نیز یک لایه Dense قرار داده شده تا خروجی را به صورت احتمال هر کلمه بدهد.

کد این بخش: q2_gru.py

برای LSTM داریم:

Batch_size = 64

vocab_size = 65

embedding_dim = 256

lstm_units = 1024

از یک لایه LSTM استفاده شده است. قبل از این لایه نیز عمل Embeding صورت گرفته است.

بعد از لایه LSTM نیز یک لایه Dense قرار داده شده تا خروجی را به صورت احتمال هر کلمه بدهد.

کد این بخش: q2_lstm.py

برای RNN داریم:

Batch_size = 64

vocab_size = 65

embedding_dim = 256

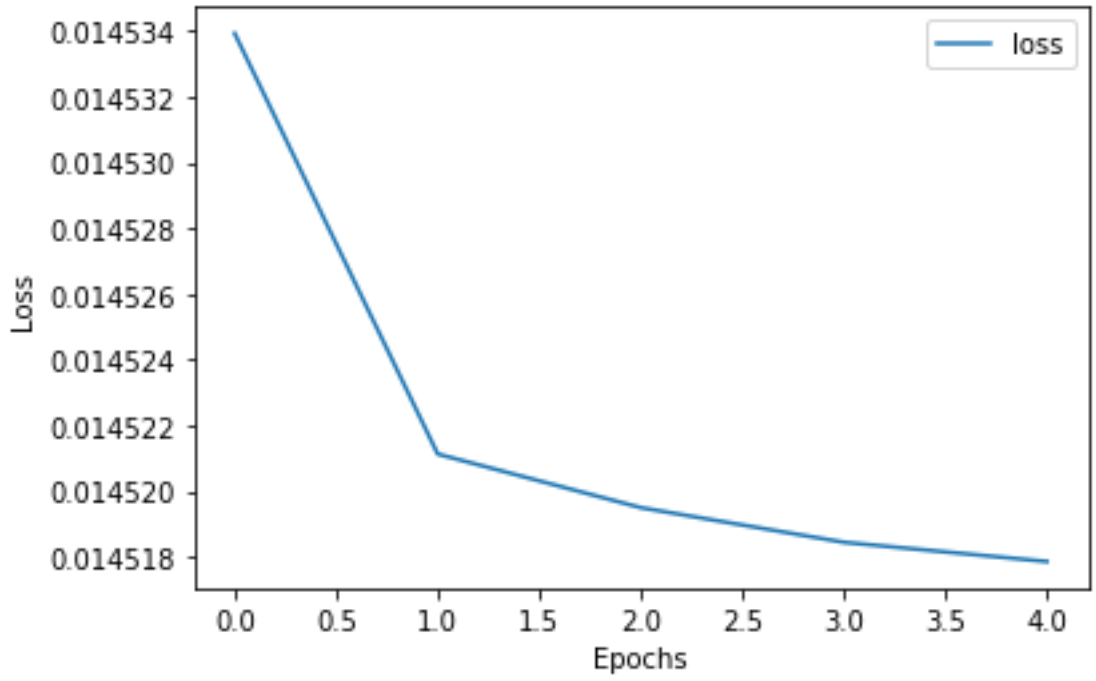
rnn_units = 1024

از یک لایه RNN استفاده شده است. قبل از این لایه نیز عمل Embeding صورت گرفته است. بعد از لایه RNN نیز یک لایه Dense قرار داده شده تا خروجی را به صورت احتمال هر کلمه بدهد.

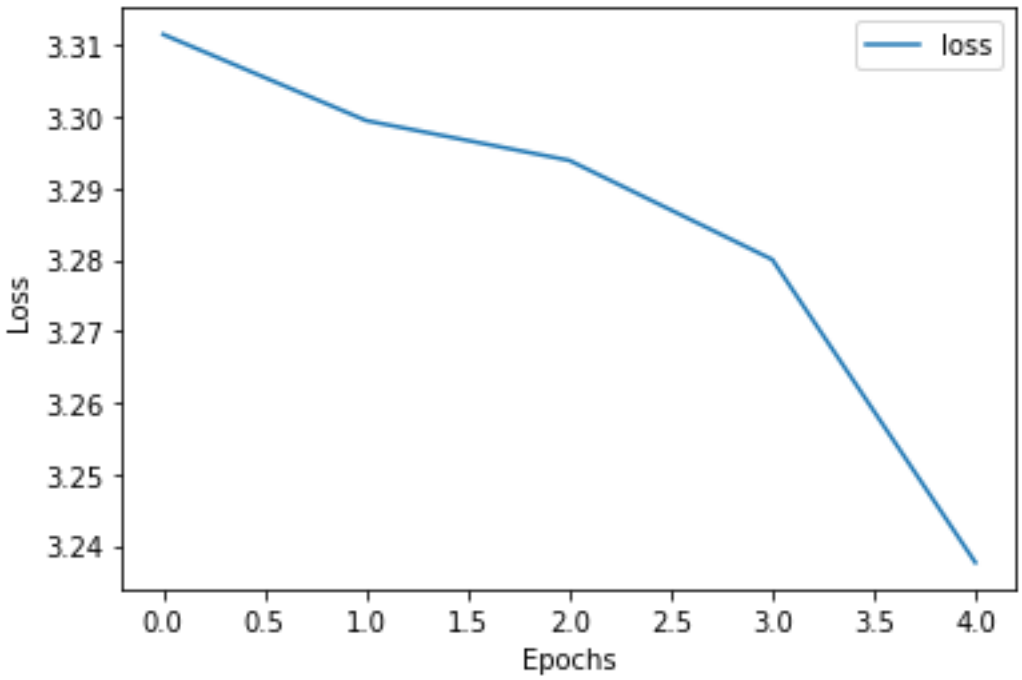
کد این بخش: q2_rnn.py

5. استفاده از توابع هزینه و بهینه‌سازها

در این بخش از دو تابع هزینه Mean Square Error و Categorical Cross Entropy استفاده شده است. در ادامه نمودار خطا برای ۵ ایپاک آمده است. مشاهده می‌شود که خطا سلول GRU برای تابع هزینه mse کمتر شده است.

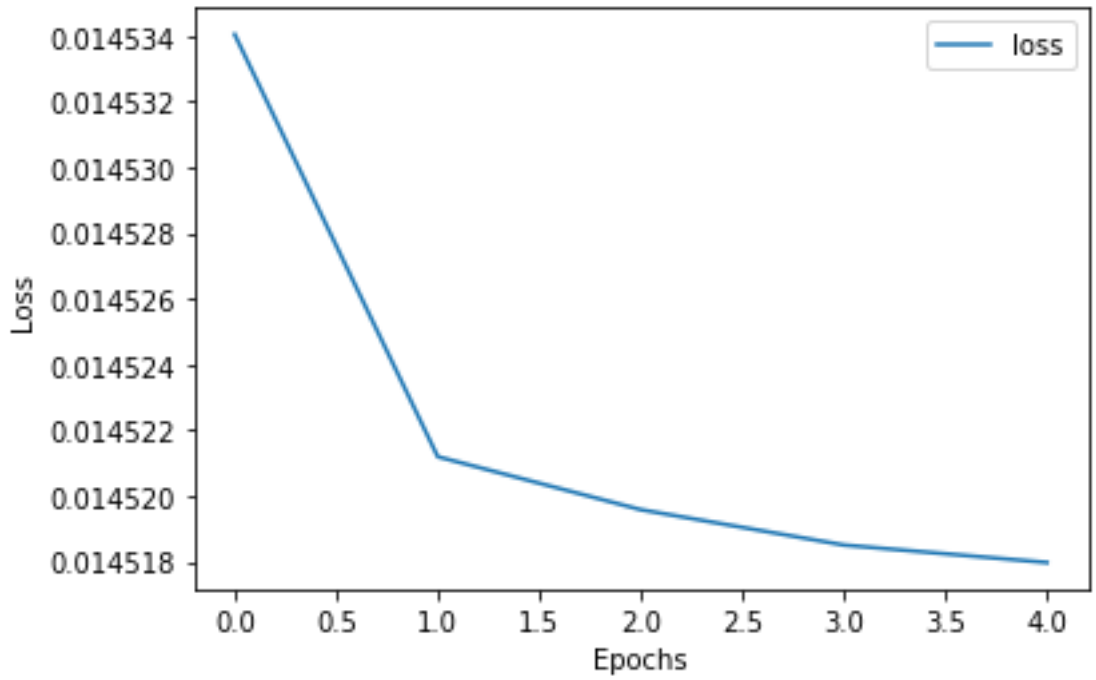


شکل – نمودار خطا برای سلول GRU و با تابع هزینه MSE

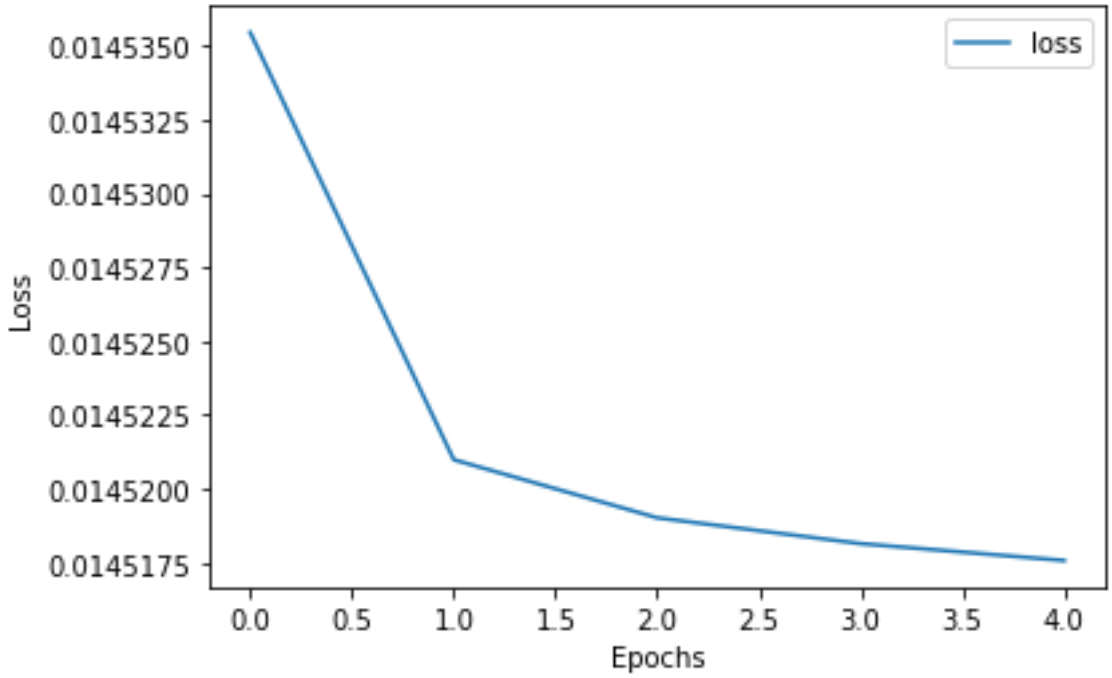


شکل – نمودار خطا برای سلول GRU و با تابع هزینه Cross Entropy

همچنین از دو بهینه‌ساز Adam و RMSProp استفاده شده است. برای مقایسه عملکرد دو بهینه‌ساز مطرح شده، تابع هزینه MSE در نظر گرفته شده است. در ادامه نمودار خطا برای این دو بهینه‌ساز آمده است. لازم به ذکر است هر دو بهینه‌ساز به خطای کمی دست‌یافتند و تفاوت چندانی با یکدیگر نداشتند.



شکل – نمودار خطا برای سلول GRU و با بهینه‌ساز Adam



شکل – نمودار خطا برای سلول GRU و با بهینه‌ساز RMSProp

درنهایت برای توابع هزینه و بهینه‌سازها یک جدول تشکیل شده که در ادامه آمده است.

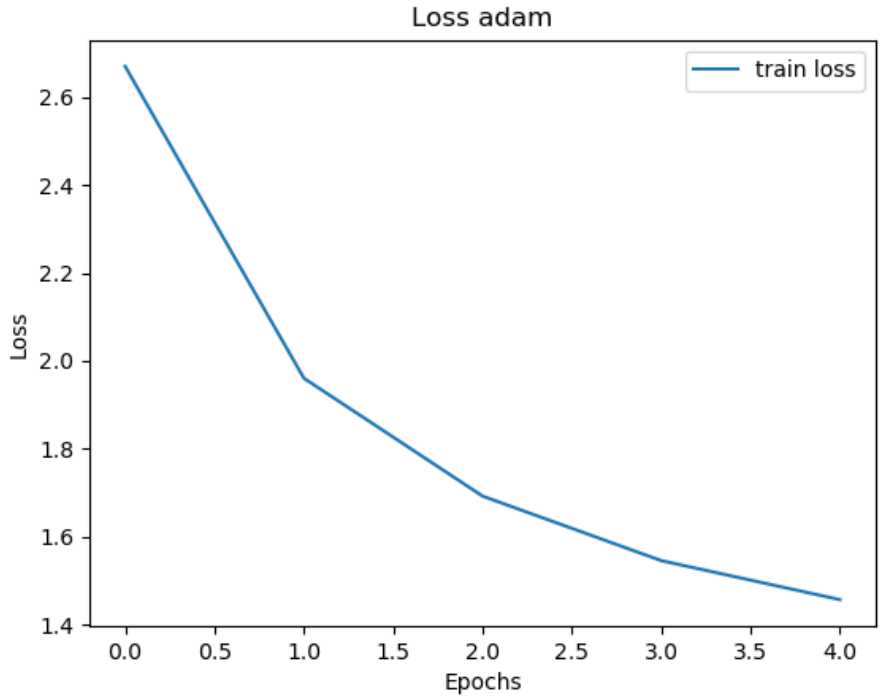
جدول – مقایسه خطا برای توابع هزینه و بهینه‌سازها

	Loss Function		Optimizre	
	MSE	CCE	Adam	RMSProp
Loss	0.0145	3.2375	0.0145	0.0145

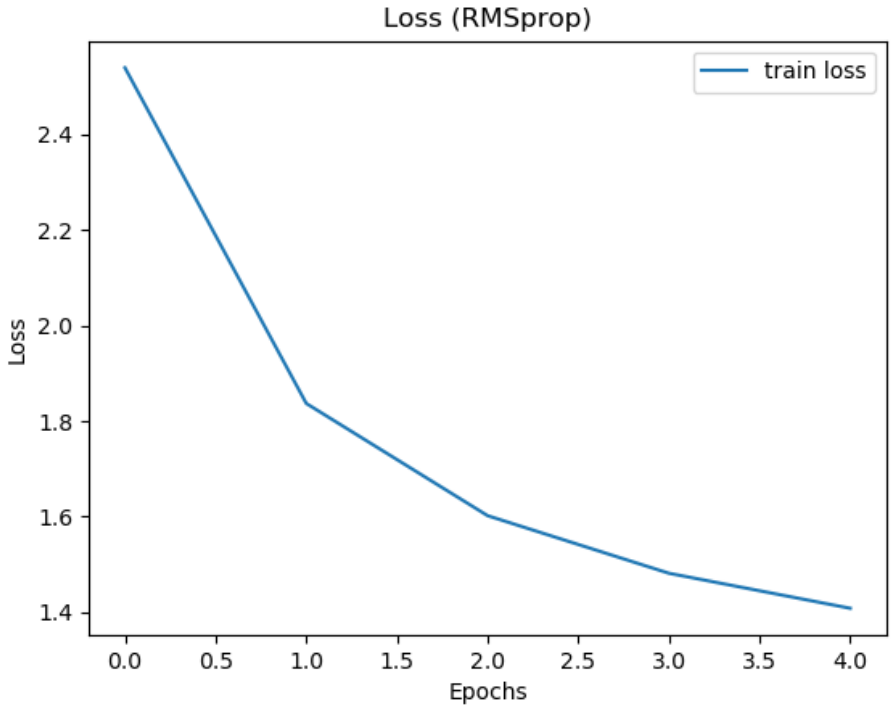
باتوجه به نتایج به دست آمده به نظر می رسد نتایج منطقی نیستند، لذا بار دیگر با استفاده از سایت تنسورفلو ران کردیم. نتایج به صورت زیر می باشند:

جدول – مقایسه خطا برای توابع هزینه و بهینه‌سازها با استفاده از سایت تنسورفلو

	Loss Function		Optimizre	
	MSE	CCE	Adam	RMSProp
Loss	0.0145	**1.1921e-7**	**1.456**	**1.40**



شکل – نمودار خطا برای سلول GRU و با بهینه‌ساز Adam



شکل – نمودار خطا برای سلول GRU و با بهینه‌ساز RMSProp

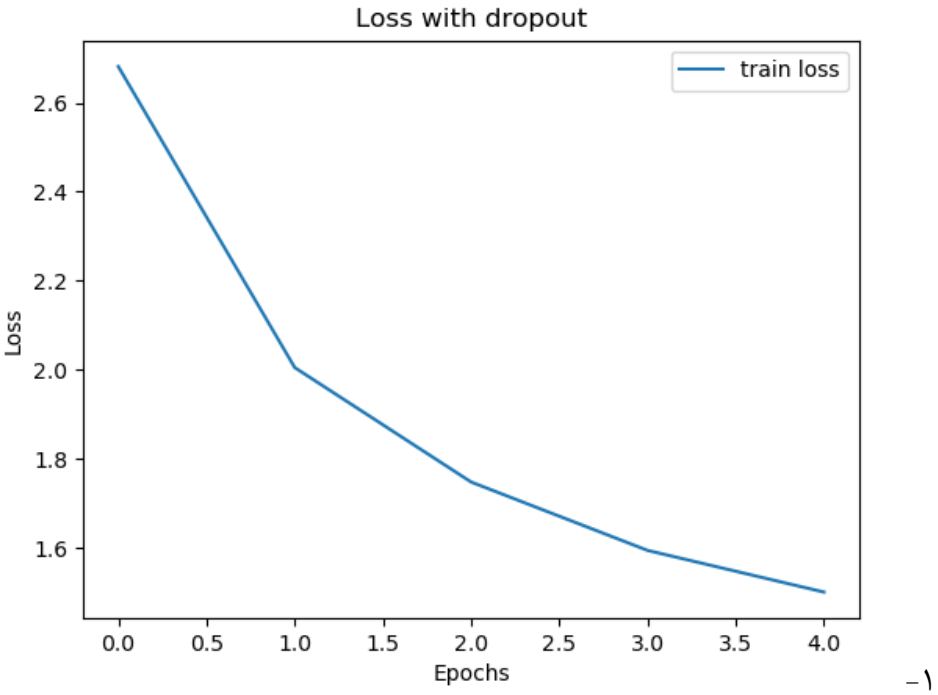
۶. استفاده از Dropout

یکی از راه‌های مقابله با overfitting استفاده از dropout است. درواقع dropout به طور تصادفی درصدی از نورون‌ها را با احتمال یکسان، خاموش می‌کند تا قدرت تعمیم شبکه را افزایش می‌دهد. برای دادگان متنی نیز در راستای جلوگیری شبکه از حفظ کردن جایگاه کرکترها، از تکنیک dropout استفاده می‌شود. در شبکه‌های recurrent میتوان در دو مسیر فوروارد و فیدبک از dropout استفاده کرد. پیش‌تر گفته شد که در شبکه‌های recurrent، خروجی توسط فیدبک توانایی استفاده از دادگان قبلی را دارد. حال استفاده از dropout برای RNN و در مسیر فیدبک توانایی شبکه برای بازیابی اطلاعات قبلی را محدود می‌کند و یادگیری را دچار مشکل می‌کند. بنابراین برای شبکه‌های recurrent در مسیرهای forward به به دقت بهتری دست پیدا می‌کنیم.

برای این منظور در کدهای زده شده مقایسه انجام می شود. (q2_rnn.py, q2_gru.py, q2_lstm.py)

a. در ابتدا برای GRU بررسی می کنیم:

با دراپوت لایه های فوروارد loss به صورت زیر می باشد. که تا 1.57 پایین آمده است.

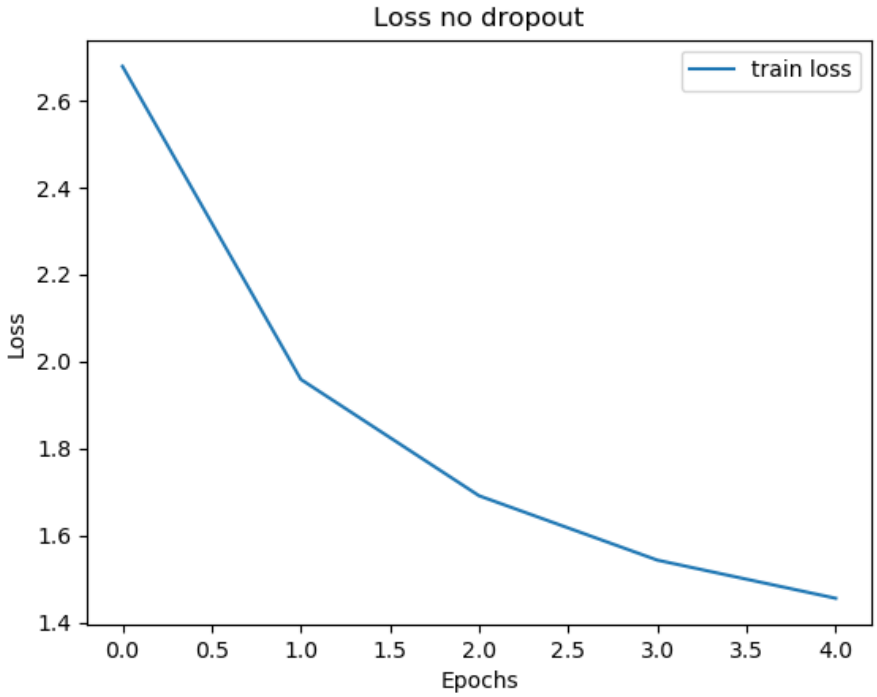


۲- شکل – نمودار خطا برای سلول GRU با dropout

با دراپوت لایه های ریکارنت loss به صورت nan درآمد و خروجی چیزی نمایش داده نشد.

```
114/Unknown - 23s 202ms/step - loss: nan
115/Unknown - 23s 202ms/step - loss: nan
116/Unknown - 23s 202ms/step - loss: nan
117/Unknown - 24s 202ms/step - loss: nan
118/Unknown - 24s 201ms/step - loss: nan
119/Unknown - 24s 201ms/step - loss: nan
120/Unknown - 24s 201ms/step - loss: nan
```

حال بدون استفاده از droupout خواهیم داشت:

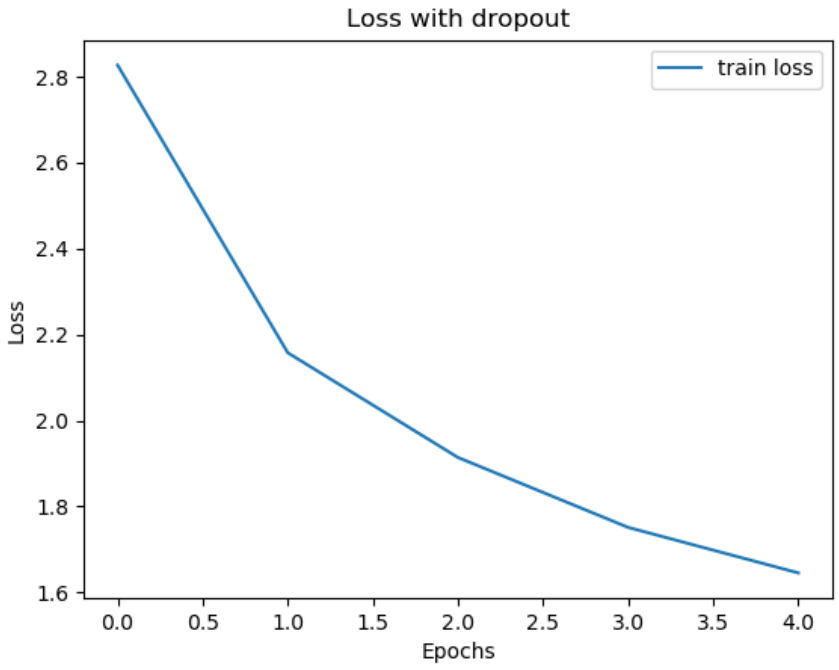


۳- شکل – نمودار خطا برای سلول GRU بدون dropout

همانطور که مشخص است مقدار آن 1.45 می باشد.

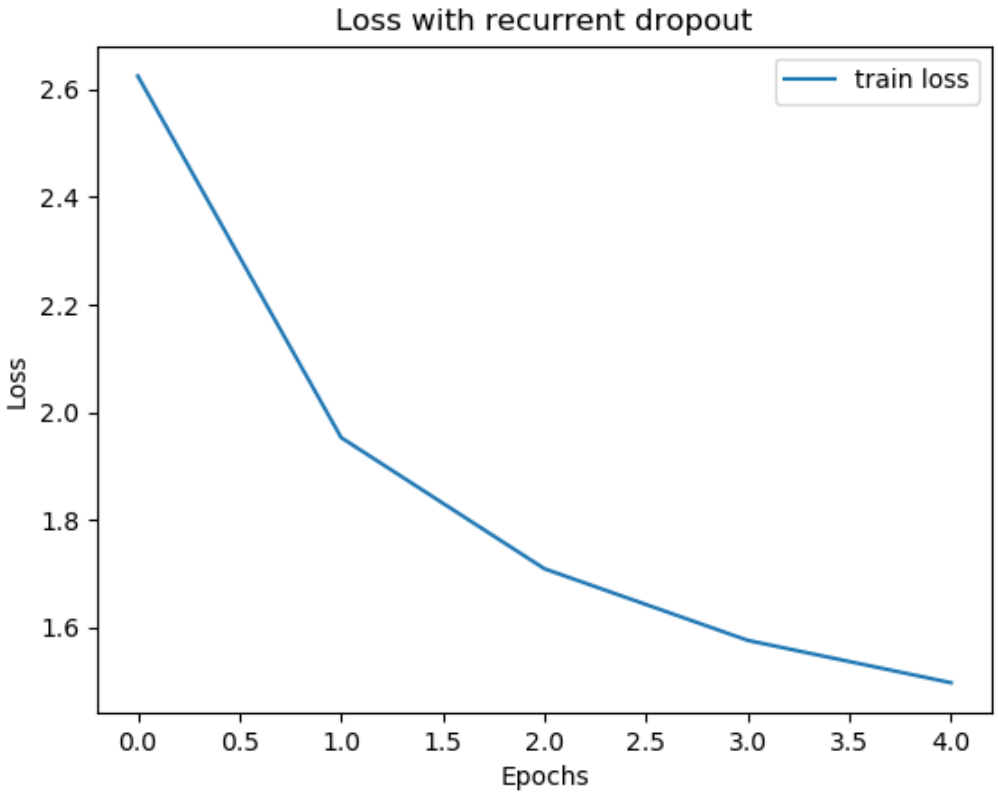
b. برای LSTM بررسی می کنیم:

با دراپوت لایه های فوروارد loss به صورت فوق می باشد. که تا 1.64 پایین آمده است.



۴- شکل – نمودار خطا برای سلول LSTM با dropout به صورت فوروارد

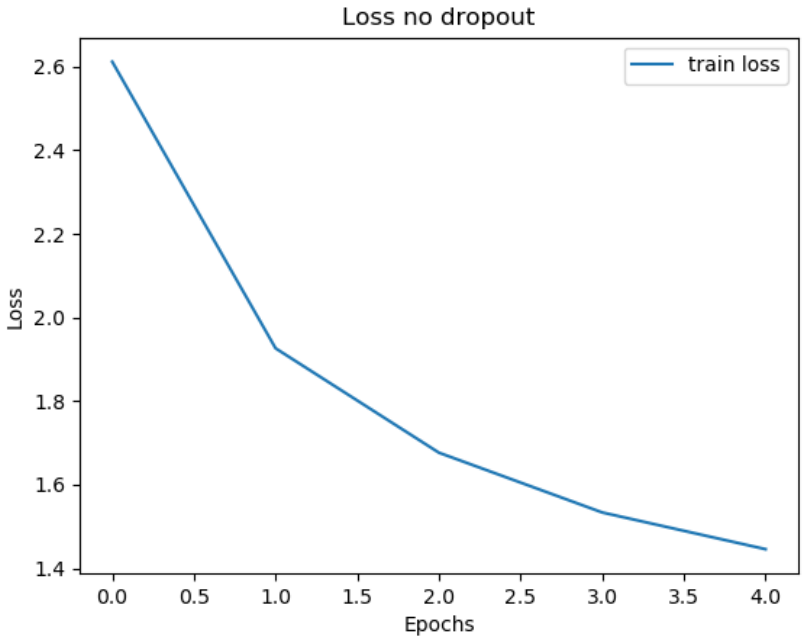
با دراپود ریکارنت نیز loss به صورت زیر در خروجی بود:



۵- شکل – نمودار خطا برای سلول LSTM با dropout به صورت ریکارنت

همانطور که مشخص است مقدار آن 1.49 می باشد.

حال بدون استفاده از droupout خواهیم داشت:

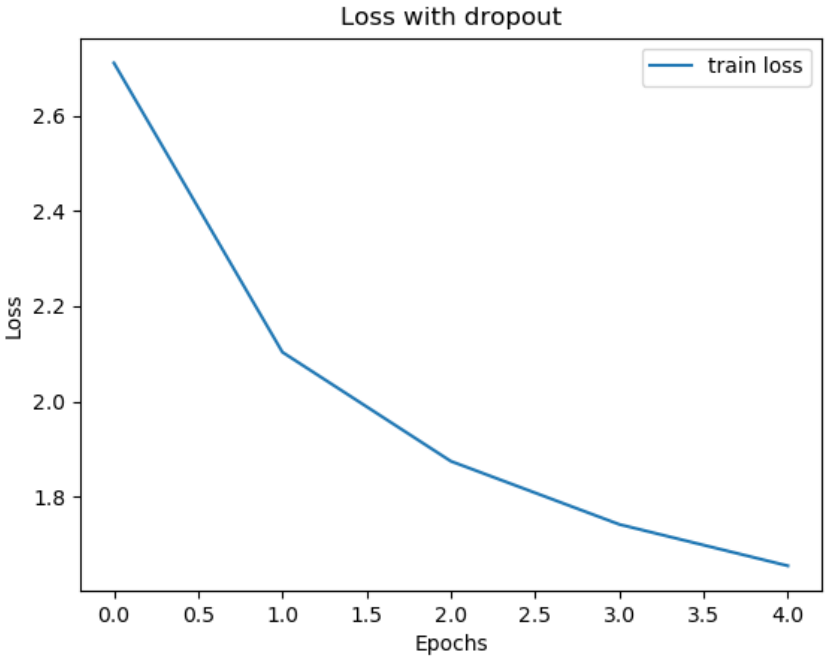


۶- شکل – نمودار خطا برای سلول LSTM بدون dropout

همانطور که مشخص است مقدار آن 1.44 می باشد.

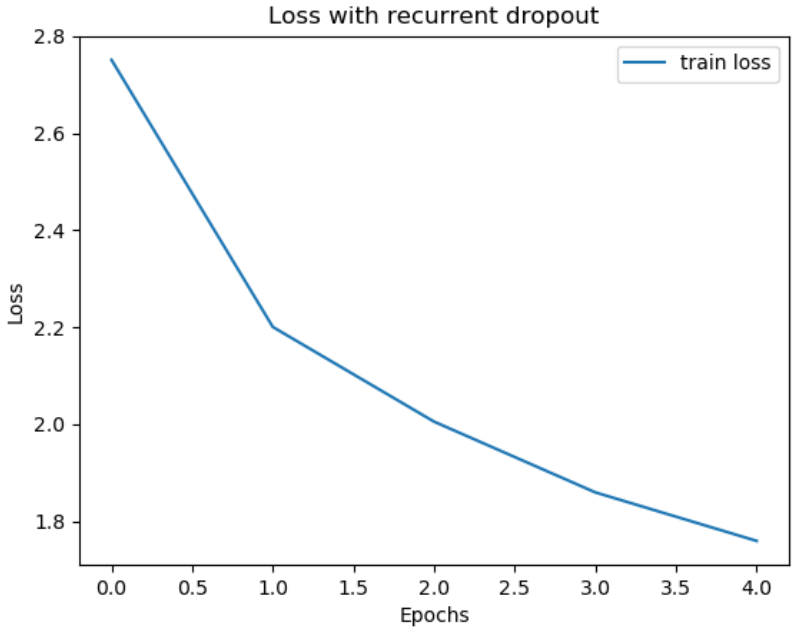
c. برای RNN بررسی می کنیم:

با دراپوت لایه های فوروارد loss به صورت فوق می باشد. که تا 1.66 پایین آمده است.



۷- شکل – نمودار خطا برای سلول RNN با dropout به صورت فوروارد

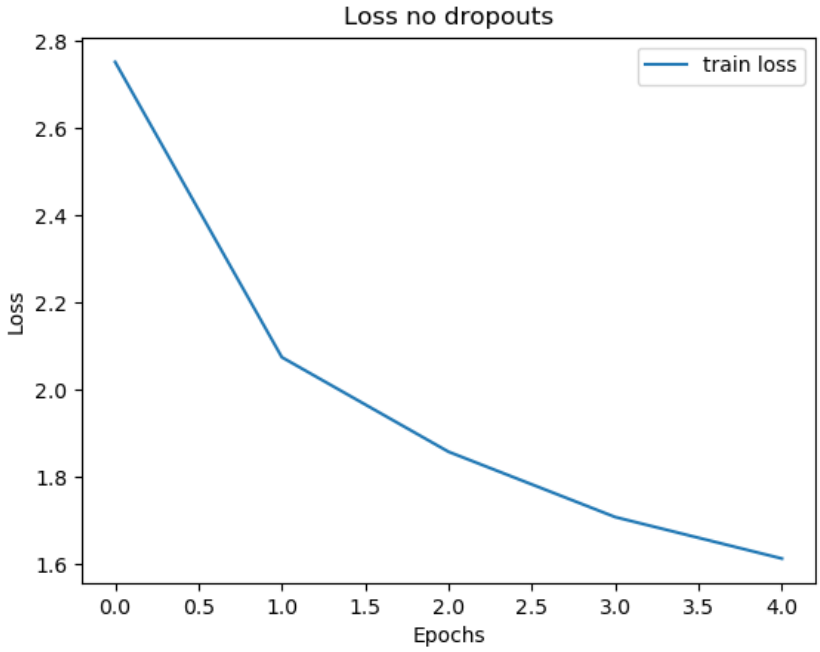
با دراپود ریکارنت نیز loss به صورت زیر در خروجی بود:



۸- شکل – نمودار خطا برای سلول RNN با dropout به صورت ریکارنت

همانطور که مشخص است مقدار آن 1.76 می باشد.

حال بدون استفاده از droupout خواهیم داشت:



۹- شکل – نمودار خطا برای سلول RNN بدون dropout

همانطور که مشخص است مقدار آن 1.61 می باشد.

در جدول زیر نیز خلاصه ای از dropout های مختلف ذکر شده است:

Network	Forward dropout	Recurrent dropout	No dropout
GRU	1.57	nan	1.45
LSTM	1.64	1.49	1.44
RNN	1.66	1.76	1.61

اگر جدول فوق را مقایسه کنیم، به صورت کلی dropout در نتایج ما بهبودی را نشان نداده است. شاید اگر مقدار epoch ها بیشتر می بود، شاهد این بهبود بودیم ولی چون رانتایم شبکه بسیار زیاد بود این مقایسه در زمان معقولی به جواب نرسید. همچنین از جدول فوق نتیجه می شود که حذف نورون ها

در recurrent ها در شبکه RNN و GRU باعث بدتر شدن عملکرد شبکه شده اند و loss افزایش یافته است.

۷. نشان دادن خروجی بر روی ساختارهای ذکر شده

در همه قسمت‌ها خروجی متناظر آورده شده است.

۸. نمایش نمودار loss و مقدار نهایی آن

برای تمامی اجرا ها نمودار loss و عدد نهایی loss قرار داده شده است. مدت زمان اجرا نیز برای آنها حدودا ثابت بوده و برابر۴ دقیقه با GPU MX۹۵۰ بوده است.

۹.

برای این بخش از کد q2_gru.py استفاده می شود. برای تولید متن نیز از فایل q2_gru_generate.py استفاده می شود.

برای حالات زیر نتایج به دست آمدند که در جدول آورده شده اند:

GRU	Given	Answer	Self score
Sequence=100 Shiftining = 1	How are you?	How are you? LUCENTIO: And mosty presently, my plussing here? Though I bolieve her, fares my hearts without a brabe of tayes:	50%

		You alt was that dearested sendy me	
Sequence=50 Shiftining = 20	How are you?	How are you?rcneW t sonEve,htYsohooln fa,tvg;duu h;bh.eeudaNf Y, iCrnr -hYtr.s tpOetdrd eoI'doc:otnesemasepshroa t htNnte at n, 'e h ec oud o igEsegeelAaep	2%
Sequence=١٦٠ Shiftining = 20	How are you?	How are you?EycO'trn asclanmrf nedmWmaeatuzTb Ga bn:eIto?adndnkdna eoadarwi:? frs od tpri h nlv or n'h eot'hmtemrehsvhsea;olf:si Ar 's oy sI retLeg: o nia uU	1%
Sequence=300 Shiftining = 2	How are you?	How are you?IA eo,Itintei esietdtya,i ral,T radng,bdtntn hlet aaltbigvnettmnsoe hm ilhdsqeriet o al w odf gsae o apnl pitoe af:at h ei,Frmnton,mn ud nwa rmtese'	2%
Sequence=400 Shiftining = 1	How are you?	How are you?	40%

		RUCKENGELI: Thee fitht sene thy lellince makede wom has, mist, come minendes. GRYADUTFOR: Hell, in to-dus: hy spimenelf ghites? ButnRUSIO: Shie so	

همانطور که مشخص است، منطقی ترین نتیجه مربوط به اجرای اول می باشد. کلمات معنی دار بیشتر از همه می باشد و در برخی جاها نیز از لحاظ گرامری نیز درست می باشد. پس بهترین حالت مربوط به sequence=100 و shiftining=1 می باشد.

پیوست 1: روند اجرای برنامه

برای تمرین‌های پیاده‌سازی، از کتابخانه tensorflow استفاده شده است و در GPU 950MX ران گرفته شده است.

مراجع

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57

https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/

https://medium.com/@bingobee01/a-review-of-dropout-as-applied-to-rnns-72e79ecd5b7b