صورت سوال: یک شبکه عصبی بازگشتی LSTM با استفاده از Keras طراحی نمایید و از آن برای تحلیل نظرات دیتابیس IMdBاستفاده نمایید. شبکه را با استفاده از این دیتابیس آموزش داده و در نهایت برنامه باید جملات را از کاربر دریافت نموده و با امتیاز دهی در صورت کسب امتیاز کمتر مساوی ۰/۳ آن را منفی و در صورتیکه بالاتر از ۰/۳ باشد آن را مثبت قلمداد کند.

این دیتابیس در آدرس زیر موجود است:

/http://ai.stanford.edu/~amaas/data/sentiment

پاسخ: کد ها رو اینجا میذارم. و همچنین هر قسمت را همراه با تئوری ای که دارد کنار آن توضیح می دهم.

```
import numpy as np
import tensorflow as tf
import numpy as np
from tensorflow import keras
from keras.preprocessing import sequence
from keras import layers
from keras.layers import Embedding
import os
import time
from keras.preprocessing.sequence import pad_sequences
import nltk
from nltk.corpus import stopwords
stop = set (stopwords.words("english"))
import pandas as pd
import matplotlib.pyplot as plt
```

```
from keras.preprocessing.text import Tokenizer

def remove_stopwords (text):
    filtered_words= [word.lower() for word in text.split() \
                     if word.lower() not in stop]
    return " ".join(filtered_words)
```

ابتدا کتابخانه هایی که لازم داریم را وارد کرده ایم. بیشتر کتابخانه ها معروف هستند و نیازی به توضیح ندارند اما چهار کتابخانه ی **nltk** و **pad_sequence** و **embedding** و **Tokenizer** مخصوص این سوال هستند که توضیح می دهم.

کتابخانه nltk: برای این استفاده کرده ام که کلمات رایج و تکراری (stopwords) را از نظرات حذف کنم. الان میتونیم چندتا از این کلمات را ببینیم که کمکی به ما نمی کنند چون تاثیر زیادی در نظرات ندارند.

```
1 stop
```

```
{'a',
 'about',
 'above',
 'after',
 'again',
 'against',
 'ain',
 'all',
 'am',
 'an',
 'and',
```

تابعی که صفحه قبل بود این کلمات را حذف می کند. مثال:

```
1 remove_stopwords("Arash is a student.")
```

```
'arash student.'
```

کتابخانه ی Tokenizer: این کتابخانه نیاز هست چون ما نمیتونیم برداری از کلمات به شبکه بازگشتی LSTM ورودی بدیم. و باید برداری از اعداد (یا بردارها) به شبکه بدیم. این کتاب خانه میاد و به هر کلمه یک عدد تخصیص میده و هر نظر کاربر میشه یک دنباله ای از اعداد. مثال:

```
1  #EXAMPLE
2  t = Tokenizer()
3
4  test_text = ['Deep Learning',
5               'Machine Learning',
6               'Data Mining ! $ % *']
7
8  t.fit_on_texts(test_text)
9
10 sequences = t.texts_to_sequences(test_text)
11
12 print("The sequences generated from text are : ",sequences)
13 print('word_index ', t.word_index)
```

```
The sequences generated from text are :  [[2, 1], [3, 1], [4, 5]]
word_index  {'learning': 1, 'deep': 2, 'machine': 3, 'data': 4, 'mining': 5}
```

همان طور که می بینیم به هر کلمه یک integer تخصیص داده و [2, 1] <= "Deep learning" هم چنین خوبی این تابع این هست که کاراکتر های اضافی را درنظر نمی گیرد مثلا [4, 5] <= "Data Mining ! $ % *" و به خوبی این تبدیل را برای ما انجام می دهد.

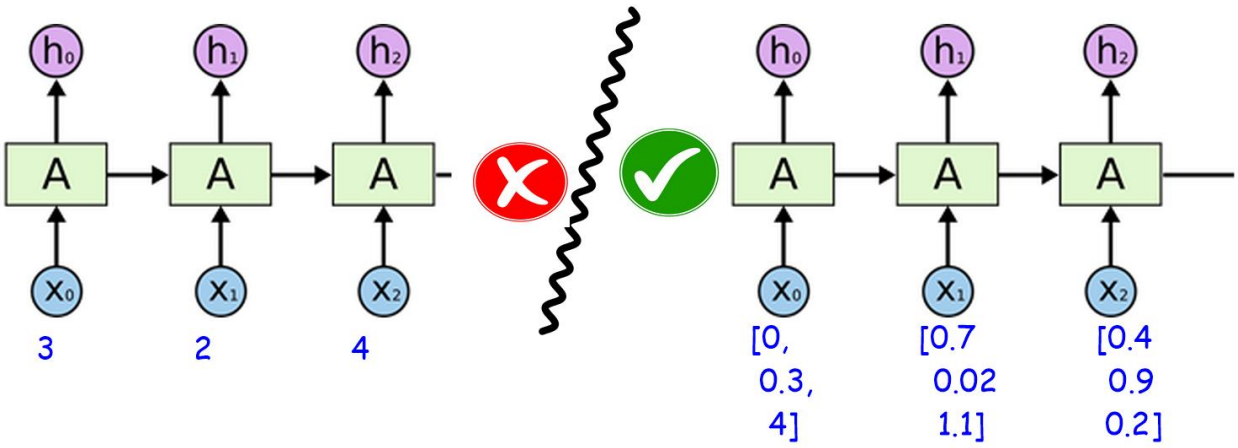
کتابخانه ی pad_sequence: این کتابخانه هم به ما کمک می کند که طول ورودی ها را یکسان کنیم. چون ما باید به شکبه عصبی بازگشتی LSTM ورودی ها با طول یکسان بدیم از نظر کامپیوتری. مثال:

```
KNTU = [np.array([1,5,3]),np.array([2]),np.array([4,1,3,5,8,7,6])]
KNTU_padded=pad_sequences(KNTU, maxlen=5, \
                padding="post", truncating="post")
```

```
print(KNTU[0], " "*9, KNTU_padded[0])
print(KNTU[1], " "*13, KNTU_padded[1])
print(KNTU[2], " "*1, KNTU_padded[2])
```

```
[1 5 3]           [1 5 3 0 0]
[2]               [2 0 0 0 0]
[4 1 3 5 8 7 6]   [4 1 3 5 8]
```

همان طور که می بینید وقتی pad می شوند (در اینجا با حداکثر طول=۵) همه طول ها یکسان ۵ می شوند آنهایی که کمتر دارند صفر می گیرند و آنهایی که بیشتر دارند از کاراکتر ۵ اُم به بعد بریده می شوند. حال می توانیم این ورودی را به شبکه عصبی بدهیم. البته الان یک مشکلی هست ما نمیخواهیم یک سری integer به واحد های ورودی شبکه عصبی دهیم. می خواهیم یکسری بردار به واحد های شبکه عصبی دهیم.



کتابخانه ی embedding: همانطور که دربالا گفتیم باید کلمات یا همون integer ها رو embed کنیم به یک بردار و بردار ها را به شبکه بارگشتی LSTM بدهیم که بتواند آن ها را در ماتریس های weights ضرب کند و واحد های hidden states را حساب کند. برای این تبدیل (کلمات_integerها) به بردار ها از embedding استفاده می کنیم. مثال صفحه ی بعد:

```
embeder=tf.keras.layers.Embedding(4, 3, input_length=3)
```

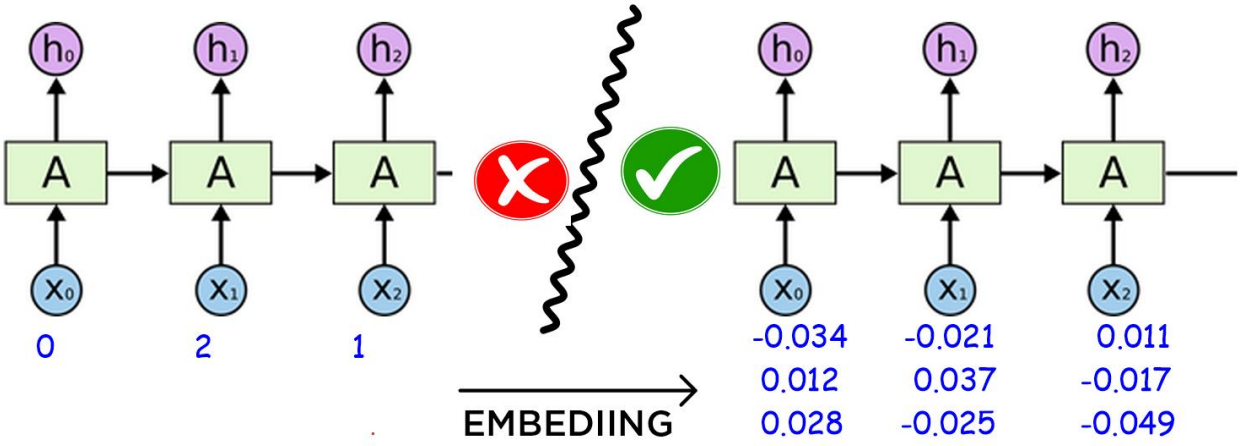
چون ورودی بعد 3 داره

```
data= np.array ([[0,2,1]])
embeder(data)
```

چون تعداد کلمات اینجا کمتر از 4 هست کل کلمات=0 1 2 (3تا)

چون می خواهیم هر عدد را به یک بردار 3 تایی مپ کنه***

```
<tf.Tensor: shape=(1, 3, 3), dtype=float32, numpy=
array([[[-0.0347145 ,  0.01235352,  0.02839499],
        [-0.02170154,  0.03740683, -0.02561101],
        [ 0.01157119, -0.01725961, -0.04921926]]], dtype=float32)>
```



خوب توضیحات کتابخانه ها و روش استفاده از آن ها تمام شد. حالا وارد کد می شویم. از اینجا به بعد کار ساده هست چون فقط می خواهیم از این تکنیک ها استفاده کنیم تا داده ها آماده شوند و فیت کنیم به مدل LSTM ای که با keras به راحتی می توان طراحی کرد.

READ TRAIN

```
path_train_positive = \
r"D:\ALL ABOUT UNIVERSITY\KNTUmaster's\Term2\DL\LMDB\aclImdb\train\pos"
path_train_negative = \
r"D:\ALL ABOUT UNIVERSITY\KNTUmaster's\Term2\DL\LMDB\aclImdb\train\neg"


x_train_positive=[]
x_train_negative=[]

start = time.time()

os.chdir(path_train_positive)
for file in os.listdir():
    if file.endswith(".txt"):
        file = open(f"{path_train_positive}\{file}", 'r', encoding='utf8')
        x_train_positive.append(remove_stopwords(file.read()))
```

```
os.chdir(path_train_negative)
for file in os.listdir():
    if file.endswith(".txt"):
        file = open(f"{path_train_negative}\{file}", 'r', encoding='utf8')
        x_train_negative.append(remove_stopwords(file.read()))

end = time.time()
print('It took', round((end-start),2),    \
      'secs to read positive and negative train')
```

```
It took 135.74 secs to read positive and negative train
```

```
x_train = x_train_positive + x_train_negative
y_train = [1]*12500 + [0]*12500
```

در این قسمت تمام ورودی های train هم مثبت pos و هم منفی neg را خوانده ایم و همچنین ۱۲۵۰۰ تای اول را کامنت مثبت =۱ و ۱۲۵۰۰ تای دوم را کامنت منفی=۰ برچسب داده ایم. فقط دقت کنید که کلمات رایج (stopwords) قبل از وارد شدن دیتا به لیست حذف شده اند.

append(remove_stopwords(file.read()))

READ TEST

```
path_test_positive = \
r"D:\ALL ABOUT UNIVERSITY\KNTUmaster's\Term2\DL\LMDB\aclImdb\test\pos"
path_test_negative = \
r"D:\ALL ABOUT UNIVERSITY\KNTUmaster's\Term2\DL\LMDB\aclImdb\test\neg"


x_test_positive=[]
x_test_negative=[]

start = time.time()

os.chdir(path_test_positive)
for file in os.listdir():
    if file.endswith(".txt"):
        file = open(f"{path_test_positive}\{file}", 'r', encoding='utf8')
        x_test_positive.append(remove_stopwords(file.read()))
```

```
os.chdir(path_test_negative)
for file in os.listdir():
    if file.endswith(".txt"):
        file = open(f"{path_test_negative}\{file}", 'r', encoding='utf8')
        x_test_negative.append(remove_stopwords(file.read()))

end = time.time()
print('It took', round((end-start),2),      \
      'secs to read positive and negative test')
```

```
It took 133.94 secs to read positive and negative test
```

مشابهاً برای داده ی آزمون هم این کد تکرار شده است.

review => bag of words => id's of word

```
x_full = x_train + x_test

t1 = Tokenizer()
t1.fit_on_texts(x_full)
num_of_unique_words= len(t1.word_counts)
print("Number of unique words in All Data: ", num_of_unique_words)
#124,252 unique words =stopwords=> 123,277
```

```
Number of unique words in All Data:  123277
```

در این قسمت همه ی دیتا را نگاه کرده ایم و با یک tokenizer تعداد کل کلمات متمایز را حساب کرده ایم (۱۲۳۲۷۷).

```
tokenizer = Tokenizer (num_of_unique_words)
tokenizer.fit_on_texts(x_train)
```

```
x_train_sequences = tokenizer.texts_to_sequences(x_train)
x_test_sequences = tokenizer.texts_to_sequences(x_test)
```

سپس توسط یک tokenizer دیگر تمامی کامنت ها به اعداد تبدیل شده اند. یک مثال میبینیم:

```
x_train[13568]
```

```
'movie terrible good effects.'
```

```
np.array(x_train_sequences[13568])
```

```
array([  2, 306,   6, 213])
```

```
for i in range (0,len(x_train_sequences)):    #change to numpy format
    x_train_sequences[i] = np.array( x_train_sequences[i])

for i in range (0,len(x_test_sequences)):
    x_test_sequences[i] = np.array( x_test_sequences[i])
```

سپس همه چیز را بفرمت numpy تبدیل می کنیم تا کار کردن شبکه عصبی با آن راحت شود.

```
max, sum, min, ind=0, 0, 2000, -1

for j in range (0, len(x_train)):

    sum = sum + len(x_train_sequences[j])
    if len(x_train_sequences[j])>max:
        max=len(x_train_sequences[j])
    if len(x_train_sequences[j])<min:
        min=len(x_train_sequences[j])
        ind=j

print("max length of a message: ", max)
print("min length of a message: ", min)
print("avg length of a message: ", sum/25000)
```

```
max length of a message:  1495
min length of a message:  4
avg length of a message:  132.69968
```

این قسمت من کل train رو گشتم ببینیم بیشترین طول نظر و کمترین طول نظر و طول متوسط نظرات چقدر هست که یک عدد مناسب برای padding پیدا کنم.

```
max_length= 140
x_train_padded = pad_sequences(x_train_sequences, \
                maxlen=max_length, padding="post", truncating="post")
x_test_padded = pad_sequences(x_test_sequences, \
                maxlen=max_length, padding="post", truncating="post")
x_train_padded.shape, x_test_padded.shape
```

```
((25000, 140), (25000, 140))
```

من دیدم بطور متوسط طول کامنت ها 132 هست پس بیشترین طول کامنت رو 140 گذاشتم. یعنی اگر کمتر بود padding صفر اضافه کند. اگرم بیشتر بود که از 140 اُمین کلمه به بعد cut کن. البته اعداد دیگری هم امتحان کردم این ظاهراً از همه بهتر بود. اعداد خیلی زیاد هم واقعاً شبکه را کند می کرد.

LSTM MODEL

```
model = keras.models.Sequential()
model.add(layers.Embedding(num_of_unique_words, 32, \
                           input_length=max_length))
model.add(layers.LSTM(80, dropout=0.15, activation="tanh"))
model.add(layers.Dense(1, activation="sigmoid"))

model.summary()
```

خوب یک مدل می سازیم که این کلمات را ابتدا خودش embed کنه و سپس وارد lstm شود. هر کلمه (integer متناظر با اون کلمه) به یک بردار ۳۲ بُعدی در embedding تبدیل می شود. سپس به LSTM(80, ...) داده می شود. 80 یعنی اون خروجی hidden state یک بردار 80 تایی خواهد بود. می توان اعداد دیگر هم امتحان کرد.
Dropout=0.15 یعنی با احتمال ۱۵ درصد یک نورون ممکن است حذف شود. (جلوگیری از overfitting)

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_35 (Embedding)    (None, 140, 32)           3944864

 lstm_4 (LSTM)               (None, 80)                36160

 dense_4 (Dense)             (None, 1)                 81

=================================================================
Total params: 3,981,105
Trainable params: 3,981,105
Non-trainable params: 0
_________________________________________________________________
```

تعداد پارامتر های لایه ی embedder رو نمی تونیم بشماریم. Embedder خودش یک مقاله هست با نام **Efficient Estimation of Word Representations in Vector Space** که خودش یک شبکه عصبی جدا هست. ولی تعداد پارامتر های LSTM را با درسی که داشتیم میتونیم راحت بشماریم.

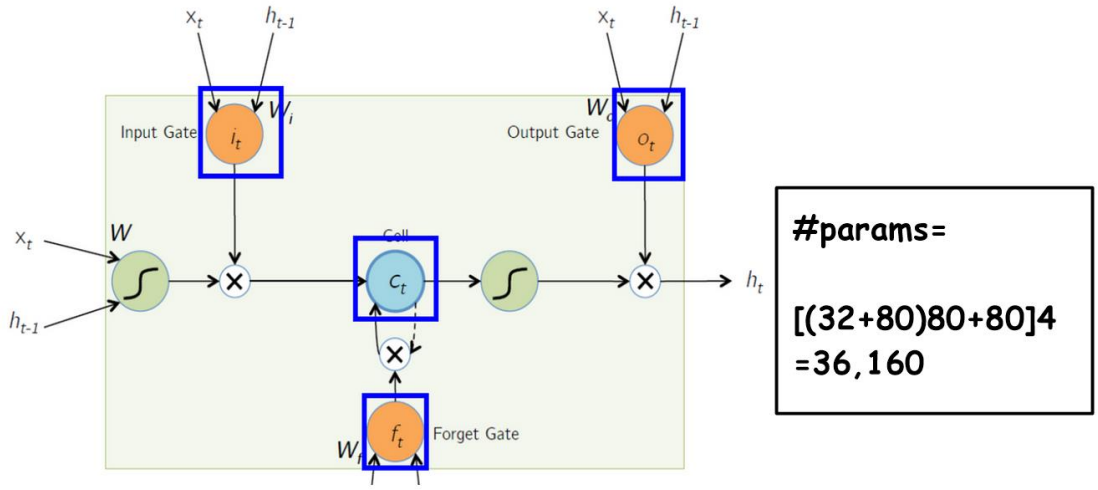
الان ورودی ها بعد از embed شدن 32 تایی هستن. و چون LSTM(80,...) هست یعنی این بردار ۳۲ تایی در کنار یک بردار hidden state قبلی که ۸۰ تایی هست قرار می گیرد و سپس باید ضرب در یک ماتریسی بشه که این بردار طولانی را دوباره به ۸۰ تایی تبدیل کنه (hidden state جدید). پس این ماتریس ابعاد

$$(80+32)*80$$

دارد همچنین 80 تا پارامتر هم برای bias ها هست. پس مجموعاً

$$(80+32)*80+80=9040$$

پارامتر داریم. اما پارامتر ها گفته 36,160 این به این دلیل است که این محاسبات برای یک RNN معمولی simpleRNN بود اینجا LSTM هست. هر LSTM چهار برابر یک simpleRNN پارامتر داره بخاطر cell, forget, input, output هر کدام به همین تعداد پارامتر دارند.




```
loss = keras.losses.BinaryCrossentropy(from_logits=False)
#optim = keras.optimizers.Adam(learning_rate=0.001)
optim = keras.optimizers.Adamax(learning_rate=0.001)
metrics = ["accuracy"]

model.compile(loss=loss, optimizer=optim, metrics=metrics)
```

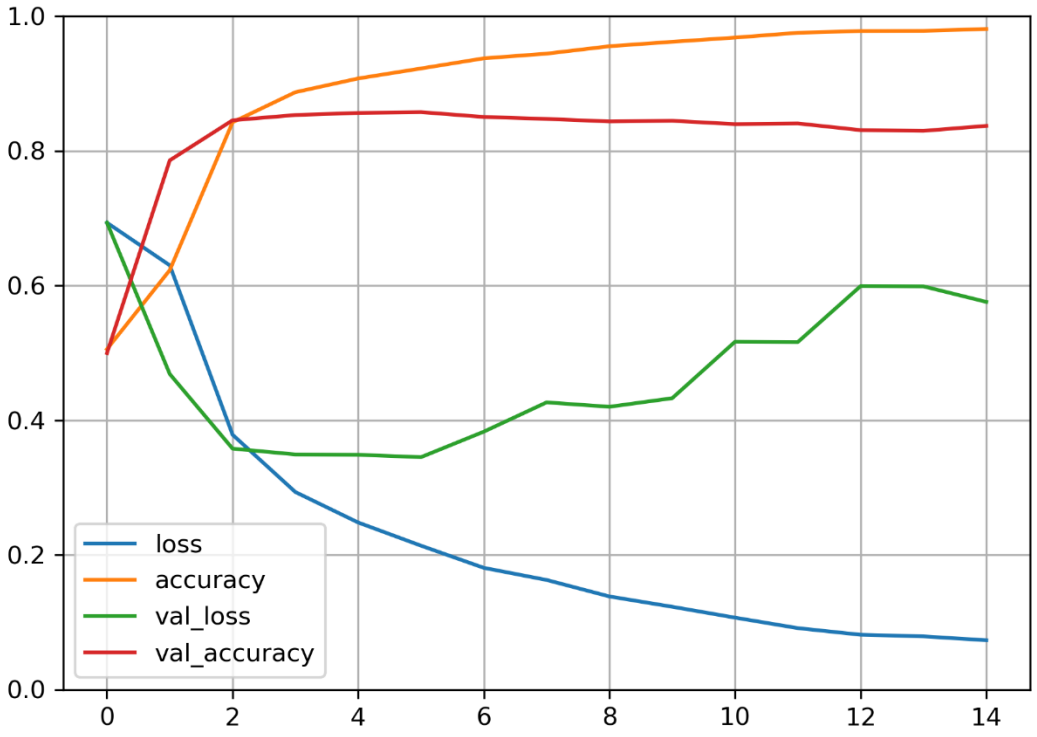
```
kee = model.fit(x_train_padded, np.array(y_train), \
                epochs=15, batch_size=150, \
                validation_data=\
                (x_test_padded, np.array(y_test)),verbose=2 )
```

خوب تابع loss و optimizer تعریف می شوند. ابتدا Adam استفاده کردم خیلی دیر همگرا می شود (بعد از ۱۰ مرحله تقریباً) سپس تو سایت خوندم نوشته بود Adamax برای مدل هایی که embedder دارند سریع تر و بهتر کار

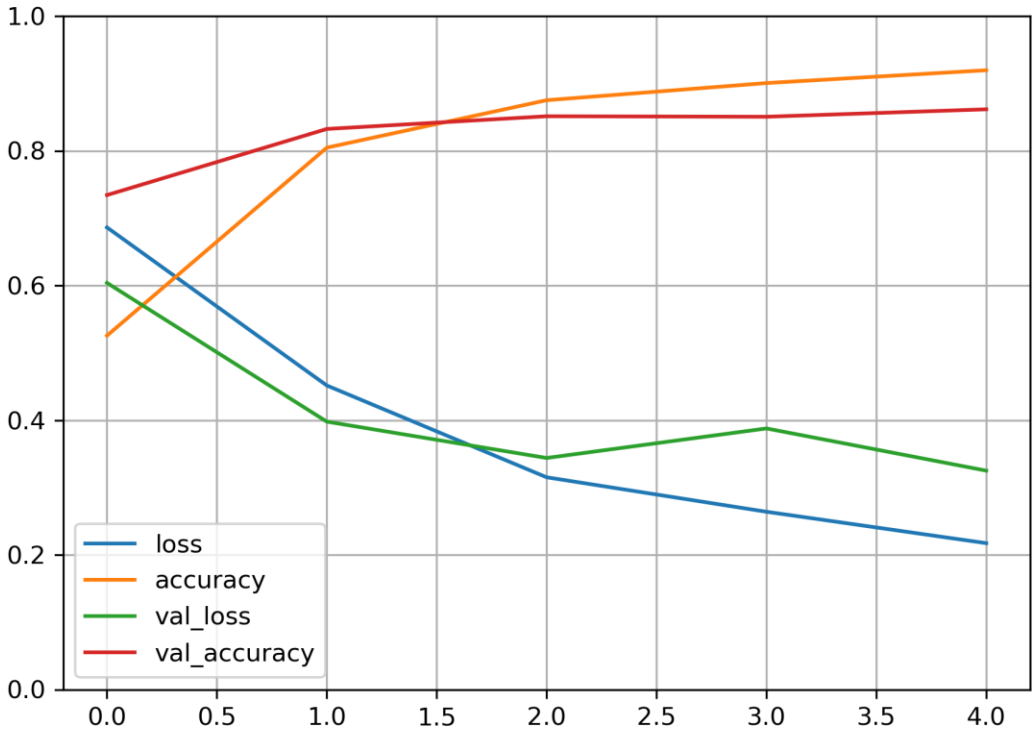
می کنه. پس از optimizer Adamax استفاده کردم. سپس دیتا را فیت می کنیم. و بعد از هر ۱۵۰ تا نمونه وزن ها را آپدیت می کنیم. نتیجه:

```
Epoch 1/15
loss: 0.6931 - accuracy: 0.5054 - val_loss: 0.6931 - val_accuracy: 0.4995
Epoch 2/15
loss: 0.6299 - accuracy: 0.6227 - val_loss: 0.4684 - val_accuracy: 0.7857
Epoch 3/15
loss: 0.3783 - accuracy: 0.8423 - val_loss: 0.3577 - val_accuracy: 0.8452
Epoch 4/15
loss: 0.2932 - accuracy: 0.8869 - val_loss: 0.3490 - val_accuracy: 0.8530
Epoch 5/15
loss: 0.2480 - accuracy: 0.9073 - val_loss: 0.3485 - val_accuracy: 0.8561
Epoch 6/15
loss: 0.2136 - accuracy: 0.9222 - val_loss: 0.3450 - val_accuracy: 0.8573
Epoch 7/15
loss: 0.1807 - accuracy: 0.9372 - val_loss: 0.3828 - val_accuracy: 0.8502
Epoch 8/15
loss: 0.1627 - accuracy: 0.9442 - val_loss: 0.4263 - val_accuracy: 0.8472
```

همان طور که میبینیم بعد از ۴ یا پنج مرحله دیگه روی test بهبودی نداریم.



پس من یکبار دیگه فقط تا ۵ epoch آموزش میدم که overfit نشده باشه. نتیجه میشه:



و نتیجه ی آخرین epoch:

```
Epoch 5/5
loss: 0.2173 - accuracy: 0.9195 - val_loss: 0.3250 - val_accuracy: 0.8615
```

پس به دقت ۹۱ درصد روی داده ی آموزش و ۸۶ درصد روی داده ی آزمون رسیده ایم.

حالا چند مورد هم خودم میخوام بنویسم ببینم این شبکه چقدر میتونه خوب عمل کنه و کجاها دیگه نمیتونه تحلیل کنه نظر من رو. من یک نظر می نویسم و کارهای tokenizing و padding و remove_stopwords و embedding همه رو انجام میدم و به شبکه عصبی می دهم. انتظار دارم **اگر کامنتم بد باشه کمتر از 0.3 بده اگرم خوب باشه بیشتر از 0.3 خروجی دهد.** تابع فعالساز تک واحد خروجی sigmoid هست پس همیشه بین 0 و 1 می دهد. این کار را با کد زیر انجام می دهم.

TEST ON MY COMMENTS

```
tt = [" It was the best movie I have seen in my life"]
tt = [remove_stopwords(tt[0])]
tkn=tokenizer.texts_to_sequences(tt)
tkn_np = np.array (tkn)
arash_padded = pad_sequences(tkn_np, maxlen=max_length, \
                             padding="post", truncating="post")
```

```
model.predict(arash_padded)
```

```
array([[0.8846246]], dtype=float32)
```

TEST ON MY COMMENTS

```
tt = [" It was the worst movie I have seen in my life"]
tt = [remove_stopwords(tt[0])]
tkn=tokenizer.texts_to_sequences(tt)
tkn_np = np.array (tkn)
arash_padded = pad_sequences(tkn_np, maxlen=max_length, \
                             padding="post", truncating="post")
```

```
model.predict(arash_padded)
```

```
array([[0.05919594]], dtype=float32)
```

خوب بنظر میرسه که داره خوب کار می کنه. یه چندتا کامنت دیگه هم بدم:

Comment	RNN OUTPUT	PERFORMANCE
"Although the movie seemed to be bad at first I really loved it by the end of it. The ending was nice."	0.592	ACCEPTABLE
"My friends said it was not good at all, but what I thought was quite different I think it was good but it could've been even better"	0.827	ACCEPTABLE
"Everyone should see this movie"	0.709	ACCEPTABLE
"Everyone should see this movie to see how bad it is. Then they can appreciate other good movies we have."	0.749*	NOT ACCEPTABLE
"To be honest, I didn't like this movie. The director didn't do well neither did the actors. It was boring and long. They could've conveyed the message in much less time."	0.067	ACCEPTABLE

نسبتاً خوب کار می کند. فقط یک نظر که با ستاره * مشخص کردم. منفی هست ولی مثبت تشخیص داده.

"Everyone should see this movie to see how bad it is. Then they can appreciate other good movies we have."

البته تشخیص همچین نظری منطقی هست که برای ماشین باید سخت باشد.

(کد ها را جدا ارسال می کنم.)

پایان