**به نام خداوند بخشنده و مهربان**

**نام و نام خانوادگی : حسین سیم چی**

**نام استاد : خانم دکتر عبدوس**

**عنوان : پروژه ی پایان ترم (گزارش کار)**

**درس : شبکه های عصبی**

**ترم پاییز و زمستان 98**

**فهرست :**

**مقدمه ............................................... 3**

**LSTM............................................. 3**

**CNN-LSTM................................... 14**

**بخش 3 و 4 پروژه ............................. 15**

**RESULT OF LSTM ........................... 17**

**مقدمه :**

در بخش اول به بررسی پارامترها و نوع ایجاد مدل LSTM در شبکه های عمیق می پردازیم . سپس با استفاده از ترکیب شبکه های کانولوشن و LSTM سعی در ایجاد مدلی با نام CNN\_LSTM می کنیم .

لازم بذکر است که در انتها و در بخش مربوط به کد CNN-LSTM با در نظر گرفتن لایه های جدید و ایجاد متدهایی سعی در افزایش دقت داده به بالای 90 درصد می نماییم .

کدهای نوشته شده در محیط برنامه نویسی Anaconda و در قسمت Spyder نوشت شده است

**LSTM\_MODEL :**

در ابتدا ذکر این نکته ضروری است که قبل از شرئع هر کاری باید کتابخانه های موردنظر را به داخل برنامه وارد نماییم تا سرعت کدنویسی ما افزایش یابد پس همانطور که در پایین ملاحظه می نمایید کتابخانه های موردنظر خود را وارد می نماییم :

from keras\_layer\_normalization import LayerNormalization

import numpy as np

from keras.models import Sequential

from keras.layers import LSTM,Dense,Dropout

from keras.utils import np\_utils

import matplotlib.pyplot as plt

from keras.utils import plot\_model

import datetime

در بخش بعد باید دیتاست خود را که همانطور که در بخش فایل توضیحات پرووژه آمده است وارد نماییم که در اینجا ما از یکی از کتابخانه های Numpy استفاده می نماییم ، کتابخانه ی استفاده شده و طریق خواندن فایل کد در پایین آمده است :

x\_train = np.genfromtxt(r'C:\\Users\\Lenovo\\Desktop\\x\_train.txt',delimiter="")

y\_train = np.genfromtxt(r'C:\\Users\\Lenovo\\Desktop\\y\_train.txt',delimiter="")

x\_test = np.genfromtxt(r'C:\\Users\\Lenovo\\Desktop\\x\_test.txt',delimiter="")

y\_test = np.genfromtxt(r'C:\\Users\\Lenovo\\Desktop\\y\_test.txt',delimiter="")

پس از اینکار اگر با فشار دادن Ctrl+Enter میتوانیم هر قسمت دلخواه را مشاهده نماییم . همانطور که در قسمت Variabel Explorer مشاهده می شود ، قسمت داده ایی ما شامل دو بخش Test و Train میباشد که هر کدام نیز شامل دو بخش داده های خام و Label مربوط به هر کدام میباشد . همانطور که مشاهده می شود داده ی آموزش ما شامل 7352 سطر و 561 ستون میباشد برای اینکه بتوانیم به صورت برداری تک بعدی داده های خود را به سیستم بدهیم نیاز داریم تا تا سایز آنها را مانند زیر تغییر دهیم ، و از طرفی باید Label های داده های خود را به برداری 7 مولفه ایی تبدیل کنیم که برای هر داده فقط یک مولفه آن یک و بقیه آنها صفر خواهد بود ، مثلا اگر داده اول ما دارای Label با مقدار 5 باشد پنجمین مولفه ی بردار ما یک و بقیه ی مولفه های ما صفر خواهد بود که به اصطلاح به آن بردار One Hot می گویند .درضمن می توان به دستور astype نوع داده ی خود را نیز تغییر دهیم

پس مراحل فوق به شرح زیر است :

y\_train = np\_utils.to\_categorical(y\_train)

y\_test = np\_utils.to\_categorical(y\_test)

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

سپس باید دنبال راهکاری باشیم تا Input\_Shape داده های خود را مشخص نماییم . چرا که در ابتدا ما دارای داده های آموزش و تست دو بعدی هستیم و میخواهیم آنها را به یک بعد تغییر دهیم . برای اینکار از دستور Resize مانند زیر استفاده می نماییم :

x\_train = np.reshape(x\_train, (x\_train.shape[0],x\_train.shape[1],1))

x\_test = np.reshape(x\_test,(x\_test.shape[0],x\_test.shape[1],1))

که با اینکار داده های ما به تنها یک ستون تبدیل میشود که میتوان در ادامه از آنها برای خواندن و ورودی مدل خود استفاده کنیم

حال به بخش اصلی و مهم طراحی خود یعنی طراحی مدل LSTM میرسیم . در شبکه های عمیق موجود در کراس ، ما میتوانیم مدل خود را وارد نماییم و در بخش طراحی از آن استفاده کنیم

میتوان مدل خود را به دو صورت Model و Sequential طراحی نمود که در بخش Model میتوان مدل خود را به هر نحوی طراحی کنیم . تنها تفاوت عمدخه بین این دو روش در مشخص کردن ورودی هر لایه میباشد . در Model ما باید ورودی هر لایه را نیز مشخص نماییم ولی وقتی از Sequential استفاده می نماییم دیگر نیاز به تعریف کردن ورودی هر لایه نداریم و لایه ها به شکل پشت سرهم به دیگری انتقال میابند . نکته حائز اهمیت از تفاوت این دو دل میتوان به این نکته اشاره نمود که در طراحی با Model میتوان چندین ورودی را به تنها یک ورودی داد ولی در مدل Sequential نمیتوانیم از ین ویژگی استفده نماییم .

در اینجا با بررسی کارکرد شبکه و دیتاست که همانطور که در فایل توضیحات شبکه آورده شده است چون دیتای ما کاملا به هم وابستگی دارند از Sequential استفاده می کنیم :

LSTM\_MODEL\_SIMCHI = Sequential()

در قسمت بعد باید لایه ی ورودی خود را تعریف نماییم به این صورت که من در اینجا از لایه ی LSTM به عنوان وردی استفاده کردم . اگر مدل خود را به صورت Model طراحی کنیم میتوان از لایه ایی به عنوان Input نیز استفاده کنیم . لایه ی اول ما دارای Input\_shape میباشد که در آن مولفه ی اول به نام Timestep شناخته میشود و مولفه دیگر به نام Feature ما میباشد که در اینجا در اثر تبدیل اولیه ایی که انجام دادیم ایم مولفه به عدد یک تبدیل شد ، از تابع فعال سازی خاصی استفاده نکردیم و میخواهیم فقط شبکه در این لایه ورودی ها را گرفته و با تعداد 32 نورون به خروجی تحویل دهد . عدد تعداد نورون خروجی کاملادلبخواهی است و من با سعی و خطا و آموزش بیش از 15 بار این تعداد برای دریافت دقت بالای 85 درصد مناسب دیدم :

LSTM\_MODEL\_SIMCHI.add(LSTM(32,return\_sequences=True,input\_shape=(x\_train.shape[1],1)))

توجه : باتوجه به تحقیقات انجام شده در این زمینه از طریق سایتها و مراجع مختلف به این موضوع رسیدم که باید در تمامی لایه های LSTM به جز لایه ی اخر منتهی به OUTPUT مقدار پارامتر return\_sequences را برابر True بگذاریم ، در عیر این صورت با پیغام خطایی شامل عدم تطابق ابعاد ورودی با خروجی و لایه های متفاوت مواجه خواهیم شد .

بعد از لایه ی اول با سعی و خطا و متوجه شدن این حقیقت که بدون استفاده از نرمال سازی داده ها دقت خوب و بالایی را بدست نمیاریم ، از لایه ی آماده ایی که کراس دارد استفاده می کنیم :

LSTM\_MODEL\_SIMCHI.add(LayerNormalization())

سپس از لایه ی دیگری که بازهم لایه ی LSTM میباشد استفاده می کنیم که تفاوت آن با لایه اول یا به اصطلاح ورودی در استفاده از تابع فعال سازی “Tanh” میباشد .

اینکه چرا از این تابع استفاده نمودیم کاملا صورت سعی و خطا و مقایسه ی دقت در حالتهای مختلف میباشد . در ابتدا بدون تابع فعال ساز و در ابتدا با تابع فعال ساز “Sigmoid” سعی در بدست آوردن نتیجه داشتم که درصد میزان دقت بر روی داده های Validation حداکثر 60 و در داده های تست نیز ماکزیمم 70 درصد میباشد . ولی استفاده از این تابع فعال ساز دقتی حدود 85 درصد با تعداد Epoch های 20 و Batch\_size با مقدار 100 را به ما برمیگرداند (بعدا در ممورد مقادیر Epoch و بقیه پارامترها مفصلا توضیح خواهم داد .

LSTM\_MODEL\_SIMCHI.add(LSTM(32, return\_sequences=True,activation='tanh'))

اگر شبکه را تنها با این لایه ها طراحی و تست کنیم متوجه میشویم که پس از مدتی دقت روی داده آموزش زیاد و بر روی داده های Validation کم میشود که این مسئله در شبکه ها عمیق به عنوان Overfitting شناخته میشودد که برای جلوگیری از این اتفاق از لایه های Dropout استفاده می کنیم که مقدار درون آن کاملا اختیاری است و من اینجا مقدار آن را 2/0 قرار داده ام بدین معنا که 20 درصد تعداد نورونها در هربار آموزش استفاده نمیشود :

LSTM\_MODEL\_SIMCHI.add(Dropout(0.2))

در قسمت بعد نیز باز از لایه ی دیگر به اسم LSTM استفاده کردم که صرفا عمل Mapping را به لایه ی خروجی انجام میدهد

LSTM\_MODEL\_SIMCHI.add(LSTM(32))

و در لایه ی اخر نیز از لای Dense استفاده کرده ام که تنها کار طبقه بندی داده ها را انجام میدهد به این صورت که 7 خروجی متناسب با تعداد مولفه های برچسب های ما دارد و تابع فعال ساز “Sigmoid” استفاده کردیم چون میخواهیم خروجی ما اعدادی 0 یا 1 مثل بردار One Hot تبدیل شود

LSTM\_MODEL\_SIMCHI.add(Dense(7, activation='softmax'))

تا الان مدل ما کاملا ساخته شده است و در ادامه به بررسی توابع و پارامترهای استفاده شده می پردازیم

* می توان مانند دستور زیر اسم هر لایه را تغییر داد :

LSTM\_MODEL\_SIMCHI.layers[0].name = 'INPUT '

* میتوان شمایی از مدل را همراه اسمی در فایل جداگانه ذخیر نمود که اینکار نیز با دستور زیر صورت می گیرد و فایل ذخیر شده با پسوند PDF به صورت جدگانه در Courseware بارگذاری شده است :

plot\_model(LSTM\_MODEL\_SIMCHI,to\_file='HOSSEIN\_SIMCHI\_LSTM\_PROJECT1.pdf',show\_shapes=True)

* در این بخش باید مدل خود را کامپایل نماییم که در اینجا پس از بررسی پارامتذهای مختلف به این نتیجه رسیدم که برای Optimizer مقدار adam را انتخاب نمایم چراکه خروجی بهتری نسبت به بقیه دارد برای تابع هزینه نیز به همین صورت و با سعی و خطا و پس از انجام چندین بار فرایند آموزش این را انتخاب کرده ام .

LSTM\_MODEL\_SIMCHI.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

* در بخش بعد میتوان با دستور زیر خلاصه ایی از مدلی که طراحی نموده ایم را در کنسول برنامه مشاهده نماییم :

print(LSTM\_MODEL\_SIMCHI.summary())

* در این برنامه و بخش بعدی که ارائه خواهم داد برای دستیابی به زمان کل مدلی که آموزش داده قبل از آموزش مدل ودر انتها از توابع آماده استفاده میکنم . برای آموزش مدل خود باید چندین پارمتر را به آن بدهیم که از جمله آنها داه های بخش آموزش و برچسب مربوط به هرکدام میباشد ، پارامتر بعدی که بسیار اهمیت دارد انتخاب تعداد Epoch ها میباشد که اگر تعداد زیادی قرار دهیم ممکن است مسئله ی Overitting صورت بگیرد و اگر خیلی کم باشد طبیعتا به دقت 85 درصد دست نخواهیم یافت با بررسی های صورت گرفته تقریبا با تعداد Epoch برابر 13 میتوان به دقتی برابر 85 درصد رسید ولی برای اینکه بدانیم درنهایت این مدل طرح شده ی ما به چه عددی همگرا میشود این تعداد را برابر 15 قرار داده ام . پارامتر مهم بعدی که باید آن را مهم بدانیم میزان سایز Batch ما در آموزش شبکه میباشد . کاملا بعد از تعدادی سعی و خطا و آموزش های متوالی میتوان مشاهده کرد که اگر اندازه را حدود 500 درنظر بگیریم نهایتا به دقت 65 درصد خواهیم رسید و هرچه سایز یا اندازه زا کاهش دهیم این درصد افزایش میباشد . پس از بررسی های فراوان دریافتم که اگر اندازه را از حدی کوچکتر بگیریم بازهم ممکن است دچار Overfitting شویم که باتوجه به بررسی های متوالی این عدد را نیز 100 درنظر گرفته ام که همانطور که از نمودارها و خروجی کار مشخص است در حدود Epoch 13 تقریبا به دقت 85 درصد میرسیم که عددی خوب و قابل توجهی است

start = datetime.datetime.now()

network\_history = LSTM\_MODEL\_SIMCHI.fit(x\_train,y\_train, epochs=20, batch\_size=100,validation\_split=0.2)

end = datetime.datetime.now()

elapsed = end - start

print('Total training time : ',str(elapsed))

در قسمت بالا میتوان قسمتی از داده ی آموزش را به Validation اختصاص داد که در اینجا ما آن را 2/0 قرار داده ایم معادل اینکه 20 درصد داده های آموزش خود را صرف این پارامتر می کنیم

* پس از گذراندن تمامی این مراحل نئبت به ارزیابی مدل خود روی داده های Test میرسد که میتوان با تابع زیر این کار رانیز به راحتی انجام داد :

test\_loss,test\_acc=LSTM\_MODEL\_SIMCHI.evaluate(x\_test,y\_test)

print(test\_loss)

print(test\_acc)

* برای رسم نمودار دقت و تابع هزینه یا خطا میتوان از کتابخانه ی Matplotlib استفاده نمود که میتوان در آن از توابعی همچون Legend برای مشخص نمودن نام هر نمودار که رنگی به آن تخصص دارد و xlabel و ylabel برای مشخص کردن اسم محور افقی و عمودی استفاده نماییم که به شرح زیر است : (نمودار ها نیز در Courseware بارگذاری شده اند)

history = network\_history.history

losses\_LSTM\_SIMCHI = history['loss']

accuracies\_LSTM\_SIMCHI = history['acc']

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.plot(losses\_LSTM\_SIMCHI)

val\_losses = history['val\_loss']

plt.plot(val\_losses)

plt.legend(['loss','val\_loss'])

plt.figure()

plt.xlabel('Epochs')

plt.ylabel('accuracy')

plt.plot(accuracies\_LSTM\_SIMCHI)

val\_accuracies = history['val\_acc']

plt.plot(val\_accuracies)

plt.legend(['acc','val\_acc'])

در اینجا به پایان بخش اول پروژه میرسیم

در بخش بعدی به بررسی مدل CNN\_LSTM میپردازیم و درهمین بخش سعی می کنیم با بررسی بیشتر و انتخاب لایه های مناسب دیگری به نوعی بخش های سوم و چهارم را نیز پوشش دهیم .

**CNN\_LSTM :**

تمامی نکات بیان شده در قسمت قبلی در اینجا نیز یکسان میباشد با تفاوتی اندک ، به این صورت که در این شبکه از لایه کانولوشن یک بعدی نیز استفاده می کنیم

همچین در اینجا برای افزایش دقت بالای 90 درصد (بخش سوم و چهارم پروژه) از لایه هایی به نام BatchNormalization استفاده می کنیم . چیزی که بارها آن را تست نموده ام و نتیجه گرفتم ان است که شبکه های CNN به شدت به تقارن لایه های آن وابسته و حساس هستند به خاطر همین موضوع اگر کمی با دقت به این شبکه ی طراحی شده نگاه کنیم خواهیم دید که تقارن در شبکه ی ما وجود دارد معماری ما برای این شبکه به شرح زیر است :

CNNLSTM\_MODEL\_SIMCHI = Sequential()

CNNLSTM\_MODEL\_SIMCHI.add(LSTM(30,return\_sequences=True,activation = 'sigmoid',input\_shape=(561,1)))

CNNLSTM\_MODEL\_SIMCHI.add(LayerNormalization())

CNNLSTM\_MODEL\_SIMCHI.add(BatchNormalization(momentum=0.99, epsilon=0.1))

CNNLSTM\_MODEL\_SIMCHI.add(Dropout(0.2))

CNNLSTM\_MODEL\_SIMCHI.add(Convolution1D(16, (3), activation='relu'))

CNNLSTM\_MODEL\_SIMCHI.add(keras.layers.BatchNormalization(momentum=0.99, epsilon=0.1))

CNNLSTM\_MODEL\_SIMCHI.add(keras.layers.Dropout(0.2))

CNNLSTM\_MODEL\_SIMCHI.add(LSTM(30,return\_sequences=True,activation = 'sigmoid'))

CNNLSTM\_MODEL\_SIMCHI.add(keras.layers.BatchNormalization(momentum=0.99, epsilon=0.1))

CNNLSTM\_MODEL\_SIMCHI.add(keras.layers.Dropout(0.2))

CNNLSTM\_MODEL\_SIMCHI.add(Convolution1D(32, (3), activation='relu'))

CNNLSTM\_MODEL\_SIMCHI.add(keras.layers.BatchNormalization(momentum=0.99, epsilon=0.1))

CNNLSTM\_MODEL\_SIMCHI.add(keras.layers.Dropout(0.2))

CNNLSTM\_MODEL\_SIMCHI.add(LSTM(30,return\_sequences=False,activation = 'sigmoid'))

CNNLSTM\_MODEL\_SIMCHI.add(keras.layers.BatchNormalization(momentum=0.99, epsilon=0.1))

CNNLSTM\_MODEL\_SIMCHI.add(Dropout(0.2))

CNNLSTM\_MODEL\_SIMCHI.add(Dense(7, activation='softmax'))

* نکاتی در مورد لایه ی BatchNormalization :

این لایه دارای پارمترهای زیادی است ولی دو مورد از اساسی ترین پارامترهای ان میتوان به ثابت momentum و ٍEpsilon اشاره کرد در واقع سرعت حرکت به سمت مینیمم بهینه و افزایش دقت را سبب میشوند

مقدار momentum را برابر بیشینه آن گذاشتیم و کاملا با سعی و خطا ان را انتخاب نموده ام . مقدار Epsilon هر عددی مثل 1/0 ، 01/0 ، 001/0 یا 0001/0 میتوان قرار داد ( به صورت روال و معمول) ولی اینجا با بررسی های صورت گرفته عدد 1/0 رامناسب دیده و اگر به اوایل بخش آموزش نگاه کنید متوجه خواهید شد چقدر سرعت رسیدن ما به حالت ثبل (LSTM) افزایش یافته است

اندازه و تعداد لایه ها در اینجا نیز کاملا اختیاری است ولی با بررسی بیشتر این تعداد یا کمتر از این مقدار به دقت بالای 90 درصد نمیرسیم

فرض بر این است که در این مدل طراحی شده مانند LSTM قبلی است با این تفاوت که ورودی و خروجی آن هردو لایه های کانوولوشنی میباشند .

* **یکی از روش ها علاوه بر BatchNormalization استفاده از validatoin و تعدا فیلترهای متفاوت در لایه های کانولوشنی میباشد که در ادامه آن را مورد بررسی قرار میدهیم (بخش 3 و 4 پروژه):**

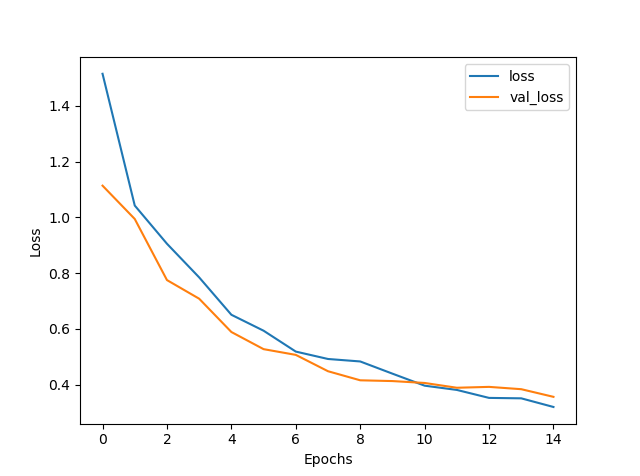
در قسمت ال پروژه نیز اگر به قسمت آموزش مدل توجه کنید ملاحظه خواهید نمود که از Validation\_split استفاده شده است ، استفاده از این دیتاها به این صورت است که بخشی از داده ی آموزش را به این بخش اختصاص میدهیم که باعث افزایش دقت در هر دو حالت میشود

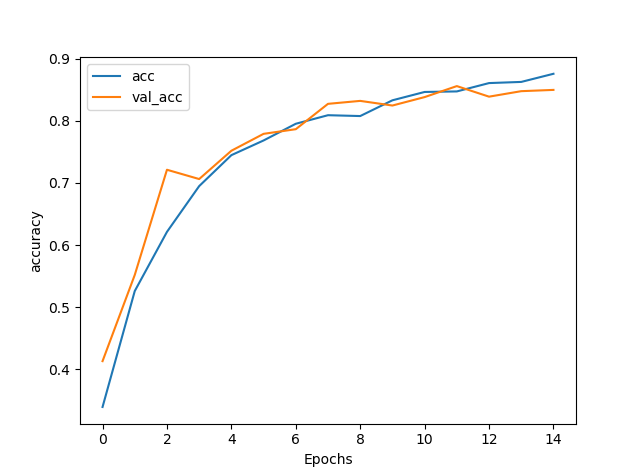
کار دیگری که در مدل دوم استفاده شده است استفاده از تعداد فیلترهای متفاوت در لایه ی اول و دوم کانولوشن میباشد که به این صورت در ابتدا با تعداد فیلترهای پایین و سپس با افزایش تعداد فیلترهها سعی در استخراج بیشتر اطلاعات می کنیم که بین این دولایه از لایه LSTM برای ذخیره ی اطلاعات لایه اول و انتقال آن به لایه دوم کانولوشن استفاده شده است . که اینکار نیز علاوه بر دو مورد قبلی باعث افزایش دقت داده های تست ما میگردد.

در انتها تصاویر مربوط به دو مدل طراحی شده را ملاحظه می نمایید

توجه : تمامی فایلها اعم از نمودار ها ، شمای کلی دو مدل به صورت PDF نیز در Courseware بارگذاری شده اند .

**تصاویر مربوط به LSTM :**





خروجی مربوط به LSTM ( با دقت حدود **86** درصد ) :

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

INPUT (LSTM) (None, 561, 32) 4352

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

HIDDEN 1 (LayerNormalizatio (None, 561, 32) 64

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

HIDDEN 2 (LSTM) (None, 561, 32) 8320

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

HIDDEN 3 (Dropout) (None, 561, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

HIDDEN 4 (LSTM) (None, 32) 8320

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

OUTPUT (Dense) (None, 7) 231

=================================================================

Total params: 21,287

Trainable params: 21,287

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

None

Train on 5881 samples, validate on 1471 samples

Epoch 1/15

5881/5881 [==============================] - 336s 57ms/step - loss: 1.5147 - acc: 0.3396 - val\_loss: 1.1136 - val\_acc: 0.4133

Epoch 2/15

5881/5881 [==============================] - 337s 57ms/step - loss: 1.0423 - acc: 0.5261 - val\_loss: 0.9942 - val\_acc: 0.5520

Epoch 3/15

5881/5881 [==============================] - 359s 61ms/step - loss: 0.9054 - acc: 0.6213 - val\_loss: 0.7752 - val\_acc: 0.7213

Epoch 4/15

5881/5881 [==============================] - 203s 35ms/step - loss: 0.7848 - acc: 0.6949 - val\_loss: 0.7084 - val\_acc: 0.7063

Epoch 5/15

5881/5881 [==============================] - 211s 36ms/step - loss: 0.6508 - acc: 0.7448 - val\_loss: 0.5890 - val\_acc: 0.7519

Epoch 6/15

5881/5881 [==============================] - 232s 39ms/step - loss: 0.5937 - acc: 0.7684 - val\_loss: 0.5276 - val\_acc: 0.7791

Epoch 7/15

5881/5881 [==============================] - 237s 40ms/step - loss: 0.5189 - acc: 0.7953 - val\_loss: 0.5071 - val\_acc: 0.7865

Epoch 8/15

5881/5881 [==============================] - 243s 41ms/step - loss: 0.4923 - acc: 0.8090 - val\_loss: 0.4484 - val\_acc: 0.8273

Epoch 9/15

5881/5881 [==============================] - 256s 44ms/step - loss: 0.4835 - acc: 0.8077 - val\_loss: 0.4159 - val\_acc: 0.8321

Epoch 10/15

5881/5881 [==============================] - 266s 45ms/step - loss: 0.4398 - acc: 0.8330 - val\_loss: 0.4132 - val\_acc: 0.8246

Epoch 11/15

5881/5881 [==============================] - 281s 48ms/step - loss: 0.3967 - acc: 0.8465 - val\_loss: 0.4064 - val\_acc: 0.8382

Epoch 12/15

5881/5881 [==============================] - 304s 52ms/step - loss: 0.3814 - acc: 0.8473 - val\_loss: 0.3892 - val\_acc: 0.8559

Epoch 13/15

5881/5881 [==============================] - 304s 52ms/step - loss: 0.3528 - acc: 0.8607 - val\_loss: 0.3923 - val\_acc: 0.8389

Epoch 14/15

5881/5881 [==============================] - 317s 54ms/step - loss: 0.3513 - acc: 0.8626 - val\_loss: 0.3840 - val\_acc: 0.8477

Epoch 15/15

5881/5881 [==============================] - 330s 56ms/step - loss: 0.3203 - acc: 0.8757 - val\_loss: 0.3566 - val\_acc: 0.8498

Total training time : 1:10:19.848246

2947/2947 [==============================] - 36s 12ms/step

0.3830552220364769 >>>>>>> Losses on Test set

0.8554462165115727 >>>>>>> Accuracy on Test set