درس : NLP

استاد: آقای دکتر هشام فیلی

دانشجوی مهمان : رقیه یزدانی

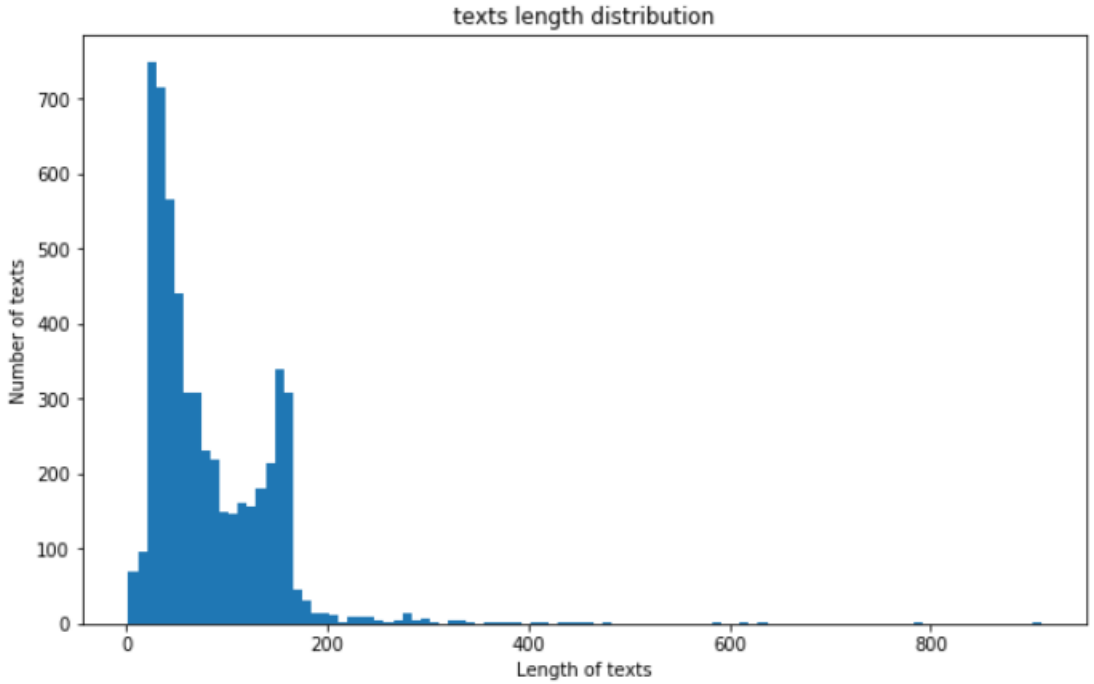
پروژه CA شماره 5

پروژه استفاده از مدلهای ELMo و Bert برای تشخیص spam و تحلیل احساس

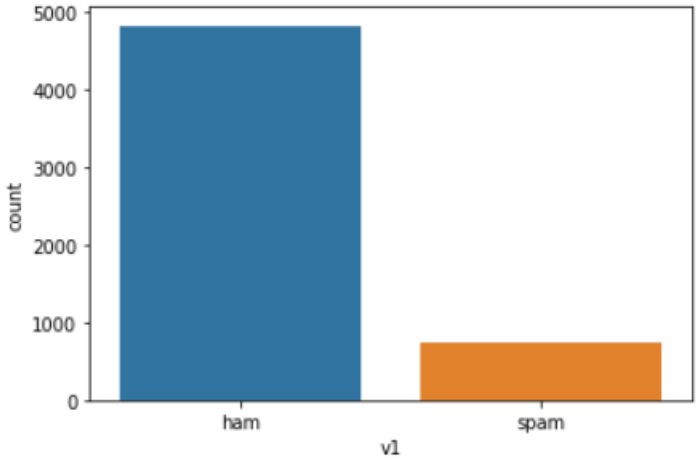
هدف این پروژه، بکاربردن روش contextual embedding و استفاده از وزنهای pre-trained مدلهای ELMo و Bert برای دو مساله spam detection و sentiment analysis می‌باشد.

سوال 1 – Spam Detection

برای این بخش، فایل spam.csv در اختیار قرار داده شده که حاوی 5572 سطر است و در ستون v1 برچسب (spam یا ham) و در ستون v2، متن موردنظر قرار دارد. در شکل زیر نمودار فراوانی تعداد متن ها بر حسب طول آنها آمده است :



کوتاهترین متن، دارای طول 2 و طولانی ترین آنها دارای طول 910 کاراکتر می‌باشد. از نظر تعداد نمونه های دارای برچسب spam و تعداد نمونه های دارای برچسب ham، نیز فایل بررسی شد که همانگونه که در شکل زیر مشخص است، تعداد داده‌های موجود برای کلاسهای spam و ham بالانس نیستند که این امر قطعا در کیفیت آموزش تاثیرگذار خواهد بود.



پیش پردازش

برای مرحله پیش پردازش، عملیات زیر صورت گرفت:

- تگهای html از متن پاک گردید.
- تمام کاراکترهای عددی و punctuation از متون پاک گردید.
- کاراکترهای تکی، حذف شدند.
- کاراکترهای blank پشت سر هم به یکی تبدیل شدند.
- برچسبهای spam و ham به 1 و 0 برای خروجی تبدیل شدند.

بعد از اینکه عملیات shuffle، روی کل داده‌ها، انجام شد، تعداد 20٪ داده،ها معادل 1114 عدد برای تست کنار گذاشته شد.

ابتدا مدلهای ELMo و Bert از مسیرهای زیر دانلود شد :

https://tfhub.dev/tensorflow/bert_en_cased_L-24_H-1024_A-16/1

https://tfhub.dev/google/elmo/3

و با توجه به اینکه استفاده از tensorflow hub برای مدل bert نیاز به حداقل نسخه 2 از tensorflow دارد، لذا روی لپ‌تاپ شخصی، نسخه قبلی tensorflow حذف شده و پس از نصب مجدد anaconda، نسخه tensorflow 2.1 نصب گردید و پس از آن برای مدل Bert امکان آموزش فراهم شد، اما متاسفانه برای مدل ELMo پس از کمی تلاش و تحقیق، مشخص شد که این مدل روی tensorflow 1.x قابلیت استفاده دارد و هنوز برای tensowflow 2 ، update های لازم release نشده است، لذا برای بخش ELMo از google colab که امکان تنظیم نسخه tensorflow را داشت، استفاده شد.

استفاده از مدل ELMo برای Spam Detection

ابتدا مدل بصورت زیر load گردید :

```
import tensorflow as tf
import tensorflow_hub as hub
elmo = hub.Module("https://tfhub.dev/google/elmo/3", trainable=True)
```

دیکشنری خروجی این ماژول شامل موارد زیر است :

lstm_outputs1 : که shape آن بصورت [batch_size, max_length, 1024] است.

lstm_outputs2 : که shape آن بصورت [batch_size, max_length, 1024] است.

elmo : که shape آن بصورت [batch_size, max_length, 1024] است.

default : که shape آن بصورت [batch_size, 1024] است.

مدل ELMo این قابلیت را دارد که متن را بصورت جملات یا بصورت tokenize شده به آن داد و سپس در خروجی default، هر ورودی که یا ترکیبی از جملات بوده و یا ترکیبی از توکنها، به یک بردار 1024 تایی embed می شود.

برای ایجاد مدل از قطعه کد زیر استفاده شده است. متن پس از پیش پردازش، به ماژول elmo داده شده و خروجی آن که از سایز batch_size * 1024 است به یک لایه Dense با سایز 1024 داده شده و برای این لایه از تابع relu به عنوان activation function استفاده شده و برای جلوگیری از overfitting تنظیمات regularization هم برای آن انجام شده و در لایه بعدی، یک لایه Dense با یک نرون و تابع sigmoid ایجاد شده که خروجی آن اگر زیر 0.5 باشد، برای کلاس خروجی، برچسب 0 و اگر بالای 0.5 باشد، برای کلاس خروجی، برچسب 1 در نظر گرفته خواهد شد(می‌شد دو نرون با softmax هم در نظر گرفت). نرخ یادگیری نیز همانگوکه که در تمرین خواسته شده، 0.0002 در نظر گرفته شده است.

```
def ELMoEmbedding(x):
  return elmo(tf.squeeze(tf.cast(x, tf.string)), signature="default",
        as_dict=True)["default"]

def build_model():
  input_text = Input(shape=(1,), dtype="string")
  embedding = Lambda(ELMoEmbedding, output_shape=(1024, ))(input_text)
  dense = Dense(1024, activation='relu',
kernel_regularizer=keras.regularizers.l2(0.001))(embedding)
  pred = Dense(1, activation='sigmoid')(dense)
  model = Model(inputs=[input_text], outputs=pred)
  model.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.0002),
 metrics=['accuracy'])
  return model
```

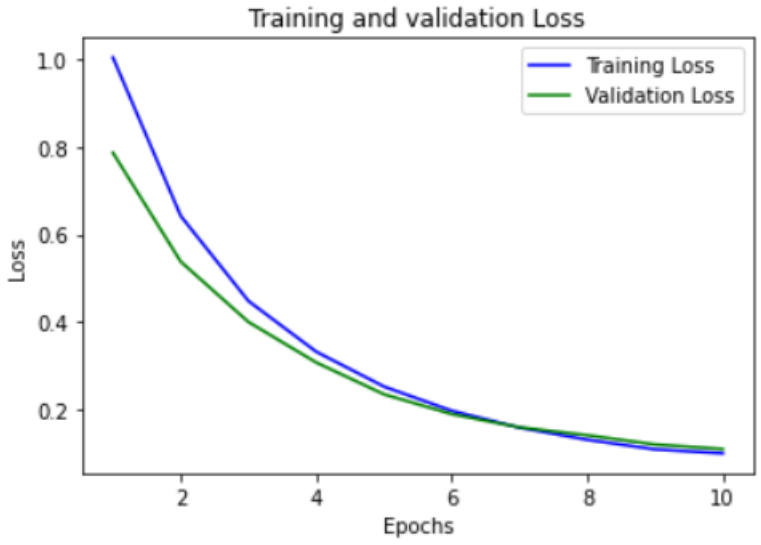
پس از کمپایل مدل، اطلاعات کلی آن بصورت زیر می‌باشد:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 1)]               0
_________________________________________________________________
lambda (Lambda)              (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              1049600
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 1025
=================================================================
Total params: 1,050,625
Trainable params: 1,050,625
Non-trainable params: 0
_________________________________________________________________
```

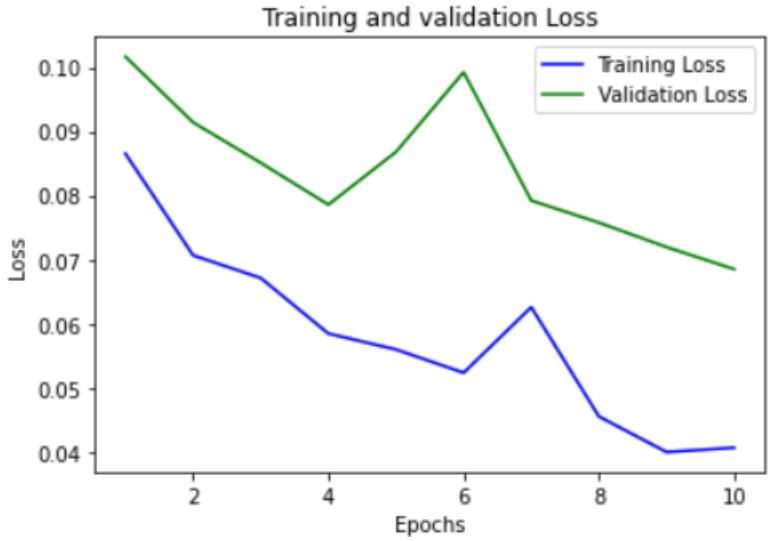
برای آموزش مدل، از قطعه کد زیر استفاده شد و batch_size=32 تعریف شده و برای 20٪ از داده های train به عنوان validation set در نظر گرفته شدند :

```
with tf.Session() as session:
  K.set_session(session)
  session.run(tf.global_variables_initializer())
  session.run(tf.tables_initializer())
  history = model_elmo.fit(np.array(train_X), np.array(train_Y),
  epochs=10, batch_size=32, validation_split = 0.2)
  model_elmo.save_weights('./model_spam_elmo.h5')
```

پس از 10 epoch نمودار loss در طول فرایند آموزش بصورت زیر بوده است :



پس از 10 epoch دیگر در ادامه آموزش بالا، نمودار loss بصورت زیر شد :



و پس از 30 epoch دیگر در ادامه آموزشهای بالا که در مجموع 50 epoch می‌شود، نمودار loss بصورت زیر شد:

مقادیر accuracy و precision، recall، AUC نیز در هریک از مراحل بالا بصورت زیر است :

Epoches	Precision	Recall	F1	Accuracy	AUC
After 10 epoches	0.982	0.953	0.967	0.985	0.953
After 20 epoches	0.983	0.975	0.979	0.990	0.975
After 50 epoches	0.983	0.956	0.969	0.986	0.956

همانطور که از نمودارها مشخص است، هر چه تعداد epoch ها افزایش یافته، روند کاهش loss از حالت یکنواختی خارج شده و نوسان دارد و بصورت متناوب، افزایش و کاهش می یابد. می توان اینطور استنباط کرد که با توجه به اینکه تعداد داده های آموزش

کم می‌باشد، وقتی از حدود 14 epoch فراتر رفته‌ایم، شبکه به سمت overfitting رفته و فاصله loss روی دیتای validation و train افزایش یافته است. در طول 20 epoch اول؛ روند loss در مجموع نزولی است اما در 30 epoch بعدی اینگونه نیست.

جدول پارامتر ها هم نشان می دهد که پس از 50 ایپاک، شبکه عملکرد ضعیف تری دارد و مقدار معیارها کم شده است. Precision ثابت مانده ولی recall کم شده است که البته با توجه به imbalance بودن داده‌ها، معیار recall می تواند برای ارزیابی بهتر از precision باشد. زیرا وقتی اکثر نمونه ها، spam نیستند، طبق تعریف precicion، تعداد نمونه هایی که spam هستند و به اشتباه ham تشخیص داده می‌شوند، در این معیار تاثیر کمی دارند اما برعکس در فرمول معیار recall این امر به خوبی خود را نشان می‌دهد.

استفاده از مدل Bert برای Spam Detection

با توجه به اینکه روی لپ‌تاپ، gpu و tensorflow 2.1 نصب شده بود، برای اجرای این بخش از colab استفاده نشد و خوشبختانه امکان اجرای آن روی لپ‌تاپ وجود داشت.

ابتدا پیش پردازش مشابه حالتی که برای مدل ELMo، صورت گرفت، انجام شد. سپس از bert tokenizer برای جدا کردن توکنهای هر متن استفاده شد :

```
bert_layer = hub.KerasLayer("F:\\Projects\\bertmodel", trainable=False)

BertTokenizer=bert.bert_tokenization.FullTokenizer
vocab_file=bert_layer.resolved_object.vocab_file.asset_path.numpy()
do_lower_case=bert_layer.resolved_object.do_lower_case.numpy()
tokenizer=BertTokenizer(vocab_file,do_lower_case)
```

با توجه به اینکه ورودی مدل Bert سه آرایه شامل آرایه توکنها، آرایه mask و آرایه segment می‌باشد و همچنین طول هر رشته ورودی باید ماکزیمم 128 توکن می بود؛ لذا به کمک متد زیر برای هر رشته از توکنها، سه آرایه مربوطه ایجاد گردید :

```
def get_ids_masks_segs(tokens, tokenizer, max_seq_length):
  segments = []
  current_segment_id = 0
  for token in tokens:
    segments.append(current_segment_id)
    if token == "[SEP]":
      current_segment_id = 1

  token_ids = tokenizer.convert_tokens_to_ids(tokens,)
  if len(token_ids) < max_seq_length:
    input_ids = token_ids + [0] * (max_seq_length - len(token_ids))
    masks = [1]*len(tokens) + [0] * (max_seq_length - len(tokens))
```

```
    segments = segments + [0] * (max_seq_length - len(tokens))
  else:
    input_ids = token_ids[:max_seq_length]
    masks = [1] * max_seq_length
    segments = segments[:max_seq_length]
  return input_ids, masks, segments
```

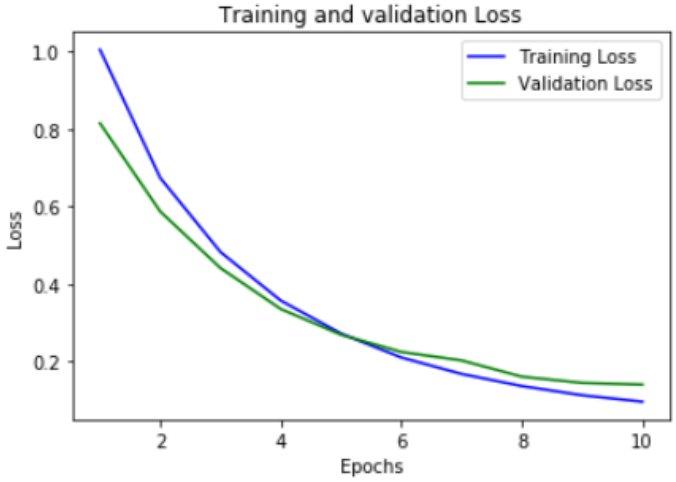
پس از اینکه ورودی شبکه با استفاده از مدل Bert آماده شد و با توجه به اینکه ابعاد آن [batch_size*3*1024] می‌باشد، ابتدا یک لایه AveragePooling روی آن اجرا شده، و سپس مشابه حالت قبل به لایه Dense با 1024 نود و سپس به لایه Dense با یک نود برای تعیین کلاس خروجی (pos یا neg) داده می‌شود :

```
hidden_units = 1024
learning_rate = 0.0002
x = tf.keras.layers.GlobalAveragePooling1D()(sequence_output)
x = tf.keras.layers.Dense(units=hidden_units, activation='relu', kernel_regularizer =
keras.regularizers.l2(0.001))(x)
out = tf.keras.layers.Dense(units=1, activation="sigmoid", name="dense_output")(x)

model = tf.keras.models.Model(
    inputs=[input_word_ids, input_mask, segment_ids], outputs=out)

model.compile(loss="binary_crossentropy",optimizer=Adam(lr=learning_rate),
metrics=["accuracy"])
```

با توجه به اینکه در صورتی که برای مدل Bert، پارامتر Trainable را true در نظر بگیریم، تعداد بیش از 333 میلیون پارامتر را شبکه باید fine-tune کند، لذا به دلیل عدم وجود سخت افزار مناسب برای امکان آموزش آنها و تولید خطای (OOM:Out Of Memory) از این بخش صرف نظر شده و Trainable=false در نظر گرفته شد. لذا نتیجه کمپایل مدل بصورت زیر شد :

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_word_ids (InputLayer)  [(None, 128)]             0
_________________________________________________________________________________________________
input_mask (InputLayer)      [(None, 128)]             0
_________________________________________________________________________________________________
segment_ids (InputLayer)     [(None, 128)]             0
_________________________________________________________________________________________________
keras_layer (KerasLayer)     [(None, 1024), (None 333579265   input_word_ids[0][0]
                                                      input_mask[0][0]
                                                      segment_ids[0][0]
_________________________________________________________________________________________________
global_average_pooling1d (Globa (None, 1024)           0           keras_layer[0][1]
_________________________________________________________________________________________________
dense (Dense)                (None, 1024)              1049600     global_average_pooling1d[0][0]
_________________________________________________________________________________________________
dense_output (Dense)         (None, 1)                 1025        dense[0][0]
=================================================================================================
Total params: 334,629,890
Trainable params: 1,050,625
Non-trainable params: 333,579,265
_________________________________________________________________________________________________
```

پس از 10 epoch نمودار loss در طول فرایند آموزش بصورت زیر بوده است :



پس از 10 epoch دیگر در ادامه آموزش بالا، نمودار loss بصورت زیر شد :

و پس از 30 epoch دیگر در ادامه آموزشهای بالا که در مجموع 50 epoch می‌شود، نمودار loss بصورت زیر شد:

مقادیر accuracy و precision، recall، AUC نیز در هریک از مراحل بالا بصورت زیر است :

Epoches	Precision	Recall	F1	Accuracy	AUC
After 10 epoches	0.980	0.915	0.944	0.977	0.915
After 20 epoches	0.962	0.940	0.951	0.979	0.940
After 50 epoches	0.966	0.957	0.961	0.983	0.957

همانطور که از نمودارها مشخص است، مشابه حالت ELMo، اینجا هم، هر چه تعداد epoch ها افزایش یافته، روند کاهش loss از حالت یکنواختی خارج شده و نوسان دارد و بصورت متناوب، افزایش و کاهش می‌یابد. در اینجا هم با اینکه در 20 epoch اول در مجموع روند loss نزولی است اما در 30 epoch آخر، شبکه به سمت overfitting رفته و فاصله loss روی دیتای validation و train افزایش یافته است. این مساله می تواند به خاطر imbalance بودن کلاسهای spam و ham هم باشد. اما جدول پارامتر ها نشان می‌دهد که در نهایت معیارهای ارزیابی، پس از 50 epoch وضعیت بهتری دارند، غیر از precision که کمتر شده، بقیه بیشتر شده اند.

برای حل مشکل imbalance بودن داده ها؛ علاوه بر اینکه معیار ارزیابی را از accuracy به سمت recall و f1 تغییر می دهیم میتوانیم از تکنیکهایی مثل under-sampling یا over-sampling یا تولید داده های مشابه استفاده کرد تا نمونه ها بالانس شوند و سپس اقدام به آموزش مدل کرد.

تاثیر پیش پردازش

برای بررسی تاثیر پیش پردازش، در یک آزمایش دیگر، متون داده شده بدون اجرای هر گونه پیش پردازشی، تحول betTokenizer گردید و سپس مدل مشابه بالا طراحی شده و تا 10 epoch آموزش داده شد. نمودار loss برای آن بصورت زیر است :

داده های تست با این مدل، ارزیابی شده و معیارهای زیر محاسبه شد که برای امکان مقایسه، معیارهای مشابه با انجام preprocessing هم در جدول آورده شده است :

Epoches	Precision	Recall	F1	Accuracy	AUC
10 epoches with preprocess	0.980	0.915	0.944	0.977	0.915
10 epoches without preprocess	0.978	0.981	0.979	0.990	0.981

در واقع در این مثال، preprocessing نه تنها تاثیر مثبتی نداشته بلکه تا اندازه‌ای، تاثیر منفی داشته است.

سوال 2 – Sentiment Analysis

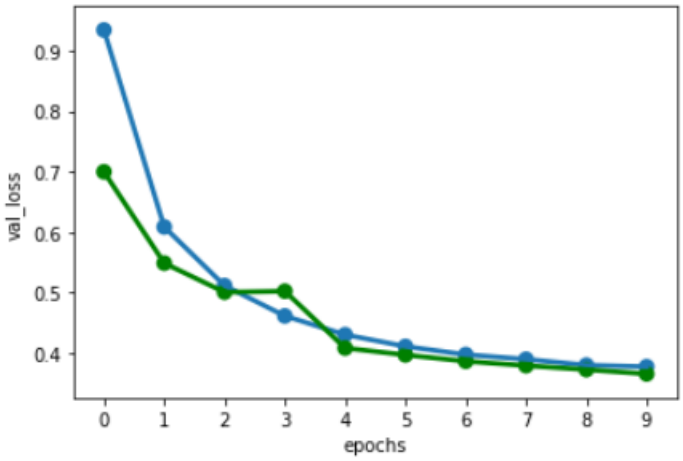
برای این بخش، فایلهای دیتاست Movie Review در اختیار قرار داده شده که حاوی 25000 متن برای آموزش و 25000 متن هم برای تست می‌باشد. در هر دو گروه داده های train و test، تعداد نمونه های هر کلاس برابر 12500 عدد می باشد. در نمونه های train تعداد کاراکترهای هر متن، بین 52 و 13604 می‌باشد و در نمونه های test تعداد کاراکترهای هر متن، بین 32 و 12730 می‌باشد.

استفاده از مدل Bert برای Sentiment Analysis

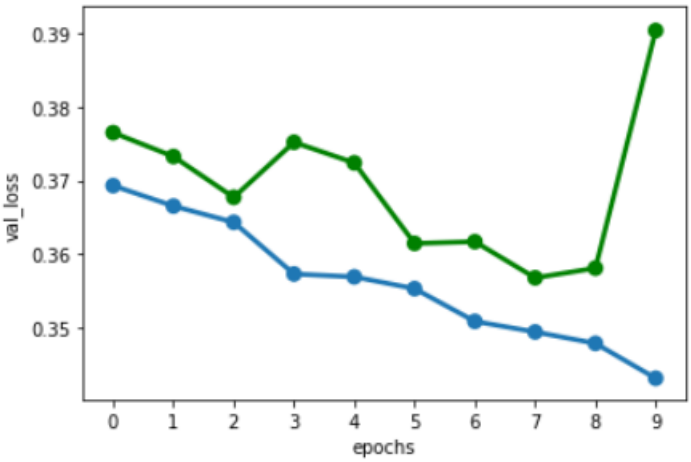
همانطور که در بالا اشاره شد، با توجه به اینکه روی لپ‌تاپ، gpu و tensorflow 2.1 نصب شده بود، برای اجرای این بخش از colab استفاده نشد و خوشبختانه امکان اجرای آن روی لپ‌تاپ وجود داشت.

ابتدا اطلاعات review ها از فایلهای مربوطه در مسیرهای aclImdb\train و aclImdb\test خوانده شده و سپس پیش پردازش مشابه حالتی که برای مدل ELMo در بالا، صورت گرفت، روی متون انجام شد و سپس از bert tokenizer برای جدا کردن توکنهای هر متن استفاده شد. و مدلی مشابه مدل استفاده شده برای تشخیص spam با استفاده از bert آموزش داده شد.

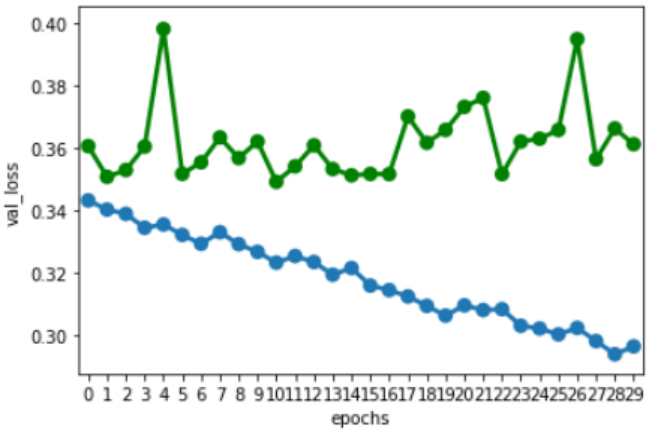
پس از 10 epoch نمودار loss در طول فرایند آموزش بصورت زیر بوده است : (نمودار سبز : val_loss و نمودار آبی: loss)



پس از 10 epoch دیگر در ادامه آموزش بالا، نمودار loss بصورت زیر بوده : (نمودار سبز : val_loss و نمودار آبی: loss)



و پس از 30 epoch دیگر در ادامه آموزشهای بالا که در مجموع 50 epoch می‌شود، نمودار loss بصورت زیر شد: (نمودار سبز : val_loss و نمودار آبی: loss)



مقادیر accuracy و precision، recall، AUC نیز در هریک از مراحل بالا بصورت زیر است :

Epoches	Precision	Recall	F1	Accuracy	AUC
After 10 epoches	0.855	0.855	0.855	0.855	0.855
After 20 epoches	0.844	0.836	0.835	0.836	0.836
After 50 epoches	0.853	0.853	0.853	0.853	0.853

همانطور که از نمودارها مشخص است، مشابه حالت Spam Detection، اینجا هم، هر چه تعداد epoch ها افزایش یافته، روند کاهش loss از حالت یکنواختی خارج شده و نوسان دارد و بصورت متناوب، افزایش و کاهش می‌یابد. خصوصاً در 30 epoch آخر، loss دائماً کمتر شده و مدل روی داده های train اصطلاحاً fit شده اما روی داده های validation، عمکرد loss اینگونه نیست و در مجموع کم نشده است.

جدول پارامترها هم نشان می‌دهد که پس از همان 10 ایپاک اول، شبکه به عملکرد مناسب رسیده و طی 40 epoch بعدی، تغییر محسوسی نداشته است. در اینجا با توجه به اینکه داده ها کاملا بالانس هستند، مشکل حالت spam detection را نداریم.

استفاده از مدل ELMo برای Sentiment Analysis

همانگونه که قبلا ذکر شد با توجه به اینکه برای استفاده از مدل Elmo نیاز به Tensorflow 1.x می‌باشد، برای اجرای این بخش از پروژه از google colab استفاده شد. ابتدا سعی شد مدلی مشابه آنچه که برای spam detection بکار رفت، برای داده های IMDB هم استفاده شود. اما خطای OOM (Out Of Memory) مانع کار شد.

```
ResourceExhaustedError: 2 root error(s) found.
  (0) Resource exhausted: OOM when allocating tensor with shape[32,128,1162,47] and type float on /job:localhost/replica:0/task:0/device:GPU:0 by allocator GPU_0_bfc
         [[{{node lambda/module_apply_default/bilm/CNN/Conv2D_3}}]]
Hint: If you want to see a list of allocated tensors when OOM happens, add report_tensor_allocations_upon_oom to RunOptions for current allocation info.

         [[metrics/acc/Identity/_219]]
Hint: If you want to see a list of allocated tensors when OOM happens, add report_tensor_allocations_upon_oom to RunOptions for current allocation info.

  (1) Resource exhausted: OOM when allocating tensor with shape[32,128,1162,47] and type float on /job:localhost/replica:0/task:0/device:GPU:0 by allocator GPU_0_bfc
         [[{{node lambda/module_apply_default/bilm/CNN/Conv2D_3}}]]
Hint: If you want to see a list of allocated tensors when OOM happens, add report_tensor_allocations_upon_oom to RunOptions for current allocation info.

0 successful operations.
0 derived errors ignored.
```

با حالتهای مختلف تست شد اما هر بار پس از مدتی، حین آموزش، دوباره این خطا تولید می‌شد. از جمله، پارامتر Trainable=false، شد، تعداد نرونهای لایه hidden به 768 و 512 تغییر داده شد، batch_size به 16 تغییر داده شد.

در نهایت با 256 نرون برای لایه Dense و batch_size=16 امکان اجرا فراهم شد که تا زمان تهیه این گزارش، سعی شد برای 10 epoch خروجی گرفته شود که تا زمان تهیه این گزارش، 3 epoch آن اجرا شده است که تصویر آن در ادامه می آید :

```
with tf.Session() as session:
    K.set_session(session)
    session.run(tf.global_variables_initializer())
    session.run(tf.tables_initializer())
    history = model_elmo.fit(np.array(train_X), np.array(train_Y), epochs=10, batch_size=16, validation_split = 0.2)
    model_elmo.save_weights('./model_imdb_elmo.h5')
```

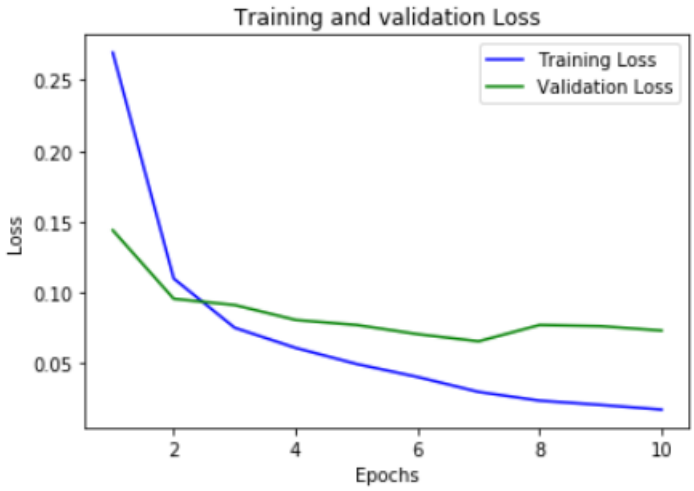
```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 3285s 164ms/sample - loss: 0.6379 - acc: 0.8069 - val_loss: 0.5072 - val_acc: 0.8420
Epoch 2/10
20000/20000 [==============================] - 3124s 156ms/sample - loss: 0.4788 - acc: 0.8399 - val_loss: 0.4629 - val_acc: 0.8366
Epoch 3/10
20000/20000 [==============================] - 3099s 155ms/sample - loss: 0.4333 - acc: 0.8490 - val_loss: 0.4279 - val_acc: 0.8436
Epoch 4/10
 1312/20000 [>.............................] - ETA: 40:58 - loss: 0.4125 - acc: 0.8605
```

سوال 3

در بخش سوالات 2 , 1، پس از لایه embedding یک لایه Dense با سایز 1024 استفاده شد که بصورت fully connected به لایه embedding متصل بود. در این بخش سعی شده تا قبل از لایه Dense از چند لایه Convolution استفاده شود و سپس از یک لایه AveragePooling و بعد هم لایه Dense. متد ایجاد مدل بصورت زیر می باشد :

```
hidden_units = 1024
learning_rate = 0.0002
x = layers.Conv1D(filters=50,kernel_size=2,padding="valid",activation="relu")(sequence_output)
x = layers.Conv1D(filters=50,kernel_size=3,padding="valid",activation="relu")(x)
x = layers.Conv1D(filters=50,kernel_size=4,padding="valid",activation="relu")(x)
x = tf.keras.layers.GlobalAveragePooling1D()(x)
x=tf.keras.layers.Dense(units=hidden_units,activation='relu',
kernel_regularizer=keras.regularizers.l2(0.001))(x)
out = tf.keras.layers.Dense(units=1, activation="sigmoid", name="dense_output")(x)

model = tf.keras.models.Model(
    inputs=[input_word_ids, input_mask, segment_ids], outputs=out)

model.compile(loss="binary_crossentropy",optimizer=Adam(lr=learning_rate),
metrics=["accuracy"])
```

نتیجه آموزش مدل برای داده های spam در 10 epoch به صورت زیر است : اگر با نمودار مشابه برای 10 epoch که مربوط به شبکه feed-forward بود مقایسه شود، مشاهده می شود که در این نمودار، loss حین آموزش از 0.2 شروع شده و طی 10 epoch به 0.01 رسیده است ولی در شبکه FF با شرایط مشابه، loss از 1 شروع شده و به 0.09 می‌رسید. نتایج ارزیابی داده‌های تست نیز با در مقایسه با حالت FF در جدول آورده شده است که بهبود را نشان می دهد.



Epoches	Precision	Recall	F1	Accuracy	AUC
10 epoches - FF	0.980	0.915	0.944	0.977	0.915
10 epoches – Conv+FF	0.977	0.958	0.967	0.985	0.958

References :

https://stackabuse.com/text-classification-with-bert-tokenizer-and-tf-2-0-in-python/

https://tfhub.dev/tensorflow/bert_en_cased_L-24_H-1024_A-16/1

https://towardsdatascience.com/transfer-learning-using-elmo-embedding-c4a7e415103c

https://towardsdatascience.com/elmo-embeddings-in-keras-with-tensorflow-hub-7eb6f0145440