به نام آنکه آموخت انسان را آنچه نمی دانست

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

درس پردازش زبان‌های طبیعی

CA5

Bert & Elmo

behzad.shayegh@ut.ac.ur

بهزاد شایق بروجنی

81 01 96 678

اردیبهشت ۱۳۹۹

فهرست

مقدمه

با سلام. در این تمرین قصد داریم دو مدل از پیش آموزش داده شده‌ی Elmo و Bert را بر روی دو مسئله تحلیل احساسات تشخیص اسپم و بررسی نظرات کاربران IMDB پیاده‌سازی کرده و Fine Tune کنیم و عملکرد این دو مدل را بررسی کنیم.

آماده‌سازی لایه‌های Embedding

قبل از اینکه به بررسی مسائل بپردازیم، ابتدا لایه‌های از پیش آموزش داده شده را به صورت لایه‌های شبکه‌ی keras آماده‌ی استفاده می‌کنیم :

لایه‌ی Elmo : برای این مدل از https://tfhub.dev/google/elmo/3 استفاده کردیم که خروجی آن بردارهای به طول 1024 بود.

لایه‌ی Bert : برای این مدل، بعد از آزمودن https://tfhub.dev/tensorflow/bert_en_cased_L-24_H-1024_A-16/1 و `transformers.BertForSequenceClassification` که موفقیت آمیز نبودند، درنهایت از مدل https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1 در قسمت اول و `uncased_L-12_H-768_A-12` برای قسمت دوم استفاده شد.

ایراداتی که در استفاده از مدل اول پیش آمد عبارت بودند از : 1- درصورتی که در شبکه و آماده‌ی تکمیل آموزش می‌شد زمان اجرا را بسیار کند می‌کرد و خیلی زود دقت آن ثابت می‌شد که دلیلی برای آن پیدا نکردیم. 2- درصورتی که خارج از شبکه استفاده می‌شد، بازدهی بسیار مقبولی داشت و همچنین زمان اجرای پایینی پیدا می‌کرد اما مخالف هدف این آزمون مبنی بر مشاهده‌ی Fine Tunning می‌بود. پس از آن صرف‌نظر کردیم. مدل دوم نیز مدل کاملا آماده با بازدهی اولیه‌ی مناسب بود اما قابلیت ایجاد تغییر در شبکه را نداشت و به عنوان لایه‌ی شبکه نیز قرار نمی‌گرفت پس از این مورد نیز صرف‌نظر شد.

مدل انتخاب شده نیز در ابعاد متفاوت وجود داشت که اندازه‌ی کوچک آن یعنی با بردار خروجی به طول 768 انتخاب شد چرا که ابزار آموزش مدل بزرگ‌تر موجود نبود.

همچنین برای مدل‌های انتخاب شده آزمون‌های فراوانی صورت گرفته و با شکست روبرو شدند که در گزارش به آنها اشاره‌ای نخواهد شد اما کد این آزمون‌ها ضمیمه شده است.

بخش اول) تشخیص اسپم

دادگان

دادگان این بخش، همانطور که از نام آن مشخص است شامل 5572 ایمیل بوده که حدود ۱۵ درصد آن‌ها برچسب اسپم و مابقی برچسب غیر اسپم دارند. ۲۵ درصد از این دادگان را بصورت تصادفی برای ارزیابی مدل‌ها جدا می‌کنیم.

پیش‌پردازش

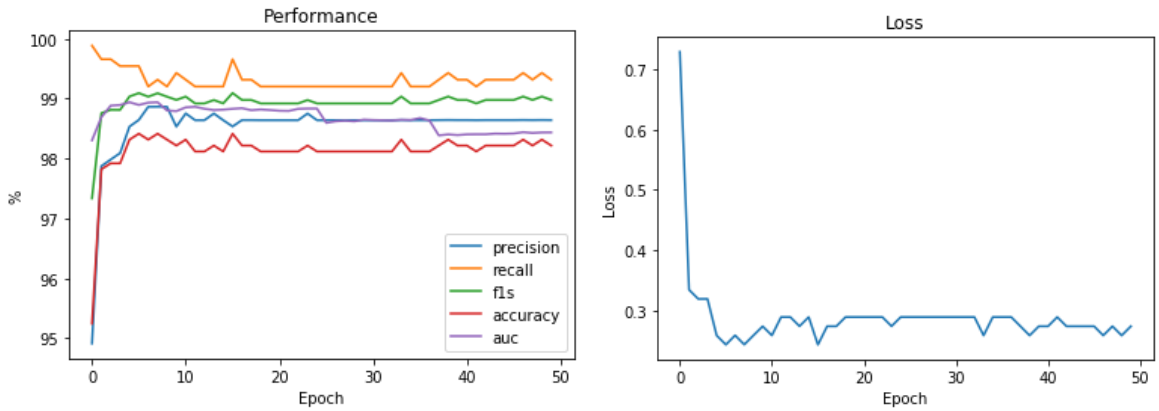
پیش پردازش دادگان شامل حذف علامت‌های نگارشی و غیر نگارشی و حذف stop words می‌باشد که در هر مدل اثر اعمال و عدم اعمال این پردازش بر روی بازدهی مقایسه شده است.

مدل Elmo

شبکه‌ی طراحی شده برای این قسمت عبارت است از :

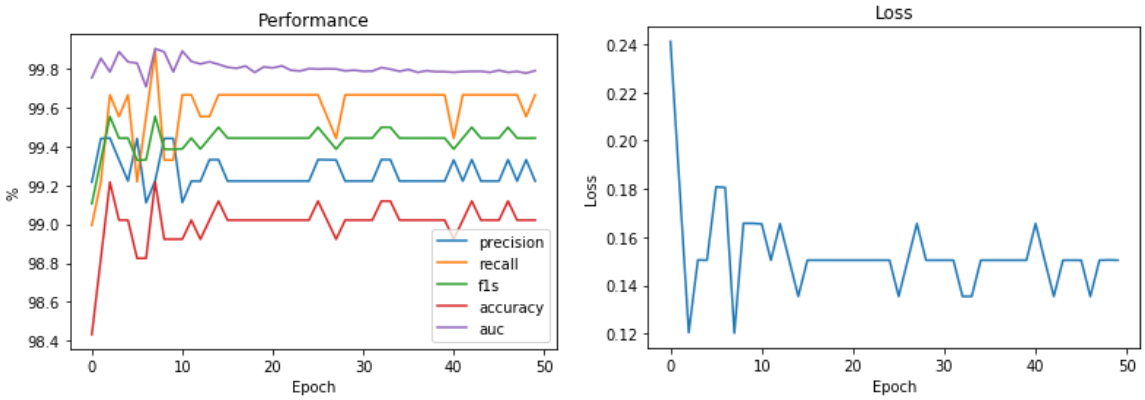
```
"Model: "model_1
_________________________________________________________________
# Layer (type)                 Output Shape              Param
=================================================================
input_1 (InputLayer)          (None, 1)                 0
_________________________________________________________________
elmo_embedding_layer_1 (Elmo  (None, 1024)              4
_________________________________________________________________
dense_1 (Dense)               (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)               (None, 1)                 1025
=================================================================
Total params: 1,050,629
Trainable params: 1,050,629
Non-trainable params: 0
_________________________________________________________________
```

نمودار Loss و دقت این مدل با اعمال پیش‌پردازش به شکل زیر بود :

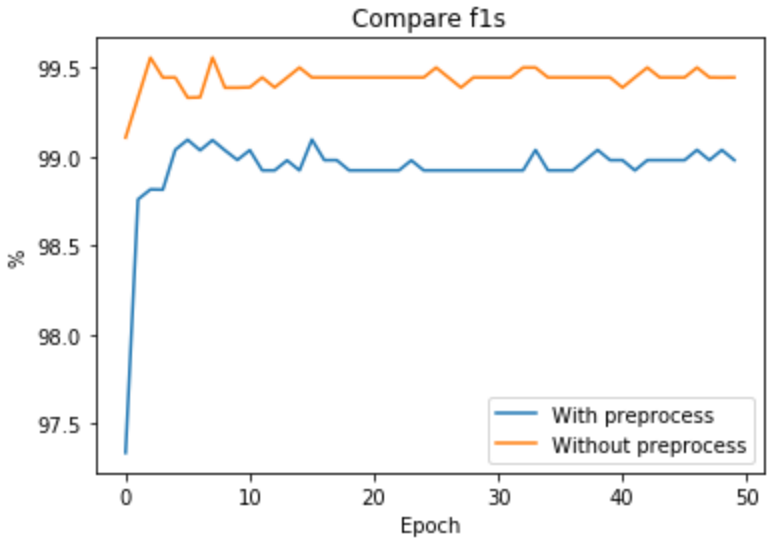


همانطور که مشخص است، از نقطه‌ای به بعد، تکرار هیچ کمکی به مدل نمی‌کند چرا که مدل‌ها از پیش آموزش داده شده‌اند و فقط نیاز به Fine Tune دارند که محاسبات زیادی نیاز ندارد.

با عدم اعمال پیش‌پردازش بر روی این مدل به نمودارهای متناظر زیر رسیدیم :



و در نمودار زیر مقایسه‌ی معیار f1 را برای نحوه عملکرد این مدل با پیش‌پردازش و بدون آن می‌بینیم :



همانطور که مشاهده می‌شود، اعمال پیش‌پردازش باعث کاهش دقت مدل می‌شود.این مشاهده را می‌توان اینگونه توجیه کرد که با توجه به این‌که از لایه‌ی از پیش‌آموزش دیده استفاده می‌کنیم، ساده سازی متن باعث کاهش overfit نمی‌شود و فقط اطلاعات مفیدی که در پیچیدگی‌های متون است را از بین می‌برد. همچنین مدل از پیش‌آموزش داده شده پیش‌پردازش خود را داشته و دخالت ما باعث ایجاد عدم تطابق می‌شود.

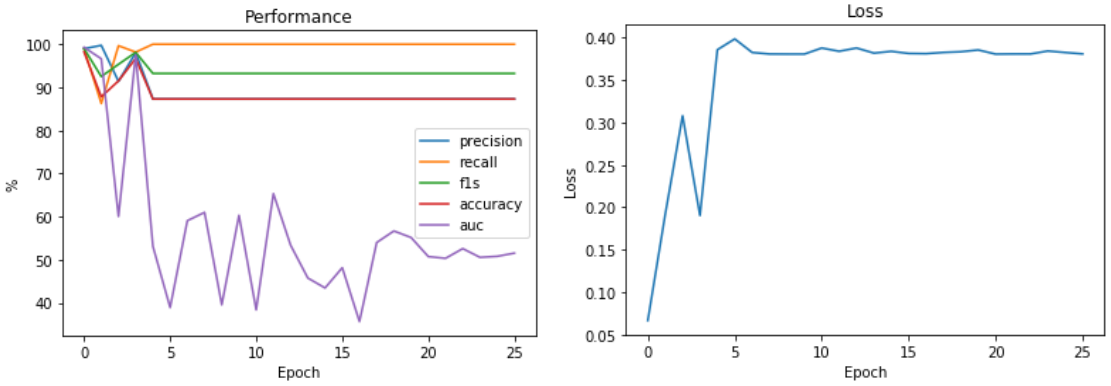
مدل Bert

شبکه‌ی طراحی شده برای این قسمت به شکل زیر است :

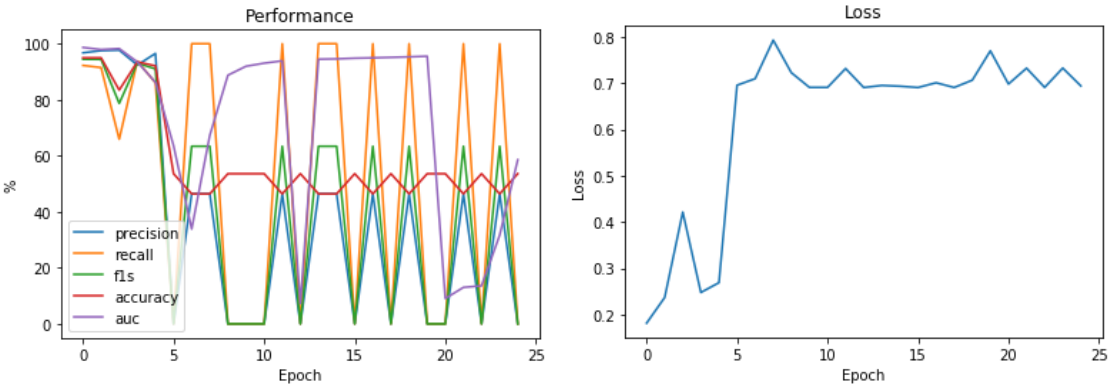
```
"Model: "model
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_ids (InputLayer)       [(None, 128)]             0
_________________________________________________________________________________________________
input_mask (InputLayer)      [(None, 128)]             0
_________________________________________________________________________________________________
segment_ids (InputLayer)     [(None, 128)]             0
_________________________________________________________________________________________________
[bert_layer (BertLayer)       (None, 768)               110104890   input_ids[0][0
[input_mask[0][0
[segment_ids[0][0
_________________________________________________________________________________________________
```

```
[dense (Dense)                  (None, 768)          590592      bert_layer[0][0
__________________________________________________________________________________________________
[dense_1 (Dense)                (None, 1)            769         dense[0][0
==================================================================================================
Total params: 110,696,251
Trainable params: 72,060,673
Non-trainable params: 38,635,578
__________________________________________________________________________________________________
```

بازدهی این مدل بر روی دادگان به شکل زیر بود (به دلیل ثابت شدن، از ادامه دادن تکرار صرف‌نظر کردیم) :

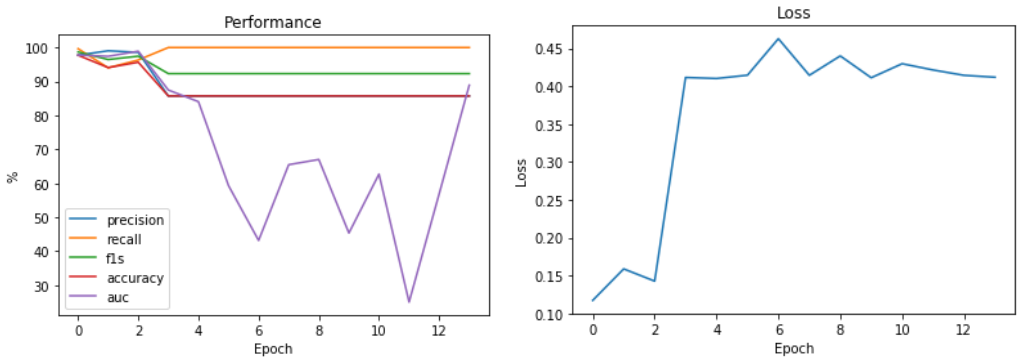


همانطور که مشاهده می‌شود در تکرار اول، مدل بسیار موفق است اما به مرور نزول کرده و بعد از تکرار چهارم به ثبات جواب یکنواخت می‌رسد. دلیل این رفتار را نتوانستیم توجیه کنیم. یکنواخت شدن پاسخ را معلول دادگان غیر متقارن حدس زدیم، پس با یک نمونه‌برداری متقارن، یک زیرمجموعه از دادگان را مهیا کرده و مدل را بر روی آن نیز اجرا کردیم :

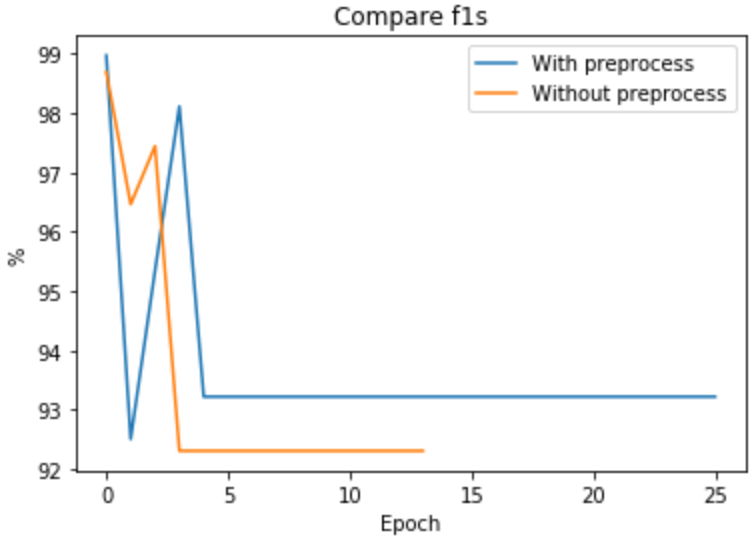


که نتیجه بسیار بدی به دست آمد. در این مدل نیز دوباره مشاهده می‌کنیم که در تکرار اولیه مدل به خوبی عمل می‌کند اما به مرور بد می‌شود. از دو مشاهده‌ی بالا می‌توان نتیجه گرفت که مدل Bert در تکرارهای اولیه قوی عمل کرده و بر روی دادگان جدید fit می‌شود اما تکرار آن بردارهای از پیش آموزش داده شده را خراب می‌کند.

با حذف مرحله‌ی پیش‌پردازش به نتایج زیر رسیدیم :



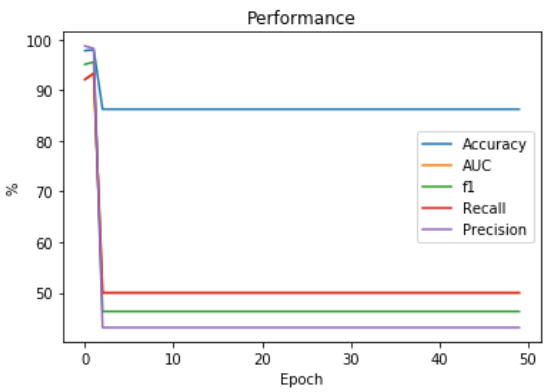
که با برای مقایسه‌ی معیار f1 آن با نسخه‌ی با پیش‌پردازش داریم :

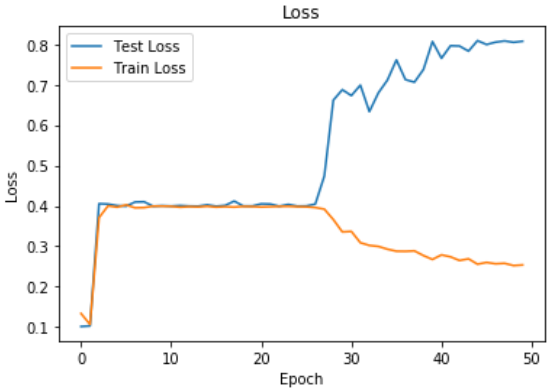


که مشاهده می‌کنیم بصورت کلی پیش‌پردازش به این مدل برخلاف Elmo کمک می‌کند و دلیل آن نیز می‌تواند این باشد که در این مدل، بردار کلمات تغییر می‌کند.

مدل Bert آماده

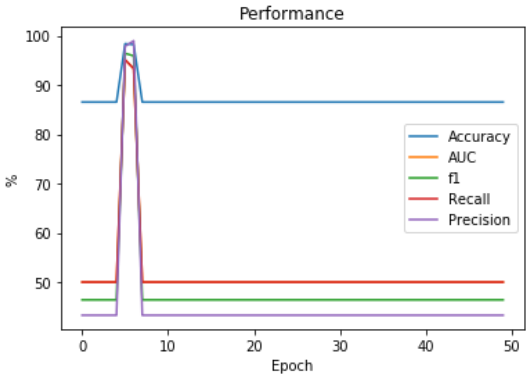
با توجه به نتایج نامطلوب حاصل از مدل ert پیادسازی شده در مرحله‌ی قبل، تصمیم گرفتیم نتایج حاصل از مدل `transformers.BertForSequenceClassification` را نیز نشان دهیم. با توجه به حجیم بودن این شبکه، گزارش آن را نمی‌آوریم، این شبکه دارای ۱۱ لایه Bert و یک شبکه FeedForward در انتها بود که می‌توانید گزارش آن را در کدهای پروژه مشاهده کنید. عملکرد این مدل بصورت زیر بود :

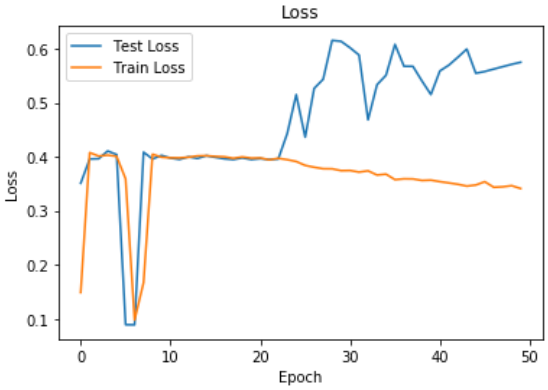




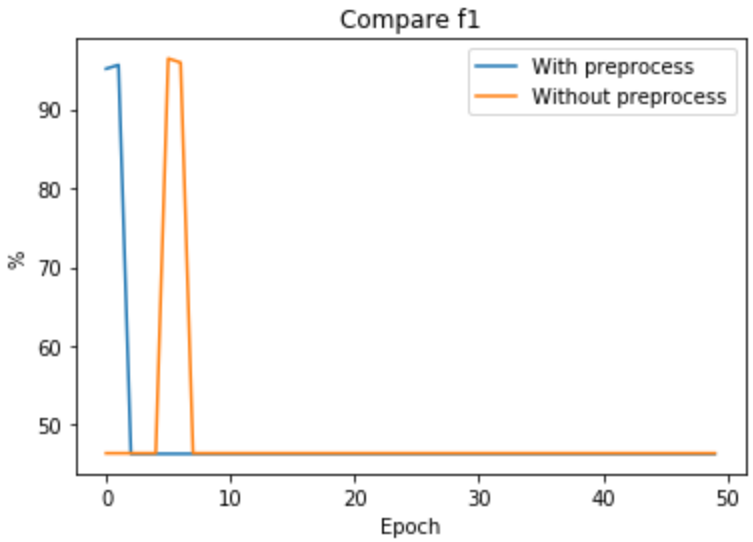
و همانطور که مشاهده می‌کنید، این نمودارها علاوه بر اینکه موضوع overfit را به وضوح نشان می‌دهند، تایید کننده‌ی نتیجه‌ای که در بخش قبل مبنی به قوت مدل Bert در تکرارهای اول نیز هستند. همچنین می‌توان پایداری بیشتری در این نمودارها دید که شاید حاصل تعدد لایه‌های شبکه باشد.

همچنین نتایج این مدل بدون پیش‌پردازش به شکل زیر بود :





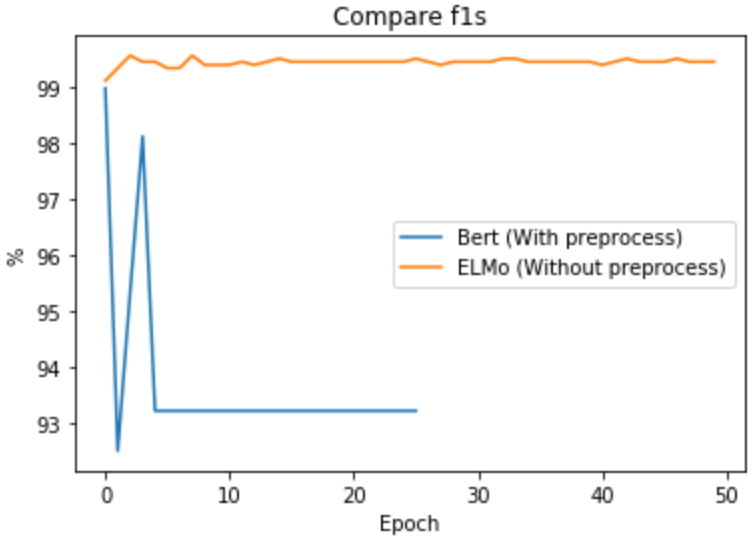
که از روی نمودار Loss مشخص است که عدم وجود پیش‌پردازش تا حدی مانع overfit می‌شود. نمودار مقایسه‌ی f1 این مدل با و بدون پیش‌پردازش در ادامه آمده است :



که نشان می‌دهد این‌مدل درصورت نداشتن پیش‌پردازش کمی زمان برای fit شدن نیاز دارد.

جمع‌بندی

نمودار زیر را در رابطه با مقایسه‌ی معیار f1 دو مدل مذکور داریم :



که مشخصا تاثیر متفاوت تکرار را بر دو مدل نشان می‌دهد.

از بین معیارهای بازدهی، معیار دقت precision برای تشخیص ایمیل اسپم مهمتر است چرا که ما دوست نداریم ایمیل غیر اسپم را از دست بدهیم و ترجیح بر آن است که ریسک در تشخیص را پایین بیاوریم.

بخش دوم) IMDB

دادگان.

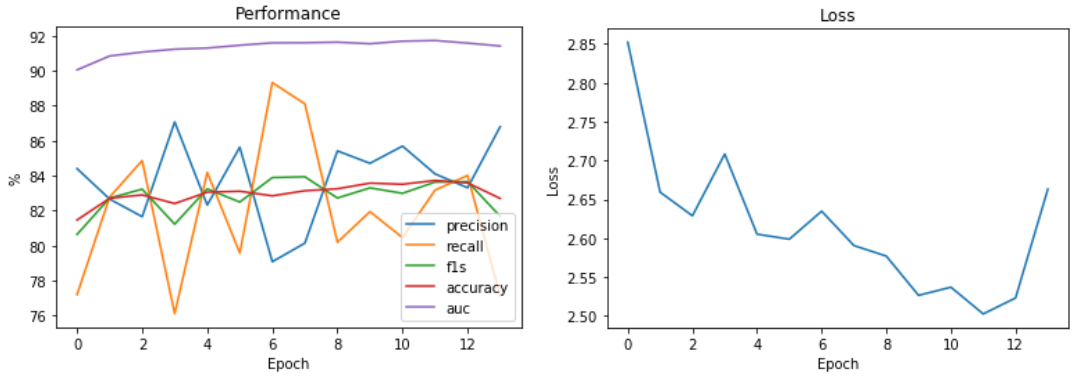
دادگان این بخش 50000 نظر متقارن مثبت و منفی نسبت به فیلم‌هاست که به دو دسته آموزش و ارزیابی مساوی تقسیم شده است.

پیش‌پردازش

تمام مواردی که برای بخش اول بیان شد برای این بخش نیز صادق است.

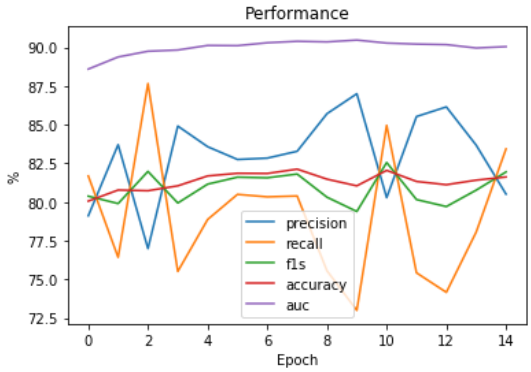
مدل Elmo

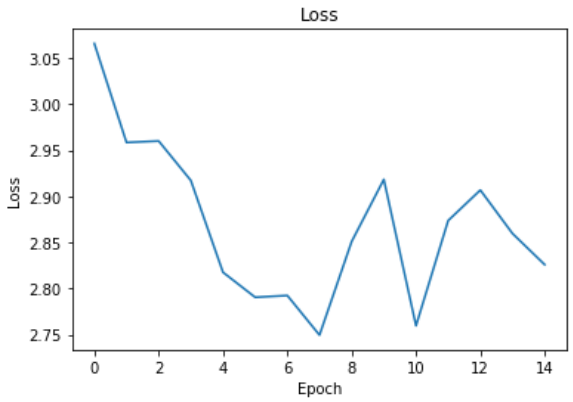
با همان شبکه‌ی تشریح شده در بخش اول، به این بخش پرداختیم. نتایج حاصل عبارت بودند از :



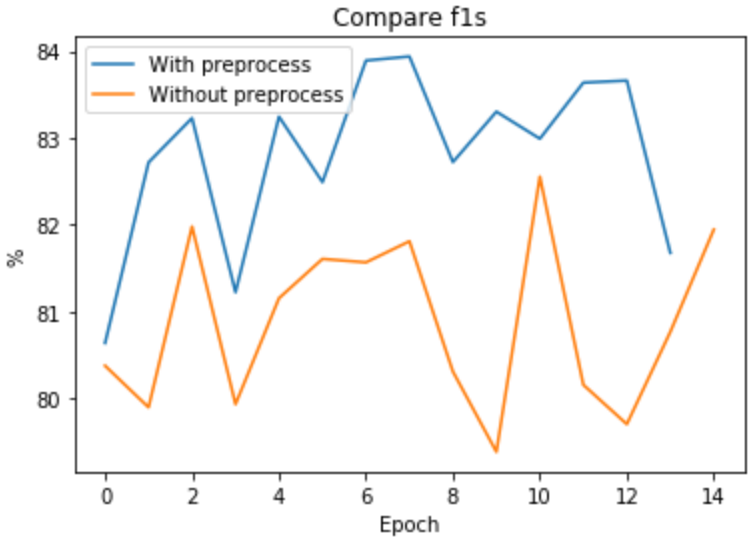
همانطور که مشاهده می‌شود، باز هم مانند بخش اول، تکرار زیاد تاثیر چندانی روی بازدهی ندارد اما با توجه به دادگان حجیم، مطالب بیشتری برای آموختن وجود دارد پس مدل دیرتر به ثبات می‌رسد (آموزش زیاد ادامه داده نشده چون آموزش شبکه بسیار کند بود و ابزار مورد نیاز موجود نبود).

عملکرد مدل بدون پیش‌پردازش بصورت زیر بود :





که نمودار مقایسه‌ی معیار f1 عبارت است از :



که برخلاف بخش قبل، در این مسئله نتیجه‌ی پیش‌پردازش مثبت است و دلیل آن را می‌توان به فرمت html دادگان مربوط دانست که فرمت معناداری نیست.

مدل Bert

بعد از مشاهده‌ی شکست صددرصدی مدل قسمت قبل در این قسمت، از یک مدل بزرگ‌تر با لایه‌های متعدد استفاده شد. توصیف شبکه‌ی مورد استفاده را در ادامه مشاهده می‌کنید :

```
Model: "model_1"
```

Layer (type)	Output Shape	Param #	Connected to
Input-Token (InputLayer)	[(None, 128)]	0	
Input-Segment (InputLayer)	[(None, 128)]	0	
Embedding-Token (TokenEmbedding	[(None, 128, 768), (	23440896	Input-Token[0][0]
Embedding-Segment (Embedding)	(None, 128, 768)	1536	Input-Segment[0][0]
Embedding-Token-Segment (Add)	(None, 128, 768)	0	Embedding-Token[0][0]

			Embedding-Segment[0][0]
Embedding-Position (PositionEmb	(None, 128, 768)	98304	Embedding-Token-Segment[0][0]
Embedding-Dropout (Dropout)	(None, 128, 768)	0	Embedding-Position[0][0]
Embedding-Norm (LayerNormalizat	(None, 128, 768)	1536	Embedding-Dropout[0][0]
Encoder-1-MultiHeadSelfAttentio	(None, 128, 768)	2362368	Embedding-Norm[0][0]
Encoder-1-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-1-MultiHeadSelfAttention[
Encoder-1-MultiHeadSelfAttentio	(None, 128, 768)	0	Embedding-Norm[0][0] Encoder-1-MultiHeadSelfAttention-
Encoder-1-MultiHeadSelfAttentio	(None, 128, 768)	1536	Encoder-1-MultiHeadSelfAttention-
Encoder-1-FeedForward (FeedForw	(None, 128, 768)	4722432	Encoder-1-MultiHeadSelfAttention-
Encoder-1-FeedForward-Dropout (	(None, 128, 768)	0	Encoder-1-FeedForward[0][0]
Encoder-1-FeedForward-Add (Add)	(None, 128, 768)	0	Encoder-1-MultiHeadSelfAttention- Encoder-1-FeedForward-Dropout[0][
Encoder-1-FeedForward-Norm (Lay	(None, 128, 768)	1536	Encoder-1-FeedForward-Add[0][0]
Encoder-2-MultiHeadSelfAttentio	(None, 128, 768)	2362368	Encoder-1-FeedForward-Norm[0][0]
Encoder-2-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-2-MultiHeadSelfAttention[
Encoder-2-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-1-FeedForward-Norm[0][0] Encoder-2-MultiHeadSelfAttention-
Encoder-2-MultiHeadSelfAttentio	(None, 128, 768)	1536	Encoder-2-MultiHeadSelfAttention-
Encoder-2-FeedForward (FeedForw	(None, 128, 768)	4722432	Encoder-2-MultiHeadSelfAttention-
Encoder-2-FeedForward-Dropout (	(None, 128, 768)	0	Encoder-2-FeedForward[0][0]
Encoder-2-FeedForward-Add (Add)	(None, 128, 768)	0	Encoder-2-MultiHeadSelfAttention- Encoder-2-FeedForward-Dropout[0][
Encoder-2-FeedForward-Norm (Lay	(None, 128, 768)	1536	Encoder-2-FeedForward-Add[0][0]
Encoder-3-MultiHeadSelfAttentio	(None, 128, 768)	2362368	Encoder-2-FeedForward-Norm[0][0]
Encoder-3-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-3-MultiHeadSelfAttention[
Encoder-3-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-2-FeedForward-Norm[0][0] Encoder-3-MultiHeadSelfAttention-
Encoder-3-MultiHeadSelfAttentio	(None, 128, 768)	1536	Encoder-3-MultiHeadSelfAttention-
Encoder-3-FeedForward (FeedForw	(None, 128, 768)	4722432	Encoder-3-MultiHeadSelfAttention-
Encoder-3-FeedForward-Dropout (	(None, 128, 768)	0	Encoder-3-FeedForward[0][0]
Encoder-3-FeedForward-Add (Add)	(None, 128, 768)	0	Encoder-3-MultiHeadSelfAttention- Encoder-3-FeedForward-Dropout[0][
Encoder-3-FeedForward-Norm (Lay	(None, 128, 768)	1536	Encoder-3-FeedForward-Add[0][0]
Encoder-4-MultiHeadSelfAttentio	(None, 128, 768)	2362368	Encoder-3-FeedForward-Norm[0][0]
Encoder-4-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-4-MultiHeadSelfAttention[

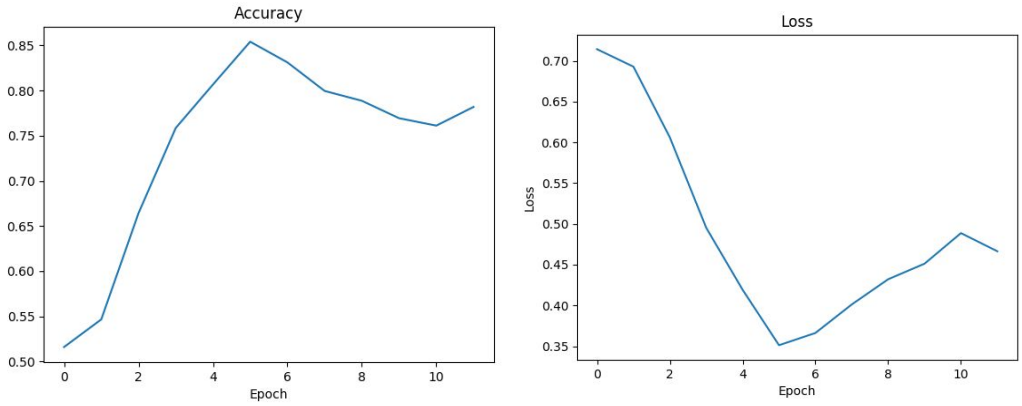
Encoder-4-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-3-FeedForward-Norm[0][0] Encoder-4-MultiHeadSelfAttention-
Encoder-4-MultiHeadSelfAttentio	(None, 128, 768)	1536	Encoder-4-MultiHeadSelfAttention-
Encoder-4-FeedForward (FeedForw	(None, 128, 768)	4722432	Encoder-4-MultiHeadSelfAttention-
Encoder-4-FeedForward-Dropout (	(None, 128, 768)	0	Encoder-4-FeedForward[0][0]
Encoder-4-FeedForward-Add (Add)	(None, 128, 768)	0	Encoder-4-MultiHeadSelfAttention- Encoder-4-FeedForward-Dropout[0][
Encoder-4-FeedForward-Norm (Lay	(None, 128, 768)	1536	Encoder-4-FeedForward-Add[0][0]
Encoder-5-MultiHeadSelfAttentio	(None, 128, 768)	2362368	Encoder-4-FeedForward-Norm[0][0]
Encoder-5-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-5-MultiHeadSelfAttention[
Encoder-5-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-4-FeedForward-Norm[0][0] Encoder-5-MultiHeadSelfAttention-
Encoder-5-MultiHeadSelfAttentio	(None, 128, 768)	1536	Encoder-5-MultiHeadSelfAttention-
Encoder-5-FeedForward (FeedForw	(None, 128, 768)	4722432	Encoder-5-MultiHeadSelfAttention-
Encoder-5-FeedForward-Dropout (	(None, 128, 768)	0	Encoder-5-FeedForward[0][0]
Encoder-5-FeedForward-Add (Add)	(None, 128, 768)	0	Encoder-5-MultiHeadSelfAttention- Encoder-5-FeedForward-Dropout[0][
Encoder-5-FeedForward-Norm (Lay	(None, 128, 768)	1536	Encoder-5-FeedForward-Add[0][0]
Encoder-6-MultiHeadSelfAttentio	(None, 128, 768)	2362368	Encoder-5-FeedForward-Norm[0][0]
Encoder-6-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-6-MultiHeadSelfAttention[
Encoder-6-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-5-FeedForward-Norm[0][0] Encoder-6-MultiHeadSelfAttention-
Encoder-6-MultiHeadSelfAttentio	(None, 128, 768)	1536	Encoder-6-MultiHeadSelfAttention-
Encoder-6-FeedForward (FeedForw	(None, 128, 768)	4722432	Encoder-6-MultiHeadSelfAttention-
Encoder-6-FeedForward-Dropout (	(None, 128, 768)	0	Encoder-6-FeedForward[0][0]
Encoder-6-FeedForward-Add (Add)	(None, 128, 768)	0	Encoder-6-MultiHeadSelfAttention- Encoder-6-FeedForward-Dropout[0][
Encoder-6-FeedForward-Norm (Lay	(None, 128, 768)	1536	Encoder-6-FeedForward-Add[0][0]
Encoder-7-MultiHeadSelfAttentio	(None, 128, 768)	2362368	Encoder-6-FeedForward-Norm[0][0]
Encoder-7-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-7-MultiHeadSelfAttention[
Encoder-7-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-6-FeedForward-Norm[0][0] Encoder-7-MultiHeadSelfAttention-
Encoder-7-MultiHeadSelfAttentio	(None, 128, 768)	1536	Encoder-7-MultiHeadSelfAttention-
Encoder-7-FeedForward (FeedForw	(None, 128, 768)	4722432	Encoder-7-MultiHeadSelfAttention-
Encoder-7-FeedForward-Dropout (	(None, 128, 768)	0	Encoder-7-FeedForward[0][0]
Encoder-7-FeedForward-Add (Add)	(None, 128, 768)	0	Encoder-7-MultiHeadSelfAttention- Encoder-7-FeedForward-Dropout[0][

Encoder-7-FeedForward-Norm (Lay	(None, 128, 768)	1536	Encoder-7-FeedForward-Add[0][0]
Encoder-8-MultiHeadSelfAttentio	(None, 128, 768)	2362368	Encoder-7-FeedForward-Norm[0][0]
Encoder-8-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-8-MultiHeadSelfAttention[
Encoder-8-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-7-FeedForward-Norm[0][0] Encoder-8-MultiHeadSelfAttention-
Encoder-8-MultiHeadSelfAttentio	(None, 128, 768)	1536	Encoder-8-MultiHeadSelfAttention-
Encoder-8-FeedForward (FeedForw	(None, 128, 768)	4722432	Encoder-8-MultiHeadSelfAttention-
Encoder-8-FeedForward-Dropout (	(None, 128, 768)	0	Encoder-8-FeedForward[0][0]
Encoder-8-FeedForward-Add (Add)	(None, 128, 768)	0	Encoder-8-MultiHeadSelfAttention- Encoder-8-FeedForward-Dropout[0][
Encoder-8-FeedForward-Norm (Lay	(None, 128, 768)	1536	Encoder-8-FeedForward-Add[0][0]
Encoder-9-MultiHeadSelfAttentio	(None, 128, 768)	2362368	Encoder-8-FeedForward-Norm[0][0]
Encoder-9-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-9-MultiHeadSelfAttention[
Encoder-9-MultiHeadSelfAttentio	(None, 128, 768)	0	Encoder-8-FeedForward-Norm[0][0] Encoder-9-MultiHeadSelfAttention-
Encoder-9-MultiHeadSelfAttentio	(None, 128, 768)	1536	Encoder-9-MultiHeadSelfAttention-
Encoder-9-FeedForward (FeedForw	(None, 128, 768)	4722432	Encoder-9-MultiHeadSelfAttention-
Encoder-9-FeedForward-Dropout (	(None, 128, 768)	0	Encoder-9-FeedForward[0][0]
Encoder-9-FeedForward-Add (Add)	(None, 128, 768)	0	Encoder-9-MultiHeadSelfAttention- Encoder-9-FeedForward-Dropout[0][
Encoder-9-FeedForward-Norm (Lay	(None, 128, 768)	1536	Encoder-9-FeedForward-Add[0][0]
Encoder-10-MultiHeadSelfAttenti	(None, 128, 768)	2362368	Encoder-9-FeedForward-Norm[0][0]
Encoder-10-MultiHeadSelfAttenti	(None, 128, 768)	0	Encoder-10-MultiHeadSelfAttention
Encoder-10-MultiHeadSelfAttenti	(None, 128, 768)	0	Encoder-9-FeedForward-Norm[0][0] Encoder-10-MultiHeadSelfAttention
Encoder-10-MultiHeadSelfAttenti	(None, 128, 768)	1536	Encoder-10-MultiHeadSelfAttention
Encoder-10-FeedForward (FeedFor	(None, 128, 768)	4722432	Encoder-10-MultiHeadSelfAttention
Encoder-10-FeedForward-Dropout	(None, 128, 768)	0	Encoder-10-FeedForward[0][0]
Encoder-10-FeedForward-Add (Add	(None, 128, 768)	0	Encoder-10-MultiHeadSelfAttention Encoder-10-FeedForward-Dropout[0]
Encoder-10-FeedForward-Norm (La	(None, 128, 768)	1536	Encoder-10-FeedForward-Add[0][0]
Encoder-11-MultiHeadSelfAttenti	(None, 128, 768)	2362368	Encoder-10-FeedForward-Norm[0][0]
Encoder-11-MultiHeadSelfAttenti	(None, 128, 768)	0	Encoder-11-MultiHeadSelfAttention
Encoder-11-MultiHeadSelfAttenti	(None, 128, 768)	0	Encoder-10-FeedForward-Norm[0][0] Encoder-11-MultiHeadSelfAttention
Encoder-11-MultiHeadSelfAttenti	(None, 128, 768)	1536	Encoder-11-MultiHeadSelfAttention
Encoder-11-FeedForward (FeedFor	(None, 128, 768)	4722432	Encoder-11-MultiHeadSelfAttention

Encoder-11-FeedForward-Dropout	(None, 128, 768)	0	Encoder-11-FeedForward[0][0]
Encoder-11-FeedForward-Add (Add	(None, 128, 768)	0	Encoder-11-MultiHeadSelfAttention Encoder-11-FeedForward-Dropout[0]
Encoder-11-FeedForward-Norm (La	(None, 128, 768)	1536	Encoder-11-FeedForward-Add[0][0]
Encoder-12-MultiHeadSelfAttenti	(None, 128, 768)	2362368	Encoder-11-FeedForward-Norm[0][0]
Encoder-12-MultiHeadSelfAttenti	(None, 128, 768)	0	Encoder-12-MultiHeadSelfAttention
Encoder-12-MultiHeadSelfAttenti	(None, 128, 768)	0	Encoder-11-FeedForward-Norm[0][0] Encoder-12-MultiHeadSelfAttention
Encoder-12-MultiHeadSelfAttenti	(None, 128, 768)	1536	Encoder-12-MultiHeadSelfAttention
Encoder-12-FeedForward (FeedFor	(None, 128, 768)	4722432	Encoder-12-MultiHeadSelfAttention
Encoder-12-FeedForward-Dropout	(None, 128, 768)	0	Encoder-12-FeedForward[0][0]
Encoder-12-FeedForward-Add (Add	(None, 128, 768)	0	Encoder-12-MultiHeadSelfAttention Encoder-12-FeedForward-Dropout[0]
Encoder-12-FeedForward-Norm (La	(None, 128, 768)	1536	Encoder-12-FeedForward-Add[0][0]
Extract (Extract)	(None, 768)	0	Encoder-12-FeedForward-Norm[0][0]
NSP-Dense (Dense)	(None, 768)	590592	Extract[0][0]
dense (Dense)	(None, 2)	1538	NSP-Dense[0][0]

Total params: 109,188,866
Trainable params: 109,188,866
Non-trainable params: 0

عملکرد این مدل بر روی این دادگان به شکل زیر بود :



که نشان می‌دهد این مدل به سرعت پیشرفت کرده و خیلی زود پیشرفت آن متوقف می‌شود. البته ممکن است با ادامه دادن آموزش بازهم بازدهی بیشتری کسب می‌کرد اما به دلیل محدودیت‌ها موفق به ادامه آموزش نشدیم. نسخه‌ی بدون پیش‌پردازش مدل نیز بصورت مشابهی، با کمی بازدهی کمتر عمل کرد که میتواند مانند Elmo برای دادگان اسپم آن‌را تحلیل کرد.

جمع‌بندی

همانطور که مشاهده شد، به طول کلی مدل Bert خیلی سریع آموزش می‌بیند و روی دادگان فیت می‌شود. در مقایسه‌ی دو مدل Bert و Elmo، ممکن است اگر آموزش را ادامه می‌دادیم مدل Bert بهتر عمل می‌کرد که می‌توان این نتیجه را گرفت با دادگان زیادتر مدل Bert بهتر عمل می‌کند.

بخش سوم) مورد اول : استفاده از شبکه‌های CNN بجای FeedForward

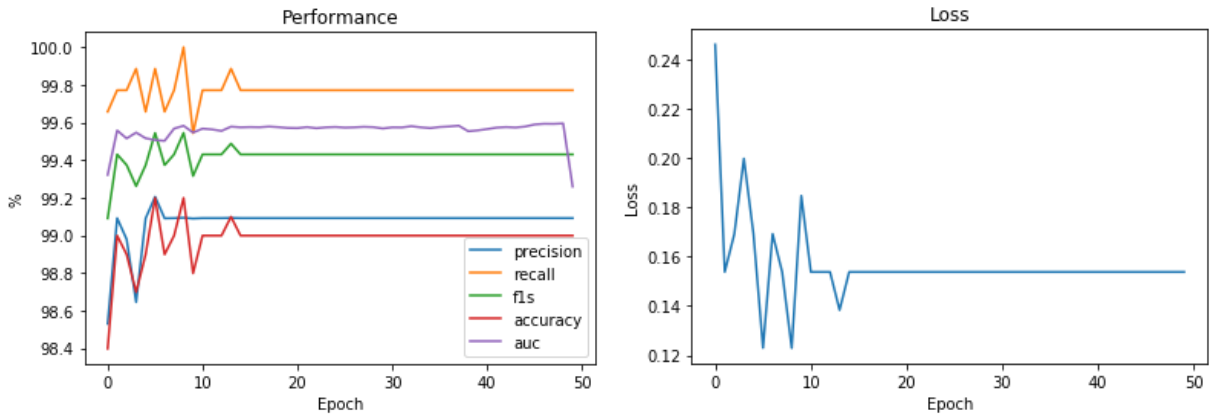
عملکرد این شبکه بر روی مدل‌های Elmo بدون پیش‌پردازش بر روی دادگان اسپم، Elmo با پیش‌پردازش بر روی دادگان IMDB و Bert بدون پیش‌پردازش بر روی دادگان اسپم انجام شده و نتایج به شکل زیر است (هر مورد را با مورد متناظر با شبکه‌ی FeedForward مقایسه کرده‌ایم) :

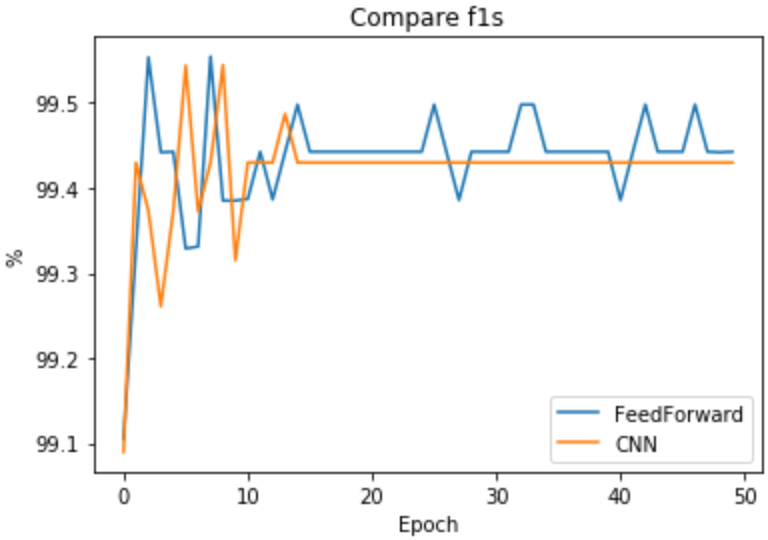
ELMO

ساختار این شبکه به شکل زیر بود :

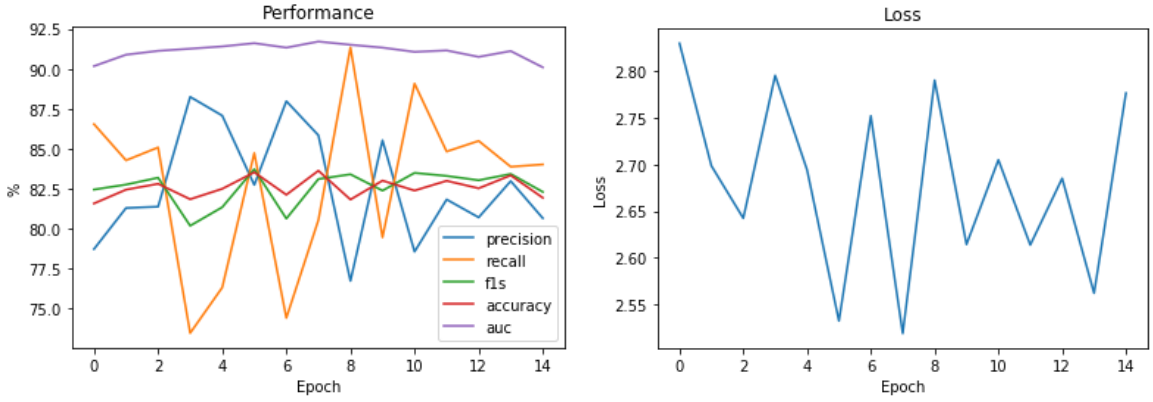
```
_________________________________________________________________
# Layer (type)                 Output Shape              Param
=================================================================
input_1 (InputLayer)         (None, 1)                 0
_________________________________________________________________
elmo_embedding_layer_1 (Elmo (None, 1024)              4
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
non_masking_1 (NonMasking)   (None, 1024)              0
_________________________________________________________________
reshape_1 (Reshape)          (None, 1, 1024)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 1, 50)             153650
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 51
=================================================================
Total params: 1,205,855
Trainable params: 1,205,855
Non-trainable params: 0
_________________________________________________________________
```

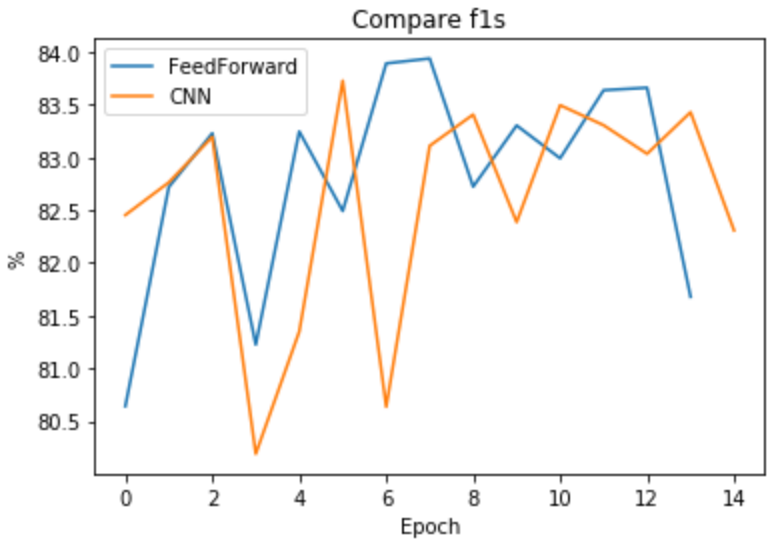
نتایج برای مسئله‌ی ایمیل اسپم به شکل زیر بود :





که مشخصا کمی از مدل اصلی عملکرد ضعیف‌تری دارد اما با ثبات‌تر است. همچنین نتایج برای مسئله‌ی IMDB به شکل زیر بود:





همانطور که مشاهده می‌شود عملکرد مشابهی دارند. برخلاف انتظار شبکه‌ی CNN کمک شایانی نکرد.

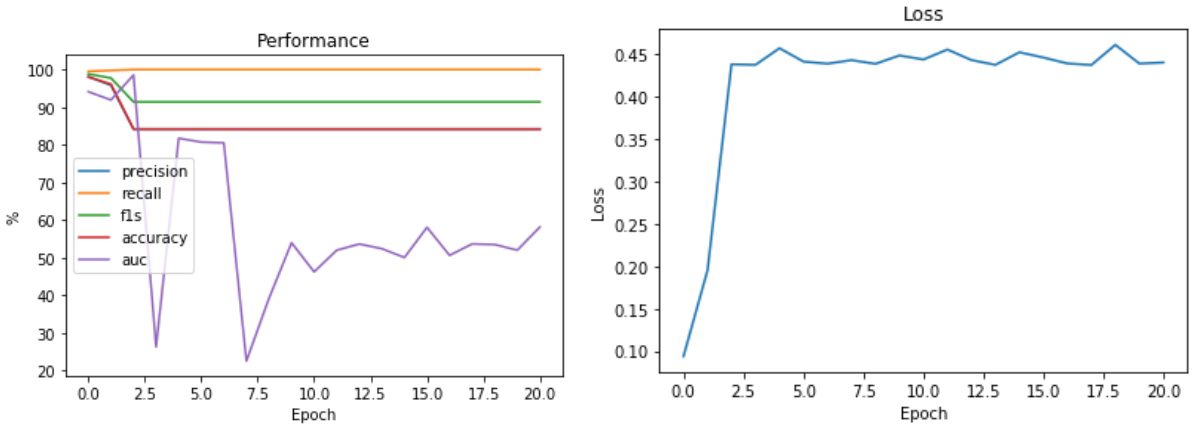
BERT

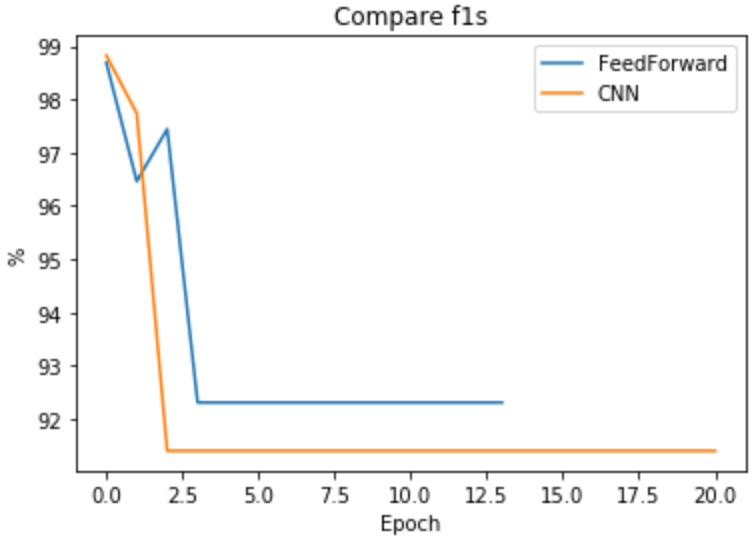
ساختار این شبکه به شکل زیر بود :

Layer (type)	Output Shape	Param #	Connected to
input_ids (InputLayer)	[(None, 128)]	0	
input_mask (InputLayer)	[(None, 128)]	0	
segment_ids (InputLayer)	[(None, 128)]	0	
[bert_layer (BertLayer) [input_mask[0][0 [segment_ids[0][0	(None, 768)	110104890	input_ids[0][0
[dense (Dense)	(None, 768)	590592	bert_layer[0][0
[non_masking (NonMasking)	(None, 768)	0	dense[0][0
[reshape (Reshape)	(None, None, 768)	0	non_masking[0][0
[conv1d (Conv1D)	(None, None, 50)	115250	reshape[0][0
[global_max_pooling1d (GlobalMax	(None, 50)	0	conv1d[0][0
[dense_1 (Dense)	(None, 50)	2550	global_max_pooling1d[0][0
[dense_2 (Dense)	(None, 1)	51	dense_1[0][0

Total params: 110,813,333
Trainable params: 72,177,755
Non-trainable params: 38,635,578

عملکرد این مدل بر روی مسئله‌ی ایمیل اسپم به شکل زیر بود :





همانطور که مشاهده می‌شود این مدل نسبت به مدل با شبکه‌ی FeedForward از نقطه‌ی بهتری شروع می‌کند (در پایان تکرار اول) و در تکرارهای اولیه عملکرد بهتری دارد ولی سریع‌تر و بیشتر نزول می‌کند. با توجه به اینکه نقطه‌ی قوت لایه‌ی Bert در تکرارهای اول است می‌توان گفت این شبکه کمک کرده است.

فایل‌های جانبی

به همراه این گزارش، یک پوشه به نام Codes ارائه می‌شود که حاوی فایل‌های زیر است :

- پوشه‌ی plots حاوی کد پلات‌های داخل صورت پروژه. توجه کنید که دادگان نمودارها به دلیل حجم زیاد حذف شده‌اند.
- پوشه‌ی Failed حاوی آزمون و خطاهای بی‌نتیجه.
- پوشه‌ی Layersحاوی پیاده سازی دو مدل ELMo و Bert به عنوان لایه‌های آموزشی شبکه‌های keras.
- پوشه‌ی Implementations حاوی قسمت‌های مختلف مسائل.

نکات نهایی

با توجه به نیاز شدید این پروژه به ابزار قوی و زمان استفاده‌ی زیاد، قادر به انجام بسیاری از آزمایشات نبودم. همانطور که مشاهده می‌کنید بسیاری از نمودارها به epochهای موردنظر (۵۰) نرسیده‌اند و دلیل آن نبودن شرایط محاسبه بود. البته اکثر اطلاعات حیاتی از عملکرد مدل‌ها در نمودارهای آورده شده موجود هستند و قسمت‌های کم اهمیت از دست رفته‌اند. لازم به ذکر است که GPU سیستم Colab به صورت محدود در اختیار مصرف کنندگان قرار می‌گیرد و من مصرف ۴ کاربر را به اتمام رساندم ولی باز هم در مسئله‌ی IMDB موفق به اتمام محاسبات نشدم. لطفا این موارد را درنظر بگیرید.

همچنین مدل Bert که در IMDB استفاده شده از TPU برای تسریع استفاده می‌کرد که colab محدودیت بسیار زیادی برای آن دارد.