به نام خدا

دانشگاه صنعتی امیرکبیر (پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

شبکه عصبی

تکلیف ششم

استاد درس: دکترصفابخش

امیرحسین کاشانی

۴۰۰۱۳۱۰۷۱

amkkashani@gmail.com

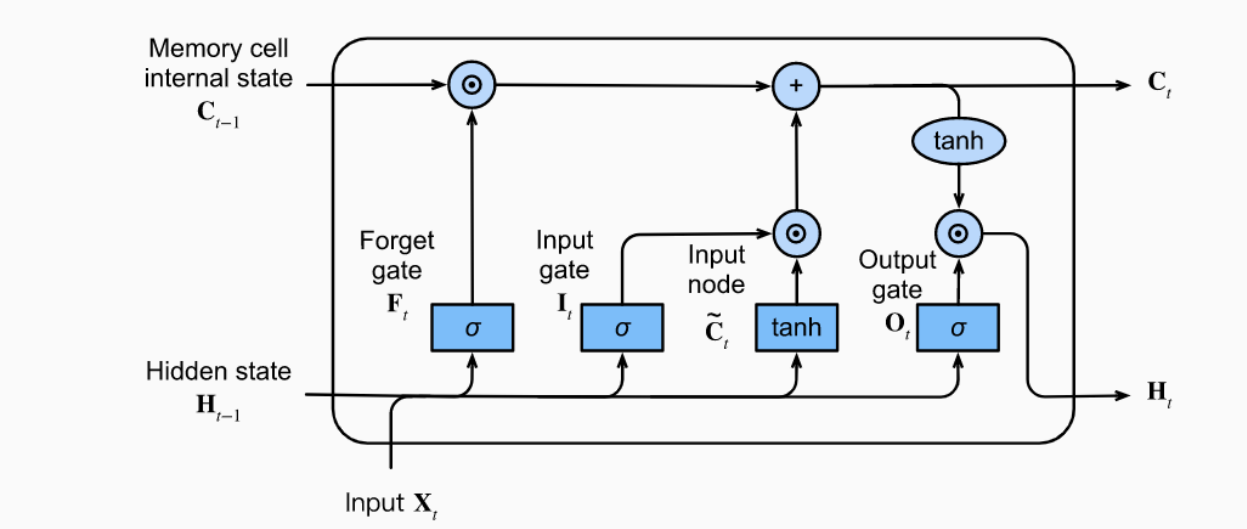
نیم سال اول ۱۴۰۱-۱۴۰۲

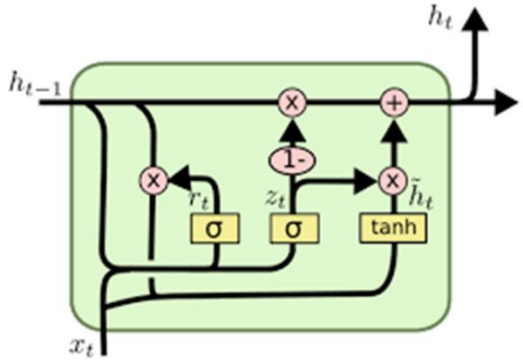
فهرست

سوال ۱)

هردو سلول در دسته ی RNN ها قرار می‌گیرند و در شبکه‌های بازرخدادی کاربرد دارند و ایده اصلی آن‌ها این بوده است که با استفاده از gateها بتوانند بر روی نحوه ذخیره اطلاعات کنترل داشته باشند و اطلاعات را به مراحل بعد منتقل کنند. در LSTM ۳ گیت وجود دارد که به ترتیب Forget gate، تصمیم گیری می‌کند که آیا اطلاعات قبلی مهم بوده اند یا خیر و می‌تواند حافظه را از بین برد، input gate مسئول این است که ورودی فعلی و نتیجه نهایی سلول قبل در حافظه چه میزان تاثیر بگذارد، output gate خروجی hidden state را مشخص می‌کند که چه میزان از استیت قبلی و چه میزان از memory state روی آن تاثیر گذارد.

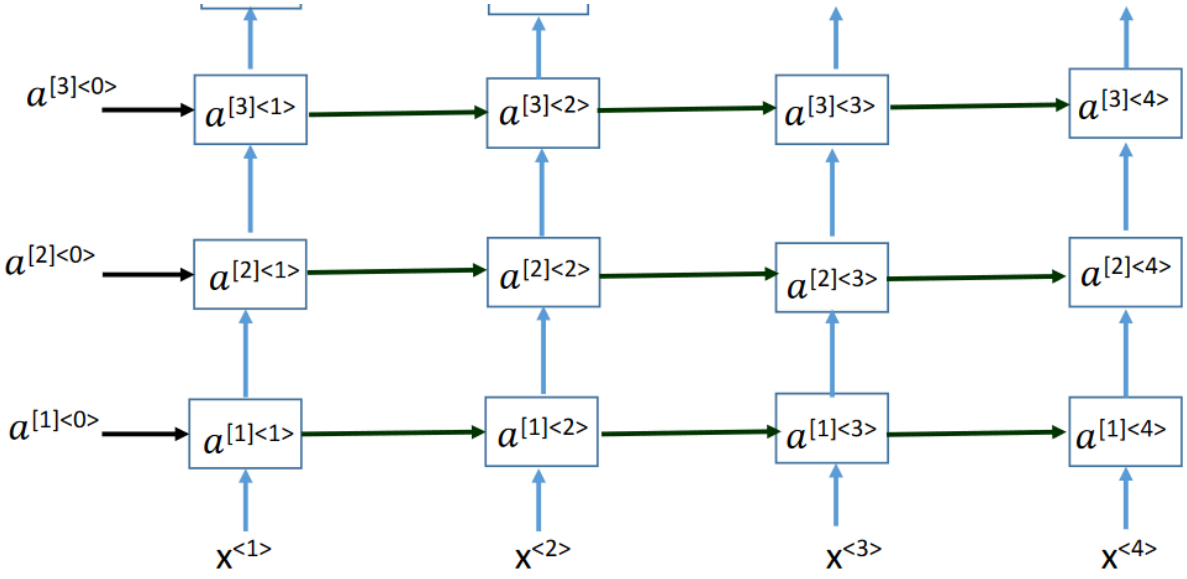
در سلول gru از دو گیت استفاده می‌شود که در آن update gate وظیفه input gate , forget gate را در lstm انجام می‌دهد و reset gate نیز مشخص می‌کند چه مقدار از اطلاعات گذشته حذف گردند. با توجه به اینکه سلول gru تعداد گیت و محاسبات کمتری دارد در بخش آموزش و تست سریع تر عمل می‌کند اما اظهار نظر کلی در رابطه با عملکرد LSTM و GRU باید متناسب با مساله مورد بحث تصمیم گیری شود.





در صورتی که این مدل‌ها را بصورت stack قرار دهیم جزو مدل‌های deep قلمداد می‌شوند اما از آن جایی که شبکه‌های بازرخدادی عموما از قدرت قابل قبولی به تنهایی برخوردارند عموما بیش از ۳ لایه از این شبکه‌ها را روی هم stack نمی‌کنند.

نکته ای که می‌توان در این جا در تفاوت LSTM و GRU به آن اشاره کرد این است که cell state , hidden state در LSTM دو ماهیت جدا هستند اما در GRU خروجی مدل فقط hidden state می‌باشد. برای زمانی که از LSTM استفاده می‌گردد می‌توانیم از هرکدام از memory state , hidden state برای ارتباط بین لایه‌ها استفاده کنیم اما در GRU فقط یک حالت برای ما وجود دارد.



سوال ۲)

خطای خروجی k ام را با V_k به شکل زیر محاسبه می‌کنیم

$$\vartheta_k(t) = f_k'(net_k(t))(d_k(t) - y^k(t))$$

$$y^i(t) = f_i(net_i(t))$$

$$net_i(t) = \sum_j w_{ij} y^j(t-1)$$

و همین طور برای خروجی‌های میانی خواهیم داشت

$$\vartheta_j(t) = f_j'(net_j(t)) \sum_i w_{ij} \vartheta_i(t+1)$$

مشتق بدست آمده برای لایه‌ها به شکل زیر خواهد بود

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \begin{cases} f'_v(net_v(t-1))w_{uv} & q=1 \\ f'_v(net_v(t-q))\sum_{l=1}^{n} \frac{\partial \vartheta_l(t-q+1)}{\partial \vartheta_u(t)} w_{lv} & q>1 \end{cases}$$

برای اینکه جلوی vanishing gradian را بگیریم باید تلاش کنیم تا عبار زیر بر قرار باشد.

$$f'_j(net_j(t))w_{jj} = 1.0$$

برای اینکه بتوانیم به این مهم دست یابیم توصیه‌های صورت می‌گیرید که عبارت اند از

۱. استفاده از وزن‌های کوچک تر در آغاز آموزش

۲. به کار گیری تابع فعال سازی relu یا تابعی که مشتق برابر یک داشته یا منفی یک داشته باشد

۳. استفاده از skip کانکشن‌ها در قبل و بعد مدل تا flow در جریان عملیات BP به راحتی عبور کند.

۴. استفاده از Gradient Clipping این روش برای جلوگیری از رشد انفجاری گرادیان است و در صورتی که گرادیان از یک مقدار آستانه بزرگ تر باشد آن را با یک مقدار پیش فرض جایگزین می‌کند.

۵. استفاده از optimizerهای دیگر مثل Adam, Adagrad and Adadelta نیز توصیه شده اند که به عنوان جواب این مساله نمی‌توان آن‌ها را مد نظر قرار داد.

لازم به ذکر است که LSTM به طور کامل نسبت vanishing gradian مقاوم نیست بلکه عملکرد مناسبی تا حدود ۱۰۰۰ گام از خود نشان داده است و ن‌می‌توان گفت که به طور کامل نسبت به این پدیده مبرا می‌باشد.

سوال ۳)

بله ، شبکه‌های مبتنی ببر LSTM امکان آموزش بصورت توزیع شده برروی GPUهای مختلف را دارند و این کار به چند حالت برای ما امکان پذیر است:

۱. Data parallelism: در این روش داده به بچ‌های مختلفی تقسیم می‌شود و آموزش بصورت موازی بر روی آن‌ها صورت می‌گیرد.(در حین آموزش مدل‌ها با هم sync خواهد شد و دستاوردهای آموزشی هریک بر دیگری تاثیر خواهد گذاشت و در آخر یک مدل جامع بدست می‌آید)

۲. Model parallelism: در این روش لایه‌ها به بخش‌های مختلف تقسیم می‌شوند و در بخش‌های مختلف آموزش می‌بینند

۳. Hybrid parallelism: این روش ترکیبی از دو روش قبلی است

در آموزش LSTM موازی و توزیع شده می‌توان از کتابخانه‌های pyTorch و Tensorflow بهره برد.

پیاده سازی

پیاده سازی بخش اول)

در این بخش مجموعه ای از پیش پردازش‌ها بر روی داده مورد نظر اعمال شده است.

Lower case

یکی از مهم ترین کارهایی که باید انجام شود این است که در صورتی که یک کلمه در برخی جاهای متن با حروف بزرگ شده است آن را با حروف کوچک جایگزین کنیم تا در زمان شمارش به عنوان کلمه ای متفاوت برداشت نشود.

remove punctuation

در این بخش علائمی‌مانند ",? .” و ... را از متن حذف می‌کنیم که محتوای خاصی در مدل فعلی ما ندارند. این علاوه بر خلاصه کردن تعداد توکن‌های استخراج شده در برخی حالات که نقطه یا علامت نگارشی به کلامت چسبیده است نیز باعث می‌شود تا از تشخیص آن‌ها به عنوان کلمه ای متفاوت جلوگیری شود. همچنین url درصورت وجود، urlهای داخل متن نیز حذف گردیده است.

Remove stop words

Stop wordها کلماتی هستند که به کررات در جملات تکرار می‌شوند و مستقل از ماهیت پیام تعداد تکرار آن‌ها زیاد است و مدل ما نباید وبستگی به این نوع کلمات داشته باشد و واکنشی به تعداد آن‌ها نباید نشان دهد. از این رو به حذف کلماتی مثل ,and ,... where , then می‌پردازیم. برای این کار از کتابخانه NLTK بهره برده شده است و مجموعه stop wordهای پیش فرض این پکیج از داده‌ها حذف گردیده است.

Stemming

در این روش کلمات هم ریشه را به یک کلمه تبدیل می‌کنیم مانند run , running یا از این قبیل کلمات به طور کلی پکیج NLTK برای اجرای این کار prefix , suffixها را از کلمات حذف می‌کند و در نتیجه شمارش مجموعه تکرارهای لغات ما دقیق تر خواهد شد.

پیاده سازی بخش دوم)

دلیل نامناسب بودن روش one_hot_encoding این است که در این روش به ازای هر کلمه یک ستون و یک بعد برای مساله ما ایجاد می‌شود(که در اکثر حالات صفر است و اطلاعاتی را در بر ندارد) در نتیجه منجر به وجود آمدن صدها هزار بعد در مساله ما می‌گردد که عملا ایجاد مدلی برای حل مساله را با توجه به پیچیدگی بیش از حد غیر ممکن می‌سازد از این رو برای این مسائل از مدل‌هایی استفاده می‌کنیم که کلمات و عبارات را در بردارهایی با اندازه کوچک تر بصورت عددی بازنمایی می‌کنند. نکته حائز اهمیت در روش‌های embedding این است که similarity preserving باشند یعنی شباهت کلمات را در بازنمایی خود حفظ کنند.

Word2vec

یک تکنیک در پردازش زبان طبیعی است که می‌تواند با آموزش برروی حجم زیادی از کلمات، کلمات هم معنی را تشخیص دهد و همچنین کلمات مناسب برای جملات ناقص را نیز پیشنهاد کند. همچنین این روش برای انجام محاسبات خود هر کلمه را به یک بردار از اعداد تبدیل می‌کند که این بردار در بر دارنده معنا ومحتوای این کلمه است. که برای مقایسه شباهت کلمات نیز می‌توان از شباهت کوسینوسی بهره برد.

دو روش برای بدست آوردن بردارهای مورد نظر بیان شده است:

این دو روش که هر دو یک شبکه عصبی ساده هستند که بدون وجود لایه پنهانی که در اغلب روشهای شبکه عصبی وجود دارد، به کمک چند قانون ساده، بردارهای مورد نیاز را تولید می‌کنند. در روش کیف لغات پیوسته (CBOW) ، ابتدا به ازای هر لغت یک بردار با طول مشخص و با اعداد تصادفی (بین صفر و یک) تولید می‌شود. سپس به ازای هر کلمه از یک سند یا متن، تعدادی مشخص از کلمات بعد و قبل آنرا به شبکه عصبی می‌دهیم و با عملیات ساده ریاضی، بردار لغت فعلی را تولید می‌کنیم (یا به عبارتی از روی کلمات قبل و بعد یک لغت، آنرا حدس می‌زنیم) که این اعداد با مقادیر قبلی بردار لغت جایگزین می‌شوند. زمانی که این کار بر روی تمام لغات در تمام متون انجام گیرد، بردارهای نهایی لغات همان بردارهای مطلوب ما هستند.

روش Skip-gram برعکس این روش کار می‌کند به این صورت که بر اساس یک لغت داده شده ، می‌خواهد چند لغت قبل و بعد آنرا تشخیص دهد و با تغییر مداوم اعداد بردارهای لغات، نهایتا به یک وضعیت باثبات می‌رسد که همان بردارهای مورد بحث ماست.

```
# Create Skip Gram model
model2 = gensim.models.Word2Vec(data, min_count = 1, vector_size = 100,
                                window = 5, sg = 1)
```

برای ایجاد استفاده از word2vec از دستور بالا کمک گرفته شده است که به در آن sg به معنی استفاده از skip_gram ، window طول پنجره کلمات، vector_size اندازه بردار نهایی ، min_count نیز بیان می‌کند که اگر تکرار کلمه ای از این مقدار کمتر بود نادیده گرفته شود.

پیاده سازی بخش سوم)

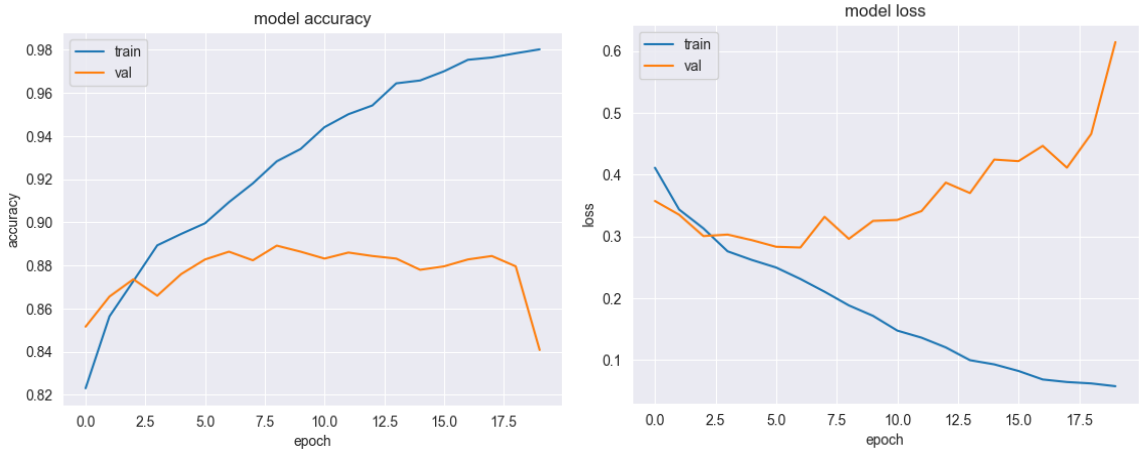
برای اینکه سایز مناسب برای lag یا تعداد کلمات مورد نیاز رو انتخاب کنیم ابتدا داده‌های طول داده‌های آموزش و تست را بدست آوردیم و بر اسال طول مرتب کردیم و به این نتیجه رسیدیم که ۸۵ درصد داده‌ها طولی کمتر از ۲۰۲ کلمه دارند که با اغماض ۲۰۰ کلمه را به عنوان طول پیشنهادی انتخاب کردیم.

پیاده سازی بخش چهارم)

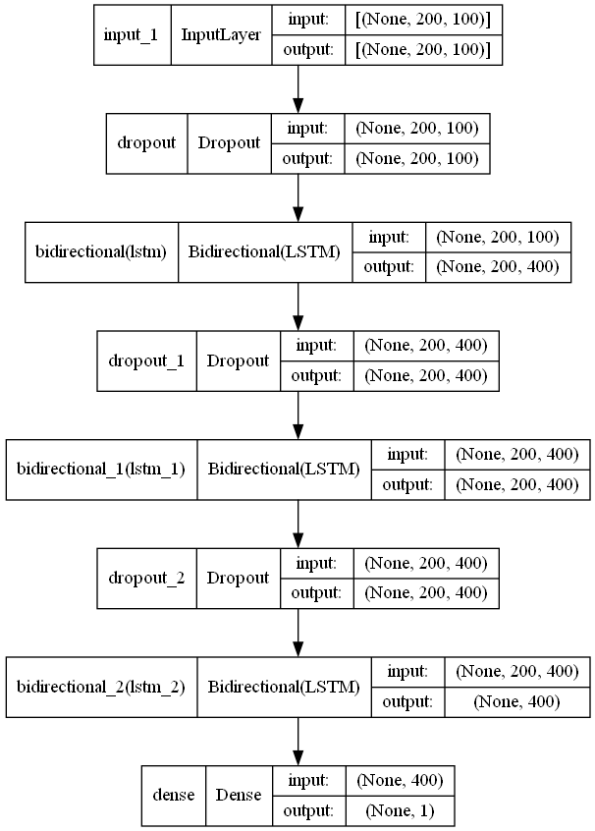
با توجه به اینکه حجم داده‌ها برای آموزش و تست بسیار زیاد بوده و از مقدار Ram سیستم بیشتر بوده است. وکتورهای داده‌های آموزش و تست ذخیره شده و در زمان نیاز بصورت مجزا load گردیده، این روال در همه مدل‌های این بخش و بخش بعد پا برجاست. (همین طور برای مدل‌های ساخته شده این فرآیند تکرار شده است به این معنی که مدل پس از آموزش ذخیره شده و بعد دوباره برای تست از حافظه load گردیده است)

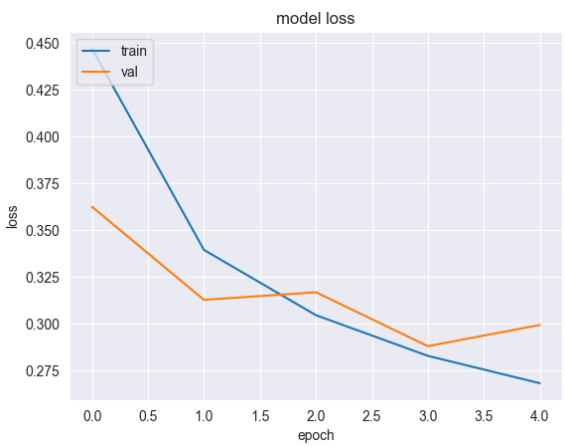
Lstm

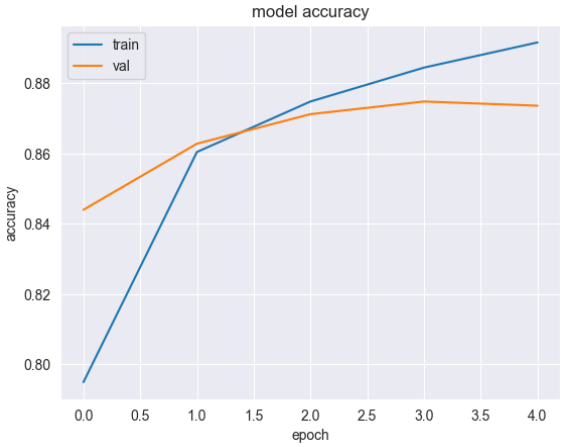
نمودار overfit شده برای مدل lstm



با توجه به نمودارها مدل ما در حدود epoch دهم به بهترین نقطه برای متوقف کردن می‌رسد از همین رو مدل را یک بار دیگر برای epoch = 5 تکرار می‌کنیم

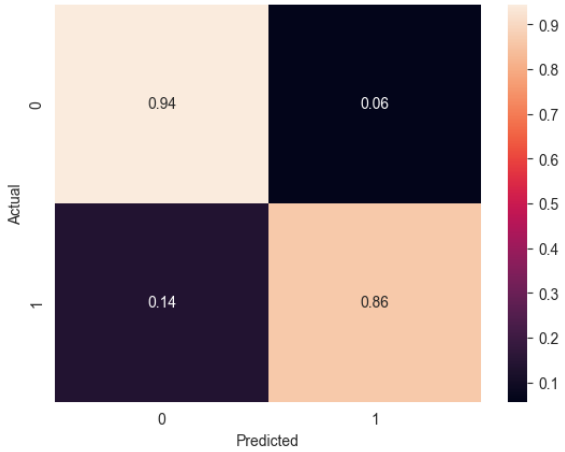






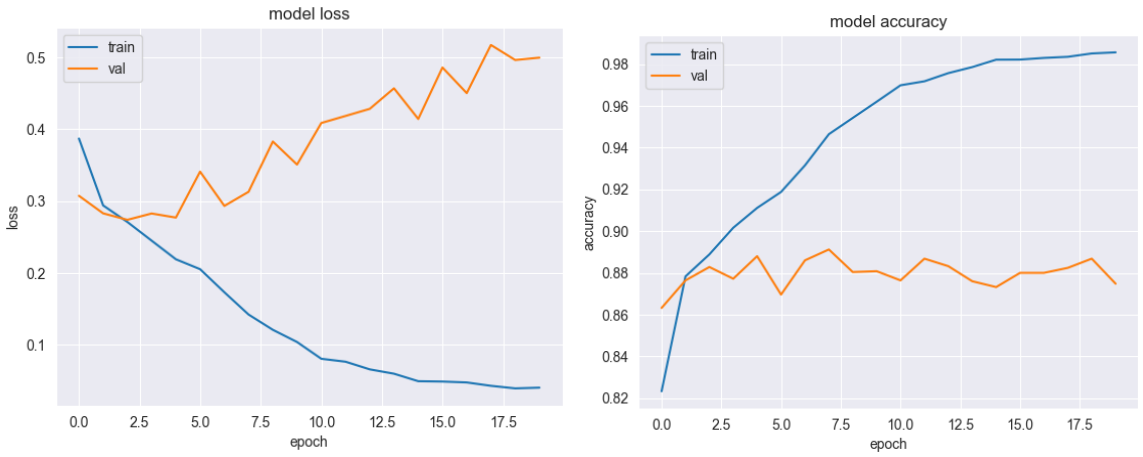
	precision	recall	f1-score	support
0	0.87	0.94	0.91	12500
1	0.94	0.86	0.90	12500
accuracy			0.90	25000
macro avg	0.91	0.90	0.90	25000
weighted avg	0.91	0.90	0.90	25000

******
accuracy model 90.256 %

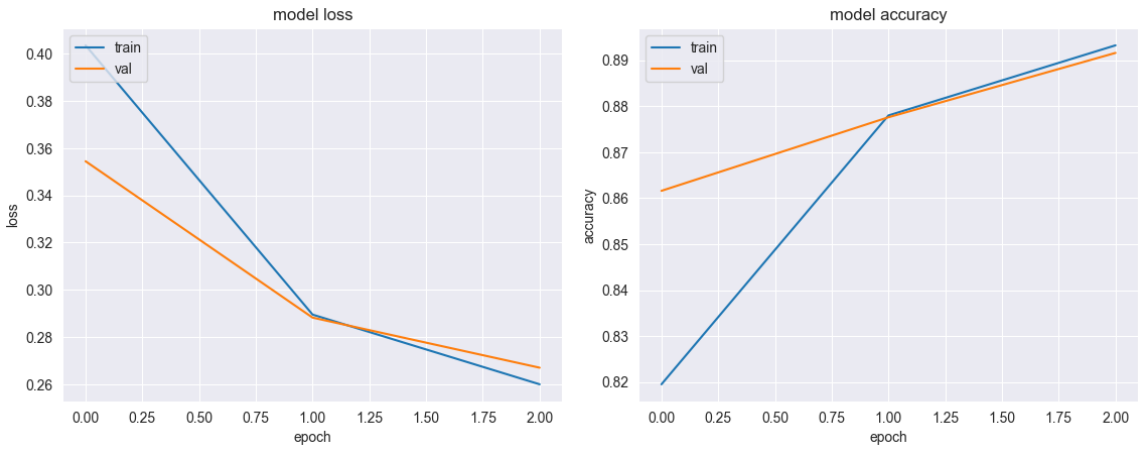


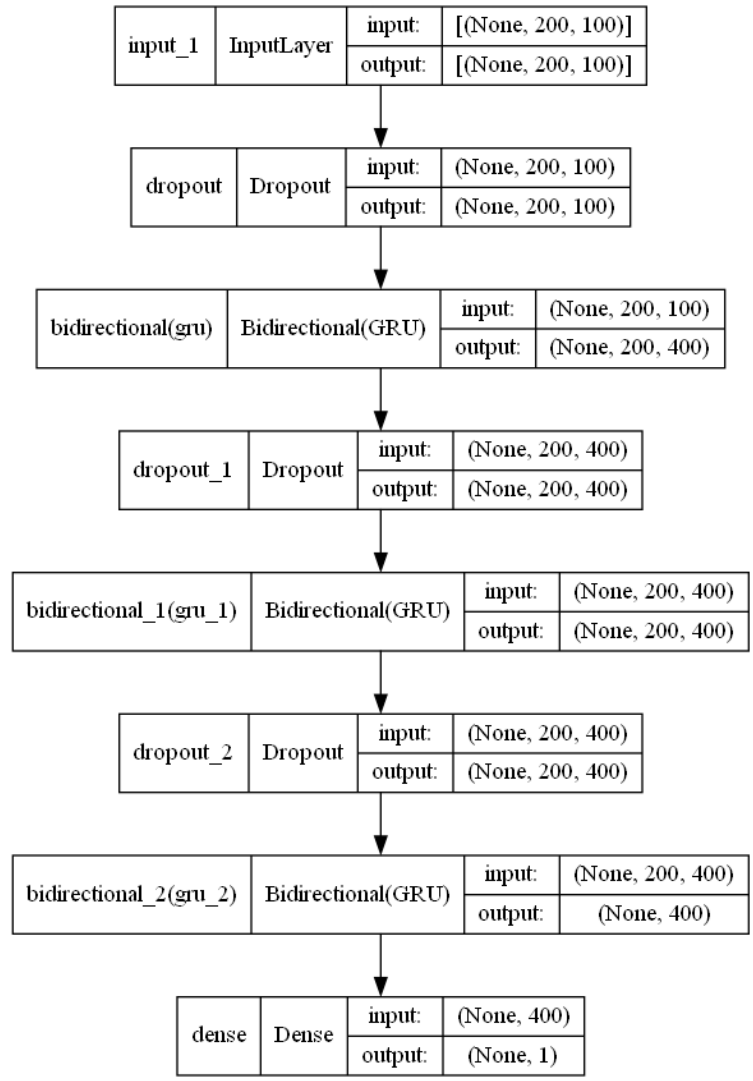
بخش **GRU**

ابتدا مدل را با ۲۰ epoch آموزش می‌دهیم تا نقطه مناسب برای توقف را دریابیم.



با توجه به مدل بدست آمده درمیابیم که مدل در حدود ۳ epoch وضعیت مناسبی دارد که این مقدار تقریبا نصف تعداد epochهای lstm بود همچنین تعداد متغیرهای قابل آموزش در این مدل نیز حدود ۲۰ درصد کمتر از lstm بود که موجب سرعت بیشتر در یادگیر هر epoch می‌شود.



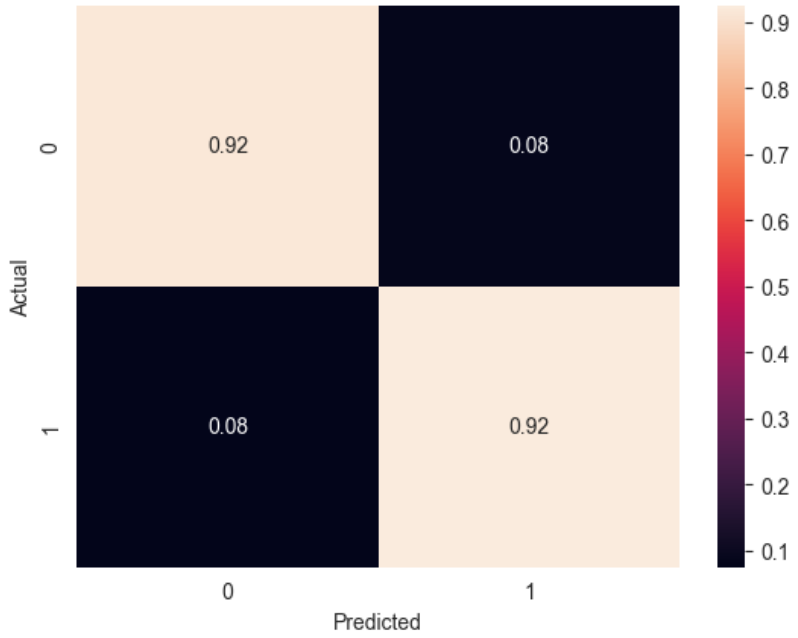



```
              precision    recall  f1-score   support

           0       0.92      0.92      0.92     12500
           1       0.92      0.92      0.92     12500

    accuracy                           0.92     25000
   macro avg       0.92      0.92      0.92     25000
weighted avg       0.92      0.92      0.92     25000

******
accuracy model 92.072 %
```

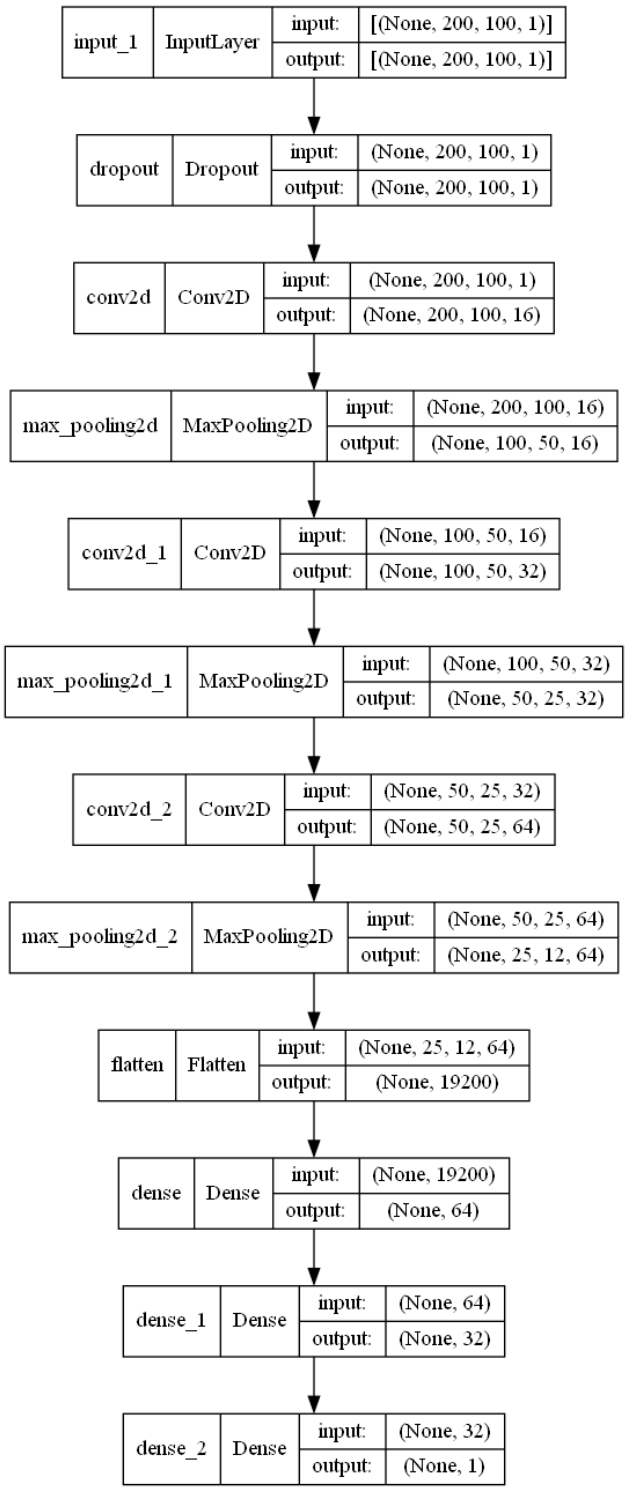


در نهایت نتایج نشان می‌دهد با وجود اینکه تعداد پارامترهای GRU در فرآیند آموزش نسبت به LSTM کمتر بود توانست علاوه بر سرعت بیشتر در آموزش (سرعت محاسبه هر epoch و تعداد epoch) توانست مقدار جزئی نیز در عملکرد بهبود حاصل کند (حدود ۲درصد).

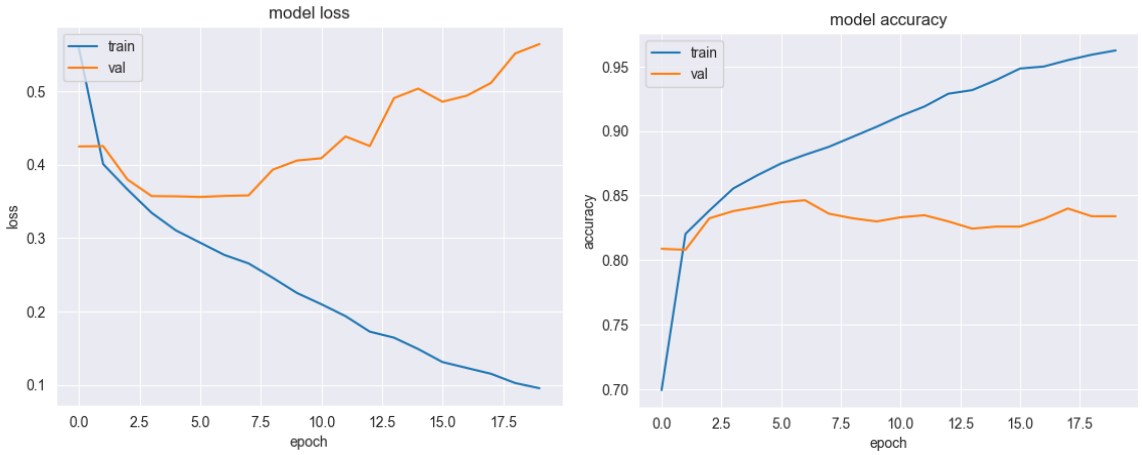
** در کنار فایل ارائه شده یک فایل دیگر نیز وجود دارد که در آن در به جای استفاده از ۳ لایه ی ۲۰۰ تایی از lstm و gru از ۱۰۰ عدد سلول استفاده شده است و برای بهبود بیشتر از چند لایه MLP در انتهای مدل بهره گرفته شده است که نتیجه عملکرد بهتر و آموزش سریع تر بوده است (حدود ۹۲ - ۹۳ درصد) که بدلیل طولانی شدن گزارشکار از آوردن نتایج آن بخش در گزارش خود داری شده است اما در فایل جداگانه ای با نام extra_lstm_gru کد و نتایج آن قرارگرفته است.

همچنین به جهت سهولت در دانلود فایل‌ها ضرایب ذخیره شده نیزبصورت یک فایل جدا آپلود شده است تا در صورت لزوم آن‌ها نیز مورد بررسی قرار گیرند.

پیاده سازی بخش امتیازی)



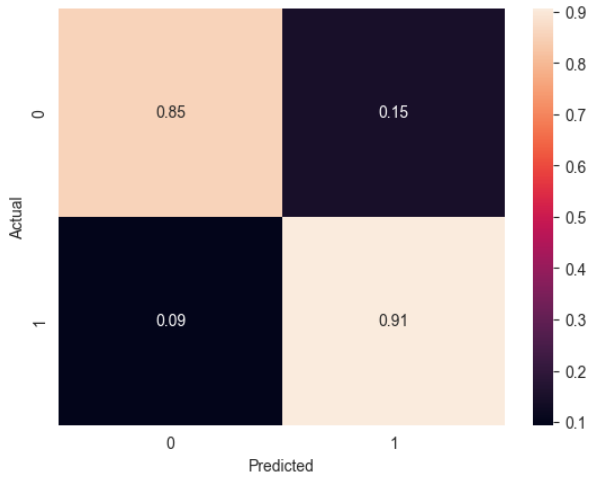
مانند بخش‌های قبل مدل را ۲۰ epoch اجرا می‌کنیم تا لحظه ای که به سمت overfit حرکت می‌کند را پیدا کنیم.



حدودا در epoch ۶ بهترین مکان برای توقف آموزش است.

	precision	recall	f1-score	support
0	0.90	0.85	0.88	12500
1	0.86	0.91	0.88	12500
accuracy			0.88	25000
macro avg	0.88	0.88	0.88	25000
weighted avg	0.88	0.88	0.88	25000

******
accuracy model 87.992 %



عملکرد مدل کانولوشنی به مقدار قابل ملاحظه ای ضعیف تر از دو مدل RNN قبلی بوده است حتی با وجود پارامترهای بیشتر در مدل یادگیری آن‌ها. این اختلاف درحدود ۵ درصد بوده است.

*** در کنار این فایل دو فایل زیپ دیگر نیز قرار داده شده است که در فایل weights خروجی وزن های مدل برای lstm، gru و cnn که lstm , gru با دقت بالای ۹۰ درصد قرار دارد.(در lstm , gru قرار داده شده از MLP در انتهای معماری استفاده نشده است) و در فایل models ، علاوه بر مدل های قبلی مدل هایی که MLP نیز در آن ها استفاده شده نیز قرار داده شده است با این تفاوت که فرمت ذخیره سازی این مدل ها h5 هستند و کار با آن ها راحت تر می باشد.