



دانشگاه تهران  
دانشکده مهندسی  
برق و کامپیوتر



درس پردازش زبان طبیعی  
تمرین سوم

پرهام بیچرانلو

نام و نام خانوادگی

۸۱۰۱۰۰۳۰۳

شماره دانشجویی

۱۴۰۲/۰۳/۲۴

تاریخ ارسال

## ۱. پیش پردازش

### ۱-۱. خواندن داده

در گام اول چون محیط اجرای برنامه سایت kaggle بود دیتا را هم از همان سایت خواندیم. ستون‌های اضافه را حذف کردیم. سپس چون در سوال گفته شده از بخشی از داده‌ها استفاده شود از دیتا ۲ درصد را با حفظ نسبت کلاس‌ها نگه داشتیم و بقیه را برای سنگین نشدن محاسبات دور ریختیم.

### ۱-۲. تقسیم دیتا

برای اینکه بتوانیم مدل را آموزش دهیم و بعد از آن ارزیابی کنیم باید دیتا را به سه بخش تقسیم کنیم که عبارتند از `.train_set`, `validation_set`, `test_set`. از دستور `train_test_split` از کتابخونه `sklearn` برای این کار استفاده کردیم. نسبت تعداد دیتا هم به صورت  $0.7/0.15/0.15$  در نظر گرفتیم.

### ۱-۳. پیش پردازش

بعد از بررسی دیتا متوجه شدم که برخی متون شامل عبارت‌هایی بدون کاربرد مثل لینک، هشتگ و ... هستند. برای اینکه دیتا تمیز بشود چهار کار کردیم:

- تبدیل کاراکترها به حروف کوچک
- حذف کاراکترهای غیر کلمه مثل لینک، منشن
- حذف اعداد
- حذف علائم نگارشی

البته کارهای دیگه هم تست کردم ولی چون در دقت به طور معنادار تاثیر گذار نبود از آن‌ها در کد نهایی صرف نظر کردم تا کد ساده‌تر باشد و خوانایی بهتری داشته باشد.

## ۲. بخش ۱

### ۱-۲. توکنایز کردن

روی جملات دیتاست حرکت می‌کنیم و کلمات را با کاراکتر space جدا کرده و بعد از انجام پیش پردازش روی آن اگر جز stopword ها نباشد و خالی نباشد آن را به لیست کلمات اضافه می‌کنیم.

سپس برای هر سه قسمت داده ایندکس کلمات را به جای آن‌ها قرار می‌دهیم.

همچنین لیبل‌ها را تبدیل به ۰ و ۱ می‌کنیم. در نهایت نتیجه این کارها را برمی‌گردانیم و به عنوان دیتا تمیز و توکنایز شده استفاده می‌کنیم.

### ۱-۲. روش‌های embedding

در سوال از ما خواسته شده که دیتا را با کمک سه embedding متفاوت به عنوان ورودی به شبکه عصبی بازگشتی خود بدهیم. که در ادامه به طور مختصر هر کدام را توضیح می‌دهیم و نحوه به کارگیری آن در مدل را شرح می‌دهیم.

#### • One hot embedding

در این بازنمایی ساده، هر المان در بردار به یک کلمه در لغت نامه اختصاص داده می‌شود یعنی هر درایه این بردار معرف یک کلمه از لغت نامه خواهد بود. برای هر کلمه نیز تنها درایه متناظر برابر با ۱ بوده و مابقی درایه‌ها با صفر مقداردهی خواهند شد. یعنی ابعاد بردار مربوط به هر کلمه اندازه تعداد لغت‌های کرپوس است.

عیب این روش تعداد ابعاد بالا آن و تنک بودن آن است. همچنین هیچ اشتراک اطلاعاتی بین کلمات وجود ندارد که یعنی این روش بازنمایی بی معنا است.

پیاده سازی:

ابتدا یک بردار با ابعاد: تعداد جملات \* طول جملات \* سایز لغت نامه می‌سازیم. سپس خانه ایندکس مربوط به کلمه را ۱ می‌کنیم و بقیه خانه‌های بردار آن کلمه را صفر قرار می‌دهیم. حالا یک برای هر کلمه یک بردار بازنمایی one-hot داریم. البته چون سایز دیتاست بزرگ است و چندده هزار کلمه داریم برای اینکه آموزش مدل خیلی کند نشود و مصرف حافظه را کمتر کنیم دیکشنری کلمات را بر اساس تعداد تکرار sort می‌کنیم و ۲۰۰۰ کلمه پرکاربرد را انتخاب می‌کنیم.

### :Word2Vec embedding •

این روش از کلمات همسایه هر کلمه برای بازنمایی آن استفاده می‌کند. با این ایده که معنی هر کلمه در کابرد آن مشخص می‌شود پس استفاده از همسایه‌ها راه مناسبی به نظر می‌رسد. در نهایت کلماتی که کاربرد مشابه دارند embedding آن‌ها انتظار داریم با این روش مشابه باشد.

پیاده سازی:

از وزن‌های kaggle pre\_train شده در یک دیتای روی سایت استفاده کردیم. که سایز embedding آن ۱۰۰ است. ابتدا خط به خط این فایل را خوانده و برای هر کلمه امبدینگ آن را استخراج کرده و به دیکشنری اضافه می‌کنیم. و در ادامه یک ماتریس embedding برای دیتا با استفاده از این دیکشنری می‌سازیم.

### :GloVe embedding •

یک الگوریتم unsupervised برای بازنمایی کلمات است. برای این کار از هم رخداد بودن کلمات در کرپوس بزرگ استفاده می‌کند. که در آخر یک بازنمایی با طول ثابت بر می‌گرداند اگر بازنمایی دو کلمه مشابه باشد آن دو کلمه مشابه هستند.

پیاده سازی:

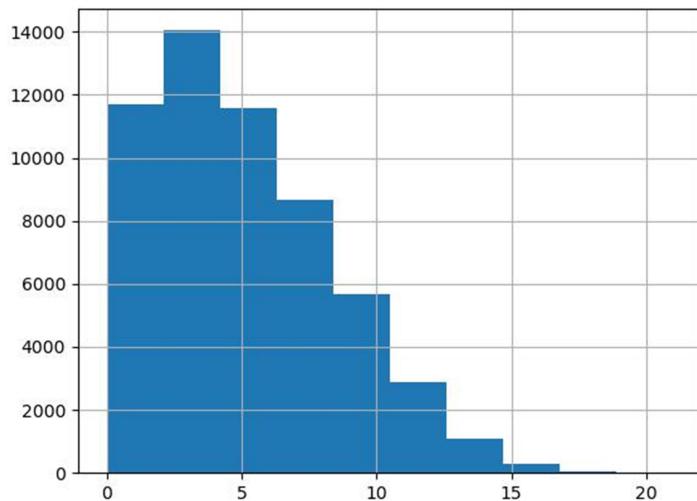
از وزن‌های kaggle pre\_train شده در یک دیتای روی سایت استفاده کردیم. که سایز embedding آن ۱۰۰ است. ابتدا خط به خط این فایل را خوانده و برای هر کلمه امبدینگ آن را استخراج کرده و به دیکشنری اضافه می‌کنیم. و در ادامه یک ماتریس embedding برای دیتا با استفاده از این دیکشنری می‌سازیم.

### ۳-۲. اضافه کردن padding

طول جملات در یک کرپوس متفاوت است. از آنجایی که برای آموزش شبکه عصبی بازگشتی نیاز است که طول جملات برابر باشد باید از تکنیک padding استفاده کنیم یعنی یک سری توکن معین به ابتدای انتهای جمله اضافه کنیم.

اما طول جمله چقدر باشد؟

بهترین کار برای تصمیم گیری دیدن توزیع طول جملات است که در زیر آورده شده است:



شکل ۱. توزیع فرآوانی طول جملات

خب همانطور که مشاهده می‌کنید طول جملات عموماً کمتر از ۲۰ است که ما هم طول جملات را ۲۰ درنظر می‌گیریم و هرچقدر طول جمله‌ای کمتر بود به آن توکن pad اضافه می‌کنیم و هر چقدر بیشتر بود از آخر آن حذف می‌کنیم.

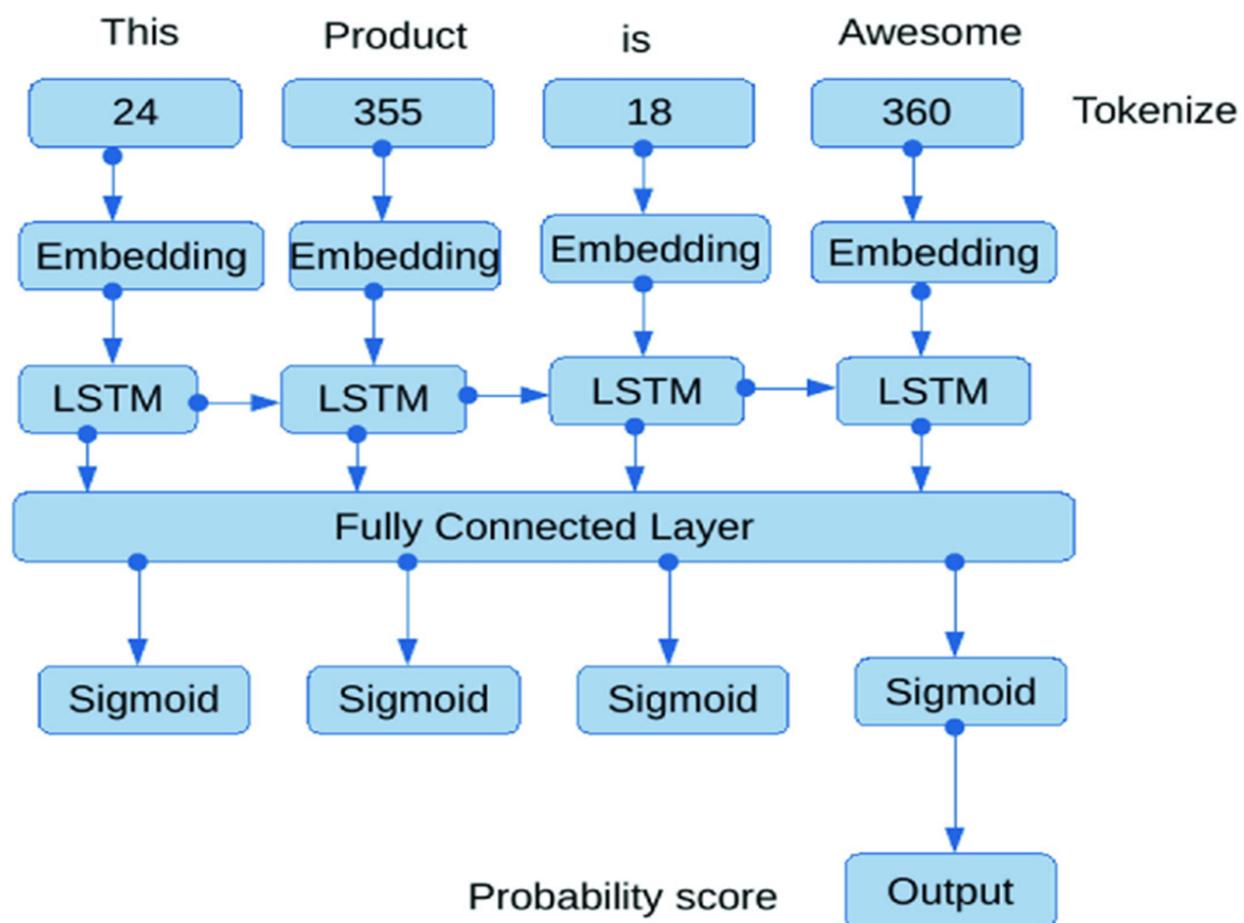
اینکه چرا کمتر یا بیشتر نگرفتیم بیشتر مربوط به تست کردن مدل‌ها بود. اما برای توجیه اگر کمتر باشد یک سری اطلاعات بالرزش از جملات طولانی حذف می‌شود و اگر طول جمله را بیشتر می‌گرفتیم تعداد توکن‌های pad به خصوص برای جملات کوتاه زیاد می‌شد و کار یادگیری را برای مدل سخت‌تر می‌کرد، همچنین حافظه بیشتری نیاز می‌داشتیم.

برای پیاده سازی ابتدا یک آرایه صفر دو بعدی (تعداد جملات، طول جملات) می‌سازیم. اگر طول آن صفر نبود، تا جایی که سایز جمله برابر ۲۰ شود به ابتدای آن توکن pad اضافه می‌کند. این کار را برای تمام دیتا انجام می‌دهیم.

## ۴-۲ و ۵. طراحی شبکه مناسب

چون در این تسک با متن سر و کار داریم و در متن توالی مهم است باید از شبکه‌های عصبی بازگشتی استفاده کنیم. که ساده‌ترین آن‌ها RNN است. و در این بخش از آن استفاده می‌کنیم و در بخش ۲ سوال از LSTM و GRU استفاده خواهیم کرد.

برای اینکه از RNN به خوبی استفاده کنیم باید نوع معماری مدل را مشخص کنیم. چون ما تحلیل sentiment است از معماری many\_to\_one استفاده خواهیم کرد. یعنی یک جمله را می‌دهیم و انتظار داریم با کمک state آخر RNN قطبیت جمله را مشخص کنیم. برای مثال معماری یک شبکه بازگشتی برای تحلیل sentiment در شکل زیر آورده شده است.



شکل ۲. معماری شبکه عصبی بازگشتی برای تحلیل sentiment

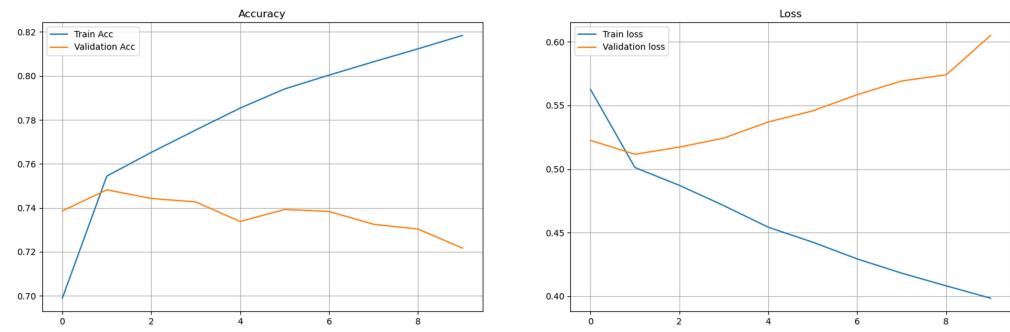
یعنی ابتدا جملات توکنایز شده را به لایه embedding می‌دهیم که می‌تواند one-hot, glove, word2vec باشد. بعد از آن خروجی بازنمایی کلمات را به شبکه عصبی بازگشته که در این بخش RNN است می‌دهیم. در نهایت خروجی state آخر شبکه RNN را از تابع خطی رد می‌کنیم و بعدش از softmax یا sigmoid استفاده می‌کنیم تا خروجی نرمال شود.

البته برای زمانی که بازنمایی one-hot داریم از لایه linear به جای embedding استفاده می‌کنیم. که هر کلمه را از سایز وکب به سایز ۱۵۰ تبدیل می‌کند.

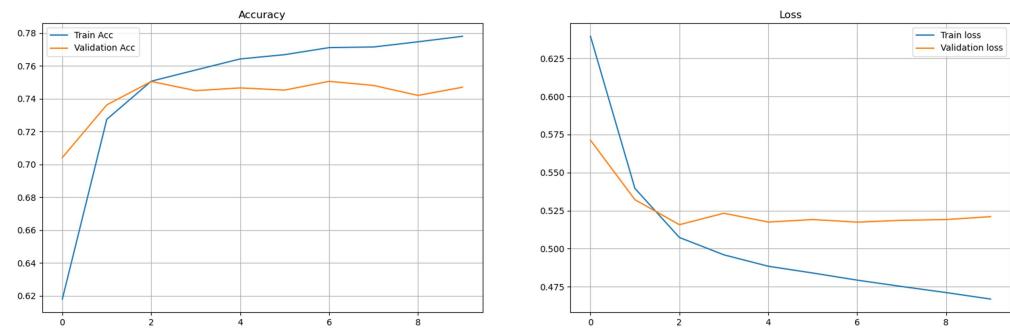
پیکربندی اولیه شبکه:

Batch\_size: 64, optimizer: Adam, learning\_rate: 0.0003,  
embedding\_dimention = 100, hidden\_dimention = 150

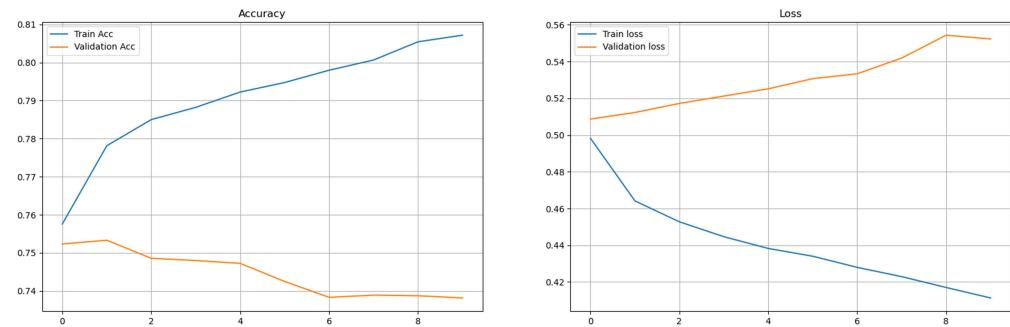
برای فاز forward دیتا را به مدل می‌دهیم. سپس در فاز backward با استفاده از cross entropy loss خروجی مدل را با لیبل واقعی مقایسه می‌کنیم. و بر اساس آن وزن‌های validation و train را آپدیت می‌کنیم. در هر ایپاک accuracy و loss را برای دیتای train و validation حساب می‌کنیم. که در آخر بتوانیم نمودار آن را رسم کنیم. این نمودارها در تعیین هایپرپارامترها بسیار کمک می‌کنند و با توجه به آن‌ها با مقادیر هایپرپارامترها بازی می‌کنیم تا به دقت مناسب برسیم. که نمودار accuracy و loss در شکل‌های زیر آمده است.



شکل ۳. نمودار خطا و دقت برای RNN-onehot



شکل ۴. نمودار خطا و دقت برای RNN-glove

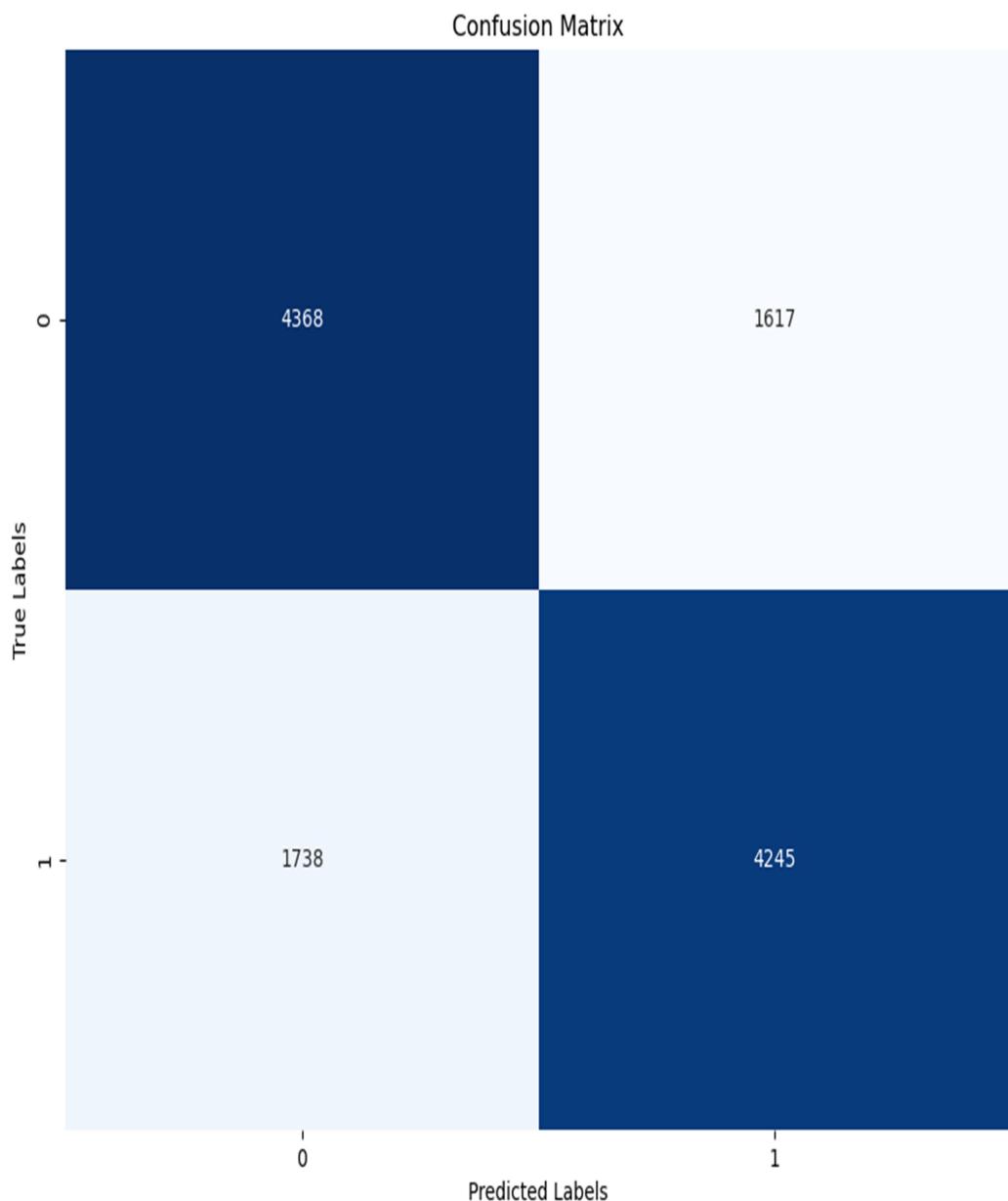


شکل ۵. نمودار خطا و دقت برای RNN-word2vec

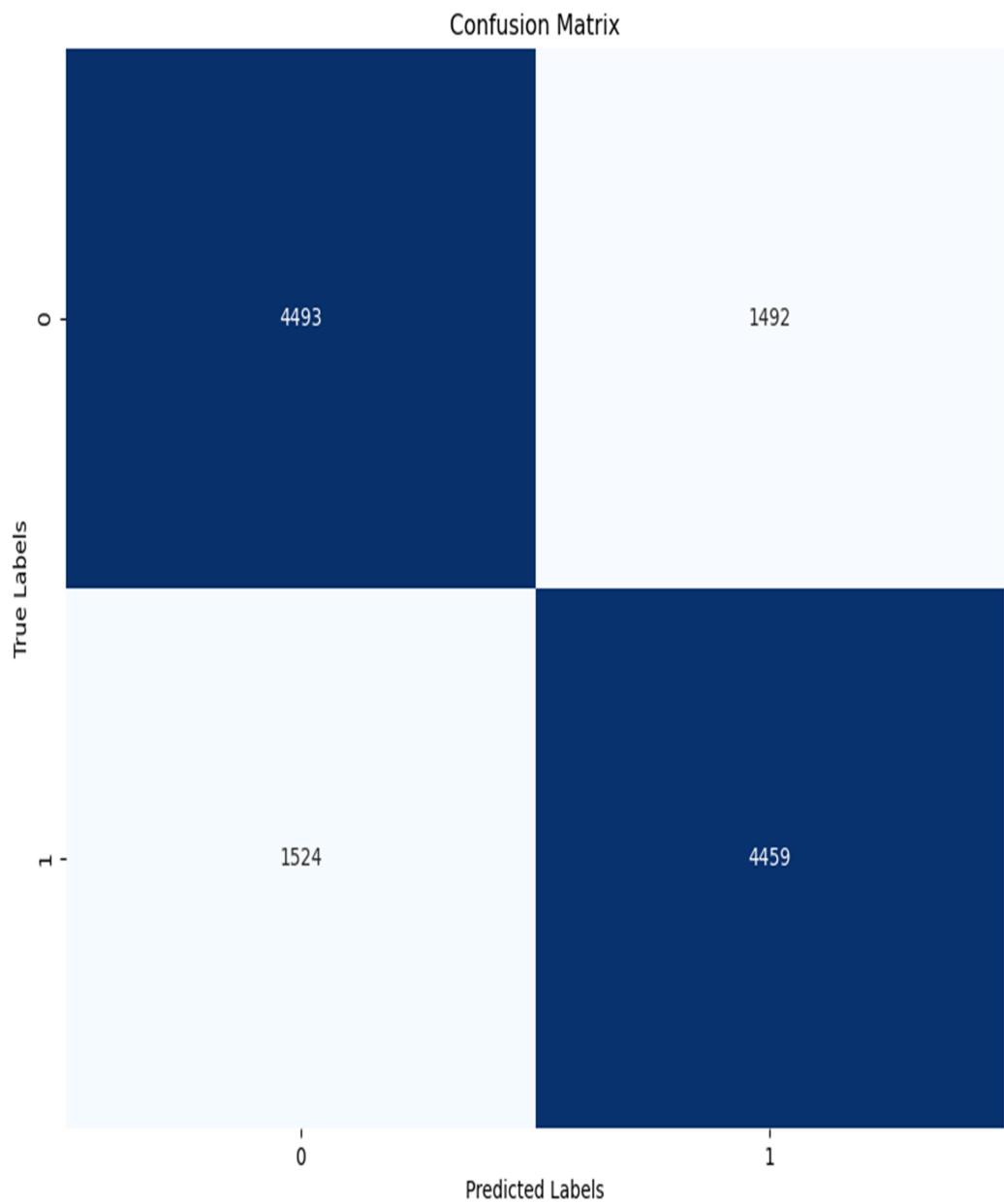
مدل‌ها ابتدا خیلی overfit شدند اما با dropout سعی کردم کمی آن را رفع کنم و تا حدی موفق شدم.

## ۶-۲. ماتریس درهم ریختگی

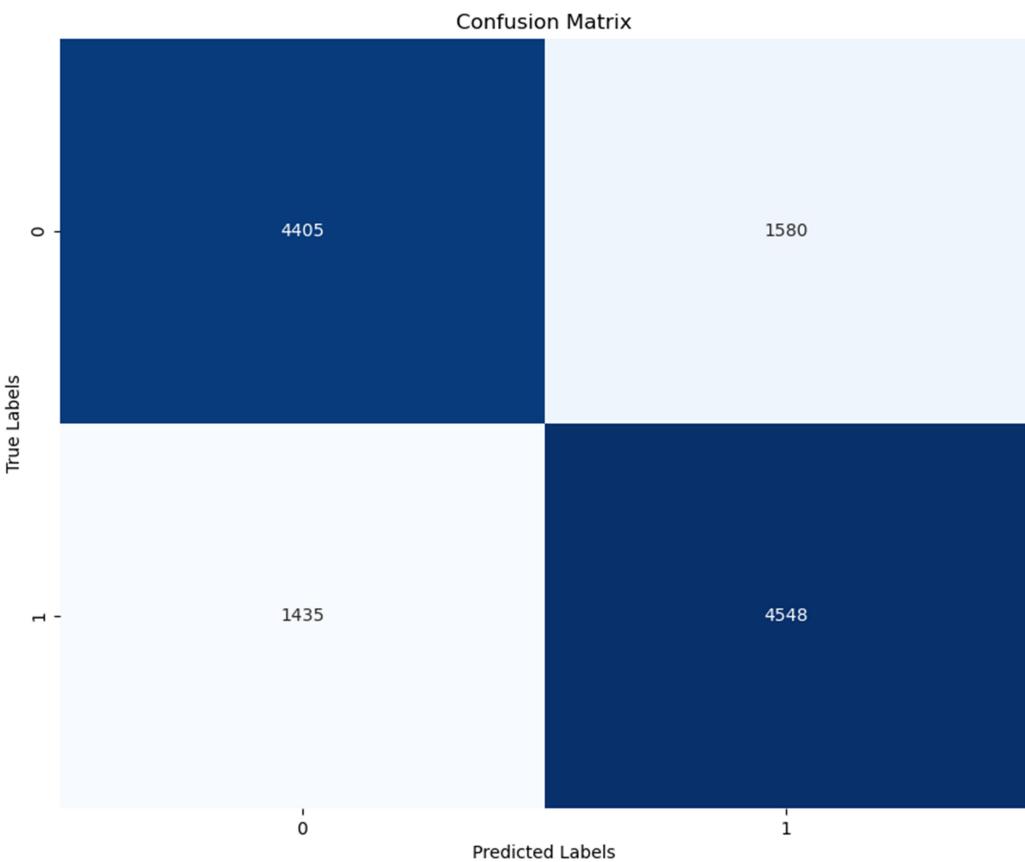
برای هر سه embedding در شکل‌های زیر ماتریس درهم ریختگی آن‌ها آورده شده است. دقیق شود در این مرحله از دیتا test استفاده کردیم نه دیتای validation.



شکل ۶. نمودار درهم ریختگی RNN\_onehot



نمودار ۷. نمودار درهم ریختگی RNN\_glove



نمودار ۸. ماتریس درهم ریختگی RNN\_word2vec

تحلیل:

هر سه مدل نزدیک هم عمل کردند. و اینطوری هم نیست که به یک لیبل بیشتر حساس باشند. تقریباً اشتباهاتشون در هر دو لیبل برابر است.

در ادامه دقیق مدل با هر سه embedding آورده شده است:

embedding	accuracy
One-hot	0.72
Word2Vec	0.75
GloVe	0.75

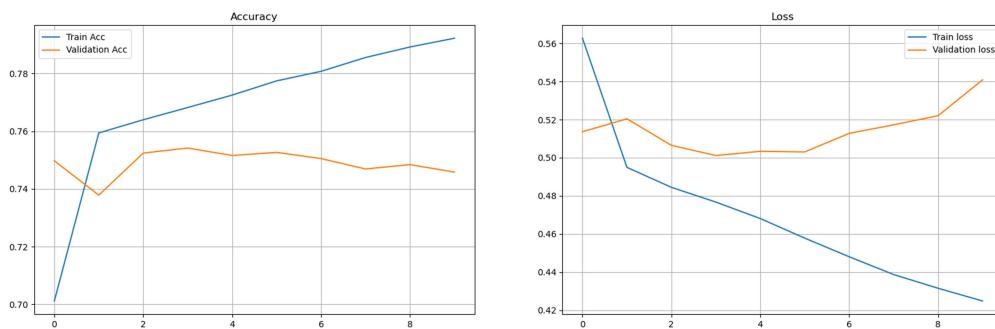
که به نظر می‌رسد one-hot embedding از GloVe و Word2Vec های embedding بهتر هستند. همچنین train آن‌ها هم سریع‌تر است. پس معنای کلمات قبل از شبکه عصبی بازگشتی به مدل کمک کرده است همانطور که انتظار داشتیم.

## ۴. بخش ۲

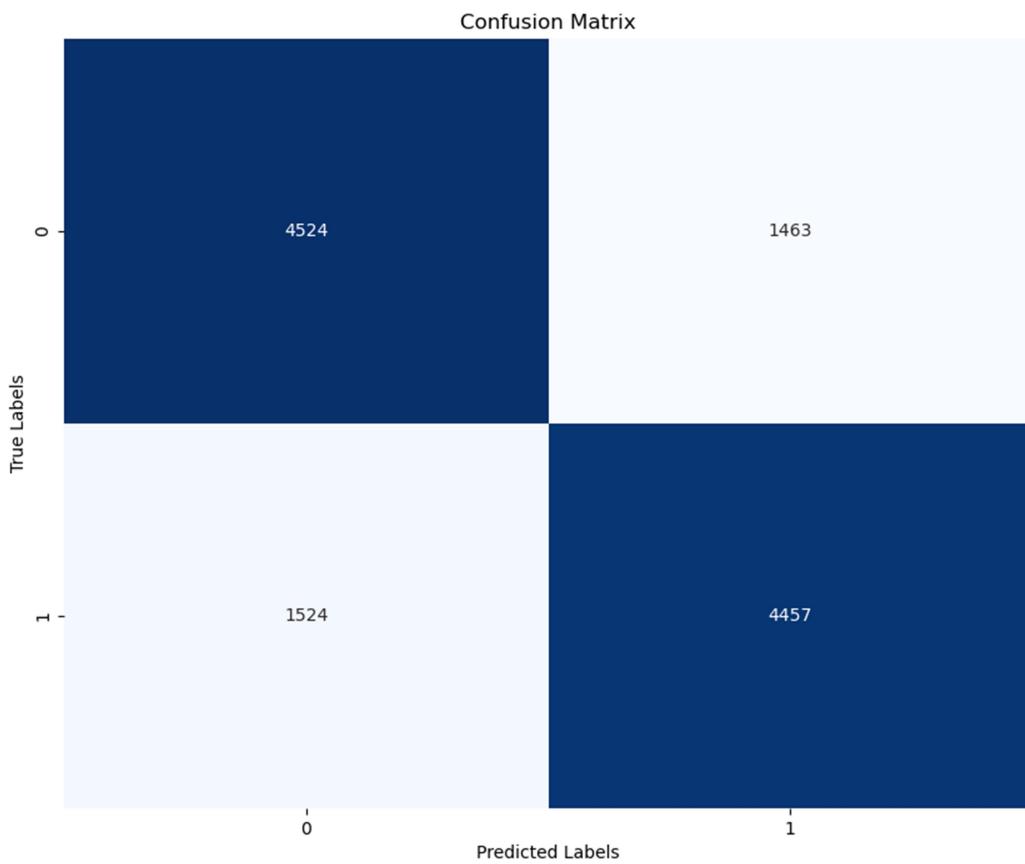
### LSTM – one hot encoding .۱-۳

از همان معماری شکل ۲ استفاده می‌کنیم. و مراحل دقیقاً مانند بخش قبل است. با این تفاوت که در LSTM ما قدرت پردازشی بیشتری به دلیل cell\_state داریم که می‌تواند وابستگی بلند مدت را هم ذخیره کند. مدل را آموزش دادیم و نمودار loss و accuracy و آن در

شکل زیر آورده شده است:



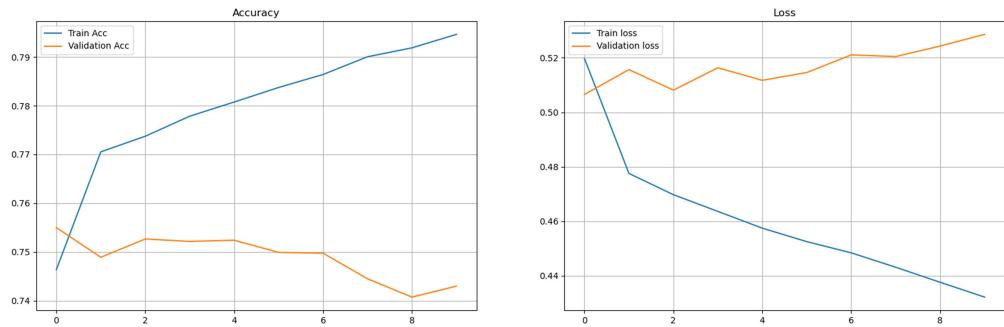
همچنین نمودار confusion matrix آن در شکل زیر آمده است.



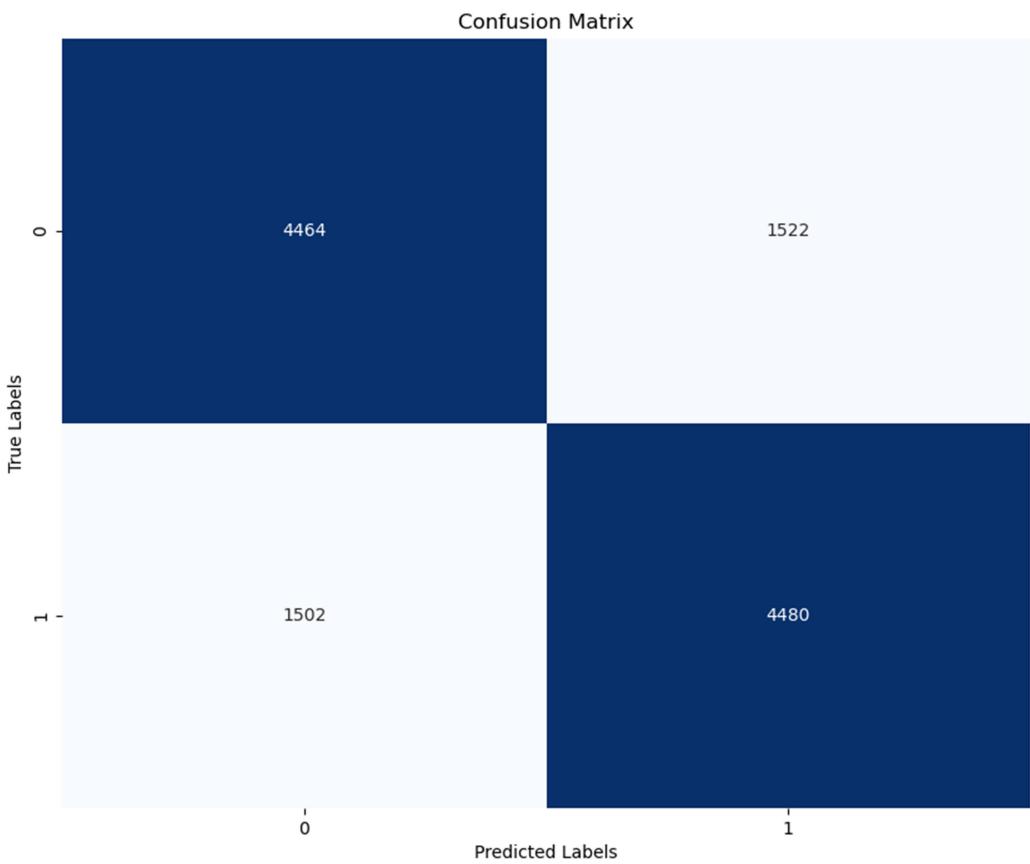
تحلیل: مدل خیلی overfit نکرده است. تا حد خوبی دقت دارد.

## LSTM – glove embedding .۲-۳

مانند بخش قبل فقط از بازنمایی glove کلمات استفاده می‌کنیم. نمودار loss و accuracy در شکل زیر آمده است.



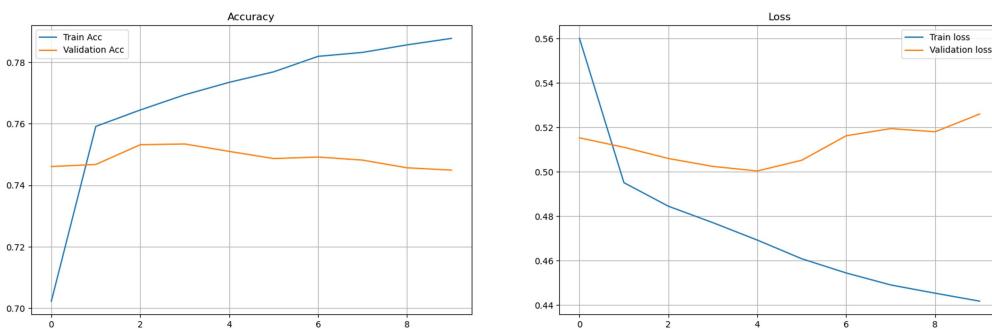
همچنین نمودار confusion matrix آن در شکل زیر آمده است.



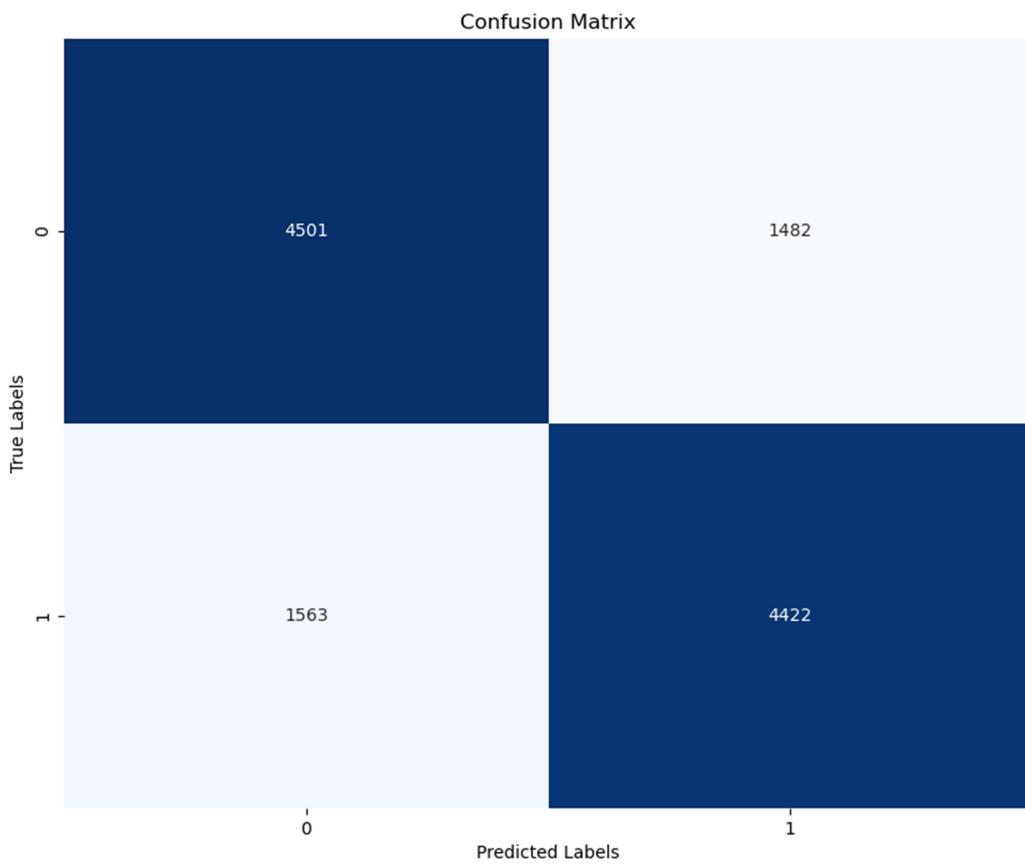
تحلیل: مدل overfit کرده است. دقتش خوب است.

### GRU – one hot encoding .۳-۳

دوباره از معماری مشابه شکل ۲ استفاده کردیم و فقط به جای LSTM از GRU استفاده کردیم. و GRU هم به دلیل داشتن حافظه قدرت پردازشی بیشتری از RNN دارد. نمودار accuracy و loss آن در زیر آورده شده است:



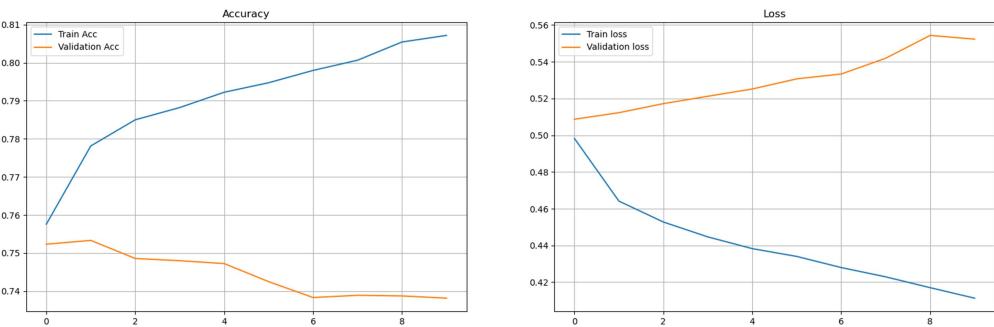
همچنین نمودار confusion matrix آن در شکل زیر آمده است.



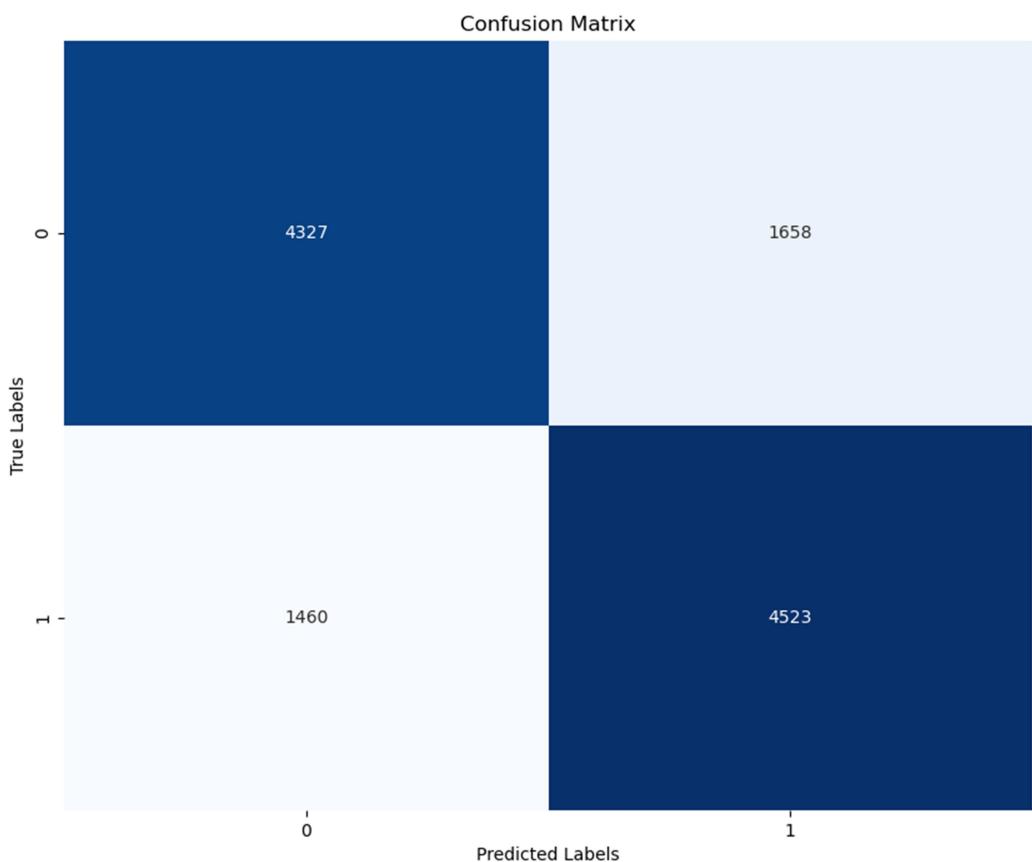
تحلیل: مدل خیلی overfit نکرده است. دقتش خوب است.

#### GRU – glove embedding .۴-۳

مانند بخش قبل فقط از بازنمایی glove کلمات استفاده می‌کنیم. نمودار loss و accuracy در شکل زیر آمده است.



همچنین نمودار confusion matrix آن در شکل زیر آمده است.



تحلیل: مدل overfit کرده است. دقت خوب است.

## ۵-۳. مقایسه LSTM با GRU

- سرعت آموزش: مدل GRU بسیار سریع‌تر از LSTM آموزش دید. و دلیلش هم به نظر این است که با ترکیب کردن گیت‌های forget, input و یکی کردن cell-state و hidden-state تعداد پارامترها کاهش قابل توجهی دارد. برای همین محاسبه گرادیان ساده‌تر می‌شود. و مدل سریع یاد می‌گیرد.
- حافظه: در حافظه هم GRU بهتر است. چون پارامترهای کمتری از LSTM دارد.
- دقیق: در جدول زیر دقیقاً برای embedding‌های متفاوت گزارش شده است.

model	embedding	accuracy
LSTM	One-hot	0.75
	GloVe	0.77
GRU	One-hot	0.75
	GloVe	0.76

همانطور که مشاهده می‌کنید بهترین مدل LSTM با Glove embedding است. دلیلش مشخص است. اول اینکه از بازنمایی با معنی‌تر نسبت به one-hot استفاده می‌کند. دوم اینکه پارامترهای بیشتری نسبت به GRU دارد. برای همین ظرفیت یادگیری آن بیشتر است. البته همین خاصیت باعث overfit شدن شدیدترش نسبت به بقیه مدل‌ها هم شده است.

مدل بعدی GRU با GloVe embedding است که به دلیل بازنمایی با معناش از دو مدل دیگر بهتر عمل می‌کند.

در آخر هم دو مدل که با one-hot encoding آموزش دیده‌اند قرار می‌گیرند. این دو مدل بسیار کندتر از دو مدل دیگر train شدند. دلیلش پردازش سنگین در

لایه feed forward قبل از لایه شبکه عصبی بازگشتی است که کلی ماتریس را باید در هم ضرب کند.