دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)
دانشکده مهندسی کامپیوتر

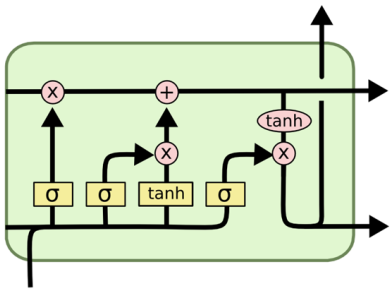
پروژه ششم درس رایانش عصبی

نگارش
سیدمهدی میرفندرسکی
مدرس
دکتر رضا صفابخش
دی ۱۴۰۱

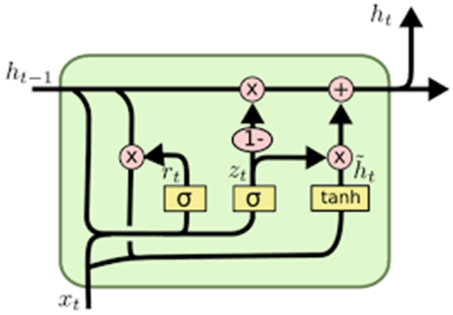
فهرست مطالب

۱ سوال اول - تشریحی

ساختار سلول LSTM و GRU به صورت زیر است:



شکل ۱: معماری LSTM



شکل ۲: معماری GRU

شباهت هر دو نوع شبکه در این است که هر دو با ارائه ساختاری بر RNN وضعیتی از زمان‌های قدیم را حفظ کنند (یک واحد حافظه). نوشتن و خواندن این حافظه توسط شبکه یاد گرفته شود. اما جزییات پیاده‌سازی آن‌ها با هم متفاوت خواهد بود. همانطور که در اشکال ارائه شده مشاهده می‌شود، سلول LSTM دارای سه گیت است (Input، Output و Forget). گیت Forget تعیین‌کننده تغییر یا عدم تغییر وضعیت سلول خواهد بود. گیت Input جریان ورودی را مشخص می‌کند و نهایتا گیت output جریان خروجی بر اساس وضعیت فعلی را کنترل خواهد کرد. اما اساسا معماری و ساختار GRU یک معماری ساده‌تر با بهره‌گیری از LSTM است. دراین معماری دو گیت reset و update وجود دارد. گیت update اهمیت اطلاعات جدید را مشخص می‌کند و در وضعیت سلول جایگزین می‌شود و گیت reset تعیین می‌کند اهمیت اطلاعات قبلی چقد است و آیا باید حفظ شود؟

با توجه به مطالب ذکر شده به نظر می‌رسد که GRU هزینه محاسباتی کمتری داشته باشد. همچنین قابلیت تغییرپذیری آن بیشتر خواهد بود. اما طبیعتا قدرت LSTM بیشتر خواهد بود. از طرفی LSTM احتمال بیش برازشش بیشتر خواهد بود.

پشته‌کردن LSTM: این کار با توجه به اینکه LSTM ها بر روی داده‌های توالی کار می‌کنند، به این معنی است که با افزودن لایه‌ها سطوح انتزاعی مشاهدات ورودی را در طول زمان اضافه می‌کند. در واقع، مشاهدات را در طول زمان تکه تکه می‌کند یا مشکل را در مقیاسه‌های زمانی مختلف نشان می‌دهد. به بیان دیگر گفته‌شده، این رویکرد به طور بالقوه به حالت پنهان در هر سطح اجازه می‌دهد تا در مقیاس زمانی متفاوتی عمل کند. پشته‌کردن GRU: مطالب بیان شده برای LSTM برای این قسمت نیز صادق خواهد بود.

اما باید توجه شود که پشته‌کردن LSTM مدل را بیش از اندازه پیچیده خواهد کرد. این بدان خاطر خواهد بود که سلول LSTM به تنهایی پیچیده است و با اینکار آموزش شبکه سخت‌تر خواهد شد زیرا درحال حاضر شبکه‌ای خواهیم داشت که هر لایه آن به اندازه کافی پیچیده خواهد بود. اما در معماری GRU چون معماری ساده‌تری دارد آموزش مقداری راحت‌تر خواهد بود.

۲ سوال دوم – تشریحی

گرادیان در این سلول شامل بردار فعال‌سازی‌های دروازه فراموشی است که به شبکه اجازه می‌دهد تا مقادیر گرادیان‌ها را در هر مرحله زمانی با استفاده از به‌روزرسانی‌های پارامتر مناسب دروازه فراموشی کنترل کند. وجود فعال‌سازی‌های دروازه فراموشی به سلول این امکان را می‌دهد تا در هر مرحله زمانی تصمیم بگیرد که اطلاعات خاصی فراموش نشود و پارامترهای مدل را متناسب با آن به‌روزرسانی کند. پس می‌توان برای اینکه گرادیان ناپدید نشود، یک به روزرسانی برای پارامتر دروازه فراموشی در مرحله زمانی بعدی پیدا کنیم که مشتق جزئی خطا نسبت به وزن در زمان بعدی صفر نشود. پس در نهایت وجود پارامتر دروازه فراموشی این امکان را در هر زمانی فراهم می‌آورد. و در نهایت مشکل از بین رفتن گرادیان حل می‌شود.

۳ سوال سوم – تشریحی

بله، شبکه های حافظه کوتاه مدت بلند را می‌توان به صورت موازی و توزیع شده آموزش داد. آموزش موازی شامل استفاده از چندین منبع پردازشی برای آموزش مدل به طور همزمان است که می‌تواند روند آموزش را سرعت بخشد. برای آموزش موازی کارهای زیر می‌تواند صورت گیرد:

- موازی‌سازی داده‌ها (تقسیم داده‌ها به دسته‌های مختلف و پردازش هر یک در ماشین متفاوت)
- موازی‌سازی مدل (تقسیم مدل به لایه‌های مختلف و پردازش هریک در ماشینی مجزا)
- ترکیب روش‌های فوق

با این کار می‌توان مدل‌های بزرگتری و یا مدلی با داده‌های بیشتری آموزش داد. همچنین سرعت آموزش نیز افزایش می‌یابد. البته برای این کار نیاز به ماژول‌های جانبی خواهد بود (برای مدیریت).

۴ سوال اول

پیش‌پردازش‌های اعمالی بر متن انواع گوناگونی دارد. در این بخش به همان موارد ذکرشده در صورت پروژه بسنده شد. همانطور که در ادامه مشاهده می‌شود، ابتدا علائم نگارشی را حذف کرده، سپس تمام حروف را کوچک و فواصل اضافی را حذف می‌کنیم. سپس کار را با حذف تگ‌های html و لینک‌ها ادامه می‌دهیم. در نهایت نیز از متن باقی کلمات پرتکرار را حذف و متن را تبدیل به لیستی از کلمات می‌کنیم.

```
test_df['comment'] = test_df['comment'].apply(lambda x: remove_punctuation(x))
test_df['comment'] = test_df['comment'].apply(lambda x: x.lower())
test_df['comment'] = test_df['comment'].apply(lambda x: remove_whitespace(x))
test_df['comment'] = test_df['comment'].apply(lambda x: cleanhtml(x))
test_df['comment'] = test_df['comment'].apply(lambda x: remove_urls(x))
test_df['comment'] = test_df['comment'].apply(lambda x: remove_stopwords(x))
test_df
```

شکل ۳: پیش‌پردازش متن

۵ سوال دوم

one hot encoding برای تعبیه کلمات توصیه نمی‌شود، زیرا اساسا طول هر بردار آن به اندازه طول کل دیکشنری خواهد بود. یعنی ما ورودی‌هایی خواهیم داشت که به شدت sparse خواهند بود. این اتفاق هزینه محاسباتی مدل را بسیار افزایش خواهد داد.

Word2Vec یک شبکه عصبی کم عمق و دو لایه است که برای بازسازی زمینه‌های زبانی کلمات آموزش داده شده است. این روش با استفاده از مجموعه بزرگی از کلمات به عنوان ورودی، یک فضای برداری را تولید می‌کند. به بیان دیگر هر کلمه منحصر به فرد به یک بردار متناظر در فضا نگاشت می‌شود. اما این بردارهای کلمه در فضای برداری به گونه‌ای قرار می‌گیرند کلمات مشابه نزدیک به هم باشند. این روش به صورت، مدل Continuous Bag-of-Words (CBOW) و مدل Skip-Gram عرضه می‌شود. به صورت دقیقتر Word2Vec یک شبکه عصبی ساده با یک لایه پنهان است و مانند همه شبکه‌های عصبی دارای وزن است و در طول آموزش هدفش تنظیم آن وزن‌ها برای کاهش یک تابع هزینه است. با این حال، Word2Vec قرار نیست برای کاری که در آن آموزش داده شده است استفاده شود، در عوض، فقط وزن‌های پنهان آن درنظر گرفته می‌شود.

در رابطه با پارامترهایی که باید تعیین شوند: اندازه پنجره هرچه بیشتر باشد به معنا بیشتر توجه می‌کند اما از طرفی زمان آموزش بیشتر خواهد شد. ما در اینجا اندازه پنجره را ۵ درنظر گرفتیم (میانه). همچنین چون مجموعه داده ما بسیار بزرگ نیست، لازم نیست از مقادیر بزرگ برای اندازه بردار ویژگی استفاده کنیم. مقدار ۱۰۰ برا آن در نظر گرفته شد.

۶ سوال سوم

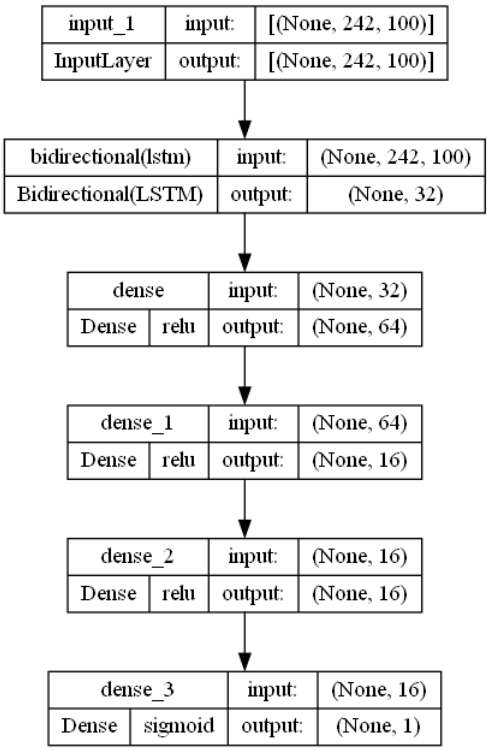
این قسمت نکته خاصی ندارد، جز اینکه از روش دوم استفاده شد و حد آستانه ۹۰ درصد داده‌ها تعیین شد. بدین ترتیب جملات با حدود طول ۲۴۲ درنظر گرفته شدند.

۷ سوال چهارم

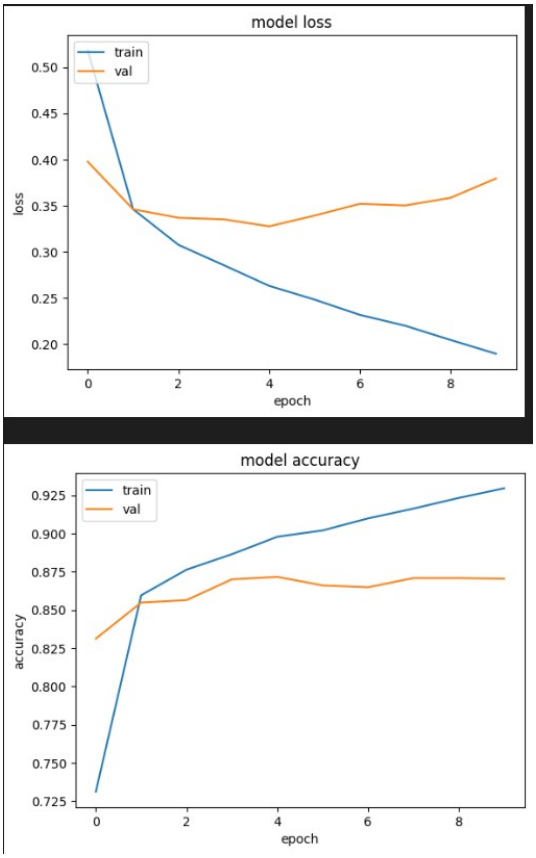
برای سعی و خطا این قسمت شبکه‌های زیر بررسی شد و نتایج آن در ادامه قابل مشاهده است. در این قسمت در ابتدا سعی شد اکثر حالت‌های ممکن بررسی شوند (معماری دوسویه، پشته‌ای و ...) اما حجم داده‌ها بسیار زیاد بود به همین دلیل به اجرای ۴ نوع شبکه

بسنده شد (هر نوع سلول ۲ شبکه). برای آموزش شبکه‌ها مجبور به ذخیره مدل، داده‌های آموزشی و داده‌های آزمون شدیم. همچنین با توجه به نوع مسئله ترجیح داده شد که از معماری دو طرفه استفاده شود. همچنین برای هر سلول یک شبکه با یک لایه و یک شبکه با دو لایه ایجاد شد. برای لایه‌های پنهان نیز از ۴ لایه استفاده کردیم (با لایه خروجی). این شبکه‌ها با ۱۰ ای‌پاک آموزش داده شدند.

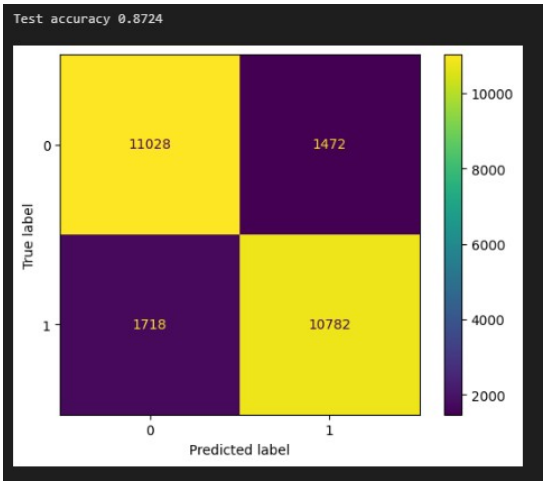
معماری چهار شبکه ذکر شده به همراه نمودارها و همچنین ماتریس درهم ریختگی آن در ادامه مشاهده می‌شود.



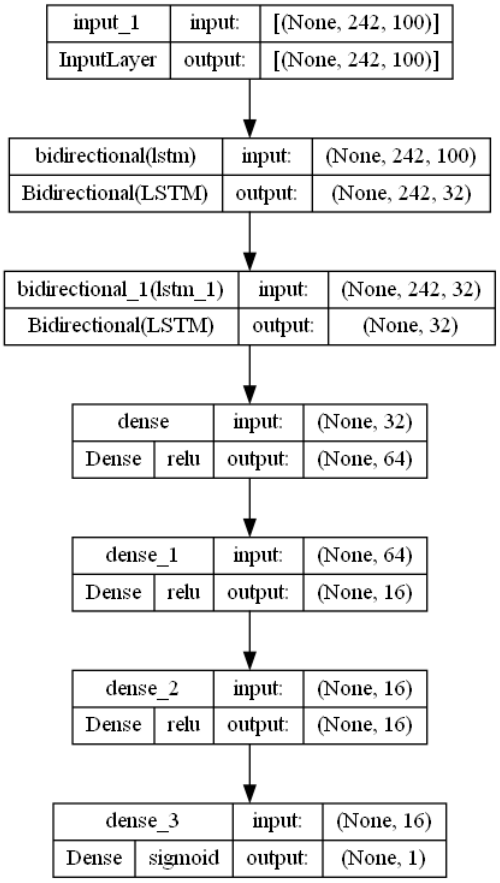
شکل ۴: LSTM-[16]-[64, 16, 16]



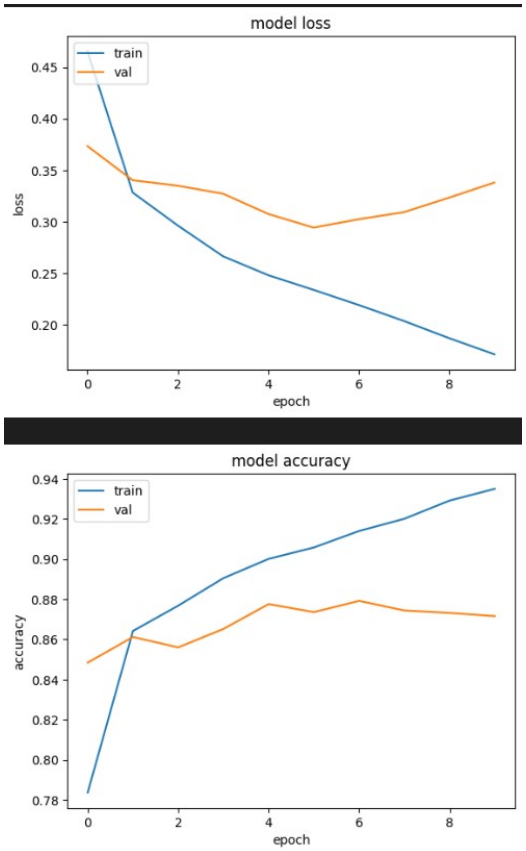
شکل ۵: LSTM-[16]-[64, 16, 16]



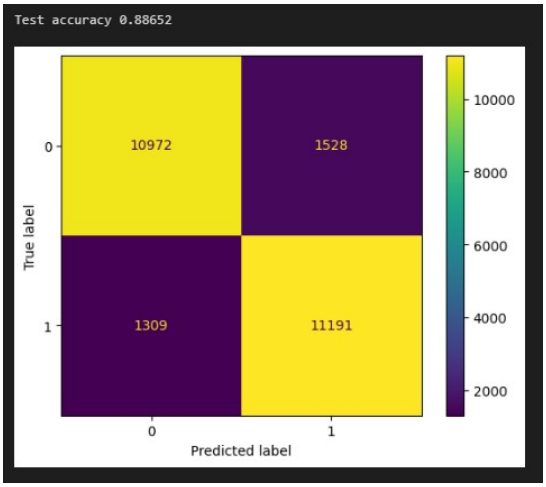
شکل ۶: LSTM-[16]-[64, 16, 16]



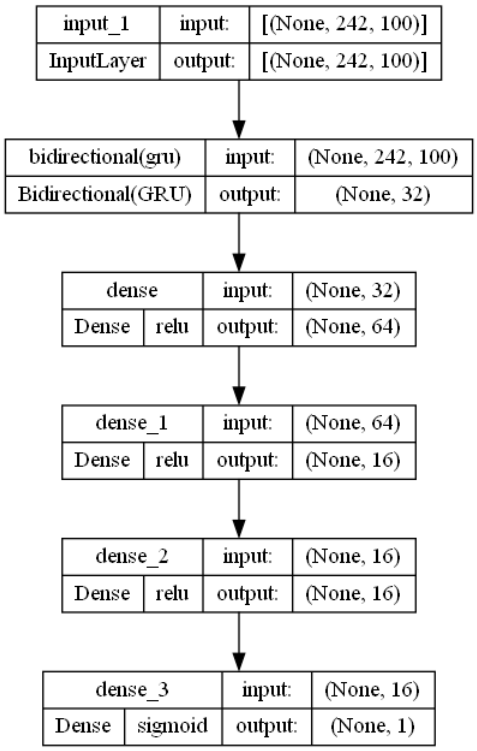
شکل ۷: LSTM-[16, 16]-[64, 16, 16]



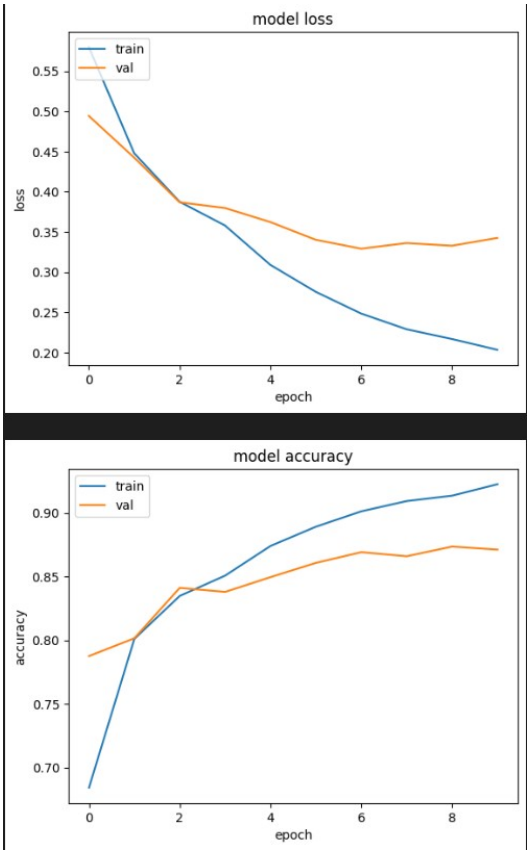
شکل ۸: LSTM-[16, 16]-[64, 16, 16]



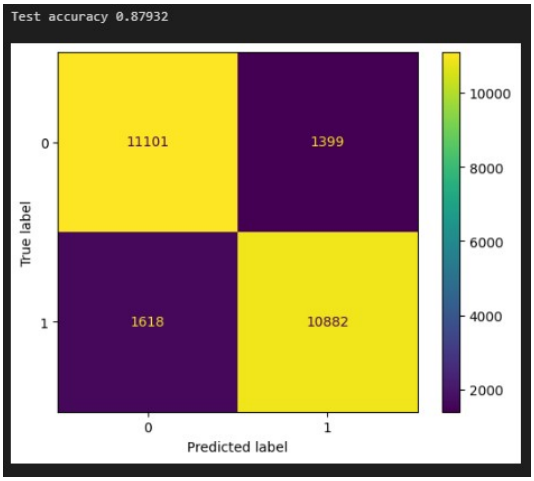
شکل ۹: LSTM-[16, 16]-[64, 16, 16]



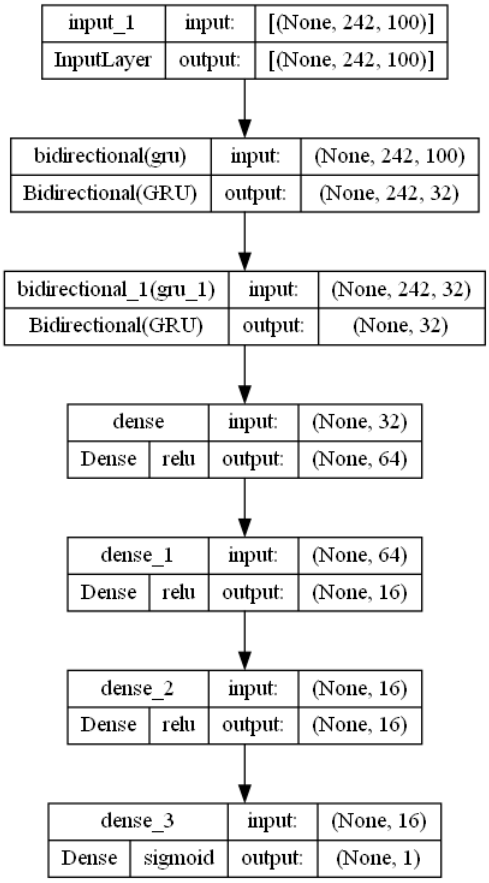
شکل ۱۰: GRU-[16]-[64, 16, 16]



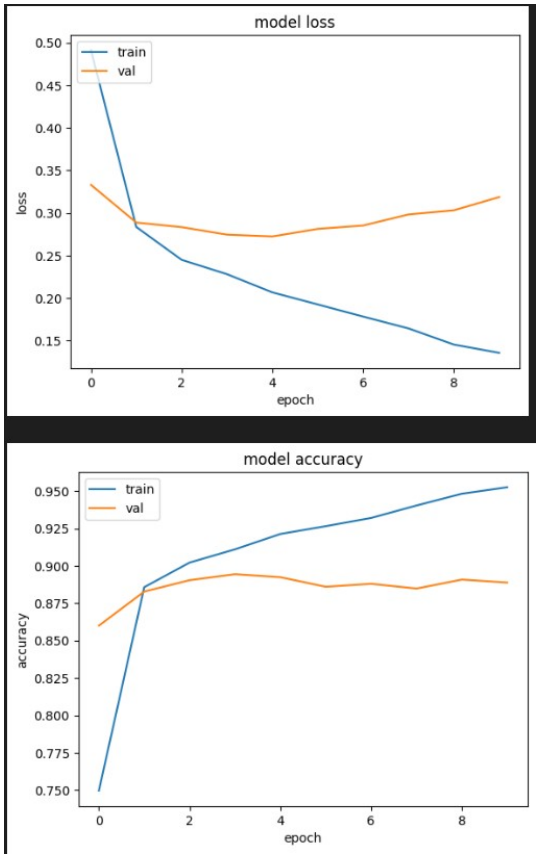
شکل ۱۱: GRU-[16]-[64, 16, 16]



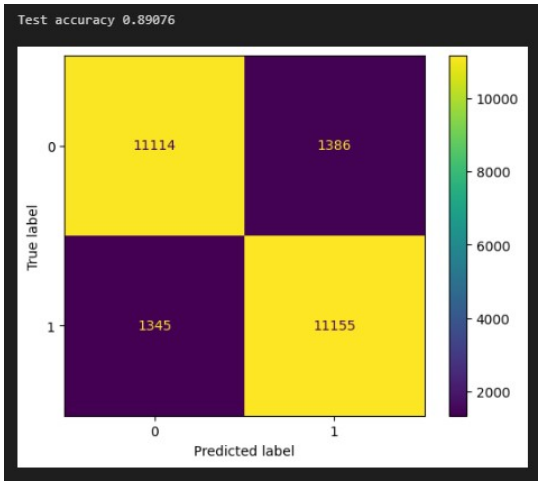
شکل ۱۲: GRU-[16]-[64, 16, 16]



شکل ۱۳: GRU-[16, 16]-[64, 16, 16]



شکل ۱۴: GRU-[16, 16]-[64, 16, 16]

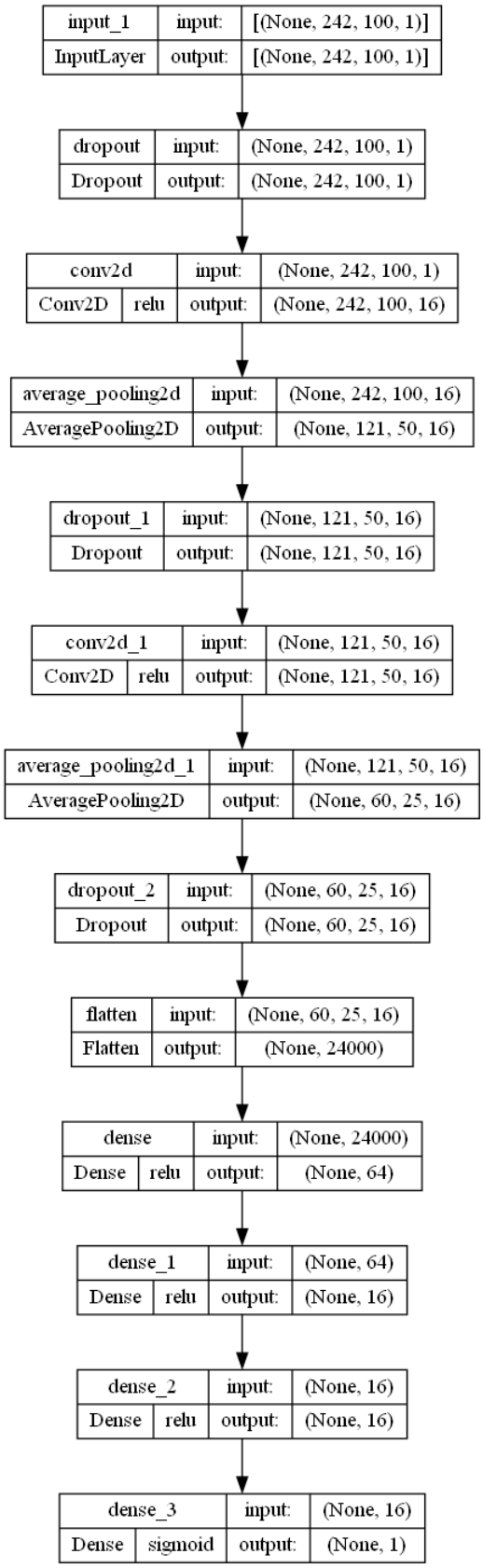


شکل ۱۵: GRU-[16, 16]-[64, 16, 16]

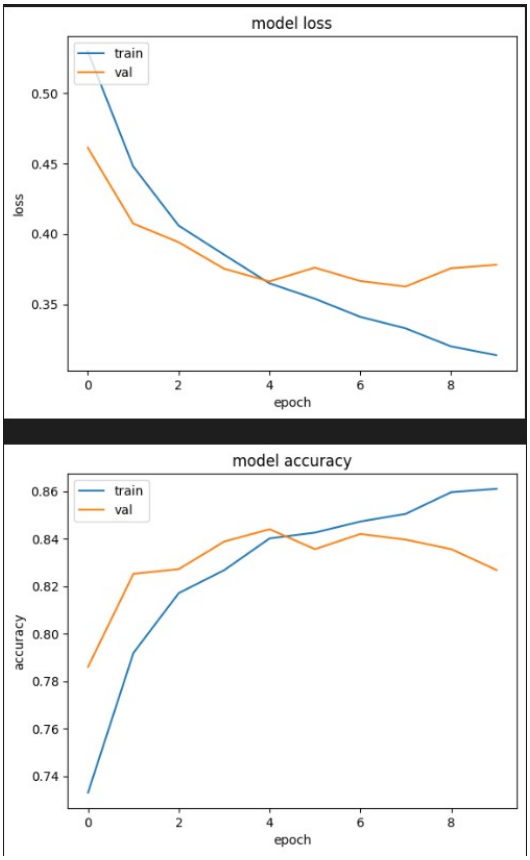
همانطور که مشاهده می‌شود بهترین معماری مربوط به شبکه با دو لایه GRU است که دقت داده‌های آزمون آن نزدیک به ۹۰ درصد بدست آمد.

۸ سوال اضافی

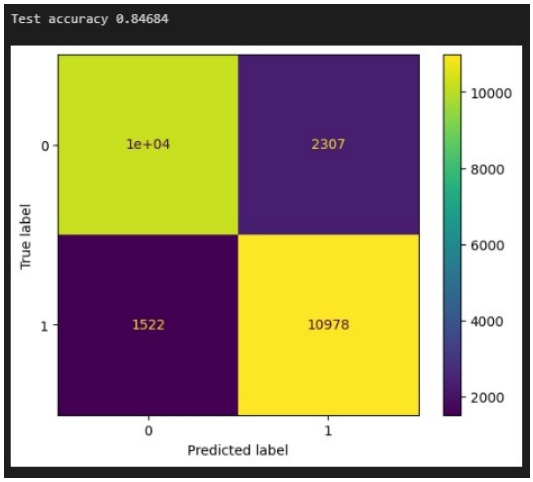
در این بخش با دو لایه کانولوشنی و ۴ لایه پنهان شبکه به سرعت آموزش داده شد و دچار بیش‌برازش شد (۹۹ درصد برای آموزش و نزدیک ۸۰ درصد آزمون). به همین خاطر چندین لایه drop out اضافه شد و نتایج زیر بدست آمد.


input_1 | input: | [(None, 242, 100, 1)]
InputLayer | output: | [(None, 242, 100, 1)]
dropout | input: | (None, 242, 100, 1)
Dropout | output: | (None, 242, 100, 1)
conv2d | input: | (None, 242, 100, 1)
Conv2D | relu | output: | (None, 242, 100, 16)
average_pooling2d | input: | (None, 242, 100, 16)
AveragePooling2D | output: | (None, 121, 50, 16)
dropout_1 | input: | (None, 121, 50, 16)
Dropout | output: | (None, 121, 50, 16)
conv2d_1 | input: | (None, 121, 50, 16)
Conv2D | relu | output: | (None, 121, 50, 16)
average_pooling2d_1 | input: | (None, 121, 50, 16)
AveragePooling2D | output: | (None, 60, 25, 16)
dropout_2 | input: | (None, 60, 25, 16)
Dropout | output: | (None, 60, 25, 16)
flatten | input: | (None, 60, 25, 16)
Flatten | output: | (None, 24000)
dense | input: | (None, 24000)
Dense | relu | output: | (None, 64)
dense_1 | input: | (None, 64)
Dense | relu | output: | (None, 16)
dense_2 | input: | (None, 16)
Dense | relu | output: | (None, 16)
dense_3 | input: | (None, 16)
Dense | sigmoid | output: | (None, 1)

شکل ۱۶: cnn



شکل ۱۷: cnn



شکل ۱۸: cnn

همانطور که مشاهده می‌شود با تست یک شبکه کانولوشنی نتیجه نزدیکی به شبکه‌های بزگشتی گرفته شد. اما همچنان شبکه‌های بازگشتی با حداقل ۵ درصد بهتر عمل کردند.