به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین چهارم

نام و نام خانوادگی	مریم دادخواه – جواد سراج
شماره دانشجویی	810101186 - 810101151
تاریخ ارسال گزارش	1402.03.03

فهرست

شکل‌ها

جدول‌ها

پاسخ 1. توصیف عکس

در این سوال به بررسی و آموزش شبکه‌ای برای تولید کپشن از تصاویر ورودی می‌پردازیم. این مدل از resnet 18 برای استخراج ویژگی از تصویر استفاده می‌کند و یک لایه LSTM برای تولید کپشن استفاده خواهد کرد. در ادامه به جزییات این شبکه و دیتاست مورد استفاده می‌پردازیم.

۱-۱. پیش پردازش داده‌ها

ما ابتدا داده‌ها را با استفاده از API کگل دانلود کردیم. سپس مسیرهای تصویر و کپشن‌ها را در یک دیتافریم pandas قرار دادیم. هر نمونه دیتاست شامل یک تصویر و 5 کپشن است که توسط افراد داده شده است. نمونه ای از تصویر و کپشن‌های مربوط به آن در شکل زیر داده شد است.

شکل 1. نمونه تصویر و کپشن‌های آن در دیتاست

در ادامه به پردازش متن‌ها می‌پرداریم. برای پردازش داده ها اولین پردازش، ابتدا داده‌ها را توکنایز میکنیم و به کلمات تبدیل می‌کنیم. در ادامه، بعد کلمات را lower case میکنیم. سپس Vocabulary را می‌سازیم. همچنین علائم نگارشی موجود در متن نیز حذف شده اند. برای ساختن Vocabulary، دیکشنری می‌سازیم و به ازای هر کلمه تعداد تکرار آن محاسبه میکنیم و اگر تعداد تکرار کلمه‌ای از حد آستانه‌ای[1] کمتر باشد آن داده را از Vocabulary حذف می‌کنیم. همچنین یک lookup table تشکیل میدهیم و به هر کلمه ایندکس نسبت می‌دهیم. در ابتدای کار حداستانه 5 در نظر گرفته شده است.

0:"<PAD>",1:"<SOS>",2:"<EOS>",3:"<UNK>" را نیز به Vocabulary اضافه می‌کنیم که به مشخص شدن ابتدا و انتهای جمله و مقابله با کلمات ناشناخته پیش‌بینی شد توسط مدل کمک می‌کند. همچنین به دلیل این که تعداد کلمات در جملات متفاوت است از padding برای یکسانسازی طول جملات استفاده می‌کنیم. برای padding، بر اساس بزرگ‌ترین جمله در یک Minibatch تمام جملات را pad میکنیم.

برای پردارش تصاویر ابتدا تصاویر ابعاد تمام تصاویر را به ابعاد 224×224 تبدیل کردیم تا ورودی به استاندارد لازم برای شبکه ResNet برسد. در نهایت مجموع دیتاست ما شامل 40455 کامنت و 40455 تصویر است که تعداد یونیک تصاویر 8091 است. یعنی به ازای هر تصویر، 5 کامنت موجود داریم.

2-۱. معماری شبکه (Encoder و Decoder)

شبکهٔ نامبرده در مقاله دارای ساختار Encoder-Decoder است. در بخش Encoder یک شبکه Resnet18 برای، تبدیل تصویر ورودی به بردار ویژگی استفاده شده است. این شبکه نقش استخراج بردار ویژگی برای ورود به بخش Decoder را بر عهده دارد. خروجی ResNet را از یک لایه Linear عبور می‌دهیم تا به ابعاد embediing متناسب با بخش Decoder برسانیم.

دربخش Decoder ابتدا یک لایه Embedding قرار میگیرد که ابعاد آن embeddingsize×vocabsize است و وظیفه آن Embedding هر کلمه به بردار ویژگی آن کلمه است. سپس embedding کلمات درون کامنت را با بردار ویژگی تصویر کانکت میکنیم و این تنسور را به بخش LSTM می‌دهیم. LSTM مطابق با خواسته‌های صورت سوال ورودی embedding size دارد و تعداد hidden size = 256 است و تعداد لایه‌های LSTM هم 1 در نظر گرفته شده است. سپس خروجی LSTM از یک لایه خطی (Fully connected) عبور داده می‌شود و خروجی این لایه به ابعاد Vocab size است تا بتوان کلمه موردنظر را پیشبینی کرد.

[1] Threshold

در بخش نهایی هدف تخمین یک کپشن مناسب است. برای این کار دقت می‌کنیم که فقط تصویر به عنوان ورودی داریم و هیچ متنی وجود ندارد و متن توسط شبکه تولید می‌شود. برای تولید متن توسط شبکه، بردار ویژگی embed شده تصویر را به لایه LSTM که در بخش قبل توضیح دادیم می‌دهیم و سپس LSTM یک بردار ویژگی تولید میکند که این بردار را به لایه fully connected می‌دهیم و یک بردار به ابعاد vocab size به ما خروجی میدهد که با argmax میتوان کلمه تخمینی با بیشترین احتمال را تخمین زد. سپس کلمه تخمین زده شده به عنوان ورودی برای تخمین کلمه بعدی استفاده می‌شود. اما توجه میکنیم در این مرحله ما ایندکس کلمه تخمینی را داریم و باید این ایندکس به لایه embedding داده شود تا بردار embedding آن کلمه استخراج شود. دو شرط مهم در این بخش رعایت می‌شود که در ادامه به آن می‌پرداریم.

1- ابتدا حلقه شرطی را طوری قرار میدهیم تا تعداد کلمات تخمینی از حدی بیشتر نشود. مثلا حداکثر برای هر تصویر 20 کلمه predict کنیم. (ماکسیمم طول کپشن تولیدی)
2- شرطی قرار می‌دهیم که هر جا کلمه "<EOS>" تولید شد همان لحظه حلقه تولید کپشن را متوقف کنیم.

پارامترهای شبکه و هایپرپارامترها

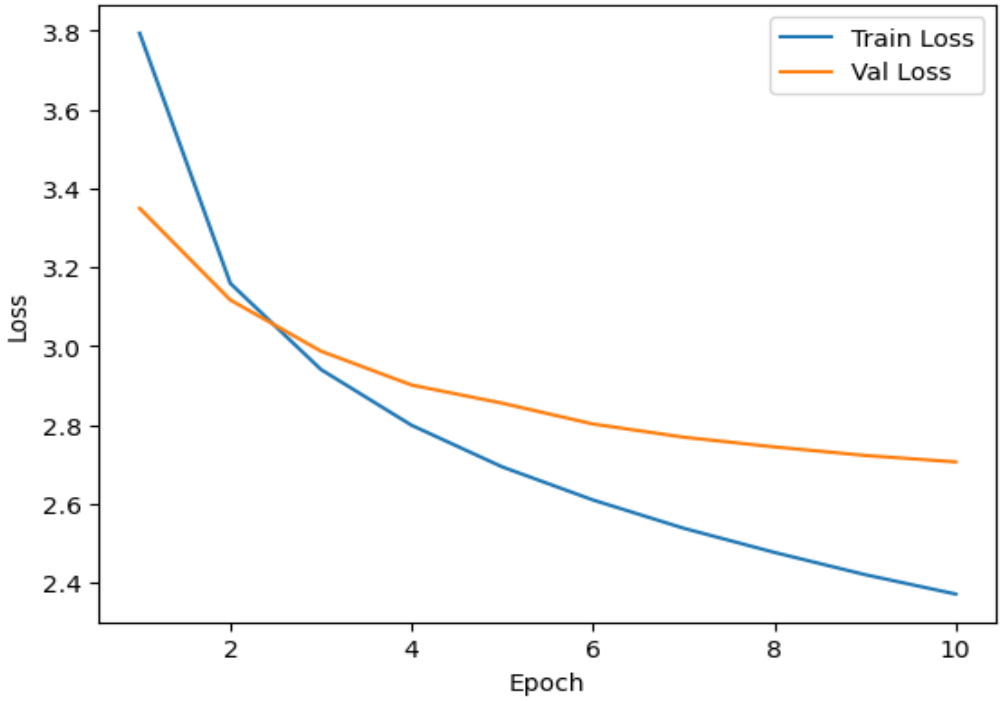
برای آموزش شبکه از تابع هزینه، cross entropy استفاده میکنیم. برای بهینه سازی، بهینه ساز Adam و با نرخ یادگیری 0.0001 را به کار می‌گیریم. Embedding Size برای هر کلمه را 200 در نظر گرفتیم و hidden size برای LSTM را 256 در نظر گرفتیم.

برای رسیدن به مدل مناسب، چند نرخ یادگیری متفاوت و embedding size های متفاوت مورد بررسی قرار گرفت تا بهینه‌ترین پارامترها برای ترین شبکه را انتخاب کنیم.

همچنین مطابق مقاله، ما برای لایه LSTM نرخ dropout = 0.30 در نظر گرفتیم. البته در مقاله dropout= 0.5 قرار داده شد که برای دیتاست ما مقدار 0.3 به نتایج بهتری رسید.

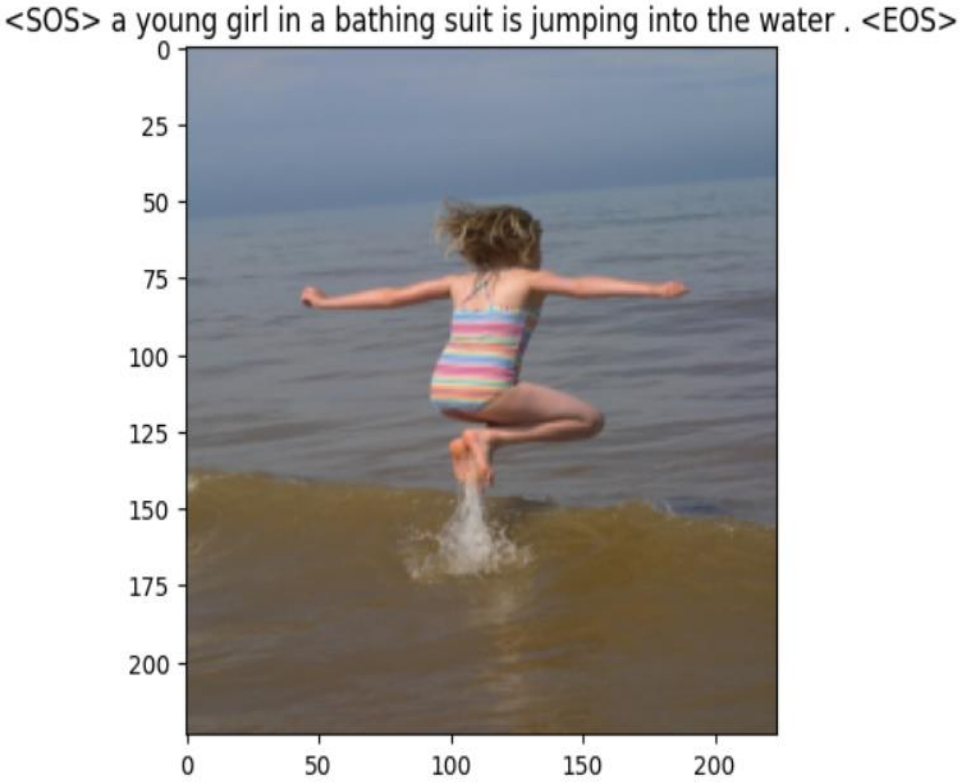
3-۱. پیاده سازی شبکه Decoder-Encoder با وزن‌های فریزشده شبکه ResNet18

با توجه به خواسته‌های این پروژه، ابتدا وزن‌های Pretraibed شده مدل ResNet18 که به عنوان Encoder به کار گرفته شده است را فریز می‌کنیم. (به جز لایه آخر). سپس مدل را روی دیتاها ترین می‌کنیم. قطعه کد زیر برای فریز کردن وزن‌ها به کار گرفته می‌شود. نمودار تابع هزینه برای این حالت در زیر آورده شده است.

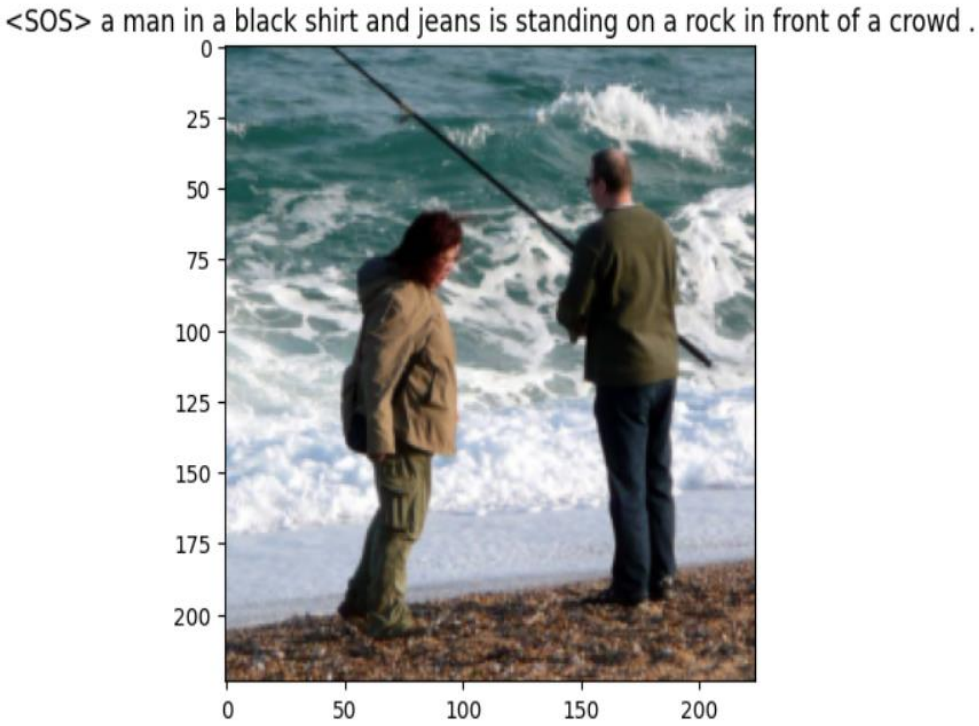


شکل 2. نمودار کاهش تابع هزینه بر روی داده‌های آموزشی و داده‌های اعتبارسنجی- وزن‌های فریز شده **ResNet18**

شکل 3. نمونه تصویر و کپشن تولید شده - **freezed ResNet 18**



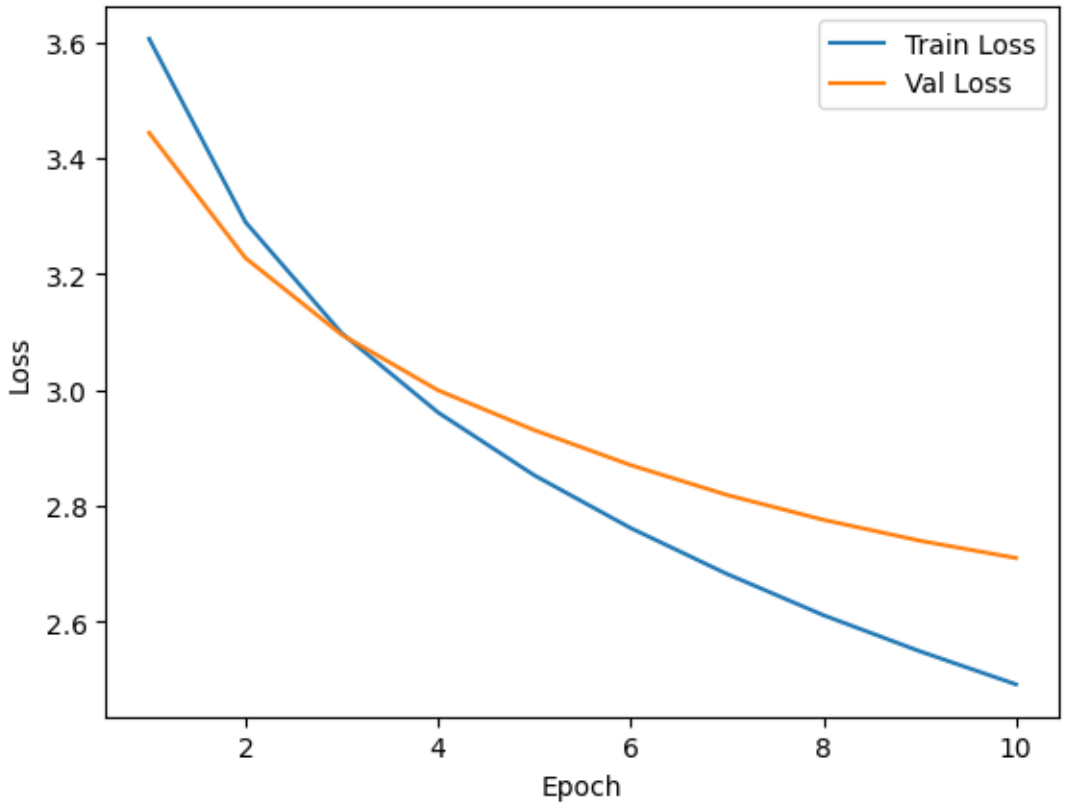
شکل 4. نمونه تصویر و کپشن تولید شده - **freezed ResNet 18**



شکل 5. نمونه تصویر و کپشن تولید شده - **freezed ResNet 18**

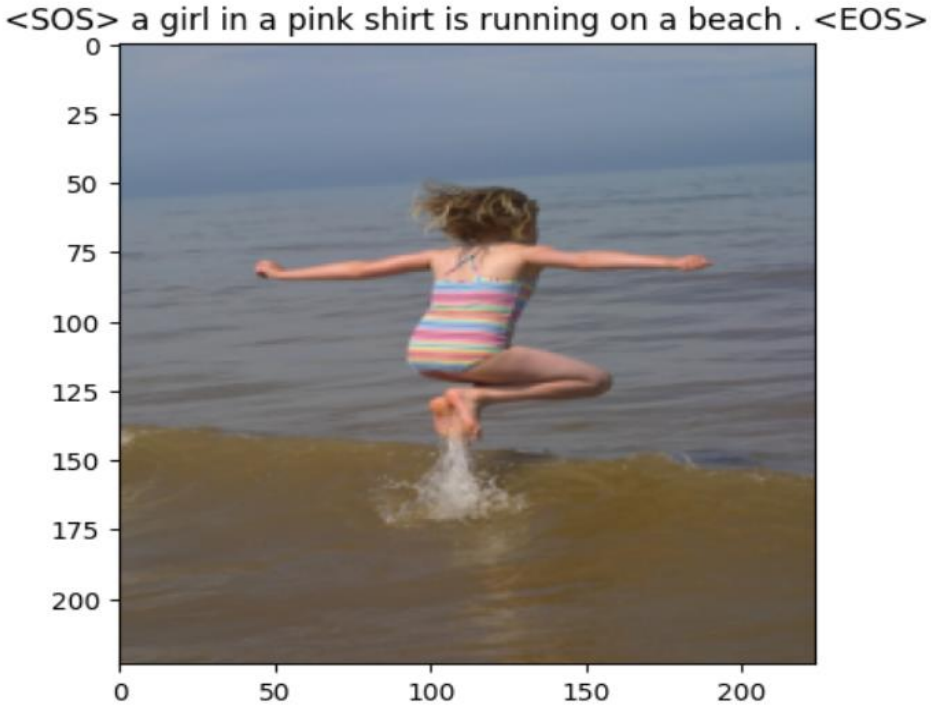
4-۱. پیاده سازی شبکه Decoder-Encoder با شبکه ResNet18 با وزن‌های trainable

در این بخش مدل Resnet18 با قابلیت trainable بودن تمامی وزن‌ها برای آموزش روی دادگان استفاده می‌شود.

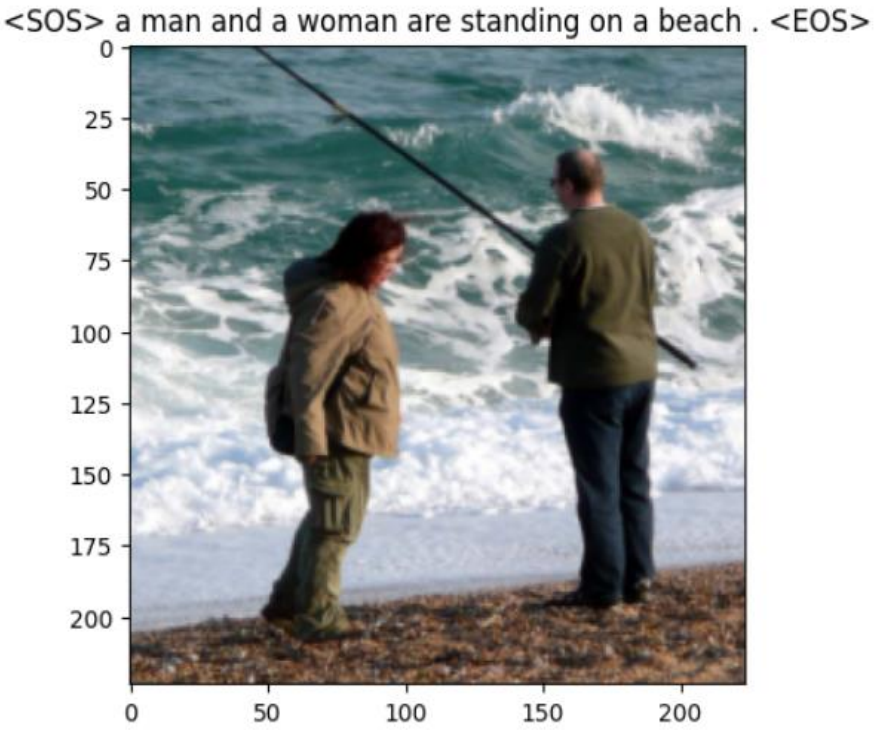


شکل 6. نمودار کاهش تابع هزینه بر روی داده‌های آموزشی و داده‌های اعتبارسنجی- وزن‌های **trainable** برای **ResNet18**

شکل 7. نمونه تصویر و کپشن تولید شده – **trainable ResNet 18**



شکل 8. نمونه تصویر و کپشن تولید شده – **trainable ResNet 18**



شکل 9. نمونه تصویر و کپشن تولید شده – **trainable ResNet 18**

5-۱. بحث و نتیجه‌گیری

در این بخش به بحث و نتیجه‌گیری مدل‌های آموزش داده شده می‌پردازیم. ما ابتدا مدل را با وزن‌های pretrained از مدل resnet18 برای استخراج ویژگی استفاده نمودیم و در بخش 4-1 از مدل resnet18 که وزن‌ها قابل آموزش باشند استفاده کردیم. با توجه به نمودارهای کاهش هزینه میتوان مشاهده نمود که کاهش هزینه در هر 2 مدل تقریبا یکسان بوده و در 10 ایپاک آموزش loss به مقدار قابل توجهی کاهش پیدا کرده است. همان طور که مشاهده می‌شود مدل پتانسیل overfitting داشته است. ما برای مقابله با overfitting ابتدا نرخ یادگیری را کاهش دادیم. تکنیک دیگری که میتوان برای جلوگیری از overfitting به کار گرفت early stopping است.

همان طور که در بخش نمایش تصاویر و کپشن‌های تولید شده توسط مدل مشاهده می‌شود نتایج و کپشن‌ها منطقی هستند و ارتباط معناداری بین کپشن‌ها و تصاویر برقرار است. البته مدل دوم بعضی جاها بهتر عمل کرده. برای مثال، در مقایسه کپشن‌های شکل 9 و شکل 5، می‌بینیم مدل دوم توانسته توضیحات مناسب تری بدهد. با کمی دقت در روی نمودارهای تابع هزینه درمیابیم که مدل دوم، اورفیت کمتری نسبت به مدل اول دارد و این می‌تواند دلیلی برای بهتر بودن مدل دوم نسبت به مدل اول بر روی داده‌های

validation و test باشد. از طرف دیگر مدل دوم تعداد 11 میلیون پارامتر بیشتر برای ترین کردن دارد و این باعث کندتر شدن روند در زمان آموزش شبکه می‌شود.

پاسخ ۲ . تشخیص اندیشه

۲-۱. معماری LSTM و embedding

LSTM (حافظه کوتاه مدت بلندمدت) نوعی (RNN) است که بر محدودیت های خاص RNN های سنتی غلبه می کند. در اینجا برخی از مزایای LSTM نسبت به RNN ها آورده شده است:

1. **Long-term dependencies**: شبکه های LSTM برای حفظ وابستگی های طولانی مدت در داده های sequential طراحی شده اند. RNN های معمولی برای یادگیری و حفظ اطلاعاتی که مربوط به time step های خیلی گذشته هستند ، دچار مشکل می شوند لذا آنها برای کارهایی که نیاز به درک وابستگی های طولانی مدت دارند، مورد استفاده نیستند. معماری LSTM شامل سلول‌های حافظه (memory cells) و gate های پیچیده‌تر است که به آن‌ها اجازه می‌دهد اطلاعات را در توالی‌های طولانی‌تر ضبط و منتشر کنند.

2. **Handling vanishing gradients**: RNN ها می توانند در طول زمان و در طی فذایند back propagation دچار مشکل vanishing gradient شوند. این امر مانع از فرآیند یادگیری، به خصوص برای توالی های طولانی می شود. LSTM با معرفی مفهوم سلول‌های حافظه و استفاده از gate های مختلف (input gate, forget gate, and output gate) برای کنترل جریان اطلاعات و گرادیان‌ها در شبکه، این مشکل را کاهش می‌دهد. این به شبکه‌های LSTM کمک می‌کند تا اطلاعات را در توالی‌های طولانی‌تر به طور مؤثرتری یاد بگیرند و حفظ کنند.

3. **Memory retention**: سلول‌های حافظه (memory cells) در شبکه‌های LSTM به آن‌ها اجازه می‌دهند تا با استفاده از مکانیسم gate ها، اطلاعات را در هر مرحله زمانی به طور انتخابی حفظ یا فراموش کنند. این به LSTMها امکان می‌دهد یاد بگیرند که کدام اطلاعات برای ذخیره کردن مهم هستند و کدام‌ها را می‌توان دور انداخت.

4. **Flexibility and adaptability**: شبکه های LSTM بسیار انعطاف پذیر هستند و می توانند با انواع مختلف داده های متوالی سازگار شوند. آنها را می توان به آسانی با افزودن سلول های حافظه اضافی یا اصلاح مکانیسم های دروازه برای تطابق با وظایف خاص یا بهبود عملکرد، گسترش یا تغییر داد.

5. **Reduced training time**: در حالی که افزایش پیچیدگی شبکه های LSTM نیاز به محاسبات اضافی دارد، آنها اغلب در طول آموزش در مقایسه با RNN های پایه سریعتر همگرا می شوند. این به این دلیل است که شبکه های LSTM می توانند یاد بگیرند که بر روی اطلاعات مهم تمرکز کنند و ورودی های نامربوط را نادیده بگیرند و در نتیجه یادگیری کارآمدتری داشته باشند.

Word embedding تکنیکی است که در پردازش زبان طبیعی (NLP) برای نمایش کلمات یا عبارات به عنوان بردارهای متراکم و کم‌بعد در یک فضای برداری پیوسته استفاده می‌شود. هدف از word embedding، به دست آوردن روابط معنایی و نحوی بین کلمات است، و ماشین‌ها را قادر می‌سازد تا زبان انسان را بهتر درک و پردازش کنند.

دلایل اصلی استفاده از word embedding به شرح زیر است:

1. **بازنمایی معنایی(Semantic representation)**: word embedding به کلمات با معانی مشابه اجازه می دهد تا نمایش های برداری مشابهی داشته باشند. این تکنیک با به دست آوردن روابط معنایی، ماشین‌ها را قادر می‌سازد تا معنی و context کلمات را به روشی دقیق تر درک کنند. به عنوان مثال، در یک مدل word embedding به خوبی آموزش دیده، کلماتی مانند "شاه" و "ملکه" دارای بردارهایی هستند که در مقایسه با کلماتی مانند "سگ" یا "ماشین" به یکدیگر نزدیکتر هستند.

2. **کاهش ابعاد**: word embedding ، بازنمایی کلمه‌های رمزگذاری‌شده به صورت one-hot با ابعاد بالا را به بردارهای متراکم با ابعاد پایین‌تر کاهش می‌دهند. این کاهش ابعاد، محاسبات را کمتر می کند و امکان مدیریت آسان تر واژگان بزرگ را فراهم می کند.

3. **Feature learning**: مدل‌های word embedding بر روی حجم زیادی از داده‌های متنی آموزش داده می‌شوند و در طول این فرآیند ، مدل‌ها ویژگی‌های مفیدی را از متن یاد می‌گیرند. این ویژگی های آموخته شده را می توان برای کارهای downstream NLP مانند تجزیه و تحلیل احساسات، ترجمه ماشینی، پاسخگویی به سوالات و طبقه بندی متن مورد استفاده قرار داد.

روش های مختلفی برای تولید word embedding وجود دارد. در اینجا چند تکنیک متداول را به اختصار توضیح می دهیم:

1. **Word2Vec** :Word2Vec یک الگوریتم محبوب برای تولید word embedding است.این الگوریتم دو رویکرد زیر را ارائه می دهد: Continuous Bag-of-Words (CBOW) و Skip-gram. CBOW کلمه هدف را بر اساس کلمات در متن پیش بینی می کند، در حالی که Skip-gram کلمات متن را بر اساس یک کلمه هدف داده شده پیش بینی می کند. مدل‌های Word2Vec با استفاده از مجموعه بزرگی از داده‌های متنی آموزش داده می‌شوند و embedding حاصل، روابط بین کلمات را بر اساس الگوهای همزمانی آنها در داده‌های آموزشی ثبت می‌کنند.

2. **Global Vectors for Word Representation**: GloVe یکی دیگر از روش های پرکاربرد برای word embedding است. این روش آمار جهانی از کاربرد همزمان (word co-occurrence) کلمات را با تکنیک فاکتورسازی برای تولید بردارهای کلمه ترکیب می‌کند. GloVe embedding بر روی داده های متنی بزرگ آموزش داده می‌شوند و روابط معنایی و نحوی بین کلمات را ثبت می‌کنند.

3. **Bidirectional Encoder Representations from Transformers**: BERT یک مدل زبانی پیشرفته است که word embedding و همچنین بازنمایی کلمات متنی (contextualized) را تولید می‌کند. BERT روی مقادیر زیادی از داده‌های متنی بدون برچسب با استفاده از یک مدل زبانی از قبل آموزش داده شده تولید و سپس برای کار های خاص fine-tuned می شود.

GloVe embeddings می‌توانند برای کلماتی که معانی متعددی دارند، مناسب باشند اما محدودیت‌های خاصی در گرفتن ظرایف معنایی ریز دارند.

GloVe embeddings بر روی مجموعه‌های بزرگی از متن آموزش داده می‌شوند، جایی که از آمار word co-occurrence برای تولید بردارهای کلمه استفاده می‌شود. این بدان معنی است که کلماتی که اغلب با هم در context مشابه ظاهر می شوند، بازنمایی های برداری مشابهی خواهند داشت. برای کلماتی با معانی چندگانه، تعبیه‌های GloVe تمایل دارند تا معنای متوسط یا کلی کلمه را بر اساس بافت کلی آن در داده‌های آموزشی نشان دهند.

در حالی که تعبیه‌های GloVe می‌توانند سطحی از چند معنایی را به تصویر بکشند، اما ممکن است بین معانی و senses مختلف یک کلمه به صورت دقیق تمایز قائل نشوند. این به این دلیل است که GloVe در طول فرایند آموزش، با تمام رخدادهای یک کلمه به عنوان یک موجودیت واحد رفتار می کند ، و برای senses و حواس مختلف کلمه به طور جزئی تمایزی قائل نمی شود.

با این حال، توجه به این نکته مهم است که GloVe embeddings هنوز هم می‌توانند در بسیاری از کارهای NLP مفید باشند، به‌ویژه زمانی که معنی یا بافت کلی یک کلمه برای کار داده شده کافی باشد. به عنوان مثال، در تجزیه و تحلیل احساسات یا وظایف طبقه بندی متن، که در آن احساس کلی یا دسته بندی یک جمله مهم است، GloVe embeddings می توانند اطلاعات ارزشمندی را ارائه دهند.

اگر نیاز به تمایز دقیق‌تر از معنای کلمه باشد، تکنیک‌ها یا مدل‌های تخصصی تمایز حس کلمه که به صراحت برای گرفتن حواس کلمه طراحی شده‌اند، مانند WordNet یا جاسازی‌های حسی، ممکن است مناسب‌تر باشند.

۲-۲. پیش پردازش دادگان

در این بخش به اختصار اطلاعاتی کلی درباره دادگان مورد استفاده را ارائه می دهیم و فرایند های پیش پردازش آنها را شرح خواهیم داد.

دیتاست مورد استفاده در این بخش شامل 5452 سطر و 3 ستون می باشد. هر سطر شامل یک سوال است که در ستون text قرار گرفته است. هر سوال نیز دارای دو برچسب است . یکی از آن دو تحت عنوان label-fine بوده که شامل مقادیر 0 تا46 است و دیگری با نام label-coarse شامل مقادیر0 تا5 می باشد.

برای فرایند پیش پردازش دادگان از دو کتابخانه tensorflow و NLTK استفاده کردیم. در زیر به شرح فرایند های مورد استقاده در این مرحله خواهیم پرداخت.

1. Convert text to lowercase
2. Word Tokenize: استفاده از nltk.word_tokenize که یک جمله ورودی را به کلمات تشکیل دهنده آن split می کند.
3. Remove Stop Words: حذف کلمات معروف به Stop Words مانند the،is ،and و ...
4. Stemming and Normalization: استخراج ریشه کلمات و حذف پیشوند ، پسوند و غیره از آنها و سپس نرمالیزه کردن آنها با تبدیل بردار کلمات به جملات حاوی آنها .
5. Tokenize with tensorflow: در این مرحله با استفاده از تابع Tokenizer از کتابخانه tensorflow، هر کلمه در جملات نرمالیزه شده در طی مراحل قبل را به مقداری عددی منحصر به فرد نگاشت می کند تا برای مدل ها قابل فهم شوند.
6. Pad Text: هدف از این مرحله آن است که جملات را به اندازه های یکسان تبدیل کنیم تا برای استفاده در مدل آماده شوند. طول هر جمله را به اندازه طول بلندترین جمله و با استفاده از zero padding تبدیل می کنیم.
7. Creating embedding matrix: در این مرحله ابتدا embedding پیشنهادی مقاله که در فایل glove.6B.300d.txt قرار دارد را داخل حافظه لود کرده و سپس ماتریس Embedding وزن مورد استفاده در مدل را با توجه به Embedding های glove.6B.300d و نیز کلمات موجود در متن دیتاست ،آماده سازی می کنیم.

   glove.6B.300d embedding به یک مدل embedding از پیش پردازش شده با نام GloVe اشاره دارد که حاوی 6 میلیارد توکن و بردار های با ابعاد 300 می باشد.(یعنی هر کلمه را به 300 بعد نگاشت می کند.)

۲-3. پیاده سازی طبقه بندی نیت

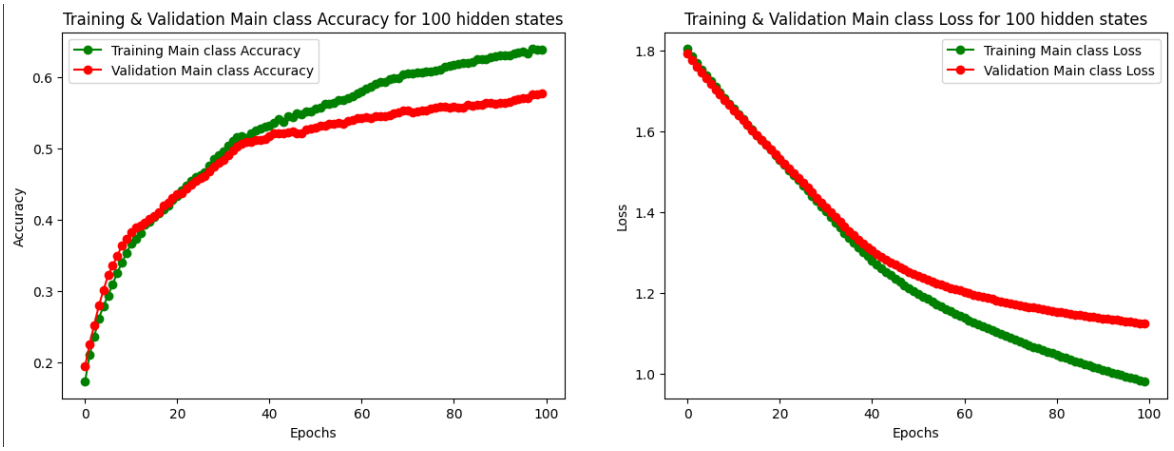
در این قسمت ابتدا به بررسی ساختار مدل ها پرداخته و سپس سعی می کنیم با استفاده از متریک های مختلف عملکرد هر کدام را بررسی کنیم.

```
Model: "First_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input1 (InputLayer)         [(None, 18)]              0

 embedding_17 (Embedding)    (None, 18, 300)           2072100

 lstm_18 (LSTM)              (None, 25)                32600

 flatten_24 (Flatten)        (None, 25)                0

 mainclass (Dense)           (None, 6)                 156

=================================================================
Total params: 2,104,856
Trainable params: 32,756
Non-trainable params: 2,072,100
_________________________________________________________________
```

شکل 10. ساختمان مدل اول

در شکل 10 ساختمان مدل اول نمایش داده شده است. در این ساختار ابتدا لایه input قرار دارد که اندازه بردار های ورودی آن به اندازه بلندترین جمله یعنی 18 است. سپس لایه embedding قرار دارد که همانطور که پیش از این نیز گفته شد حاوی 18 بردار 300 بعدی می باشد بدین بعنی که هر کلمه ورودی را به یک فضای برداری 300 بعدی نگاشت می کند. سپس لایه LSTM قرار دارد که در این شکل تعداد hidden state های آن 25 است. نکته قابل توجه این است که در این مدل ما تنها به خروجی سلول آخر لایه LSTM نیاز داریم به همین علت خروجی این لایه یک بردار 25 بعدی است. سپس خروجی لایه LSTM را برای دادن به لایه بعدی یعنی لایه linear، flat می کنیم و در نهایت به یک لایه dense با 6 نورون (به اندازه تعداد main class) و با تابع فعالساز softmax می دهیم. تعداد کل پارامتر ها و نیز پارامتر های trainable و non-trainable نیز در شکل مشخص است. non-trainable params در واقع همان وزن های لایه embedding هستند که به صورت transform learning از وزن های glove.6B.300d استفاده کردیم.

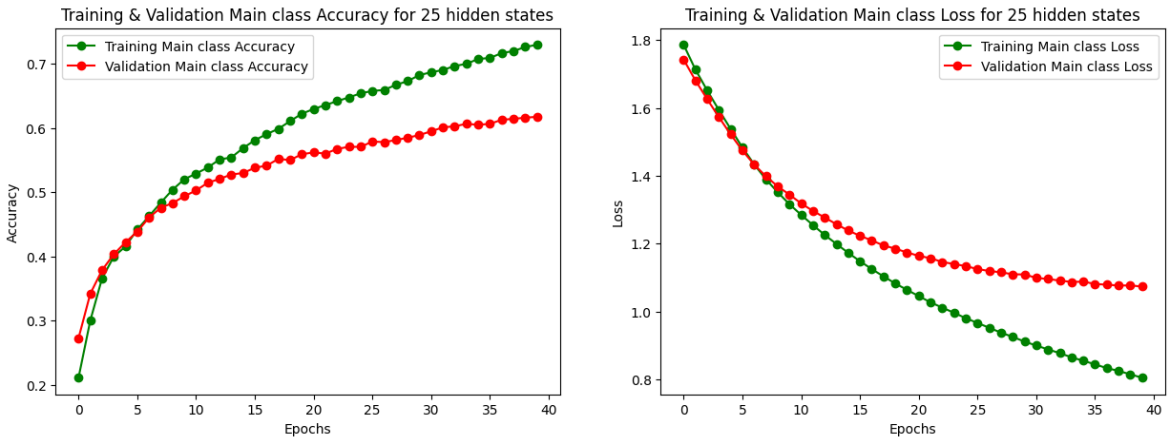
نهایتا مدل را با تابع loss، "sparse_categorical_crossentropy" و Adam optimizer کامپایل کرده و در طی 100 ایپاک و با validation_split = 0.4 آموزش دادیم. در ادامه نتایج کسب شده را بررسی خواهیم کرد.



شکل 11. نمودار **loss** و **accuracy** برای داده های **train** و **validation** برای **100 hidden state**

شکل بالا، نمودار lossو accuracy را برای داده های train و validation در طی 100 ایپاک و برای تعداد hidden states 100 نشان می دهد. برای این مدل از learning_rate = 1e-5 استفاده کردیم. همانطور که مشخص است در طی هر ایپاک دقت مدل افزایش و loss آن کاهش یافته است تا در نهایت در ایپاک آخر به مقادیر زیر رسیده است :

loss: 0.9803 - **accuracy**: 0.6396 - **val_loss**: 1.1248 - **val_accuracy**: 0.5777



شکل 12. نمودار **loss** و **accuracy** برای داده های **train** و **validation** برای **hidden state 25**

در شکل 12 ، نمودار lossو accuracy را برای داده های train و validation در طی 40 ایپاک و برای تعداد hidden states 25 نشان می دهد. برای این مدل از learning_rate = 1e-4 استفاده کردیم. همانطور که مشخص است در این مدل نیز طی هر ایپاک دقت مدل افزایش و loss آن کاهش یافته است تا در نهایت در ایپاک آخر به مقادیر زیر رسیده است :

loss: 0.8047 - **accuracy**: 0.7297 - **val_loss**: 1.0734 - **val_accuracy**: 0.6176

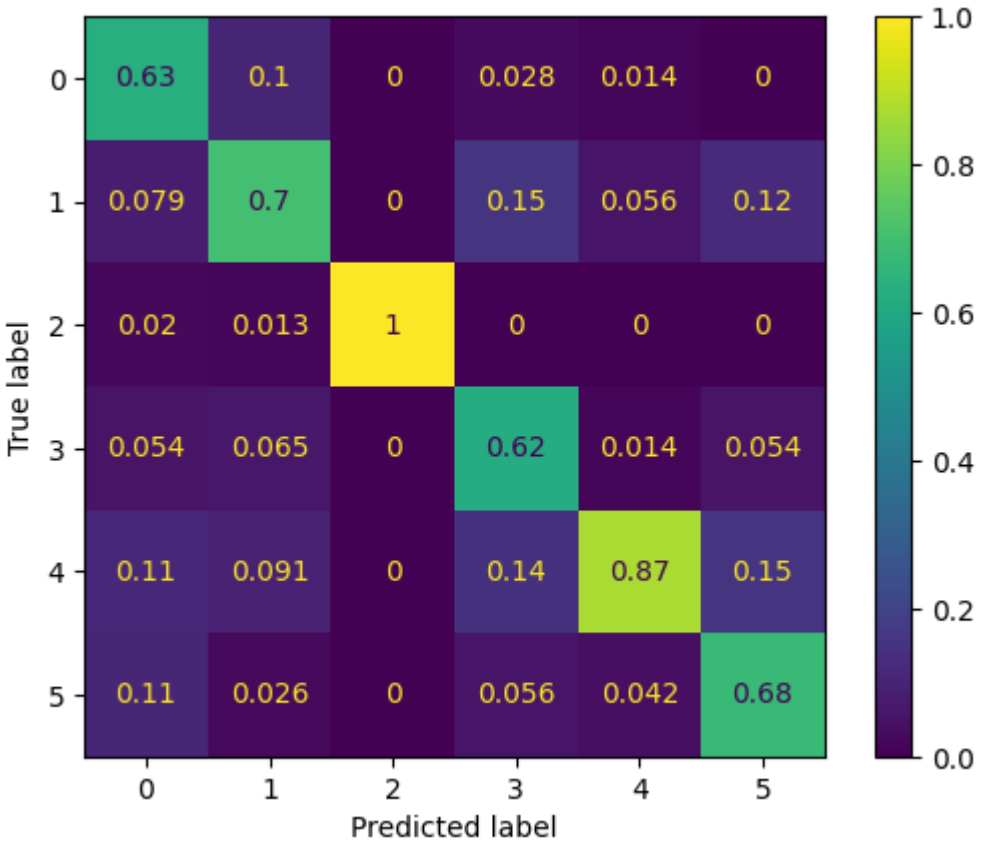
	precision	recall	f1-score	support
0	0.55	0.95	0.70	138
1	0.69	0.48	0.57	94
2	0.00	0.00	0.00	9
3	0.65	0.55	0.60	65
4	0.88	0.50	0.64	113
5	0.62	0.59	0.61	81
accuracy			0.63	500
macro avg	0.57	0.51	0.52	500
weighted avg	0.67	0.63	0.62	500

شکل 13. نتایج پیاده سازی مدل اول بر داده های **test** برای 25**hidden state**

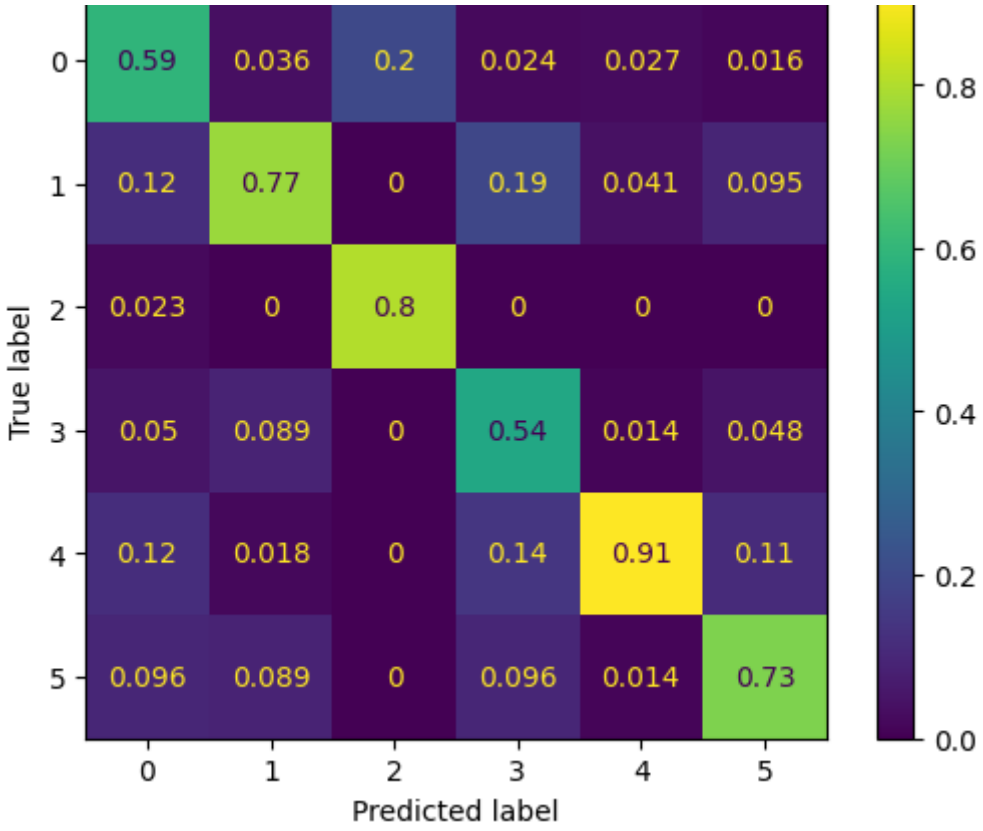
	precision	recall	f1-score	support
0	0.52	0.96	0.67	138
1	0.59	0.40	0.48	94
2	0.00	0.00	0.00	9
3	0.56	0.43	0.49	65
4	0.92	0.43	0.59	113
5	0.59	0.56	0.57	81
accuracy			0.59	500
macro avg	0.53	0.46	0.47	500
weighted avg	0.63	0.59	0.57	500

شکل 14. نتایج پیاده سازی مدل اول بر داده های **test** برای **100 hidden state**

شکل بالا نتایج حاصل از مدل با 25 hidden states و شکل پایین با 100 عدد است. در هر دو مورد دقت حاصل با دقت مقاله بسیار فاصله دارد. در کل مدل پیشنهادی با تعداد 25 hidden states توانسته به دقت های بهتری به نسبت دیگر مدل برسد.



شکل 15. ماتریس پراکندگی مدل اول با **hidden state**25



شکل 16. ماتریس پراکندگی مدل اول با **100 hidden state**

همچنین ماتریس آشفتگی مربوط به مدل با هر دو تعداد LSTM hidden states در شکل های بالا نمایش داده شده است. در هر دو مورد قطر اصلی به نسبت سایریرین مقادیر بیشتری دارد که این نشانه خوبی است . مدل اول بهترین عملکرد را با جملات کلاس 2 داشته است و توانسته به طور کامل همه آنها را پیش بینی کند . مدل دوم بهترین عملکرد را برای کلاس 4 داشته است. مقایسه ماتریس پراکندگی دو مدل نشان می دهد که هر مدل در پیش بینی برخی کلاس ها از دیگری بهتر عمل کرده است.

```
#Building the base model
NUM_HEIDDEN_STATES = 25
input_layer =  Input(shape=(max_length+1,), name = 'input1')
embeddig = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length =max_length+1 ,trainable=False)(input_layer)
lstm = LSTM(NUM_HEIDDEN_STATES , return_sequences=True)(embeddig)

#subclass prediction
flat =  Flatten()(lstm[:,-1])
first_predictions = Dense(num_sub_class,activation='softmax',name='subclass')(flat)
#mainclass prediction
flat_2 = Flatten()(lstm[:,-2])
second_predictions = Dense(num_main_class,activation='softmax',name='mainclass')(flat_2)

second_model = Model(inputs=input_layer, outputs=[first_predictions , second_predictions] , name="Second_model")
opt = optimizers.Adam(learning_rate=1e-4)
second_model.compile(optimizer=opt,
                     loss={
                           'subclass': 'sparse_categorical_crossentropy',
                           'mainclass': 'sparse_categorical_crossentropy'
                           },
                     metrics={
                           'subclass': 'accuracy',
                           'mainclass': 'accuracy'
                           })
second_model.summary()
```

```
Model: "Second_model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input1 (InputLayer)            [(None, 19)]         0           []

 embedding (Embedding)          (None, 19, 300)      2072100     ['input1[0][0]']

 lstm (LSTM)                    (None, 19, 25)       32600       ['embedding[0][0]']

 tf.__operators__.getitem (Slic  (None, 25)          0           ['lstm[0][0]']
 ingOpLambda)

 tf.__operators__.getitem_1 (Sl  (None, 25)          0           ['lstm[0][0]']
 icingOpLambda)

 flatten (Flatten)              (None, 25)           0           ['tf.__operators__.getitem[0][0]'
                                                                 ]

 flatten_1 (Flatten)            (None, 25)           0           ['tf.__operators__.getitem_1[0][0
                                                                 ]']

 subclass (Dense)               (None, 47)           1222        ['flatten[0][0]']

 mainclass (Dense)              (None, 6)            156         ['flatten_1[0][0]']

==================================================================================================
Total params: 2,106,078
Trainable params: 33,978
Non-trainable params: 2,072,100
__________________________________________________________________________________________________
```

شکل 17. ساختمان مدل دوم

شکل 17، نمایش دهنده ساختمان مدل دوم هست. تفاوت این مدل با مدل قبلی در این است که در اینجا ما دو خروجی main class و sub class داریم که برای آنها از خروجی آخر و یکی مانده به آخر لایه LSTM استفاده می کنیم . لذا ما به خروجی همه cell های lstm نیاز داریم و برای همین return_sequence را برای لایه LSTM برابر True قرار می دهیم تا خروجی cell های داخلی را نیز داشته باشیم.سپس خروجی cell های مورد نظر را به لایه های dense با تعداد نورون برابر با تعداد sub classو main class داده و از تابع softmax عبور می دهیم. مدل ما دو خروجی دارد که برای هر دو از "sparse_categorical_crossentropy" به عنوان تابع loss استفاده می کنیم. تعداد پارامتر های trainable و non-trainable مدل نیز در شکل نمایش داده شده است. نکته قابل توجه در این معماری آن است که برای ورودی شبکه ، در آخر هر آرایه یک 0 اضافه می کنیم تا از خروجی LSTM آن برای طبقه بندی subclass استفاده کنیم. Hyperparameter های استفاده شده تعداد hidden state لایه LSTM و نیز learning rate برای Adam optimizer است که در جدول زیر مقادیر استفاده شده را آورده ایم:

جدول 1. مقادیر **learning rate** برای **lstm** با مقادیر **25,100 hidden states**

# Hidden states for LSTM layer	Learning rate
25	**3e-5**
100	**e-51**

Hyperparameter های استفاده شده با آزمون و خطا به دست آمدند و آنهایی که بهترین دقت را برای مدل طراحی شده ، داشتند به عنوان پارامتر های نهایی استفاده شدند. اگر چه نهایتا نتوانستیم به دقت گزارش شده در مقاله دست پیدا کنیم و چون در مقاله درباره پارامتر های استفاده شده چیزی گفته نشده ، نهایتا مدل را با بهترین نتیجه ای که توانستیم به آن دست پیدا کنیم گزارش کردیم .

گزارش های مربوط به عملکرد مدل دوم برای 25 hidden states به شرح زیر می باشد:

train_subclass_loss: 2.2034

train_mainclass_loss:0.9109

train_subclass_accuracy: 0.3925

train_mainclass_accuracy: 0.6796

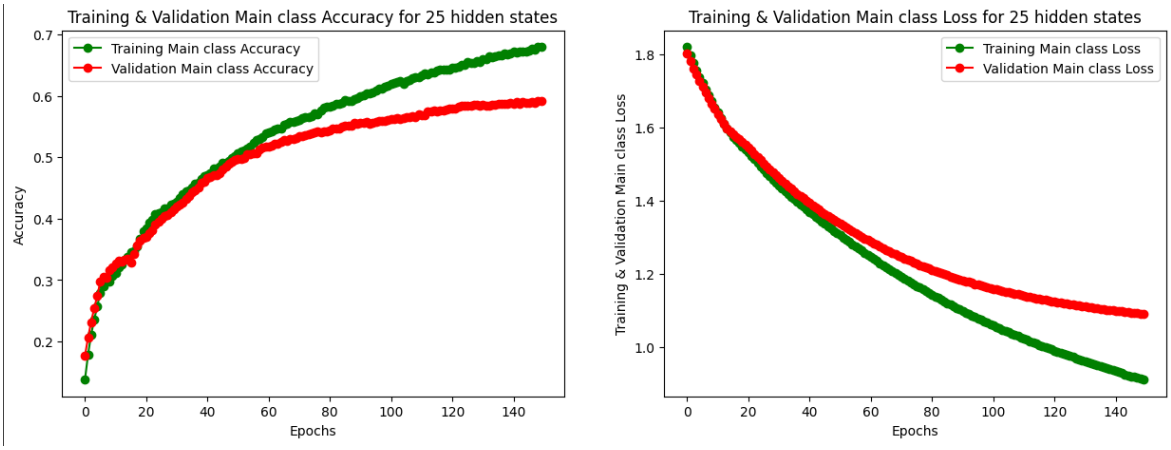
val_subclass_loss: 2.4075

val_mainclass_loss: 1.0905

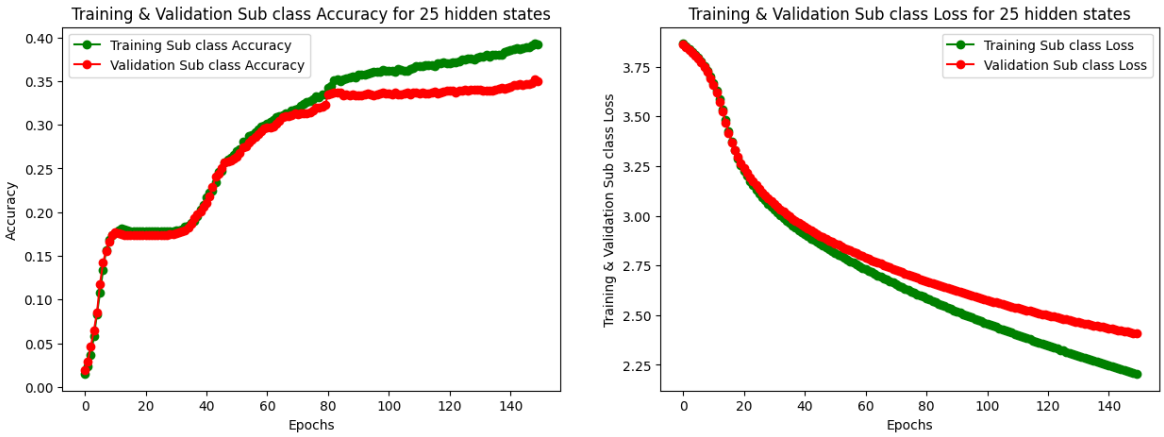
val_subclass_accuracy: 0.3494

val_mainclass_accuracy: 0.5924

همانطور که گفتیم دقت ها ی به دست آمده از دقت های مقاله فاصله دارد ولی نکته آن است که مدل ما در این مرحله و با این دقت ها overfit نشده است و برای به دست آوردن دقت های بیشتر روی training data مدل overfit می شد.



شکل 18.. نمودار **loss** و **accuracy** برای مدل دوم کتگوری **main class** روی داده های **train** و **val** ، با **hidden state** 25



شکل 19. نمودار **loss** و **accuracy** برای مدل دوم کتگوری **sub class** روی داده های **train** و **val** ، با **hidden state**25

دو شکل بالا نمودار های accuracy و loss برای داده های train و val برای دو گروه subclassو main class برای تعداد hidden states 25 را نشان می دهد . همانطور که مشخص است در اجرای الگوریتم برای 150 ایپاک ، نمودار accuracy برای هر دو کلاس تقریبا صعودی و نمودار loss نزولی است . این نکته که مدل در نهایت overfit نشده است نیز در شکل مشخص است چرا که اختلاف بین نتایج train و val زیاد نیست.

	precision	recall	f1-score	support
0	0.52	0.93	0.67	138
1	0.60	0.39	0.47	94
2	0.00	0.00	0.00	9
3	0.59	0.52	0.55	65
4	0.87	0.36	0.51	113
5	0.57	0.62	0.60	81
accuracy			0.58	500
macro avg	0.53	0.47	0.47	500
weighted avg	0.62	0.58	0.56	500

شکل 20. نتایج حاصل از مدل دوم بر روی داده های **test** برای **main class** با **25 hidden state**

accuracy			0.40	500
macro avg	0.07	0.08	0.06	500
weighted avg	0.29	0.40	0.30	500

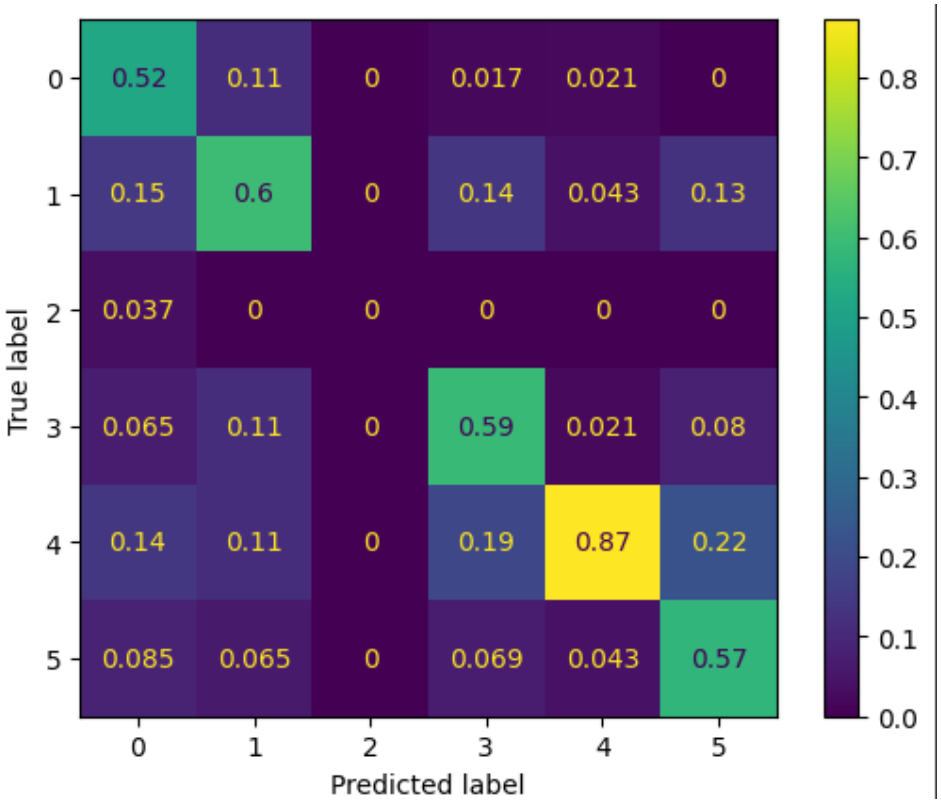
شکل 21.نتایج حاصل از مدل دوم بر روی داده های **test** برای **sub class** با 25 **hidden state**

سپس مدل را بر روی داده های test آزمایش کرده و از متریک هایی که در دوشکل بالا دیده می شود برای ارزیابی عملکرد مدل استفاده کردیم. بر طبق آنچه در نتایج حاصل دیده می شود ، عملکرد مدل در پیش بینی subclass به نسبت main class بدتر بوده است و یکی از دلایل آن میتوان تعداد زیاد کلاس ها (47) موجود در این کتگوری باشد که برای train شدن به دیتای بیشتری نیاز دارد . برای پیش بینی main class، مدل بهترین عملکرد را در پیش بینی کلاس 0و بدترین عملکرد را در پیش بینی کلاس 2 داشته است . برای کلاس های موجود در main class به علت آنکه تعداد آنها کم بود مقایسه متریک های آن راحت تر است ولی برای sub class به علت تعداد زیاد کلاس ها این مقایسه راحت نیست و به همین علت فقط گزارش نهایی آن را آورده و کلاس به کلاس بررسی نکردیم. (نتایج حاصل از آن در بخش بعدی و در ماتریس آشفتگی بهتر قابل قیاس است.)

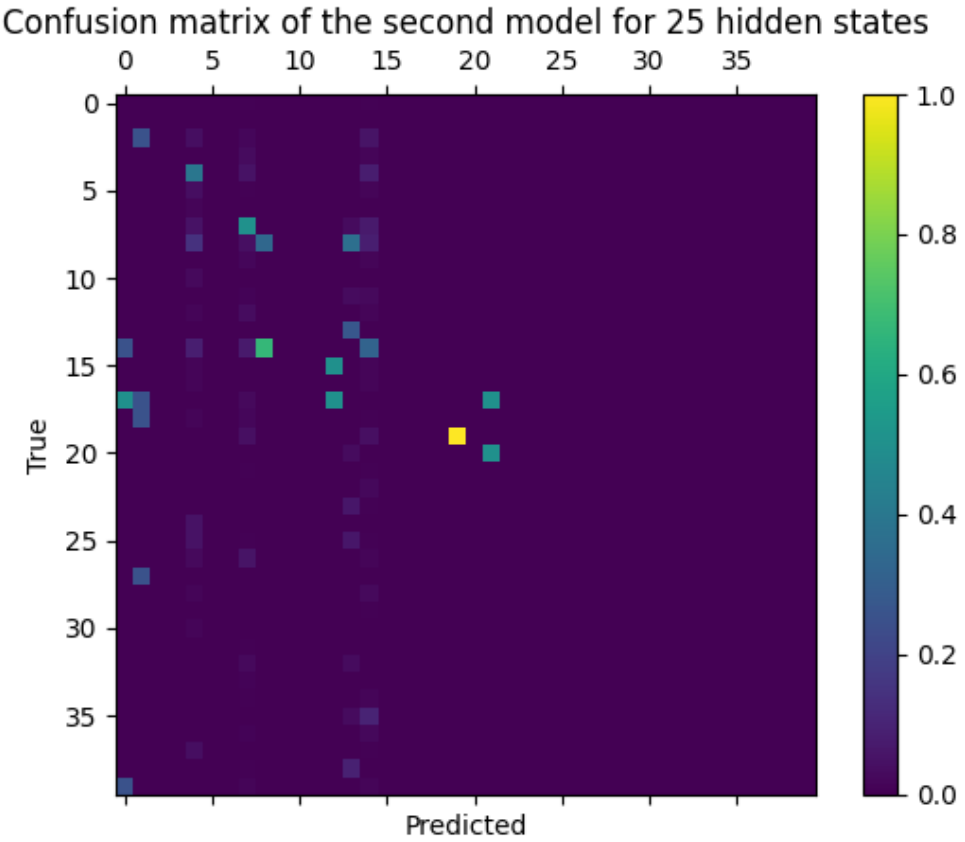
برای یادآوری !

Precision: دقت پیش‌بینی‌های مثبت را اندازه‌گیری می‌کند،

Recall: کامل بودن پیش‌بینی‌های مثبت را اندازه‌گیری می‌کند.



شکل 22ماتریس پراکندگی مدل دوم با 25 **hidden state** بر روی داده های **test** برای **main class** s



شکل 23. ماتریس پراکندگی مدل دوم با 25 **hidden state** بر روی داده های **test** برای **sub class**

یکی دیگر از روش های مورد استفاده برای مقایسه عملکر مدل ها استفاده از ماتریس آشفتگی آنهاست که در شکل 22,23 نمایش داده شده است. با نگاه به این ماتریس برای خروجی main class میتوان دید که مدل در پیش بینی کلاس 4 عملکرد به نسبت خوبی داشته است چرا که بیشترین تعداد پیش بینی درست آن برای این کلاس بوده است . یک نکته که در این ماتریس مشخص می شود و قبلا در سایر متریک ها متوجه آن نشدیم آن است که تعداد داده های با main class = 2 به طور کلی در دیتاست کم بوده است و این غنی نبودن دیتا ست یکی از دلایل عملکرد ضعیف مدل است . البته این مسئله می تواند ناشی از تقسیم اشتباده داده ها به train و test نیز باشد که برای بررسی آن می توانیم از Splitting Strata استفاده کنیم تا داده های هر کلاس به نسبت مساوی تقسیم شوند . دز خصوص ماتریس مربوط به sub class ، اول اینکه به علت تعداد بالای کلاس ها یکی از بهترین روش ها برای مقایسه عملکرد مدل همین استفاده از ماتریس آشفتگی است. ثانیا همانطور که دیده میشود تعداد پیش بینی های درست مدل زیاد نبوده و تنها برای یک کلاس توانسته به تعداد خوبی پیش بینی درست داشته باشد .

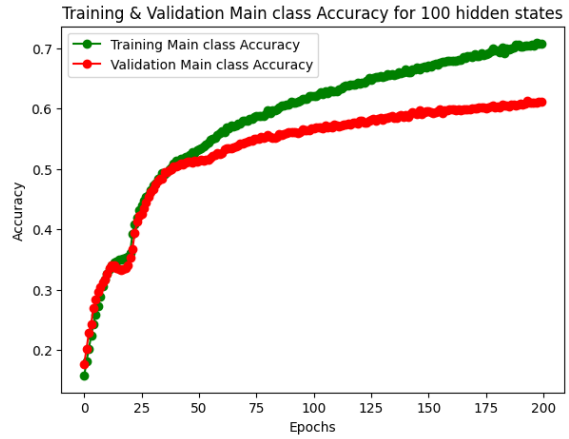
در ادامه همین نتایج را برای تعداد 100 hidden states بررسی خواهیم کرد.

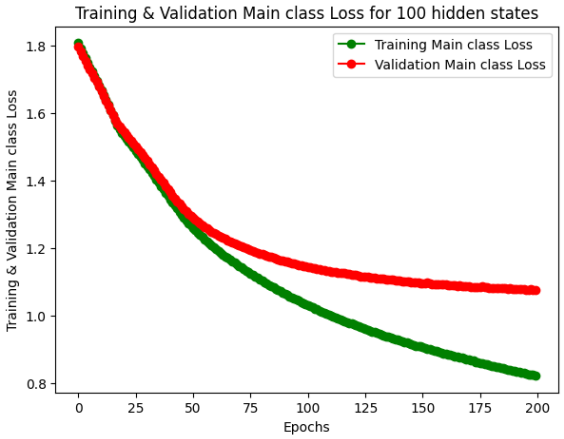
train_subclass_loss: 1.8541
train_mainclass_loss: 0.8229
train_subclass_accuracy: 0.5139
train_mainclass_accuracy: 0.7083
val_subclass_loss: 2.1766
val_mainclass_loss: 1.0759
val_subclass_accuracy: 0.4269
val_mainclass_accuracy: 0.6112

جدول 2. مقایسه عملکرد مدل دوم برای تعداد **25** و **100 hidden state** برای لایه **LSTM**

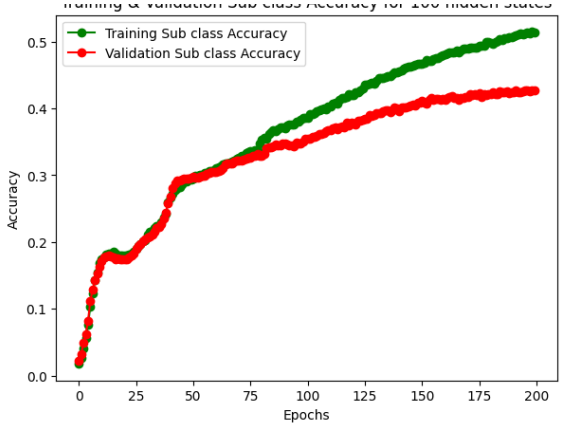
	25 LSTM hidden states	100 LSTM hidden states
Train subclass loss	2.2034	**1.8541**
Train mainclass loss	0.9109	**0.8229**
Train subclass accuracy	0.3925	**0.5139**
Train mainclass accuracy	0.6796	**0.7083**
Validation subclass loss	2.4075	**2.1766**
Validation mainclass loss	1.0905	**1.0759**
Validation subclass accuracy	0.3494	**0.4269**
Validation mainclass accuracy	0.5924	**0.6112**

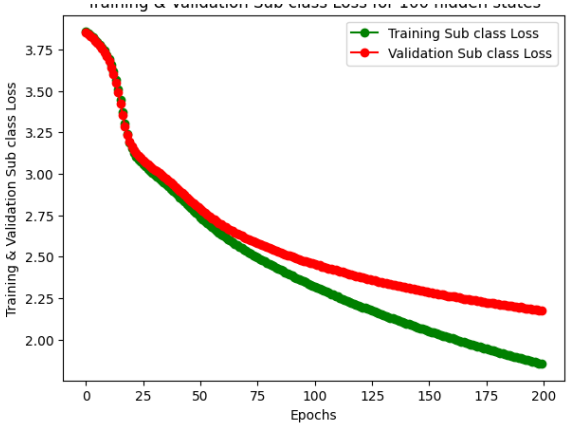
در جدول بالا مقایسه عملکرد مدل برای تعداد 25 و 100 hidden states برای لایه LSTM مشاهده می کنید. همانگونه که مشخص است عملکرد مدل برای تعداد 100 hidden states با اختلاف خوبی به نسبت دیگری بهتر است . مدل مجددا نتوانسته به دقت های مقاله برسد اما هم در train و هم در validation دقت مدل بهبود یافته است.





شکل 24 . نمودار **loss** و **accuracy** برای مدل با 100 **hidden state** با داده های **train** و **val** برای **main class**





شکل 25.نمودار **loss** و **accuracy** برای مدل با 100 **hidden state** با داده های **train** و **val** برای **sub class**

دو نمودار بالا دقت و loss مدل با 100 hidden states را در طی 200 ایپاک برای داده های train و test نشان می دهد. در اینجا نیز در طی ایپاک ها دقت بهبود یافته و loss کم شده البته به نسبت مدل قبلی در نهایت اختلاف بین train و validation بیشتر است ولی به اندازه ای نیست که بتوان گفت مدل overfit شده است.

	precision	recall	f1-score	support
0	0.53	0.95	0.68	138
1	0.66	0.44	0.53	94
2	0.00	0.00	0.00	9
3	0.56	0.51	0.53	65
4	0.92	0.42	0.57	113
5	0.60	0.59	0.60	81
accuracy			0.60	500
macro avg	0.55	0.48	0.48	500
weighted avg	0.65	0.60	0.58	500

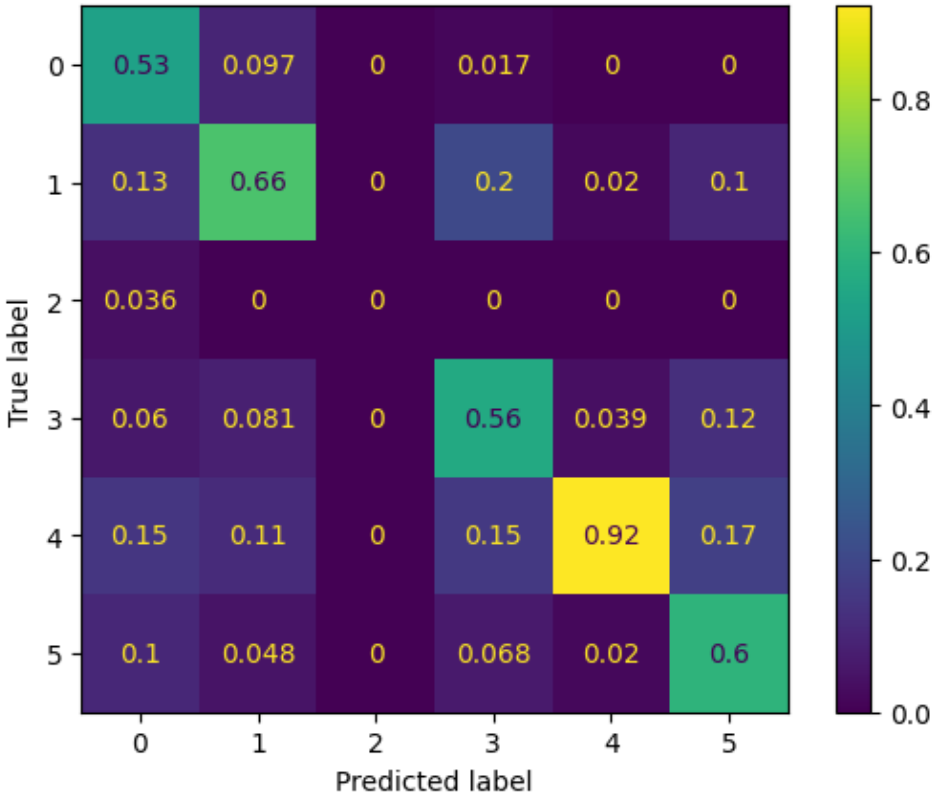
شکل 26.. نتایج حاصل از مدل دوم بر روی داده های **test** برای **main class** با 100 **hidden state**

accuracy			0.47	500
macro avg	0.15	0.13	0.13	500
weighted avg	0.38	0.47	0.39	500

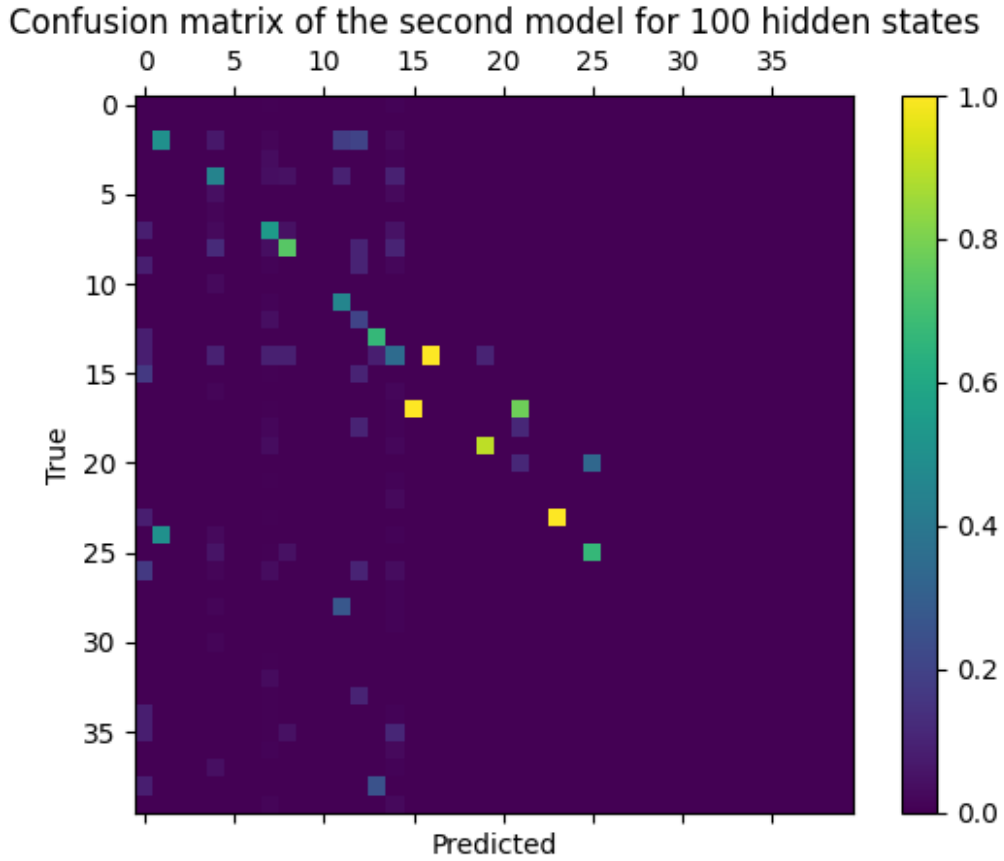
شکل 27.. نتایج حاصل از مدل دوم بر روی داده های **test** برای **sub class** با **hidden state 100**

با مقایسه نتایج حاصل از اجرای مدل دوم بر روی داده های test که در اشکال بالا نمایش داده شده است ، می توان دید که برای main class مجددا کلاس 0 بهترین نتایج را داشته است و دوباره به طور کلی نتایج حاصل از اجرای مدل بر روی subclass نتایج بدتری به نسبت main class داشته است.

همچنین ماتریس آشفتگی ای مدل نیز در اشکال 28,29 نمایش داده شده است. برای داده های با main class نتایج حاصل از این مدل مشابه نتایج حاصل از مدل قبلی می باشد ولی برای داده های subclass نتایج حاصل از این مدل به مراتب بهتر است چرا که تعداد داده های قرار گرفته در قطر اصلی بیشتر است اگرچه باید به این نکته نیز توجه شود که برخی از خانه های خارج از قطر اصلی نیز تعداد زیادی را به خود اختصاص داده اند که این نشان از ضعف مدل در پیش بینی داده های با آن لیبل ها است .



شکل 28.ماتریس پراکندگی مدل دوم با 100 **hidden state** بر روی داده های **test** برای **main class**



شکل 29.ماتریس پراکندگی مدل دوم با 100 **hidden state** بر روی داده های **test** برای **sub class**

۲-4. پیاده سازی مدل Responder

در نهایت یک مدل responder بر روی مدل های طراحی شده سوار کرده تا نحوه استفاده از intent classification در پاسخگویی به سوالات را بررسی کنیم. بدین منظور از دیتاست QA_data که در فایل تمرین قرار دارد استفاده کردیم. مراحل preprocessing اعمال شده بر روی این دیتاست نیز مشابه مراحل اعمال شده بر روی دیتاست قبلی که پیش تر توضیح دادیم است.

```
Model: "Responder"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_layer (InputLayer)       [(None, 12)]         0           []

 embedding (Embedding)          (None, 12, 300)      287400      ['input_layer[0][0]']

 lstm (LSTM)                    (None, 12, 100)      160400      ['embedding[0][0]']

 tf.__operators__.getitem_2 (Sl  (None, 11, 300)     0           ['embedding[0][0]']
 icingOpLambda)

 tf.__operators__.getitem_1 (Sl  (None, 100)         0           ['lstm[0][0]']
 icingOpLambda)

 tf.__operators__.getitem (Slic  (None, 100)         0           ['lstm[0][0]']
 ingOpLambda)

 responder_lstm (Bidirectional)  (None, 11, 200)     320800      ['tf.__operators__.getitem_2[0][0
                                                                 ]',
                                                                  'tf.__operators__.getitem_1[0][0
                                                                 ]',
                                                                  'tf.__operators__.getitem[0][0]'
                                                                 , 'tf.__operators__.getitem_1[0][
                                                                 0]',
                                                                  'tf.__operators__.getitem[0][0]'
                                                                 ]

 Dense_layer (Dense)            (None, 11, 781)      156981      ['responder_lstm[0][0]']

==================================================================================================
Total params: 925,581
Trainable params: 638,181
Non-trainable params: 287,400
__________________________________________________________________________________________________
```

شکل 30. ساختمان مدل **responder**

در شکل 30 ساختمان مدل responder را مشاهده می کنید. در لایه input جملات pad شده به عنوان ورودی شبکه داده می شود و سپس مشابه آنچه در مدل دوم داشته جریان داده های در شبکه ادامه میابد. نکته قابل توجه استفاده از یک لایه bidirectional lstm در شبکه طراحی شده است. خروجی cell آخر و یکی مانده به آخر لایه lstm که قبلا از آنها برای پیش بینی subclass و main class استفاده می کردیم، در این شبکه با یکدیگر ترکیب شده و به عنوان initial state به لایه bidirectional LSTM داده می شود. این همان استفاده از intent classification در پاسخ به سوالات است . توجه شود که تعداد hidden states های لایه bidirectional ، 100 عدد است و چون خروجی همه cell های آنرا استخراج می کنیم نهایتا یک

خروجی (200*11) خواهیم داشت. در آخر یک لایه dense قرار میدهیم با تعداد نورون به اندازه تعداد کلمات خاص موجود در دیتاست. وظیف این لایه این است که خروجی bidirectional LSTM را از200 به 781نگاشت کند تا بتوانیم از آن برای پیش بینی پاسخ استفاده کنیم .

تعداد پارامتر های trainableو non trainable این شبکه را نیز میتوان مشاهده کرد.

```
train_ans_data =[]
for sent in train_ans:
  temp =[]
  for word in sent:
    b = np.zeros(vocab_size_ans)
    b[int(word)] = 1
    temp.append(b)
  train_ans_data.append(temp)

train_ans_data =np.array(train_ans_data)
print(train_ans_data.shape)
```

```
(500, 11, 781)
```

شکل 31.نحوه **reshape** کردن داده های جواب

با توجه به خروجی شبکه طراحی شده ، نیاز داریم تا برای استفاده از gradient decent ، داده های مربوط به جواب را نیز reshape کرده و با استفاده از روش one-hot encoding ، آنها را به فضای 781 نگاشت کنیم. کد مربوط به این بخش نیز در شکل 31 مشاهده می شود.

نهایتا شبکه را بر روی دیتاست داده شده train می کنیم و بعد سوال های داده شده در متن سوال را از آن می پرسیم. شکل 32 نحوه تبدیل خروجی شبکه به متن پاسخ را نشان می دهد. برای اینکار ابتدا یک reversed map از کلمات tokenize شده در قسمت قبل ایجاد می کنیم که در آن key ها همان اعداد یونیک اختصاص داده شده به هر کلمه و value ها کلمات مربوط به آن عدد هستند. سپس با استفاده از این map اعداد خروجی را به کلمات مرتبط نگاشت می کنیم.

```
# Creating a reverse dictionary
reverse_word_map = dict(map(reversed, tokenizer_ans.word_index.items()))

for idi , ans in enumerate(answer__):

  # Function takes a tokenized sentence and returns the words
  def sequence_to_text(list_of_indices):
      list_of_indices = list_of_indices[list_of_indices != 0]
      if list_of_indices.size == 0:
        return ""
      # Looking up words in dictionary
      words = [reverse_word_map.get(letter) for letter in list_of_indices]
      return(words[0])

  # Creating texts
  my_texts = list(map(sequence_to_text, ans))
  str_list = list(filter(None, my_texts))

  print("Question: " , test_questions[idi])
  if len(str_list) == 0:
    print("No Answer")
  else:
    print(str_list[0])

  print("************")
```

شکل 32. کد مربوط به نحوه تبدیل خروجی شبکه به پاسخ های متنی

خروجی یا همان پاسخ های هر سوال در شکل 33 نمایش داده شده اند. همانطور که انتظار می رفت ، پاسخ ها کاملا نامرتبط با متن سوالات است که این با توجه به حجم بسیار پایین دیتاست برای چنین تسکی ، قابل توجیه است.

```
Question:  How many people speak French?
caffein
************
Question:  What day is today?
9
************
Question:  Who will win the war?
narrow
************
Question:  Who is Italian first minister?
blue
************
Question:  When World War II ended?
dollar
************
Question:  When Gandhi was assassinated?
metal
************
```

شکل 33. پاسخ های شبکه طراحی شده به سوال های مطرح شده