به نام خدا

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

درس پردازش زبان طبیعی

پاسخ تمرین ۳

نام و نام خانودگی: زهرا ریحانیان

شماره دانشجویی: ۸۱۰۱۰۱۱۷۷

اردیبهشت ماه ۱۴۰۲

فهرست

پاسخ سوال اول

کد مربوط به این بخش در فایل NLP_CA3_Q1.ipynb موجود است. پاسخ ها را بر اساس ترتیب عناوینی (Title) که در این فایل موجود است، توضیح می دهم.

پاسخ بخش اول: آماده کردن مجموعه داده

- **Load Data**

در این بخش بعد از نصب و import کتابخانه های لازم، داده ها را ابتدا لود و با استفاده از کتابخانه json تبدیل به دیکشنری کردم. سپس هر کدام را در یک دیتافریم ذخیره کردم. در اینجا ۵ ردیف اول دیتاست آموزش را مشاهده می کنید:

	text	srl_frames	verb_index	words_indices
0	[A, primary, stele, ,, three, secondary, stele...	[O, O, O, O, O, O, O, O, O, O, B-ARG1, O]	10	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
1	[The, progress, of, this, coordinated, offensi...	[O, O, O, O, O, B-ARG1, O, O, O, O, O, O]	4	[13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 2...
2	[Well, ,, on, the, battlefield, behind, enemy,...	[O, O, O, O, O, O, O, O, O, O, O, O, O, O, ...	12	[26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 3...
3	[What, you, are, interested, in, is, exactly, ...	[O, B-ARG1, O, B-ARG2, I-ARG2, O, O, O, O, O, ...	2	[54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65]
4	[Well, ,, from, the, information, and, the, si...	[O, O, O, B-ARG1, I-ARG1, I-ARG1, I-ARG1, I-AR...	10	[66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 7...

- **Some helper function**

در اینجا متغیر هایی که در ادامه نیاز داریم را تعریف کردم. دیکشنری خواسته شده برای برچسب ها را کمی تغییر دادم. چون بعضی از برچسب ها یعنی I-ARGM و I-ARGM-TMP در دیتاست داده شده، نبود. تابع srl_frames_to_number را تعریف کردم که لیستی از برچسب ها را می گیرد و با توجه به این دیکشنری، آن ها را به فرم عددی تبدیل می کند و لیستی با همان سایز که عناصر آن عدد متناظر با برچسب است را برمیگرداند.

تابع add_padding فیلد text، طول دلخواه و pad_token را میگیرد و هر کدام از جمله ها در فیلد text را با افزودن pad_token به اندازه داده شده تبدیل می کند. به این ترتیب طول همه جملات یکسان می شود.

- **Create Vocab Class**

در این بخش به پیاده سازی کلاس Vocab مطابق ساختار داده شده پرداختم. در متد سازنده، دیکشنری word2id را می گیرد و اگر None بود، یک دیکشنری می سازد و توکن های Pad_token, Unknown_word, Start_token, End_token را به آن اضافه می کند. یک دیکشنری id2word هم می سازد که key و value آن برعکس این دیکشنری است. متد های __getitem__ ، __len__ و add همانند توضیحات داده شد، پیاده شد. برای متد word2indices دو حالت را در نظر گرفتم که آیا یک لیستی از کلمه داریم یا لیستی از لیست های کلمه و برای این کار نوع اولین عنصر را چک میکند. اگر لیست باشد پس لیستی از لیست هایی که اندیس متناظر با کلمه را دارند بر می گرداند در غیر این صورت لیستی از کلمه هاست و لیستی از اندیس

های متناظر را بر می گرداند. indices2words برعکس این متد است ولی فرض می کند لیستی از اندیس های کلمه را می گیرد و باید لیستی از کلمات متناظر را برگرداند. متد to_input_tensor شبیه متد word2idices است با این تفاوت که باید با padding متناسب با طول جمله، طول جملات را یکسان کند و بعد اندیس های متناظر را برگرداند. در اینجا من پارامتر padding هم اضافه کردم چون در ادامه به این پارامتر نیاز می شد. به صورت دیفالت هم مقدار ۰ دارد که در این صورت طول بزرگترین جمله را پیدا می کند و به همه ی جملات را تا جایی که هم اندازه با آن شوند، Pad_token اضافه می کند. متد from_corpus هم که فیلد text را میگیرد به همراه پارامتر های size, remove_frac, freq_cutoff. خروجی آن یک object از همین کلاس vocab است. این متد ابتدا یک دیکشنری که key آن کلمات و value آن فرکانس حضور در text است، را می سازد. سپس از بین این کلمات، آن هایی که فرکانس حضور کمتر از freq_cutoff دارند را حذف می کند. سپس کلمات را بر اساس فرکانس به صورت نزولی مرتب می کند. سپس محاسبه می کند بر اساس remove_frac چند تا کلمه باید باقی بمانند. سپس مینیمم سایز و عدد بدست آمده را میگیرد تا بداند در کل چند تا کلمه را ذخیره کند. بعد به اندازه عدد بدست آمده، کلمه به vocab اضافه می کند و آن را بر میگرداند.

- **Prepare dataset**

در این بخش آبجکت vocab با توجه به فیلد text داده train با استفاده از متد from_corpus با این پارامتر ها ساخته می شود:

Size=20000

Remove_frac=0.3

Freq_cutoff=1

در نهایت اندازه وکب بدست آمده برابر 13361 شد و طول هر جمله هم برابر 50 .

در ادامه به srl frames هم تا جایی که هم اندازه طولانی ترین جمله یعنی ۵۰ شوند، padding اضافه کردم. که توکن padding در اینجا O است. در ادامه این فیلد را به فرمت عددی تبدیل کردم و با ستون متناظر جایگزین کردم. ستون word_indices هم چون کاربردی نداشت، حذف کردم. در نهایت فیلد text که برای هر سطر لیستی از کلمات را داشت، با استفاده از متد to_input_tensor کلاس وکب تبدیل می کنیم به این که هر سطر لیستی از اندیس کلمه را داشته باشد. در نهایت دیتافریم ها را به دیتاست تبدیل کردم که بتونم به دیتالودر تبدیلش کنم.

پاسخ بخش دوم: LSTM Encoder Model

- **LSTM Encoder**

در این بخش مدل LSTM را مطابق آنچه که خواسته شده بود، پیاده سازی کردم. در متد سازنده متغیر های word_embeddings, lstm, fc را به عنوان فیچر های کلاس تعریف کردم. از کتابخانه پایتورچ برای تعریف این لایه ها استفاده کردم. در متد forward که همان فرآیند forward در شبکه عمیق است، پارامتر های text و verb_index دریافت می شود. ابتدا ورودی که همان فیلد متن است را به لایه embedding داده و جانمایی های کلمات بدست آورده می شود. سپس این جانمایی

ها به lstm داده می شود. با استفاده از این خروجی و verb_index هایی که به عنوان پارامتر متد داریم، hidden state فعل های batch داده شده استخراج می شوند. با استفاده از متد repeat آن را به اندازه طول دنباله، در بعد دوم تکرار می کنیم. به صورت زیر:

```
repeated_verb_hidden = verb_hidden_state.repeat(1, seq_len, 1)
```

در نهایت با خروجی lstm آن را concat می کنیم. به صورت زیر:

```
concat_words = torch.cat((output,repeated_verb_hidden) , dim = 2)
```

در نهایت concat_words به لایه خطی fc داده می شود و خروجی آن برگشت داده می شود.

- **Train Model**

برای آموزش مدل از این هایپرپارامتر ها استفاده کردم:

Epoch = 10

Batch_size = 64

Hidden_dim=64

Num_embeddings = Vocab Size (در بالا ذکر شد اندازه وکب چقدر است)

Embedding_dim=200

Output_size=9

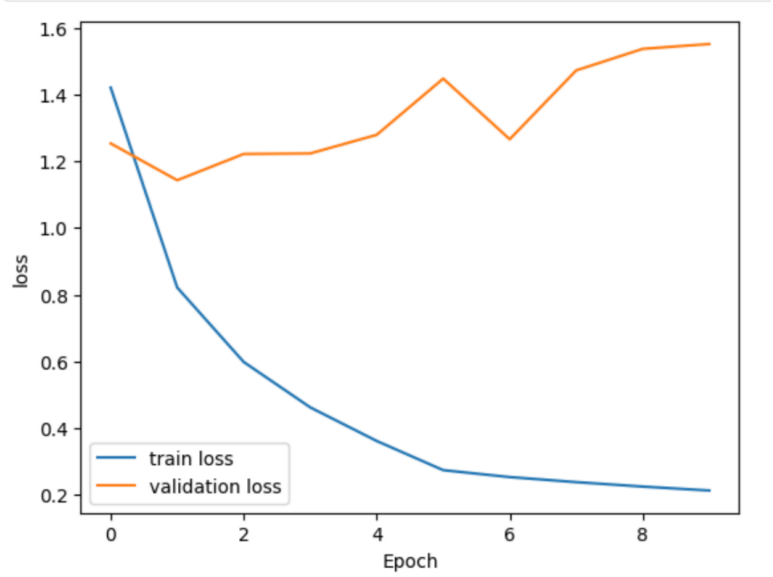
Learning_rate=2e-3

Gamma=0.2

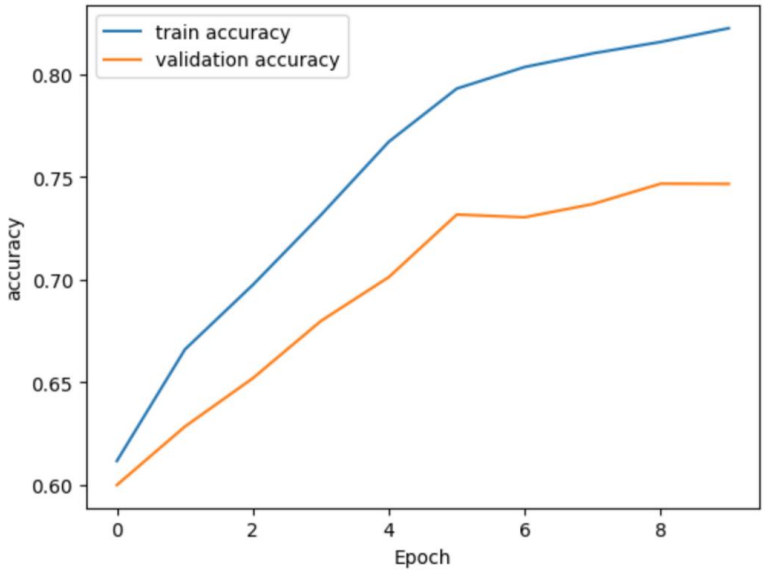
Step_size=5

دو مورد آخر را برای learning scheduler استفاده کردم.

برای loss هم از CrossEntopyLoss استفاده کردم که با استفاده از متد compute_class_weight از کتابخانه sklearn وزن هر کلاس را بدست آوردم و به عنوان پارامتر وزن به آن دادم. همچنین از بهینه ساز Nadam و scheduler=StepLR نیز استفاده کردم. در نهایت با استفاده از تابع train که آن را تعریف کرده بودم، مدل را آموزش دادم. همچنین بعد از هر ایپاک بعد از بدست آوردن دقت و خطای کلی، یک بار مدل روی داده validation ارزیابی می شود و خطا، دقت و f1 score آن محاسبه و ذخیره می شود. در نهایت این نمودار ها بعد از آموزش مدل بدست آمد:

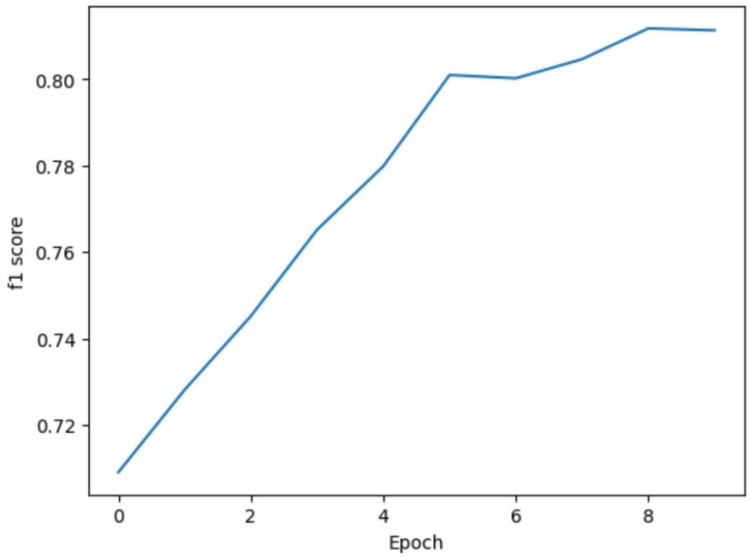


شکل ۱ نمودار **loss** در حین آموزش مدل **LSTM**



شکل ۲ نمودار **accuracy** در حین آموزش مدل **LSTM**

طبق این نمودار ها، در طول آموزش خطای train کمتر شده ولی خطای valid تقریبا صعودی بود. در عوض دقت هر دو در طول آموزش بهتر شده است. داده train در ایپاک آخر به دقت ۸۱ درصد و داده valid در ایپاک آخر به دقت حدود ۷۵ درصد رسید.



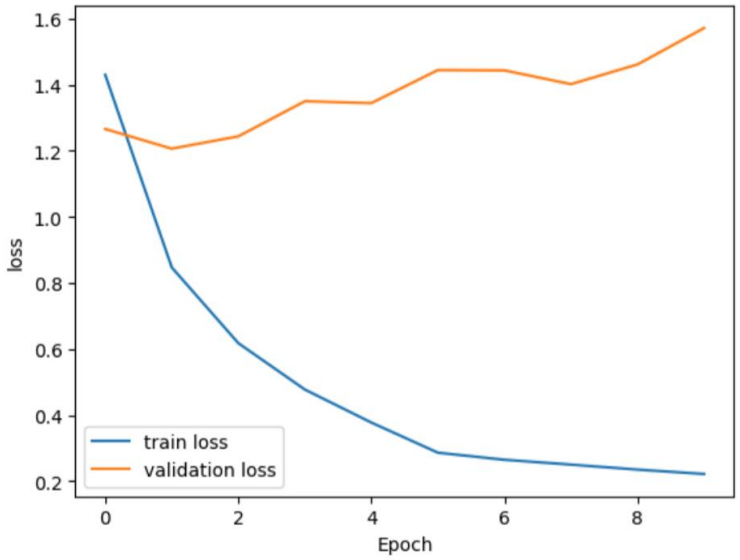
شکل ۳ نمودار f1-score مدل LSTM برای داده اعتبار سنجی

همان طور که مشاهده می شود، f1-score داده valid در طول زمان بهتر می شود و به حدود ۰.۸۱ در ایپاک آخر رسید.

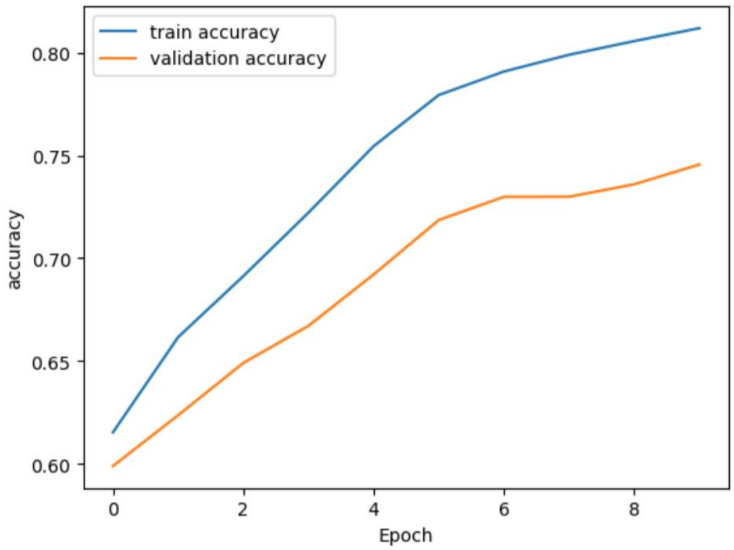
پاسخ بخش سوم: GRU Encoder Model

- **GRU Encoder**

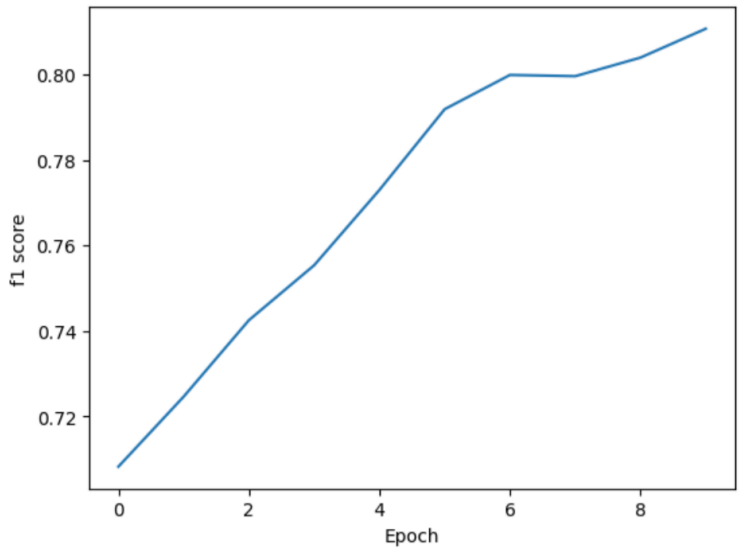
طبق چیزی که خواسته شد، این مدل دقیقا شبیه مدل قسمت قبل است. با این تفاوت که به جای lstm از gru استفاده شد. هایپر پارامتر ها را هم تغییر ندادم. نتیجه به صورت زیر حاصل شد:



شکل ۴ نمودار **loss** در حین آموزش مدل **GRU**



شکل ۵ نمودار **accuracy** در حین آموزش مدل **GRU**



شکل ۶ نمودار f1-score مدل GRU برای داده اعتبارسنجی

قسمت ۳-۲:

۱. مزیت LSTM به RNN در این است که در مدیریت وابستگی های طولانی مدت بسیار بهتر هستند و این به دلیل توانایی آنها در به خاطر سپردن اطلاعات برای مدت طولانی است. LSTM ها نسبت به مشکل ناپدید شدن گرادیان بسیار کمتر مستعد هستند. این به این دلیل است که آنها از نوع متفاوتی از عملکرد فعال سازی استفاده می کنند که به عنوان سلول LSTM شناخته می شود، که به حفظ اطلاعات در توالی های طولانی کمک می کند. در نهایت، LSTM ها در مدل سازی داده های متوالی پیچیده بسیار کارآمد هستند. این به این دلیل است که آنها می توانند نمایش های سطح بالایی را یاد بگیرند که ساختار داده ها را ضبط می کند.

۲. GRU بسیار مشابه شبکه LSTM است، با این تفاوت که به‌جای سه گیت، فقط دو گیت تنظیم مجدد (Reset Gate) و گیت به‌روزرسانی (Update Gate) دارد؛ هم‌چنین شبکه GRU چیزی به‌نام حالت سلول (Cell State) ندارد و برای انتقال اطلاعات از حالت نهان (Hidden State) استفاده می‌کند. در کل LSTM در مقایسه با GRU معماری پیچیده تری دارد. LSTM معمولاً پارامترهای بیشتری نسبت به GRU به دلیل گیت اضافی (فراموش گیت) دارد. این می‌تواند LSTM را قدرتمندتر کند، اما در عین حال مستعد بیش از حد برازش، به‌ویژه در مجموعه‌های داده کوچک‌تر است. همچنین موجب می شود تا در مقایسه با GRU، به خصوص در مجموعه داده های بزرگتر، زمان های آموزشی کمی کندتر داشته باشد. LSTM به دلیل معماری پیچیده تر، LSTM به طور بالقوه می تواند الگوها و روابط پیچیده تری را در داده ها یاد بگیرد. برای کارهایی که گرفتن وابستگی های طولانی مدت حیاتی است، مناسب است. GRU گرچه ساده تر است اما هنوز هم می تواند یاد بگیرد که وابستگی های بلندمدت را به طور موثر درک کند. در بسیاری از وظایف پردازش زبان طبیعی به خوبی عمل می کند و یک انتخاب محبوب برای کارهای مختلف مدل سازی توالی است.

۳. این مدل به نوعی وظیفه pos tagging را انجام می دهد که اختصاص هر یک از این تگ ها وابسته به فعل هم هست. بنابراین با concatenate کردن hidden state فعل با hidden state همه توکن ها، نوعی توجه به فعل داریم که این باعث می شود اگر جمله طولانی بود، فعل را فراموش نکند و برای تگ زدن هر توکن، فعل را در نظر بگیرد و در نهایت مدل بهتر آموزش ببیند و بهتر کار کند.

۴. برای این که از ناپدید شدن گرادیان ها جلوگیری کنیم چند راه حل وجود دارد:

- میتوان از توابع فعال سازی ای استفاده نمود که کمتر مستعد vanishing gradient هستند مثل ReLU و انواع آن.
- مقدار دهی اولیه صحیح وزن ها می تواند مشکل vanishing gradient را کاهش دهد.
- می توان gradient clipping انجام داد که شامل تنظیم یک مقدار آستانه برای محدود کردن اندازه گرادیان ها در حین backpropagation که از کوچک شدن یا بزرگ شدن بیش از حد آنها که منجر به exploding gradient میشود، جلوگیری می کند.
- Batch normalization: اعمال batch_normalization خروجی هر لایه را normal می کند تا میانگین ۰ و واریانس ۱ داشته باشند. این می تواند به حفظ شیب پایدار در طول فرآیند آموزش کمک کند.

پاسخ بخش چهارم: Encoder-Decoder Model

قسمت ۴-۱ و ۴-۲

- **Load Glove**

ابتدا Glove با بعد ۲۰۰ را لود کردم و مقادیر آن را در دیکشنری words ذخیره کردم.

- **Preprocess data**

لیبل های مورد نیاز را در یک مجموعه ذخیره کردم. داده ها را لود کردم.

در مرحله بعد طبق صورت مساله، لازم بود تا یک دیتاست جدید با این داده ها ساخته شود. به همین منظور ابتدا فعل هر جمله را با توجه به ستون verb_index پیدا شد و در یک ستون جدید verbs اضافه شد. در تابع create_QA_df چهار تا لیست تعریف کردم که در دیتای جدید این ۴ ستون را داشته باشیم: question, answer, question_len, answer_len. دو تای اولی همان سوال و جواب به فرمت صورت مساله است. دو تای بعدی طول سوال و طول جواب هستند که در ادامه به آن ها نیاز پیدا خواهیم کرد. این تابع دو پارامتر ورودی می گیرد. اولی base_df که همان داده پایه ای هست که از آن دیتاست جدید را بسازیم. دومی aug است که به صورت پیش فرض برابر ۲ است. علت آن این است که تعداد زیادی از جواب ها خالی و به صورت <s></s> بود و مدل در آموزش یاد گرفته بود که فقط همین را تولید کند. در صورتی که باید لیبل ها را یاد می گرفت و اگر مثلا جمله ای ARG0 داشت جواب را برگرداند. به همین خاطر من فقط در دیتاست train آن سوال هایی که جواب تهی نداشتند را ۲ بار وارد کردم که مدل بتواند این ها را بهتر یاد بگیرد. در ادامه هم بعد از این که دیتاست جدید تولید شد، دیتاست را به دو قسمت مساوی تقسیم کردم و در هر قسمت جمله ها را با توجه به طول جواب به ترتیب نزولی مرتب کردم. این کار باعث شد مدل دقت و f1 score خیلی بهتری بگیرد.

ادامه توضیحات تابع create_QA_df: در این تابع به ازای هر سطر در base_df و هر لیبلی که داریم، سوال را به فرمتی که در صورت مساله بود، تبدیل می کنیم. اگر لیبل به فرمت {B-{label در srl_frames آن بود، جواب را از روی srl_frames و text در تابع find_full_answer پیدا می کند و سطر جدید به دیتاست جدید اضافه می شود (به تعداد aug) و البته اگر دو تا جواب برای label داشته باشیم، هر دو را ذخیره می کند(به تعدا aug). در غیر این صورت هم سطر جدید ایجاد می شود ولی جواب <s></s> خواهد بود. در نهایت هم یک دیتافریم ساخته شده و برمیگرداند. این تابع را روی دیتای train با aug=2 (دلیلش را در بالا گفتم) و برای دیتای valid با aug=1. دیتای تست ستون srl_frames را نداشت و نمی شد با آن دیتاست خواسته شده را ساخت.

- **Create dataset and dataloader**

در این جا ابتدا وکب مورد نیاز را با اضافه کردن SEP_TOKEN و لیبل ها، آماده کردم. سوال و جواب ها در دیتای Train و valid را به فرمت عددی تبدیل کردم. داده train را به دو قسمت مساوی تقسیم کردم و در هر قسمت جمله ها را با توجه به طول جواب به ترتیب نزولی مرتب کردم. دلیل این کار را در زیربخش قبلی گفته بودم. در نهایت دیتاست و دیتالودر با batch_size=64 ساخته شد.

- **Create embeddings of vocab**

در این بخش با توجه به Glove بردار جانمایی وکب با همان ترتیب کلمات ساخته می شود و اگر کلمه ای در وکب در Glove وجود نداشت، بردار صفر می گذارد.

- **Implement model**

برای پیاده سازی مدل خواسته شده، کلاس های زیر را تعریف کردم (این لینک[1] تا حدی کمک کرده است):

۱. Encoder

پارامتر های این مدل padding_idx (اندیس padding)، embs_npa (بردار جانمایی کلمات)، embedding_size (سایز جانمایی)، hidden_size (اندازه hidden state)، num_layers (تعداد لایه های lstm) و p (درصد dropout) هستند. در این مدل یک لایه embedding با مقادیر اولیه ی Glove و قابل آموزش، یک lstm دوطرفه با dropout=p و دو لایه خطی داریم برای hidden و cell .

در مسیر forward بعد از این که کلمات از لایه جانمایی عبور داده شد، از torch.nn.utils.rnn.pack_padded_sequence استفاده شد که تاثیر padding ها را کمتر کند. اینجاست که از طول سوالات استفاده می شود. بعد از آن که خروجی آن از lstm عبور داده شد، از torch.nn.utils.rnn.pad_packed_sequence استفاده شد که خروجی های lstm هم اندازه باشند. در نهایت هم hidden و cell از لایه های خطی عبور داده می شوند که از بین دو مسیر رفت و برگشت هر کدام با یک وزنی انتخاب شوند. در نهایت خروجی ها را برمی گرداند.

۲. Attention

[1] https://github.com/aladdinpersson/Machine-Learning-Collection/blob/master/ML/Pytorch/more_advanced/Seq2Seq_attention/seq2seq_attention.py

این کلاس برای اجرای مکانیسم توجه ساخته شد. در این جا از یک softmax و relu و یک لایه خطی استفاده شد. در مسیر forward خروجی encoder که همان hidden state های هر کلمه است و hidden state قبلی را می گیرد. آن ها را concat می کند و از لایه خطی عبور می دهد. خروجی را از relu و سپس از softmax می گذارند و خروجی بدست آمده را با خروجی انکدر ضرب داخلی می کند و خروجی می دهد.

۳. Decoder

پارامتر های این مدل padding_idx (اندیس padding)، embs_npa (بردار جانمایی کلمات)، embedding_size (سایز جانمایی)، hidden_size (اندازه hidden state)، output size (سایز خروجی که همان تعداد وکب است)، num_layers (تعداد لایه های lstm) و p (درصد dropout) هستند. در این مدل یک لایه embedding با مقادیر اولیه ی Glove و قابل آموزش، یک lstm یک طرفه با dropout=0.2 و یک لایه خطی، یک لایه dropout و Attention داریم.

در مسیر forward ورودی را از لایه embedding و بعد از relu رد می کند. hidden state را به تعداد طول دنباله تکرار می کند و آن را به همراه خروجی انکدر به Attention میدهد. خروجی آن را با جانمایی concat می کند و به Lstm می دهد. در نهایت خروجی lstm از یک لایه خطی و dropout رد می شود و در نهایت return می شود.

۴. Seq2seq

این مدل encoder, decoder را می گیرد.

در متد forward که زمان training اجرا می شود، ابتدا ورودی را به انکدر می دهد، سپس در یک حلقه ی for به اندازه طول target، دیکدر را صدا می زند و اولین بار توکن <S> را به عنوان ورودی، طول ورودی (source_len)، خروجی انکدر و hidden و cell انکدر را به دیکدر می دهد و خروجی آن را ذخیره می کند. این مدل بر اساس teacher_force_ratio کار می کند، یعنی اگر ۱ باشد target را به دیکدر بعدی می دهد و هر چقدر کمتر باشد با همان احتمال بین target و بهترین خروجی انتخاب می کند و به دیکدر بعدی می دهد. به همین ترتیب در ادامه ی حلقه ی for، ورودی انتخاب شده، طول ورودی، خروجی انکدر و و hidden و cell دیکدر قبلی را به دیکدر بعدی می دهد و همین طور تا آخر. در نهایت خروجی را return میکند.

در این مدل دو متد دیگر برای انجام beam search برای تولید خروجی داریم. برای این کار من در متد beam_decode برای هر داده در batch متد batch_beam_decode را فراخوانی کردم. این متد را بر اساس الگوریتمی که در کتاب آقای جورافسکی موجود است (فصل ۱۳ صفحه ۲۸۱) و این لینک[2] نوشتم. با این تفاوت که یک متغیر با مقدار دیفالت 0.6 برای این متد قرار دادم، به این صورت که بهترین جواب ها را با این حتمال انتخاب کند. چون مدل احتمال زیادی به padding و end_token می داد. برای همین این کار را انجام دادم که توکن های دیگر هم شانس بالاتری داشته باشند. همچنین در آخر هم چک می کنم که تعداد جواب هایی که پیدا شد با beam_width داده شده برابر است یا خیر. اگر نبود از بین جواب های برتری که پیدا شده بود تا جایی که لازم

[2] https://www.kaggle.com/code/stevelukis/translating-with-bidirectional-lstm-beam-search

است به آن اضافه می کنم. به فرمت عددی و تنسور تبدیل می کنم و برگشت می دهم. این جواب ها همان اندیس کلمات هستند و به ترتیب بیشترین احتمال در لیست ذخیره شده اند. پس برای گرفتن بهترین جواب کافیست که اولین جمله را بگیریم.

- **Train model**

برای آموزش مدل از این هایپرپارامتر ها استفاده شد:

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

num_epochs = 12
learning_rate = 4e-3
output_size = train_vocab.__len__()

embedding_size = Dim
hidden_size = 256
num_layers = 1
enc_dropout = 0.2
dec_dropout = 0.8
```

همچنین مدل، loss و optimizer هم ساخته شد:

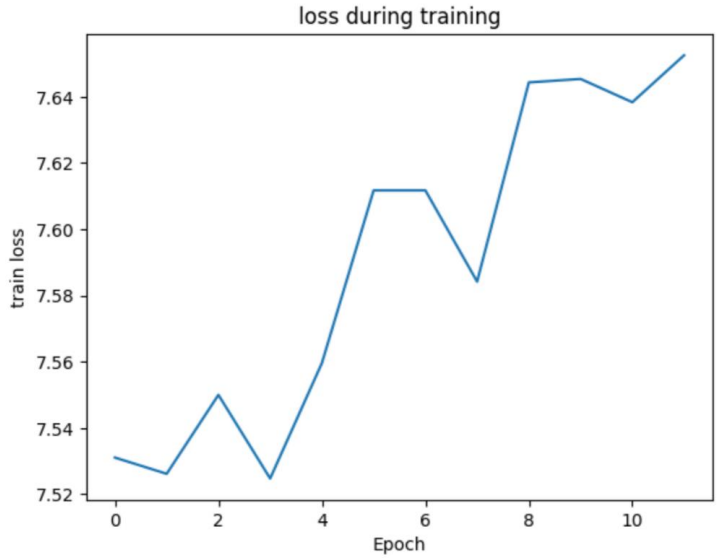
```
model = Seq2Seq(encoder_net, decoder_net, device).to(device)
optimizer = NAdam(model.parameters(), lr=learning_rate)

pad_idx = train_vocab.word2id[Pad_token]
criterion = nn.CrossEntropyLoss(ignore_index=pad_idx)
```

در اینجا تنظیم شده که loss توکن های pad را در نظر نگیرد و به این صورت محاسبات دقیق تر خواهد بود.

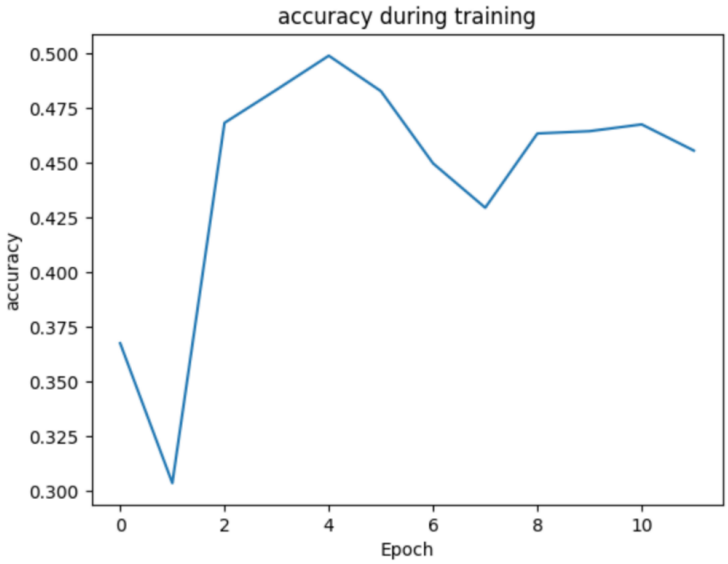
در نهایت مدل آموزش داده شد. قبل از هر optimizer.step() از

```
torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1)
```

استفاده شد که مطمئن شویم گرادیان در range مناسبی قرار دارند و از ناپدید شدن گرادیان در طول آموزش جلوگیری می کند و در نهایت این نمودار ها برای این مدل حاصل شد:



شکل ۷ نمودار loss مدل seq2seq در طول آموزش



شکل ۸ نمودار accuracy مدل seq2seq در طول آموزش

همان طور که مشاهده می شود خطا در حدود 7.65 و دقت در حدود ۴۶ درصد رسید. همچنین f1-score آن را هم حساب کردم که حدود ۵۸ درصد شد.

قسمت ۴-۳: برای مجموعه داده valid، هر batch این داده را به model.beam_decode دادم. خروجی همان طور که گفته شد به ترتیب از احتما بالا به پایین پر شده است. پس اولین خروجی هر batch ذخیره شد و در نهایت f1-score محاسبه شد و 0.89 بدست آمد.

قسمت ۴-۴:

سوال ۱: محدودیت های آن این است که مدل پیچیده تر شده است. به ram بیشتر و برای سرعت بیشتر به gpu نیاز است. در واقع فرآیند آموزش کند تر شده است. چالش های بیشتری دارد. چون تعداد سوال هایی که بدون جواب بود زیاد بود و باید این مساله حل می شد. در واقع به داده های بیشتری نیاز است که به اندازه کافی تنوع نقش های معنایی را پوشش دهد که این می تواند چالش برانگیز باشد. ممکن است داده های کافی برای پوشش دادن همه تفاوت های ظریف SRL وجود نداشته باشد که منجر به مشکلات پراکندگی می شود. SRL شامل شناسایی ساختار یک جمله است که می تواند پیچیده و با جزئیات باشد. تبدیل آن به قالب QA ممکن است این روابط را بیش از حد ساده کند یا به تعداد زیادی الگوی سؤال برای پوشش همه نقش‌های ممکن نیاز داشته باشد. تبدیل از SRL به QA ممکن است منجر به از دست دادن برخی نشانه ها و جزئیات های زبانی شود که برای درک نقش های درون یک جمله مهم هستند.

سوال ۲: این توکن ها به وضوح شروع و پایان یک دنباله را مشخص می کنند، که برای مدل بسیار مهم است تا بفهمد خروجی باید از کجا شروع شود و کجا متوقف شود. در طول آموزش، این توکن ها اهداف ثابتی را برای مدل در موقعیت های اول و آخر خروجی فراهم می کنند و به آن کمک می کنند شروع و پایان فرآیند تولید را بیاموزند.

پاسخ بخش پنجم: تحلیل

قسمت ۵-۱: برای این بخش یعنی مقایسه کمی دو مدل، داده های validation را بر اساس semantic role ها تقسیم می کنیم (تابع create_label_df) و هر کدام از مدل ها را جداگانه روی این داده ها اجرا می کنیم و f1-score آن ها را مقایسه می کنیم.

۰. برای لیبل ARGM-LOC مدل lstm بهتر عمل کرده است:

Lstm f1-score = 0.8

Encoder-decoder = 0.7

۱. برای لیبل ARG2 مدل encoder-decoder بهتر عمل کرده است:

Lstm f1-score = 0.8

Encoder-decoder = 0.86

۲. برای لیبل ARGM-TMP هر دو مدل عملکرد نسبتا خوبی داشتند و f1-score آن ها برابر شد:

Lstm f1-score = 0.81

Encoder-decoder = 0.81

۳. برای لیبل ARG0 مدل encoder-decoder بهتر عمل کرده است:

Lstm f1-score = 0.82

Encoder-decoder = 0.86

۴. برای لیبل ARG1 مدل encoder-decoder بهتر عمل کرده است:

Lstm f1-score = 0.81

Encoder-decoder = 0.86

همان طور که از مقایسه f1-score ها برای هر مدل روی لیبل های جداگانه مشاهده می کنید، به جز لیبل ARGM-LOC و ARGM-TMP مدل encoder-decoder بهتر از lstm بوده. البته برای این که مشاهده کرد آیا این تفاوت معنادار است یا خیر نیاز به آزمون های آماری داریم.

قسمت ۵-۲: برای مقایسه کیفی این دو سطر داده valid را انتخاب کردم:

[67]:

	index	text	srl_frames	verb_index	verbs
0	4	[Because, ,, as, everyone, knows, ,, our, Disn...	[O, O, O, B-ARG0, O, O, O, O, O, O, O, O, O...	4	knows
1	5	[They, add, to, what, we, already, have, ,, li...	[O, O, O, B-ARG1, B-ARG0, B-ARGM-TMP, O, O, O,...	6	have

چون تعداد صفر ها در پاسخ زیاد است دقت بالا شده است ولی مدل توانسته در جمله اول ARG0 و در جمله دوم هم ۳ تا از لیبل ها را درست پیش بینی کند. هر چند در بعضی موارد آنهایی که O بودند را اشتباه برچسب زده است.

به مدل encoder-decoder هم همین دو جمله را دادم ولی اصلا نتوانست حتی یکی از لیبل های غیر از O را تشخیص دهد. این نشان می هد در این تسک عملکرد مدل LSTM بهتر است. علت آن این است که در QA دچار پراکندگی داده ها شدیم و مدل پیچیده تر شده است. در حالی که در مدل LSTM، مدل ساده تر است و به داده کمتری نیاز داریم. همچنین توالی کلمات و جایگاه آن ها در جمله، نشانه ای برای برچسب آن است. در حالی که در مدل encoder-decoder این گونه نیست.