به نام خدا

پدرام رستمی – ۸۱۰۱۰۰۳۵۳

تمرین سوم پردازش زبان طبیعی

تعیین نقش کلمات

الف – تمام دیتاست‌های مربوط به POS tagging شامل برچسب‌های یکسانی نیستند و ممکن است تعداد یا تنوع برچسب‌هایشان با همدیگر فرق کند. مجموعه برچسب‌های universal برای حل این معضل به وجود آمده‌اند. این مجموعه شامل ۱۲ برچسب هستند و در کتابخانه‌ی NLTK می‌توان از این مجموعه برای برچسب‌گذاری تمام دیتاست‌های مربوط به POS tagging استفاده کرد.

ب – در این بخش جملات تمام فایل‌های مجموعه داده‌ی Treebank را خوانده و سپس با استفاده از دستور train_test_split مقدار ۸۰ درصد داده‌ها را برای آموزش انتخاب کرده و از ۲۰ درصد باقی مانده هم ۱۵ درصد را برای تست و ۵ درصد را برای اعتبارسنجی انتخاب می‌کنیم.

پ – پیاده سازی الگوریتم Viterbi در فایل Q1.ipynb آمده است. در سلول مربوط به پیاده سازی این الگوریتم عینا از الگوریتم اصلی سودو کد آن استفاده شده است که در شکل ۱ قابل مشاهده است. دقت این الگوریتم بر روی داده‌های تست برابر ۹۳,۳۸ درصد به دست آمد.

Algorithm 2 Viterbi Algorithm($\{o_1, ..., o_T\}, \{q_0, q_1, ..., q_N, q_F\}$)

Create a table `viterbi`[T][N+2] ▷ Create DP table
for each state q from 1 to N **do** ▷ Initialization
    `viterbi`[1][q] $= p(q|q_0) \times p(o_1|q)$
    `backpointer`[1][q] $= 0$
end for
for $t = 2$ to T **do** ▷ DP recursion step
    for each state q from 1 to N **do**
        `viterbi`[t][q] $= p(o_t|q) \cdot \max_{q'=1}^{N}$ `viterbi`$[t-1][q'] \cdot p(q|q')$
        `backpointer`[t][q] $= \arg\max_{q'=1}^{N}$ `viterbi`$[t-1][q'] \cdot p(q|q')$
    end for
end for
`viterbi`[T][q_F] $= \max_{q=1}^{N}$ `viterbi`$[T][q] \cdot p(q_F|q)$ ▷ Termination
`backpointer`[T][q_F] $= \arg\max_{q=1}^{N}$ `viterbi`$[T][q] \cdot p(q_F|q)$
return the best path by following the backtrace of the backpointers through time

شکل ۱- سودو کد الگوریتم Viterbi

ت – در مجموعه داده‌های تست، داده‌ی ۵۴ کلمه‌ای است دارای ۳ خطا می‌باشد. با بررسی ۳ خطای این جمله می‌توانیم به دلایلی که ممکن است خطا رخ دهد بپردازیم. اولین کلمه‌ای که در این جمله به اشتباه تشخیص داده شده است، کلمه‌ی 'appropriators' است. دلیل وقوع خطا برای این کلمه این است که این کلمه در مجموعه داده‌های آموزش آورده نشده بود. ۲ کلمه‌ی دیگری که به اشتباه تشخیص داده شده اند، کلمات overseas و private هستند که هر دو در کلمات مجموعه داده‌ی آموزش وجود داشتند. برچسب‌های واقعی این دو کلمه 'NOUN' بوده است و با کمی بررسی بیشتر متوجه می‌شویم که دلیل برچسب دهی اشتباه به این دو کلمه این بوده است که در مجموعه داده‌ی آموزش، هیچ کدام با برچسب 'NOUN' وجود نداشته اند (در شکل ۲ نتایج این بررسی‌ها قابل مشاهده است). پس می‌توان گفت در این روش اکثر اشتباهات مربوط به کلماتی می‌شود که یا در مجموعه داده‌ی آموزش وجود نداشته‌اند یا در بعضی از نقش‌هایی که در مجموعه داده‌ی تست داشته اند در مجموعه داده‌ی آموزش نداشته اند.

```
print('appropriators' in all_words)
print('overseas' in all_words)
print('private' in all_words)
print(emission_freq['NOUN']['overseas'])
print(emission_freq['NOUN']['private'])
```

[67] ✓ 0.5s

```
False
True
True
0.0001
0.0001
```

*شکل ۲ – بررسی مربوط به خطاهای الگوریتم **Viterbi***

ث – استراتژی برخورد با کلمات ناشناخته در این الگوریتم بسیار مهم است. اگر از احتمال صفر برای برخورد با کلمات ناشناخته استفاده شود، تمام سطرهای ستون مربوط به آن کلمه صفر شده و در نتیجه تمام ستون‌های بعد از آن (که مربوط به کلمات بعدی است) هم صفر می‌شوند و وجود یک کلمه‌ی ناشناخته باعث می‌شود الگوریتم برای تمام کلمات بعد از آن هم پیشبینی اشتباهی داشته باشد. برای جلوگیری از همچین مشکلی از یک احتمال بسیار کم (احتمال اپسیلون که دراین پروژه برابر است با ۰,۰۰۰۰۱) استفاده شده است. همچنین ممکن است در مجموعه داده‌ی تست بعضی کلمات نقش‌هایی داشته باشند که آن نقش‌ها را در مجموعه داده‌ی آموزش نداشتند.

در صورتی که این احتمال هم صفر در نظر گرفته شود، باز هم همان مشکل قبلی پیش می‌آید. برای حل این مشکل هم باز از احتمال اپسیلون برای همچین شرایطی استفاده می‌شود. دقتی که برای این الگوریتم در بخش **پ** ذکر شد (۹۳,۳۸ درصد) با در نظر گرفتن احتمال اپسیلون بوده است.

ج – در این بخش مدل RNN برای مسئله‌ی POS Tagging طراحی می‌شود. این مدل شامل یک لایه‌ی embedding ، یک لایه RNN و یک لایه‌ی Linear است. در این مدل سایز بردارهای embedding و سایز لایه‌ی hidden به عنوان پارامتر به مدل داده می‌شوند. برای بررسی تاثیر سایز لایه‌ی hidden، سایز این لایه را ۳ مقدار ۶۴، ۱۲۸ و ۲۵۶ می‌دهیم. سایز بردارهای embedding در هر ۳ حالت برابر ۲۵۶ در نظر گرفته شده است. همچنین در هر سه حالت مدل به اندازه‌ی ۱۰ ایپاک آموزش دیده و در صورتی که در دو ایپاک از بهترین عملکردش بهتر نشود، آموزش متوقف می‌شود (early stopping با مقدار patience برابر با ۲)

اهمیت استفاده از داده‌ی validation این است که داده‌ی تست باید دست نخورده باقی بماند و تنها برای تست نهایی مدل استفاده شود. پس نیاز به مجموعه داده‌ی دیگری است تا مطمئن شویم مدل بر روی داده‌های آموزش overfit نمی‌شود یا هایپر پارامترهای مدل به خوبی تنظیم شده اند. به این دلایل باید از مجموعه داده‌ی validation استفاده کنیم و عملکرد مدل را بعد از هر ایپاک بر روی این مجموعه داده بسنجیم.

چ – دقت مدل در حالتی که سایز لایه‌ی hidden برابر ۶۴ بود برابر ۹۰,۹ درصد، در حالتی که سایز این لایه برابر با ۱۲۸ بود برابر ۹۱,۷۷ درصد و در حالتی که سایز این لایه برابر با ۲۵۶ بود برابر با ۹۱,۰۰ درصد بود. با توجه به نتایج می‌توان به این نتیجه رسید که سایز لایه‌ی hidden تاثیر خیلی زیادی بر روی عملکرد مدل نخواهد داشت. در حالی که برای مثال سایز بردارهای embedding تاثیر قابل توجه‌تری بر روی عملکرد مدل خواهد گذاشت.

ح – مدل طراحی شده در بخش **ج** را با لایه‌های LSTM و GRU هم تست می‌کنیم. نتیجه‌ی این تست‌ها در جدول ۱ قابل مشاهده است. همانطور که در این جدول مشخص است، مدل LSTM بهترین عملکرد و دقت را داشته و همچنین بهترین سایز برای لایه‌ی پنهان ۱۲۸ بوده است.

جدول ۱- مقایسه‌ی عملکرد مدل‌های RNN ، LSTM و GRU با سایزهای متفاوت لایه‌ی پنهان

	Hidden dim=64	Hidden dim=128	Hidden dim=256	average
RNN	90.90%	91.77%	91.00%	91.22%
LSTM	92.12%	93.81%	93.86%	93.26%
GRU	91.03%	90.79%	91.22%	91.01%
average	91.35%	92.14%	92.02%	

مزیت مدل‌های LSTM و GRU نسبت به مدل RNN این است که تاثیر کلمات پیشین را برای مدت بیشتری حفظ می‌کند. در حالی که در مدل RNN تنها چند کلمه‌ی آخر تاثیر گذار هستند و کلمات قبل‌تر خیلی زود تاثیرشان را از دست می‌دهند (مشکل vanishing gradient). در این مسئله شاید تاثیر کلمات پیشین با فاصله‌ی زیاد تاثیر خیلی زیادی در نقش کلمه‌ی فعلی نداشته باشد. برای همین عملکرد مدل RNN تقریبا مانند عملکرد مدل GRU است ولی مدل LSTM عملکرد بهتری از هر دوی آن‌ها داشته است.

خ – LSTM دارای ۳ گیت ورودی، فراموشی و خروجی است. گیت ورودی مشخص می‌کند که چه داده‌هایی در حافظه‌ی بلند مدت ذخیره شود. این گیت تنها با ورودی همان لحظه و حافظه‌ی کوتاه مدت گام قبلی کار می‌کند. گیت فراموشی مشخص می‌کند که چه اطلاعاتی در حافظه‌ی بلند مدت بماند و چه اطلاعاتی پاک شود و این کار را با ضرب برداری که به وسیله‌ی ورودی آن لحظه و بردار حافظه‌ی کوتاه مدت به دست می‌آید در بردار حافظه‌ی بلند مدت انجام می‌دهد. گیت خروجی هم با توجه به ورودی آن لحظه و بردار حافظه‌ی کوتاه مدت قبلی و بردار حافظه‌ی بلند مدت به روز شده، بردار حافظه‌ی کوتاه مدت جدید را درست می‌کند.

GRU دارای ۲ گیت به روز رسانی و بازنشانی (reset) است. گیت به روز رسانی مشخص می‌کند که چه مقدار از اطلاعات گذشته به آینده منتقل شود. این گیت می‌تواند تمام داده‌های گذشته را منتقل کند یا هیچ کدام از داده‌های گذشته را منتقل نکند. گیت بازنشانی هم مشخص می‌کند که چه مقدار از داده‌های گذشته نادیده گرفته شوند.

LSTM نسبت به GRU دارای یک گیت بیشتر است. همین باعث می‌شود که عملکرد آن دقیق‌تر از GRU باشد ولی گیت‌های کمتر GRU باعث می‌شود پارامترهای کمتری نسبت به LSTM برای آموزش داشته باشد.

د – داده‌های مجموعه داده‌ی Treebank در مجموع شامل ۳۹۱۴ جمله است. این تعداد جمله برای الگوریتم‌های آماری و یادگیری ماشین مناسب است ولی برای مدل‌های شبکه‌ی عصبی، داده‌های بزرگتر باعث عملکرد بهتر آن‌ها می‌شود. در نتیجه نمی‌توان انتظار داشت که عملکرد مدل‌های شبکه‌های عصبی خیلی بهتر از Viterbi باشد. همچنین در مدل‌های شبکه‌ی عصبی از embedding های pre-train استفاده نشده است و خود لایه‌ی embedding هم باید آموزش ببینند. در نتیجه فرایند آموزش طولانی‌تر می‌شود. در حالی که مدل‌های بخش پ به دلیل کمبود منابع محاسباتی تنها ۱۰ ایپاک آموزش دیده‌اند و شاید اگر بیشتر آموزش داده می‌شدند عملکردشان هم بهتر می‌شد.

تشخیص گروههای اسمی

ب – در این بخش هم مانند بخش قبل از احتمال اپسیلون برای مشاهده‌ی حالت‌های پیشبینی نشده استفاده شده است تا از رخداد توالی‌های غیر ممکن هم جلوگیری شود و رخداد غیر ممکن فقط یک بار و برای همان کلمه اتفاق بیفتد. تنها تغییر این الگوریتم نسبت به الگوریتم سوال اول این است که از tag های BIO به جای tag ها POS استفاده کرده ایم.

پ – در شکل ۳ عملکرد مدل Viterbi در معیارهای recall، precision و F1 آمده است. با توجه به نامتوازن بودن کلاس‌ها، میانگین macro ی امتیازات معیار مناسبی نیست و باید از میانگین وزن دار آن‌ها استفاده شود.

	precision	recall	f1-score	support
I-FACILITY	0.89	0.97	0.93	166
B-LOCATION	0.72	0.84	0.78	89
B-PERSON	0.76	0.58	0.66	5960
I-ORGANIZATION	0.58	0.58	0.58	2865
I-GPE	0.72	0.93	0.81	555
I-LOCATION	0.74	0.79	0.77	89
I-PERSON	0.58	0.60	0.59	2983
B-GPE	0.82	0.74	0.77	5131
O	0.98	0.99	0.98	228479
B-FACILITY	0.82	0.98	0.89	168
B-GSP	1.00	0.38	0.55	77
I-GSP	0.00	0.00	0.00	2
B-ORGANIZATION	0.64	0.56	0.60	4462
accuracy			0.96	251026
macro avg	0.71	0.69	0.69	251026
weighted avg	0.96	0.96	0.96	251026

*شکل ۳- عملکرد مدل **Viterbi** در معیارهای مختلف*

ت – استفاده از مدل‌های بازگشتی به صورت عادی عملکرد مناسبی برای این مسئله ندارد. زیرا بر خلاف مسئله‌ی POS tagging که نقش هر کلمه تنها به کلمات قبلی‌اش وابسته است، در مسئله‌ی NER محتوای جمله در تمام آن پخش می‌شود و برای حدس اینکه آن کلمه named entity هست یا نه نیاز به دانستن محتوای کلمات بعدی هم هست.

برای حل این مسئله می‌توان از مدل‌های بازگشتی دو طرفه استفاده کرد (برای مثال مدل bi-lstm). در این شبکه‌ها یک rnn از سمت چپ شروع به پردازش اطلاعات می‌کند و یک rnn از سمت راست. در این صورت ما همزمان هم اطلاعات کلمه‌های گذشته و هم اطلاعات کلمات آینده را داریم.