به نام خدا

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

پردازش زبان‌های طبیعی

تمرین سوم

نام و نام خانوادگی : حسین سیفی

شماره دانشجویی : ۸۱۰۱۰۰۳۸۶

اردی‌بهشت ۱۴۰۱

۱-تعیین نقش کلمات

الف

تفاوت بارگزاری کردن مجموعه داده Penn Treebank با استفاده از tagset=”universal” و بدون استفاده از آن در تعداد برچسب‌ها و جزئیات در انتخاب برچسب برای کلمات می‌باشد. در صورتی که مجموعه داده به صورت عادی بارگزاری شود تعداد برچسب‌ها برابر ۴۶ می‌باشد و در صورتی که در حالت universal بارگزاری شود برچسب‌ها و نقوش کلمات در جمله ساده سازی می‌شود و تعداد برچسب‌ها به ۱۲ عدد کاهش می‌یابد.(به اضافه یک برچسب آغاز جمله(START) که به صورت دستی اضافه می‌کنیم برابر ۱۳ می‌شود.) طبیعتا در حالت universal جزئیات بسیار کمتری نسبت به حالت عادی در برچسب زنی وجود دارد، برای مثال در حالت عادی به دسته‌های متفاوتی از جمله VBD و VBG تقسیم می‌شوند ولی در حالت universal تنها با یک برچسب verb مشخص می‌شوند. به عنوان نمونه جمله زیر با دو حالت برچسب زنی متفاوت قابل مشاهده است:

Such belts already are required *-89 for the vechiles’ front seats.

1. Normal tags: 'JJ', 'NNS', 'RB', 'VBP', 'VBN', '-NONE-', 'IN', 'DT', 'NNS', 'POS', 'JJ', 'NNS', '.'
2. Universal tags: 'ADJ', 'NOUN', 'ADV', 'VERB', 'VERB', 'X', 'ADP', 'DET', 'NOUN', 'PRT', 'ADJ', 'NOUN', '.'

در حالت عادی قابل مشاهده است که اسامی جمع دارای برچسب متفاوتی با اسامی مفرد هستند(قابل مشاهده در بقیه جملات مجموعه داده) در حالی که در حالت universal تنها یک برچسب برای اسم‌ها وجود دارد. در ادامه قابل مشاهده است که در حالت عادی دو بخش فعل موجود در جمله فوق برچسب‌های متفاوتی دارند(VBNوVBP) اما در حالت universal هر دو با برچسب verb شناخته می‌شوند.

این مجموعه داده به صورت زیر بارگزاری شده است:

```
dataset = list(treebank.tagged_sents())

uni_dataset = list(treebank.tagged_sents(tagset='universal'))
```

ب

داده‌ها را به کمک تابع زیر می‌توان ابتدا به دو بخش آموزش و تست تقسیم کرد و سپس بخش Validation را از یکی از دو دسته ایجاد شده استخراج کرد. برای ما اهمیت دارد که قرار دادن این دادگان در دسته‌های متفاوت کاملا تصادفی باشد به این معنی که هیچ داده‌ای برای بهبود عملکرد مدل در دسته خاصی قرار نگیرد، همچنین به هیچ عنوان این سه دسته نباید اشتراکی داشته باشند.

```
from random import gauss

def split(ds, train_size):
    train = list()
    test = list()
    for sen in uni_dataset:
        if gauss(0,1) > train_size:
            test.append(sen)
        else:
            train.append(sen)
    return train,test
```

درصد در نظر گرفته شده برای داده‌های Train، Test و Validation به ترتیب برابر ۶۰ درصد، ۲۰ درصد و ۲۰ درصد می‌باشد اما قابل ذکر است که بخش Validation برای الگوریتم اول(Viterbi) کاربردی ندارد و می‌توان از آن به عنوان داده آموزش در کنار داده‌های آموزش استفاده کرد.

پ

برای برچسب زنی نقش کلمات الگوریتم ویتربی(Viterbi) انتخاب شده است. پیش از اجرای این الگوریتم بر روی داده تست نیاز داریم تا به کمک مجموعه داده‌ای برای آموزش، ماتریس احتمال توالی هر دو برچسب مشخص و احتمال تولید هر کلمه از هر برچسب را ایجاد کنیم.(شبه کد این بخش‌ها بسیار ساده است و در گزارش ذکر نشده است) سپس با استفاده از ماتریس‌های ایجاد شده و شبه کد زیر، برچسب‌های نقش کلمات را برای هر جمله پیش‌بینی می‌کنیم:

```
Function Viterbi(input of length T,state graph of len N) returns best-path
    Create a path probability matrix viterbi[N+2,T]
    For each state s from 1 to N do:
        viterbi[s,1] = transition[0,s] * emission[s,In_1]
        backpointer[s,1] = 0
    For each word w from 2 to T do:
        For each state s from 1 to N do:
            If In_w in words:
                viterbi[s,w] = max^N_{i=1}(viterbi[I,w-1] * transition[i,s] * emission[s,In_w])
                backpointer[s,w] = argmax^N_{i=1}(viterbi[I,w-1] * transition[i,s] * emission[s,In_w])
            else:
                viterbi[s,w] = max^N_{i=1}(viterbi[I,w-1] * transition[i,s] * emission[s,"[UNK]"])
                backpointer[s,w] = argmax^N_{i=1}(viterbi[I,w-1] * transition[i,s] * emission[s, "[UNK]"])
    viterbi[q_f,t] = max^N_{i=1}(viterbi[s,T] * transition[s,q_f])
    backpointer[q_f,t] = argmax^N_{i=1}(viterbi[s,T] * transition[s,q_f])
    return backtrace path from backpointer matrix
```

دقت پیاده سازی الگوریتم Viterbi که با استفاده از ۸۰ درصد مجموعه داده Penn treebank آموزش دیده است و بر روی ۲۰ درصد این مجموعه داده تست شده است، برابر ۰.۸۹۵ درصد می‌باشد.

ت

نقش برخی کلمات در داده‌های تست به دلایل متفاوت درست تشخیص داده نشده‌اند و برچسب اشتباه برای آنها انتخاب شده است. در ادامه چند مورد از این خطاهای پیش آمده را بررسی خواهیم کرد.

۱. کلمه Next در ابتدای یکی از جمله‌های مجموعه تست، در دسته Det قرار گرفته است در حالی که می‌بایست در دسته Adj قرار می‌گرفت. در این مثال چون کلمه Next در ابتدا جمله ظاهر شده است، هیچ دید درستی از برچسب کلمه قبلی وجود ندارد و در نتیجه بخش Prior برای آن به شکل تقریبا تصادفی درمی‌آید. اگر این کلمه در میانه یک جمله واقعی دیده می‌شد احتمال کمی وجود داشت که برچسب آن به درستی تشخیص داده نشود. اتفاقی مشابه برای کلماتی که بعد از کلمه‌ای با کلاس X ظاهر می‌شوند

نیز می‌افتد و همچنین به طور کلی انتخاب کلاس برای کلمات ابتدا جمله دارای خطای بسیار زیادی است و بیشتر کلماتی که در این جایگاه برچسب صحیح می‌خورند در یکی از کلمات دارای احتمال بسیار بیشتری نسبت به بقیه کلاس‌ها هستند.

۲. کلمه right در به اشتباه در دسته Noun قرار گرفته و کلاس صحیح Adv می‌باشد. این خطا می‌تواند به این دلیل باشد که نقش انتخاب شده برای کلماتی که احتمال تولید بالایی در هیچکدام از دسته‌ها ندارند(توزیع یکنواخت(Uniform) دارند) و یا به عبارتی دیگر می‌توانند نقوش متفاوتی را بپذیرند، تحت تاثیر جایگاه آن کلمه در جمله و برچسب انتخاب شده برای کلمه پیش از آن قرار می‌گیرد.

۳. گرچه در داده‌های تست چنین موردی دیده نشد اما برخی اسامی خاص نیز می‌تواند در کلاس اشتباه قرار بگیرند چرا که با احتمالی زیاد، پیش از این در داده‌های آموزش دیده نشده‌اند.

ث

روشی که برای برخورد با کلمه‌های ناشناخته(کلماتی مانند اسامی خاص که در مجموعه داده‌ی آموزش دیده نشده) انتخاب شده بدین شکل است که توکن [UNK] را مجموعه کلمات دیده شده در مجموعه داده اضافه کردیم و احتمال تولید(Emission) این توکن را توسط هر کدام از کلاس‌ها محاسبه کردیم. با توجه به اینکه توکن اضافه شده در مجموعه داده آموزش دیده نشده است(احتمال دیده شدن آن بسیار پایین است)، احتمال محاسبه شده برابر مقدار هموار شده(Smoothed) به ازای فرکانس صفر می‌باشد و عدد بسیار کوچکی است اما نکته مثبت در اضافه کردن توکن ذکر شده این است که با هر احتمال غیر صفری از تولید این توکن از کلاس‌های متفاوت، ارزش توالی برچسب‌ها در یک جمله در نظر گرفته می‌شود. جمله قبل بدین معنی است که در صورتی که این کلمه در نظر گرفته نمی‌شد و هموار سازی انجام نمی‌گرفت، احتمال تولید آن از هر کلاس برابر صفر می‌شد و در ادامه حاصلضرب تولید کلمه در انتقال(Transition) از یک کلاس به کلاس دیگر برابر صفر می‌شد و در نتیجه هیچ تفاوتی بین انتخاب یک کلاس با دیگری وجود نداشت. با پیاده‌سازی این روش درست است که اطلاعات خاصی از کلمه تولید شده برای برچسب زدن آن نمی‌توانیم به دست بیاوریم اما توالی کلاس‌ها در جمله‌های دیده شده در مجموعه داده آموزش می‌تواند به ما کمک کند. برای مثال می‌دانیم که احتمال ظهور یک صفت(ADJ) یا اسم(NOUN) بعد از یک صفت بیشتر از بقیه کلاس‌ها است و در نتیجه انتخاب کلاس دقیق‌تری را می‌توان انجام داد.

ج

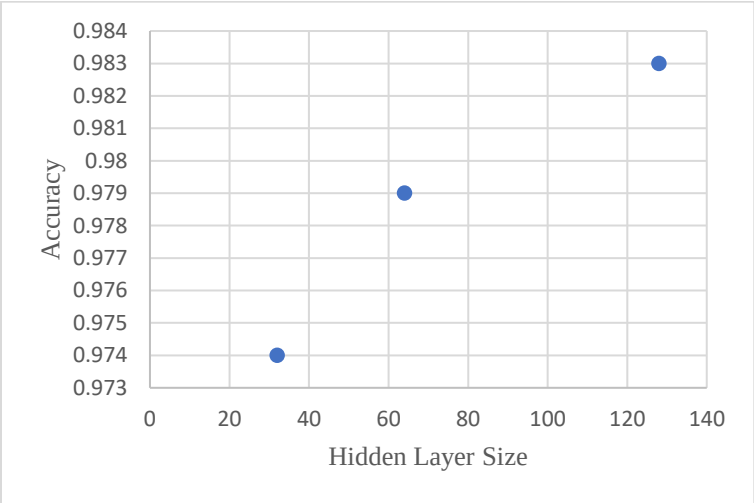
برای پیاده سازی شبکه‌های عصبی بازگشتی RNN با استفاده از زبان Python به کتابخانه Tensorflow نیاز داریم. پیش از آموزش مدل مورد نظر، باید طول جملات را به کمک Padding(برای جملات کوتاه‌تر از مقدار مورد نظر) و Truncating(برای جملات طولانی‌تر از مقدار مورد نظر) یکسان کنیم و کلمات موجود در مجموعه داده Penn treebank را به کمک روش Word2Vec یا Glove به Embeddingهایی تبدیل کنیم(در این تمرین از روش Word2Vec استفاده شده است) و در نتیجه هر یک از جملات موجود در این مجموعه داده به عنوان یک توالی از Embeddingها در نظر گرفته می‌شوند. سپس به سه دسته آموزش، تست و Validation تقسیم می‌شوند و برای هر کدام از این دسته‌ها به ترتیب ۶۰ درصد، ۲۰ درصد و ۲۰ درصد داده‌ها در تخصیص داده می‌شود.

در این بخش ۳ مدل RNN با اندازه لایه نهان(Hidden Layer) ۳۲، ۶۴ و ۱۲۸ ایجاد شده‌اند و با داده‌های یکسانی آموزش داده شده و تست شده‌اند. در این سه مدل سه تفاوت دیده می‌شود. تفاوت اول این است که به نظر می‌رسد با افزایش اندازه لایه نهان از ۳۲ تا ۱۲۸ زمان آموزش مدل افزایش می‌یابد، تفاوت دوم نیز افزایش مقدار دقت است و تفاوت سوم کاهش مقدار Loss می‌باشد. به عبارتی دیگر، مدت زمان آموزش و دقت مدل RNN با اندازه لایه نهان آن رابطه مستقیم و با مقدار Loss رابطه عکس دارد.

همچنین با استفاده از داده‌های Validation می‌توان مطمئن شد که مدل شبکه عصبی ایجاد شده داده‌های آموزش را در حافظه ذخیره نکرده است و در واقع بخشی از جنبه‌ها و ویژگی‌های با معنی آن‌ها را یاد گرفته است و در آینده می‌توان از این مدل روی داده‌ها دیده نشده و تست به خوبی عمل کند یا به عبارتی دیگر می‌توان عمومی سازی(Generalization) در این حوزه انجام دهد.

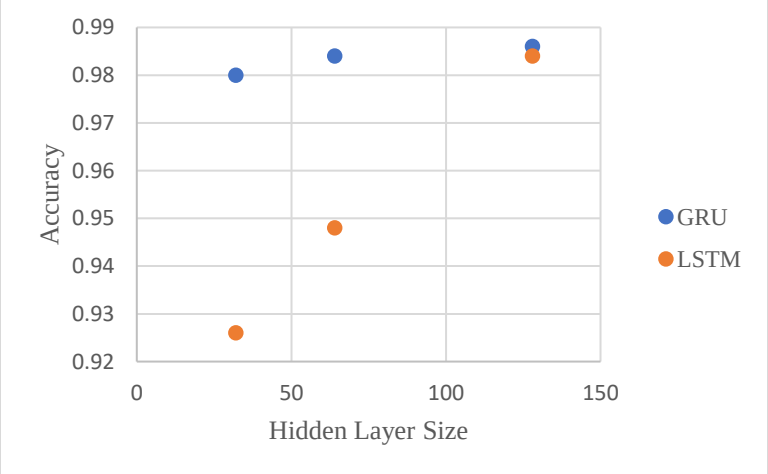
ج

دقت به دست آمده برای مدل‌هایی با اندازه لایه نهان متفاوت در نمودار زیر قابل مشاهده است:



ح

شبکه‌های عصبی LSTM و GRU نیز همانند RNN با اندازه‌های متفاوت لایه نهان بررسی شدند و نتایج مشابهی با RNN از نظر مقدار دقت، Loss و زمان آموزش مدل به دست آمد اما نکته قابل توجه در این مدل‌های تاثیرگذاری ناچیز اندازه لایه نهان در مدل GRU و تاثیر نسبتا شدیدتر آن روی مدل LSTM می‌باشد. نمودار زیر مقادیر دقت را نسبت به اندازه لایه نهان برای هر یک از مدل‌های LSTM و GRU نشان می‌دهد.



خ

شبکه‌های عصبی LSTM دارای سه گیت زیر می‌باشند:

۱. **گیت Input**: این گیت برای به‌روزرسانی اطلاعات موجود تعبیه شده است. اطلاعات ورودی گام جدید، به‌همراه اطلاعات لایه نهان گام قبلی، به این گیت وارد می‌شوند و از تابع فعالساز عبور می‌کنند تا این تابع تصمیم بگیرد کدام اطلاعات (نزدیک به صفر) دور انداخته و کدام (نزدیک به ۱) به‌روزرسانی شوند. همچنین اطلاعات ورودی گام جدید، به‌همراه اطلاعات لایه نهان گام قبلی، به تابع

تانژانت هایپربولیک وارد می‌شوند تا مقادیرشان بین -۱ تا ۱ قرار بگیرد. درنهایت خروجی تابع فعالساز و تانژانت هایپربولیک با هم ضرب می‌شوند تا تابع فعالساز تصمیم بگیرد چه مقادیری از خروجی تابع تانژانت هایپربولیک باید حفظ شوند.

۲. **گیت Forget**: این گیت تصمیم می‌گیرد کدام اطلاعات حفظ و کدام فراموش شود. اطلاعات ورودی گام جدید به‌همراه اطلاعات حالت نهان (Hidden State) گام قبلی به این گیت وارد می‌شوند و از تابع فعالساز(Activation function) عبور می‌کنند. خروجی این تابع عددی بین صفر و ۱ است و هر چه عدد خروجی به صفر نزدیک‌تر باشد یعنی باید اطلاعات فراموش شود و هر قدر به ۱ نزدیک‌تر باشد یعنی باید حفظ شود.

۳. **گیت Output**: این گیت درنهایت تصمیم می‌گیرد که لایه نهان بعدی چه مقداری باشد. همان‌طور که می‌دانیم، لایه نهان اطلاعات ورودی‌های قبلی را همراه خودش دارد. اول اطلاعات ورودی گام جدید به‌همراه اطلاعات لایه نهان گام قبلی به تابع فعالساز وارد می‌شوند. مقدار به روز شده‌ی cell state به تابع تانژانت هایپربولیک وارد می‌شود. خروجی این دو تابع با هم ضرب می‌شود تا تصمیم گرفته شود لایه نهان چه اطلاعاتی را با خودش به گام بعدی ببرد. درنهایت cell state جدید ول ایه نهان جدید به گام زمانی بعدی منتقل می‌شوند.

شبکه‌های عصبی GRU بر خلاف LSTM که دارای ۳ گیت بود، تنها ۲ گیت دارند که شامل گیت‌های Update و Reset هستند. گیت اول معادل حافظه طولانی مدت و گیت دوم معادل حافظه کوتاه مدت است.

این دو شبکه از نظر مقدار دقت به دست آمده روی داده‌های مشابه تفاوت زیادی ندارند اما دارای تفاوت‌های مهمی هستند:

۱. هر دو شبکه جریان اطلاعات را کنترل می‌کنند اما GRU برخلاف LSTM از حافظه برای کنترل این جریان استفاده نمی‌کند.
۲. GRU سرعت آموزش بهتری نسبت به LSTM دارد و از نظر زمانی بسیار بهینه‌تر است.
۳. در حالتی که داده‌های آموزشی کمی داشته باشیم GRU عملکرد بهتری نسبت به LSTM از خود نشان می‌دهد.
۴. GRU ساختار ساده‌تری دارد و در نتیجه می‌توان آن‌را نسبتا ساده‌تر بهبود بخشید و باعث افزایش کارایی و سرعت آن شد.

د

بهترین نتیجه(بیشترین دقت و کمترین مقدار Loss) بین ۹ مدل آموزش داده شده متعلق به مدل GRU با اندازه لایه نهان ۱۲۸ می‌باشد که دقت ۰.۹۸۶ و مقدار Loss برابر ۰.۰۵ را دارد. این مدل دقت بسیار بالاتری نسبت به الگوریتم Viterbi پیاده شده در قسمت پ سوال با مقدار دقت ۸۹ درصد دارد. یکی از دلایل واضح این برتری استفاده از توالی تمام ورودی‌های پیشین در تمام مدل‌های شبکه عصبی بازگشتی از جمله GRU می‌باشد در حالی که در الگوریتم پیاده سازی شده در بخش پ انتخاب برچسب کلمه فعلی تنها بر اساس کلمه فعلی و برچسب کلمه قبلی انجام می‌گیرد. این فرض اگرچه مقداری از دقت الگوریتم می‌کاهد اما تاثیر به سزایی در قابل پیاده سازی شدن(Feasible) آن دارد به طوریکه اگر این فرض در پیاده سازی روش مورد نظر اعمال نشود، این الگوریتم دارای هزینه زمانی و فضایی بسیار سنگینی خواهد شد و همچنین پیچیدگی آن بسیار بالا می‌رود و از قابلیت درک الگوریتم نیز کاسته می‌شود. در نقطه مخالف آن، الگوریتم‌های شبکه عصبی بازگشتی قرار دارند که با حداقل هزینه در آموزش و پیاده سازی می‌توانند توالی کاملی از کلمات موجود در هر جمله را در نظر بگیرند و با دید باز و اطلاعات کامل(در صورت نیاز با تصمیم گیری گیت‌های مربوطه) اقدام به انتخاب برچسب Part of Speech برای هر یک از کلمات می‌کند.