به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

درس پردازش زبان طبیعی

تمرین کامپیوتری شماره سوم

نام و نام خانوادگی

امیرحسین دبیری اقدم

شماره دانشجویی

810197502

اردیبهشت ماه 1401

فهرست

بخش 1 - تعیین نقش کلمات

الف) وقتی با tagset='universal' لود می‌کنیم، تگ‌های Universal[1] که کلی‌تر و با جزئیات کمتر و با تعداد کمتر هستند (12 تگ) ، جایگزین تگ‌های Penn (36 تگ) می‌شوند مثلا همانطور که در نمونهٔ زیر مشاهده می‌شود کلمه 'Pierre' که تگ 'NNP' (Proper **noun**, singular) و کلمه 'years' که تگ 'NNS' (**Noun**, plural) دارند، در حالت استفاده از تگ‌های Universal هر دو تگ 'NOUN' گرفته‌اند یعنی جزئیات تگ مربوطه مثلا اینکه اسم خاص مفرد است یا اسم جمع است حذف شده و هر دو صرفاً تگ اسم گرفته‌اند.

Without tagset='universal':

```
[('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ','), ('61', 'CD'), ('years', 'NNS'), ('old', 'JJ'), (',', ','), ('will', 'MD'),
('join', 'VB'), ('the', 'DT'), ('board', 'NN'), ('as', 'IN'), ('a', 'DT'), ('nonexecutive', 'JJ'), ('director', 'NN'),
('Nov.', 'NNP'), ('29', 'CD'), ('.', '.')]
```

With tagset='universal':

```
[('Pierre', 'NOUN'), ('Vinken', 'NOUN'), (',', '.'), ('61', 'NUM'), ('years', 'NOUN'), ('old', 'ADJ'), (',', '.'),
('will', 'VERB'), ('join', 'VERB'), ('the', 'DET'), ('board', 'NOUN'), ('as', 'ADP'), ('a', 'DET'), ('nonexecutive',
'ADJ'), ('director', 'NOUN'), ('Nov.', 'NOUN'), ('29', 'NUM'), ('.', '.')]
```

بنابراین مشاهده می‌شود که استفاده از tagset='universal' باعث ساده‌تر شدن تگ‌ها می‌شود و بنابراین در ادامه از این حالت استفاده می‌کنیم.

ب) ابتدا جملات موجود در دیتاست را shuffle می‌کنیم؛ اگرچه الگوریتم Viterbi نیازی به استفاده از validation ندارد زیرا پارامتری برای تنظیم کردن ندارد اما چون در ادامه از شبکه‌های عصبی بازگشتی استفاده می‌کنیم که نیاز به تنظیم پارامتر دارند پس از همین ابتدا 10٪ جملات را برای validation و بعد 75% داده‌ها را به عنوان داده train و 15٪ ما بقی را برای test در نظر می‌گیریم. (برای تکرارپذیر شدن نتایج random_seed را برابر با شماره دانشجویی قرار داده‌ام.)

پ) با توجه به شبه‌کد زیر که در اسلایدهای درس موجود بود و براساس توضیحات استاد، الگوریتم Viterbi را پیاده می‌کنیم. به این ترتیب که ابتدا دیکشنری S شامل استیت‌ها و شماره متناظرشان در نظر می‌گیریم؛ استیت‌ها شامل 12 تگ Universal tagset است.

```
S = {0:'ADJ', 1:'ADP', 2:'ADV', 3:'CONJ', 4:'DET', 5:'NOUN', 6:'NUM',
 7:'PRON', 8:'PRT', 9:'VERB', 10:'X', 11:'.'}
```

همچنین دو تگ خاص START و END به صورت ضمنی در هنگام پیاده‌سازی در نظر می‌گیریم که مربوط به ابتدا و انتهای جمله هستند (این دو استیت به اصطلاح non-emmiting هستند.). در ادامه نیاز به محاسبه احتمالات Emission و Transition است که به ترتیب با ماتریس‌های B و A نمایش می‌دهیم.

[1] https://universaldependencies.org/u/pos

این احتمالات را با استفاده از داده‌های train بر اساس روش MLE (تخمین با استفاده از شمارش) مشابه الگوریتم Naïve Bayes بدست می‌آوریم. در نهایت با داشتن این دو ماتریس و استیت‌ها می‌توان sequence تگ‌های POS بهینه (بهینه‌ترین مسیر در گراف حالت) را براساس فرض‌های ساده‌کننده HMM، با کمک الگوریتم Viterbi که یک الگوریتم برنامه نویسی پویا است، بدست آوریم.

function VITERBI(*observations* of len 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** ;initialization step
$viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
$backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ;recursion step
for each state s **from** 1 **to** N **do**
$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$
$viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
$backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
return the backtrace path by following backpointers to states back in time from $backpointer[q_F,T]$

با اجرای الگوریتم پیاده شده روی داده‌های تست، دقت برابر با 85.7٪ بدست می‌آید.

ت) در ادامه چند نمونه از جملات با تگ‌های تخمین زده شده توسط الگوریتم Viterbi و تگ‌های اصلی آمده است:

Prediction POS tags:
[('A', 'DET'), ('player', '.'), ("'s", 'PRT'), ('**commitment**', '.'), ('to', 'PRT'), ('practice', 'NOUN'), ('and', 'NOUN'), ('team', 'NOUN'), ('image', 'NOUN'), ('is', 'VERB'), ('as', 'ADP'), ('important', 'ADJ'), ('as', 'ADV'), ('his', 'PRON'), ('batting', 'NOUN'), ('**average**', 'ADJ'), ('.', '.')]

Actual POS tags:
('A', 'DET'), ('player', 'NOUN'), ("'s", 'PRT'), ('**commitment**', 'NOUN'), ('to', 'PRT'), ('practice', 'NOUN'), ('and', 'CONJ'), ('team', 'NOUN'), ('image', 'NOUN'), ('is', 'VERB'), ('as', 'ADV'), ('important', 'ADJ'), ('as', 'ADP'), ('his', 'PRON'), ('batting', 'NOUN'), ('**average**', 'NOUN'), ('.', '.'),

---

Prediction POS tags:
[('Ten', 'NOUN'), ('**shirt-sleeved**', 'NOUN'), ('ringers', 'NOUN'), ('stand', 'VERB'), ('in', 'ADP'), ('a', 'DET'), ('circle', 'ADJ'), (',', '.'), ('one', 'PRON'), ('foot', 'X'), ('ahead', 'ADV'), ('of', 'ADP'), ('the', 'DET'), ('other', 'ADJ'), ('in', 'ADP'), ('a', 'DET'), ('**prize-fighter**', '.'), ("'s", 'PRT'), ('stance', 'VERB'), (',', '.'), ('each', 'DET'), ('pulling', 'ADJ'), ('a', 'DET'), ('rope', 'ADJ'), ('that', 'ADP'), ('*T*-225', 'NOUN'), ('disappears', 'ADJ'), ('through', 'ADP'), ('a',

'DET'), ('small', 'ADJ'), ('hole', '.'), ('in', 'ADP'), ('the', 'DET'), ('high', 'ADJ'), ('ceiling', 'NOUN'), ('of', 'ADP'), ('the', 'DET'), ('**ringing**', 'VERB'), ('chamber', 'VERB'), ('.', '.')]

Actual POS tags:
('Ten', 'NUM'), ('**shirt-sleeved**', 'ADJ'), ('ringers', 'NOUN'), ('stand', 'VERB'), ('in', 'ADP'), ('a', 'DET'), ('circle', 'NOUN'), (',', '.'), ('one', 'NUM'), ('foot', 'NOUN'), ('ahead', 'ADV'), ('of', 'ADP'), ('the', 'DET'), ('other', 'ADJ'), ('in', 'ADP'), ('a', 'DET'), ('prize-fighter', 'NOUN'), ("'s", 'PRT'), ('stance', 'NOUN'), (',', '.'), ('each', 'DET'), ('pulling', 'VERB'), ('a', 'DET'), ('rope', 'NOUN'), ('that', 'DET'), ('*T*-225', 'X'), ('disappears', 'VERB'), ('through', 'ADP'), ('a', 'DET'), ('small', 'ADJ'), ('hole', 'NOUN'), ('in', 'ADP'), ('the', 'DET'), ('high', 'ADJ'), ('ceiling', 'NOUN'), ('of', 'ADP'), ('the', 'DET'), ('**ringing**', 'NOUN'), ('chamber', 'NOUN'), ('.', '.'),

---

Prediction POS tags:
[('The', 'DET'), ('**rise**', 'VERB'), ('in', 'ADP'), ('the', 'DET'), ('stock', 'NOUN'), ("'s", 'VERB'), ('price', 'NOUN'), ('may', 'VERB'), ('also', 'ADV'), ('reflect', 'DET'), ('the', 'DET'), ('fact', 'NOUN'), ('that', 'ADP'), ('USX', 'NOUN'), ("'s", 'VERB'), ('steel', 'NOUN'), ('segment', 'NOUN'), ('fared', 'NOUN'), ('better', 'ADJ'), ('than', 'ADP'), ('some', 'DET'), ('other', 'ADJ'), ('**steelmakers**', '.'), ("'", 'PRT'), ('.', '.')]
Actual POS tags:
('The', 'DET'), ('**rise**', 'NOUN'), ('in', 'ADP'), ('the', 'DET'), ('stock', 'NOUN'), ("'s", 'PRT'), ('price', 'NOUN'), ('may', 'VERB'), ('also', 'ADV'), ('reflect', 'VERB'), ('the', 'DET'), ('fact', 'NOUN'), ('that', 'ADP'), ('USX', 'NOUN'), ("'s", 'PRT'), ('steel', 'NOUN'), ('segment', 'NOUN'), ('fared', 'VERB'), ('better', 'ADV'), ('than', 'ADP'), ('some', 'DET'), ('other', 'ADJ'), ('**steelmakers**', 'NOUN'), ("'", 'PRT'), ('.', '.'),

مشاهده می‌شود که اکثر کلماتی که تگ‌شان اشتباه تخمین زده شده است، کلماتی هستند که در داده آموزش وجود نداشته‌اند و به اصطلاح out-of-vocab هستند مثل 'commitment'، 'shirt-sleeved'،'prize-fighter'،'steelmakers'،'37-year-old' و... یا کلماتی مثل 'average'،'ringing'،'rise' و... هستند که هم می‌توانند 'NOUN' و هم 'ADJ' و یا حتی 'VERB' باشند و به اصطلاح ambiguity دارند و ممکن است برای مثال در داده آموزش تنها حالت 'ADJ' آن وجود داشته باشد اما در داده تست حالت 'NOUN' هم موجود باشد که در اینجا هم با توجه به اینکه تعداد داده‌های آموزش چندان زیاد نیست اینکه چنین حالاتی رخ بدهد محتمل است و الگوریتم Viterbi حتی با توجه به احتمالات transition هم نتوانسته تگ‌های صحیح را تخمین بزند.

ث) برای برخورد با کلمات جدید و out-of-vocab داده تست از آنجا که این کلمات در داده آموزش نبودند بنابراین احتمال آن‌ها صفر است (در ماتریس B) که باعث صفر شدن احتمال کل sequence می‌شود که درست نیست، بنابراین برای هندل کردن این اتفاق بهتر است تا حد امکان داده‌های تست را زیاد کنیم تا کمتر این حالت رخ دهد اما به هر صورت چون این حالت اجتناب ناپذیر است ناچار هستیم که از smoothing استفاده کنیم؛ به این ترتیب که برای کلمات OOV هم یک احتمال خیلی کوچک در نظر

می‌گیریم تا حاصل کل توالی صفر نشود (در قسمت‌های قبل smoothing اعمال شده بود و دقت و... بر این اساس محاسبه شده بود). البته با اعمال smoothing الزاماً تگ این کلمات به درستی تخمین زده نمی‌شود که این موضوع را در قسمت ت هم دیدیم. برای تخمین تگ این کلمات می‌توان از روش‌های rule-based استفاده کرد؛ به بیان دیگر الگوریتم Viterbi را با روش‌های rule-based که مثلاً براساس ساختار morphological یا شکل نوشتاری کلمات یا... هستند، ترکیب کرد و برای تخمین تگ کلمات OOV از rule ها استفاده کرد و در موارد دیگر از الگوریتم Viterbi. همچنین به صورت کلی می‌توان به جای HMM از MEMM استفاده کرد و ویژگی‌های مختلف morphological و... را هم علاوه بر احتمالات emission و transition در نظر گرفت و مدل پیچیده‌تری نسبت به HMM استفاده کرد برای هندل کردن مشکلات مذکور.

ج) برای استفاده از RNNها نیاز است تا ابتدا داده متنی به بردار عددی تبدیل شود؛ برای این کار می‌توان از embeddingهایی نظیر GloVe، Word2Vec و... که pre-train شده‌اند استفاده کرد (و در صورت لزوم برای این دیتاست و تسک pos tagging وزن‌های آن را fine-tune کرد) و یا embedding را از صفر train کرد. با توجه به اینکه حجم داده‌های موجود چندان زیاد نیست و آموزش embeddingها هم نیاز به داده زیاد و نیز صرف زمان زیاد است از GloVe استفاده می‌کنیم که هر کلمه را به یک بردار 300بعدی می‌نگارد. همچنین در پایتورچ اگر وزن‌های مربوط به embedding، freeze نشوند، در حین آموزش شبکه این وزن‌ها هم آپدیت می‌شوند که در ابتدا این وزن‌ها را فریز نمی‌کنیم تا fine-tune شوند. اندازه hidden-layer را برابر با مقادیر 16،32 و 64 در نظر می‌گیریم و با epochs=20 و با learning-rate=0.1 شبکه را آموزش می‌دهیم. (این پارامترها را براساس داده validation طوری تنظیم کردیم که بهترین دقت را روی این داده‌ها داشته باشیم) . نتایج زیر در نهایت حاصل شد؛ به عنوان رفرنس دقت الگوریتم Viterbi روی داده‌های train و validation هم آمده است.

Hidden-layer size	Train accuracy (%)	Validation accuracy (%)
16	91.096	86.249
32	92.591	85.928
64	93.661	85.978
Viterbi (as ref.)	89.584	85.395

مشاهده می‌شود که دقت RNNها روی داده‌های آموزش و validation به طرز قابل توجهی بیشتر است نسبت به Viterbi. بین خود RNNها نیز هر قدر مدل پیچیده‌تر شده (اندازه hidden-layer بزرگتر) دقت آن روی داده‌های آموزش بیشتر شده اما دقت آن روی داده‌های validation کمتر شده که نشان دهنده این است که مدل پیچیده‌تر با توجه به حجم نسبتاً کم داده‌های آموزش، روی این داده‌ها overfit می‌شود.

نمودارهای loss و accuracy مدل‌ها در زیر آمده است. مشاهده می‌شود که از یک جایی به بعد با افزایش epochها دقت مدل‌ها بر روی داده train افزایش (loss کاهش) و دقت آن روی validation کاهش یافته (loss افزایش) که نشان می‌دهد مدل‌ها رفته رفته به سمت overfit شدن پیش می‌روند.

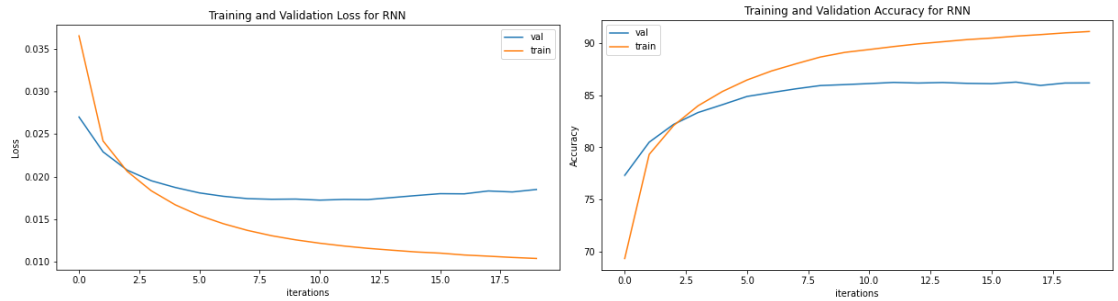

Figure1 - Training and Validation Accuracy/Loss for Vanilla RNN (Hidden-layer=16)

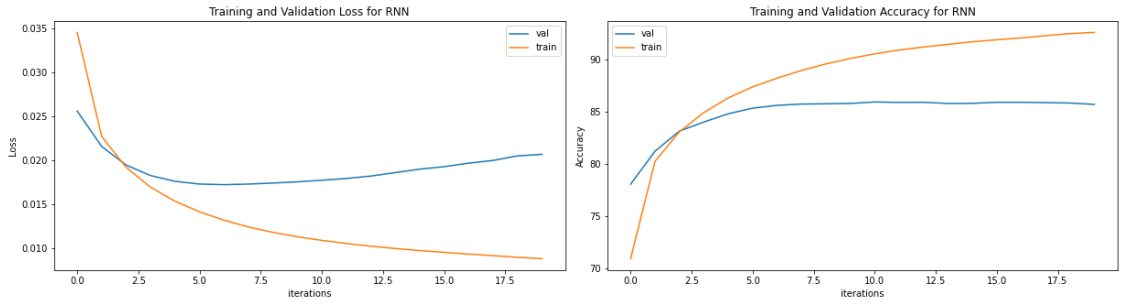

Figure2 -Training and Validation Accuracy/Loss for Vanilla RNN (Hidden-layer=32)

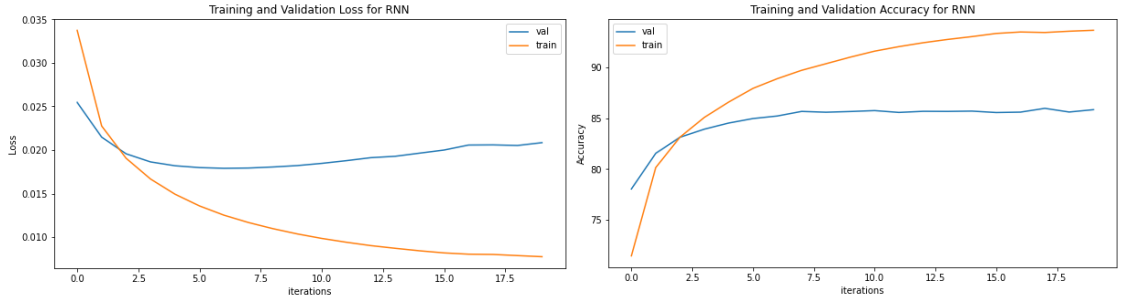

Figure3 - Training and Validation Accuracy/Loss for Vanilla RNN (Hidden-layer=64)

همچنین یکبار هم با فریز کردن وزن‌های embedding و با epochs=200 و با learning-rate=0.01 شبکه را آموزش می‌دهیم که نتایج آن بدتر از حالات فوق می‌شود. نمودارهای مربوطه و... در نوتبوک ضمیمه شده آمده است و برای جلوگیری از شلوغ شدن، در گزارش نیامده است. اما در این حالت هم

مدل‌ها رفته رفته به سمت overfit شدن پیش می‌روند و از یک جایی به بعد با اینکه دقت مدل‌ها روی داده train افزایش می‌یابد اما دقت مدل روی داده‌های validation ثابت مانده و حتی رفته رفته کمتر می‌شود.

چ) در نهایت دقت‌های جدول زیر برای Vanilla RNN حاصل می‌شود. در اینجا هم به عنوان رفرنس دقت الگوریتم Viterbi آمده است.

Hidden-layer size	Test accuracy (%)
16	85.664
32	85.376
64	85.274
Viterbi	85.705

مشاهده می‌شود که عملکرد مدل‌ها تقریبا یکسان است البته در این حالت Viterbi اندکی دقت بیشتری دارد که علت آن افراز داده‌ها به test,train,validation است و اگر افراز دیگری را در نظر بگیریم احتمالاً دقت RNNها اندکی بیشتر شود همانطور که در validation این چنین بود. به صورت کلی هم شبکه‌های عصبی عمیق به دلیل پیچیدگی بیشتر نیاز به داده بیشتری برای آموزش دارند تا بهتر generalize شوند و overfit نشوند اما در اینجا حجم داده‌ها چندان زیاد نبود و شبکه‌ها بعضاً overfit می‌شدند؛ همچنین شاید با تنظیم بهتر هایپرپارامترها یا استفاده از dropout می‌شد دقت بهتری هم بدست آورد اما به دلیل کمبود وقت و محدودیت‌های colab و... فرصت نشد.

ح) مراحل قبل را این بار برای شبکه مبتنی بر LSTM و GRU تکرار می‌کنیم. مشابه قبل اندازه hidden-layer را برابر با مقادیر 16،32 و 64 در نظر می‌گیریم و با epochs=20 و با learning-rate=0.1 شبکه‌ها را آموزش می‌دهیم. (این پارامترها را براساس داده validation طوری تنظیم کردیم که بهترین دقت را روی این داده‌ها داشته باشیم). در پایان نتایج زیر حاصل شد؛ به عنوان رفرنس دقت الگوریتم Viterbi روی داده‌های train و validation هم آمده است.

Hidden-layer size	Train accuracy (%)	Validation accuracy (%)
16 (LSTM)	92.629	86.159
32 (LSTM)	93.594	86.551
64 (LSTM)	94.764	86.631
16 (GRU)	92.876	86.601

32 (GRU)	93.655	86.742
64 (GRU)	94.923	87.083
Viterbi	89.584	85.395

مشاهده می‌شود که دقت LSTM و GRUها روی داده‌های آموزش و validation بیشتر است نسبت به Viterbi. بین خود شبکه‌های عصبی نیز برعکس حالت قبل هر قدر مدل پیچیده‌تر شده (اندازه hidden-layer بزرگتر) دقت آن روی داده‌های آموزش و نیز دقت روی داده‌های validation بیشتر شده که نشان از برتری این مدل‌ها نسبت به Vanilla RNNها است که احتمالاً دچار مشکلاتی نظیر Vanishing gradient و... می‌شوند. همچنین ظاهراً شبکه GRU با توجه به اینکه ساختارش اندکی ساده‌تر است نسبت به LSTM، توانسته بهتر generalize شود هرچند اگر حجم داده‌های آموزش بیشتر بود احتمالاً برعکس می‌بود اما در هر صورت، نسبت به الگوریتم Viterbi و حتی Vanilla RNN، برای هر دو مدل GRU و LSTM افزایش دقت قابل توجه است که به دلیل ساختار و گیت‌های موجود در شبکه است که برای تسک‌های NLP نسبت به دیگر مدل‌ها مناسب‌تر هستند. (البته بدون در نظر گرفتن transformerها)

نمودارهای loss و accuracy مدل‌ها نیز در زیر آمده است. مشاهده می‌شود که مشابه Vanilla RNNها، در اینجا هم از یک جایی به بعد با افزایش epochها دقت مدل‌ها بر روی داده train افزایش (loss کاهش) و دقت آن روی validation کاهش یافته (loss افزایش) که نشان می‌دهد مدل‌ها رفته رفته به سمت overfit شدن پیش می‌روند.

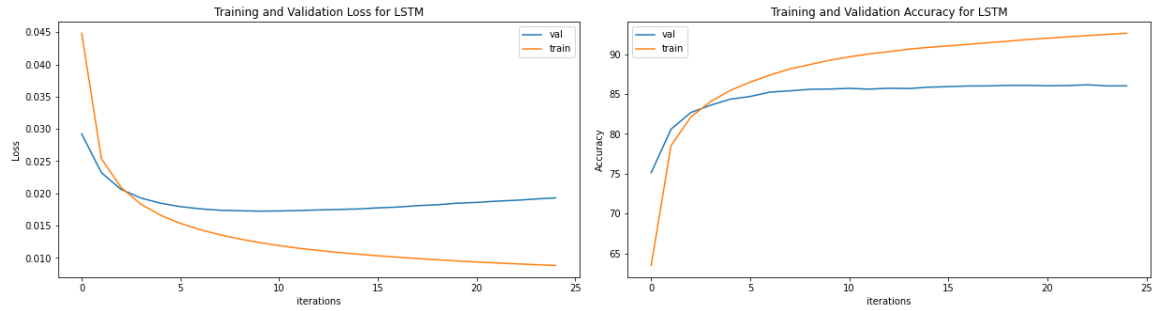

Figure 4- Training and Validation Accuracy/Loss for LSTM (Hidden-layer=16)

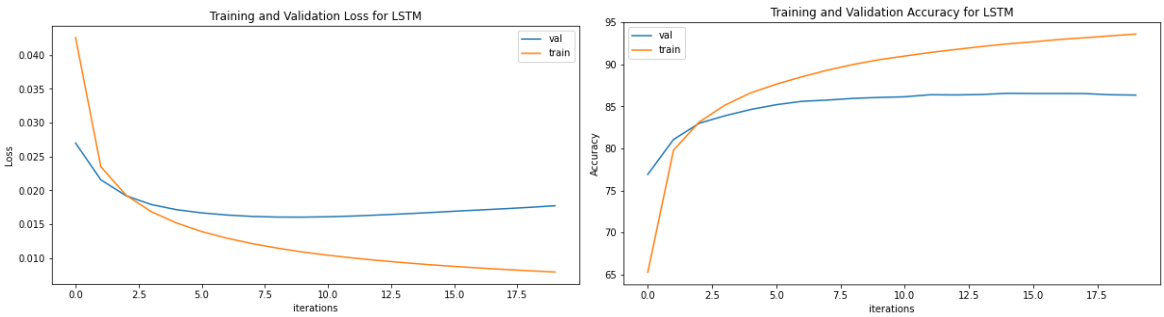

Figure 5-Training and Validation Accuracy/Loss for Vanilla LSTM (Hidden-layer=32)

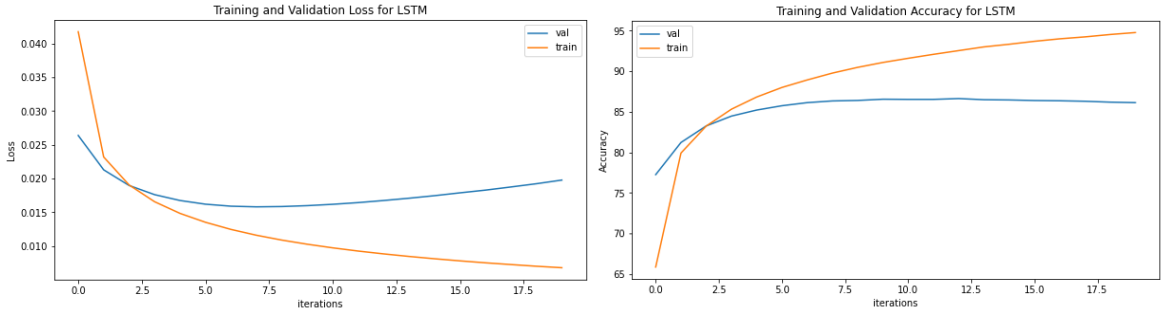

Figure 6 - Training and Validation Accuracy/Loss for LSTM (Hidden-layer=64)

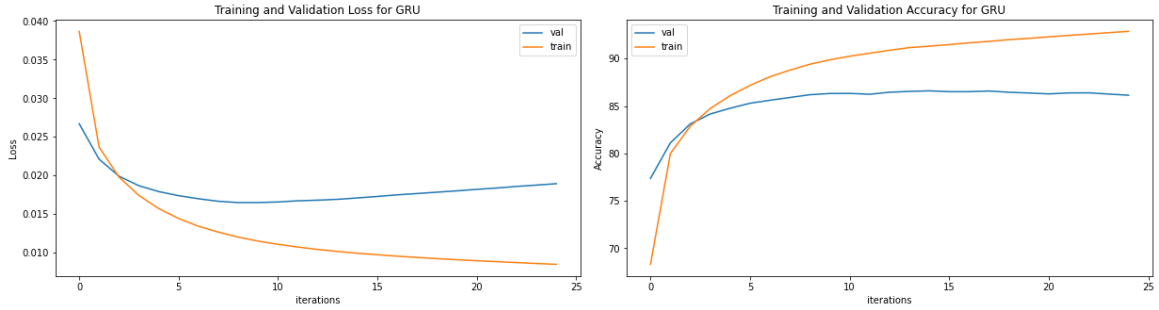

Figure 7- Training and Validation Accuracy/Loss for GRU (Hidden-layer=16)

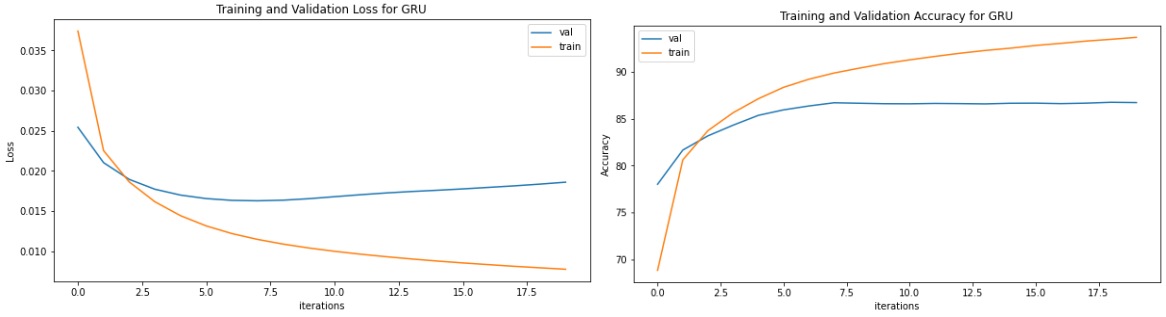

Figure 8-Training and Validation Accuracy/Loss for Vanilla GRU (Hidden-layer=32)

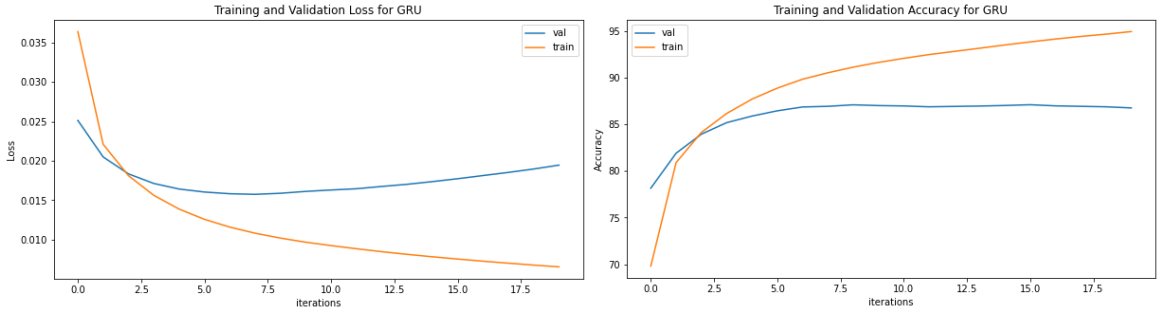

Figure 9 - Training and Validation Accuracy/Loss for GRU (Hidden-layer=64)

همچنین با فریز کردن وزن‌های embedding و با epochs=200 و با learning-rate=0.01 شبکه‌های LSTM و GRU را آموزش دادیم که عین Vanilla RNN این بار هم نتایج بدتر از حالتی شد که embedding هم fine-tune می‌شد. نمودارهای مربوطه و... در نوت‌بوک ضمیمه شده آمده است اما برای جلوگیری از شلوغی در گزارش نیامده است. اما در این حالت هم مدل‌ها رفته رفته به سمت overfit شدن پیش می‌روند و از یک جایی به بعد با اینکه دقت مدل‌ها روی داده train افزایش می‌یابد اما دقت مدل روی داده‌های validation ثابت مانده و حتی رفته رفته کمتر می‌شود.

در نهایت دقت‌های بدست آمده در جدول زیر آمده است. در اینجا هم به عنوان رفرنس دقت الگوریتم Viterbi و نیز بهترین مدل Vanilla RNN آمده است.

Hidden-layer size	Test accuracy (%)
16 (LSTM)	85.431
32 (LSTM)	85.987
64 (LSTM)	85.952
16 (GRU)	85.856
32 (GRU)	**86.048**
64 (GRU)	86.021
16 (Vanilla RNN)	85.664
Viterbi	85.705

مشاهده می‌شود که بهترین مدل GRU (با اندازه لایه پنهان 32) است که بیش از 0.3٪ بهبود نسبت به Viterbi و Vanilla RNN (اندازه لایه پنهان 16) داشته و بعد از آن مدل LSTM (با اندازه لایه پنهان 32) است که بیش از 0.2٪ افزایش دقت داشته است.

مجدداً لازم به ذکر است که برتری LSTM و GRU نسبت به Vanilla RNN ساختار گیت‌های آن است که باعث می‌شود مشکل صفر یا اشباع شدن گرادیان (vanishing & exploding gradient) علی الخصوص در جملات طولانی که وابستگی با فاصله زیاد (long distance dependency) وجود دارد، تا حدی برطرف شود و اطلاعات با ارزش در طول زمان از بین نرود و حفظ شود و اطلاعاتی که دیگر مورد نیاز نیستد از بین بروند. همچنین در مورد GRU هم که یک ورودی کمتر از LSTM دارد (تصویر زیر)، چون تعداد پارامترهایش کمتر است مدل اندکی ساده‌تر نسبت به LSTM است که وقتی حجم داده‌های آموزش کم است مدل ساده‌تر بهتر می‌تواند جنرالایز شود و به همین دلیل هم برای این دیتاست بهترین عملکرد را GRUها دارند (توضیحات بیشتر درباره گیت‌های LSTM و مقایسه آن با GRU در قسمت خ).

البته ممکن است با افزایش حجم داده آموزش،تنظیم بهتر هایپرپارامترها و یا به ازای افرازهای دیگر داده به train,test,valid عملکرد LSTM از GRU هم بهتر بشود اما به صورت کلی این دو مدل عملکرد نزدیک به هم دارند و اکثر مواقع نسبت به Vanilla RNN و Viterbi عملکرد بهتری دارند.

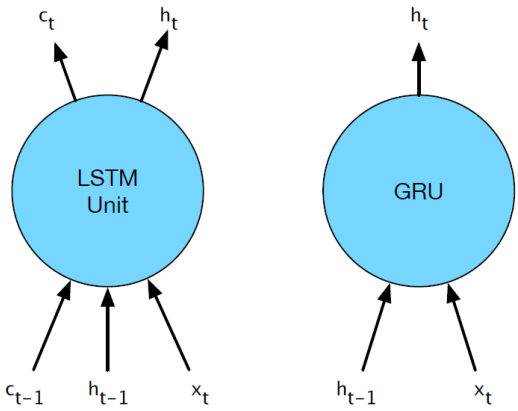

Figure10 - **GRU unit & LSTM unit**

خ) در Vanilla RNN همانطور که در شکل 11 دیده می‌شود تنها یک تابع اکتیویشن (معمولاً tanh) داریم که همانطور که گفته شد این ساختار وقتی جملات طولانی باشند مشکل vanishing gradient ممکن است رخ دهد. برای رفع این مشکل، مدل LSTM ارائه شده است که چند گیت شامل گیت‌های Forget,input,output مشابه شکل 12 دارد. هدف اصلی این گیت‌ها این است که اطلاعات مهم در طول زمان حفظ شده و اطلاعات کم ارزش حذف شوند که این کار به کمک گیت‌های مذکور انجام می‌شود. (اطلاعات در cell state ذخیره یا از آن حذف می‌شوند. این اطلاعات تعیین کننده hidden state مدل LSTM هستند.)

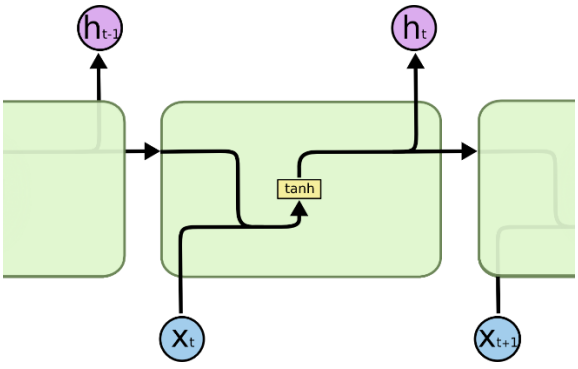

Figure11 **-Vanilla RNN structure**

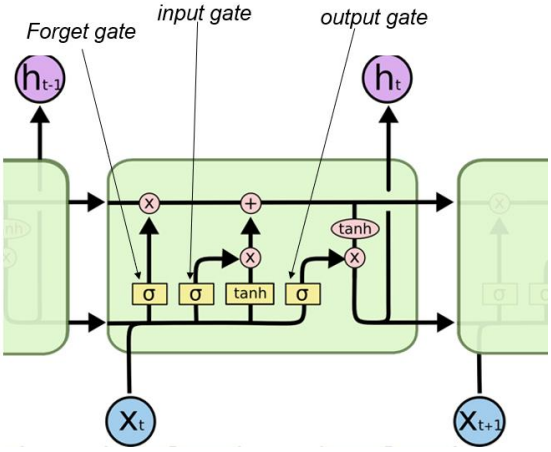

Figure12 - LSTM structure

فرآیند حذف اطلاعات cell state که مدل LSTM تصمیم می‌گیرد دور بریزد (اطلاعات کم ارزش) توسط گیت Forget انجام می‌شود که یک تابع sigmoid دارد که براساس h_{t-1} و x_t خروجی آن تعیین می‌شود که یک یعنی اطلاعات کامل حفظ شود و صفر یعنی کامل از بین برود.

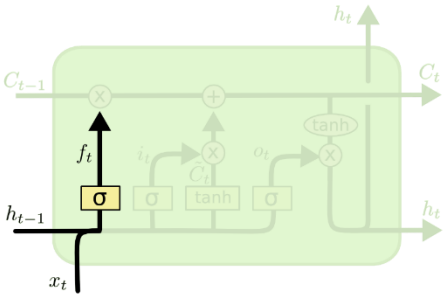

Figure 13-Forget gate

فرآیند اضافه کردن اطلاعات به cell state از طریق گیت input است. مشابه قبل یک تابع sigmoid دارد که براساس h_{t-1} و x_t خروجی آن تعیین می‌شود و براساس مقدار آن تصمیم گرفته می‌شود که مقدار خروجی تابع اکتیویشن tanh (مقدار جدید cell state) به cell state اضافه شود یا نه. به این ترتیب و با استفاده از دو گیت forget و input اطلاعات از cell state حذف یا به آن اضافه می‌شوند.

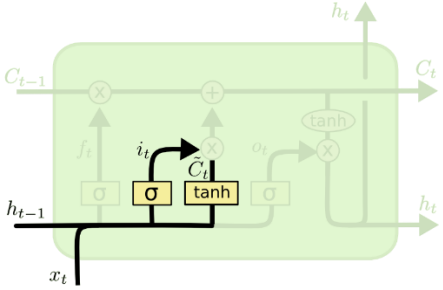

Figure 14 – Input gate

در پایان مدل LSTM به کمک گیت output تصمیم می‌گیرد که چه مقدار از cell state حاضر بعد از عبور از تابع اکتیویشن tanh به عنوان h_t (hidden-state) استفاده شود.

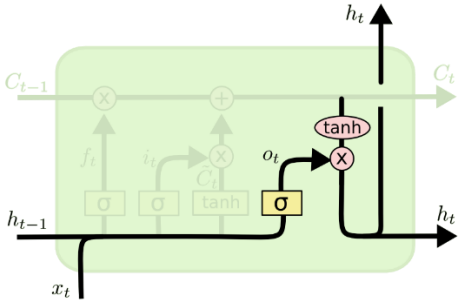

Figure 15-Output gate

همانطور که در شکل16 مشاهده می‌شود ساختار GRU تقریباً شبیه LSTM بوده و به نوعی ساده شده LSTM است به این ترتیب که گیت‌های forget و input آن با هم ترکیب شده و گیت update خواهیم داشت. همچنین در GRU دیگر cell state جدا از hidden state نداریم (ادغام شده‌اند). این موارد باعث کاهش تعداد پارامترهای مدل و ساده‌تر شدن آن می‌شود.

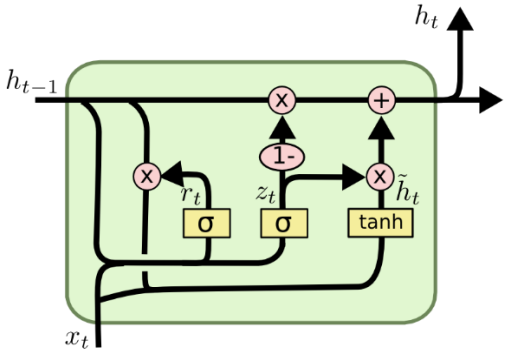

Figure16 - GRU structure

د) همانطور که در قسمت ح دیدیم بیشترین دقت (86.048٪) با مدل GRU (با اندازه لایه پنهان 32) بدست آمد که نسبت به الگوریتم Viterbi با دقت 85.705٪ افزایش 0.343٪ را شاهد هستیم که افزایش قابل قبولی است. هرچند همانطور که چندین بار هم ذکر شد، شبکه‌های عصبی عمیق بهترین عملکرد را وقتی دارند که حجم داده برای آموزششان زیاد باشد و در اینجا هم اگر حجم داده بیشتری برای آموزش مدل در اختیار بود قطعاً تفاوت دقت Viterbi و شبکه‌های عصبی بیشتر و قابل توجه‌تر می‌شد؛ اما وقتی حجم داده کم است روش‌های کلاسیک ماشین لرنینگ بعضاً دقت عملکرد بهتری دارند. همچنین دیگر تفاوت این است که train و نیز inference شبکه‌های عصبی عمیق به دلیل محاسبات پیچیده ، هزینه زمانی بیشتری دارد و نیاز به سخت افزارهای پیشرفته‌تر نظیر GPU ,TPU های گران قیمت وجود دارد اما روش‌های کلاسیک نظیر Viterbi معمولاً نیاز به train ندارند یا به سادگی با شمارش و محاسبه احتمالات

train می‌شوند و inference آن‌ها نیز معمولاً خیلی الگوریتم پیچیده‌ای نیست و نیاز به سخت افزار پیشرفته ندارد و در زمان کم محاسبات آنها انجام می‌شود.

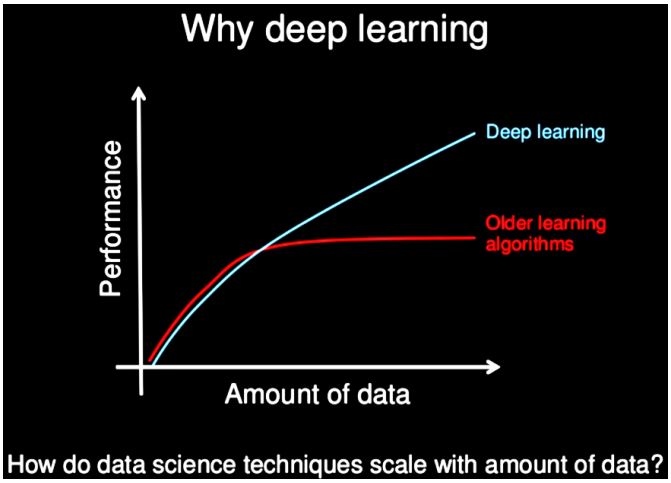

Figure17 – Deep Learning vs Classic machine learning

بخش 2 - تشخیص گروه‌های اسمی

الف) با استفاده از کتاب‌خانه NLTK این بار به جای POS Tag، برای همان دیتاست قبلی IOB tag ها را به عنوان لیبل کلمات در نظر می‌گیریم. یک نمونه از داده‌ها با تگ‌های IOB متناظر در ادامه آمده است:

```
[('Mr.', 'B-PERSON'), ('Nixon', 'B-PERSON'), ('also', 'O'), ('proposed', 'O'), ('that', 'O'),
('China', 'B-GPE'), ('restore', 'O'), ('its', 'O'), ('participation', 'O'), ('in', 'O'), ('the', 'O'),
('Fulbright', 'B-ORGANIZATION'), ('Program', 'I-ORGANIZATION'), (',', 'O'), ('a', 'O'),
('U.S.', 'B-GPE'), ('government-funded', 'O'), ('academic', 'O'), ('exchange', 'O'), ('.', 'O')
```

ب) این بار چون فقط الگوریتم Viterbi را اجرا خواهیم کرد بنابراین نیازی به داده validation نیست پس 25٪ داده‌ها را برای test و 75٪ ما بقی را برای train در نظر می‌گیریم.

این بار دیکشنری S که مربوط به استیت‌ها بود را براساس تگ‌های IOB به صورت زیر تعریف می‌کنیم و بعد مشابه قبل احتمالات transition و emission را با استفاده از داده train حساب می‌کنیم و در نهایت مشابه قبل الگوریتم Viterbi را با استفاده از این موارد اجرا می‌کنیم.

```
S = {0:'O', 1:'B-PERSON', 2:'I-PERSON', 3:'B-ORGANIZATION', 4:'I-
ORGANIZATION', 5:'B-GPE', 6:'I-GPE', 7:'B-LOCATION', 8:'I-
LOCATION', 9:'B-FACILITY', 10:'I-FACILITY', 11:'B-GSP', 12:'I-GSP'}
```

تفاوت عمده IOB تگ‌ها نسبت به تگ‌های SOP وجود تگ‌های با پیشوند I است که نمی‌توانند مستقلاً بعد از تگ O ظاهر شوند و صرفاً می‌توانند بلافاصله بعد از تگ‌های با پیشوند B متناظر ظاهر شوند؛ به عبارت دیگر توالی:

```
'B-ORGANIZATION', 'O', 'I-ORGANIZATION'
```

نامعتبر است و نمی‌تواند چنین حالتی رخ دهد یا به طور مشابه توالی:

```
'B-ORGANIZATION', 'B-FACILITY', 'I-ORGANIZATION'
```

نیز نامعتبر است.

نکته قابل توجه این است که نیازی به ایجاد تغییر در پیاده‌سازی الگوریتم Viterbi که در قسمت قبل نوشته بوده‌ایم نیست زیرا توالی غیر ممکن در هنگام اجرای این الگوریتم رخ نمی‌دهد زیرا احتمال transition از تگ O به تگ با پیشوند I یا دیگر حالات نامعتبر نظیر رفتن از استیت START به تگ با پیشوند I صفر است (در ماتریس مربوط transition یعنی A) و بنابراین ممکن نیست که بعد از اجرای این الگوریتم توالی غیر ممکن دارای بیشترین احتمال باشد و به عنوان خروجی نهایی الگوریتم ظاهر شود.

پ) عملکرد الگوریتم Viterbi را با استفاده از کتاب‌خانه seqeval روی داده تست محاسبه می‌کنیم:

Precision	Recall	F1
38.16 %	41.41 %	39.71 %

ت) بله می‌توان استفاده کرد اما چالش اصلی که وجود دارد همان احتمال رخداد توالی‌های غیر ممکن است که در قسمت قبل گفته شد؛ در صورت استفاده از مدل‌های بازگشتی برای تسک NER چون این توالی‌های غیرمجاز ممکن است رخ دهد و در واقع هیچ چیزی در این مدل‌ها وجود ندارد که تضمین کند این حالات رخ نمی‌دهند پس استفاده از این مدل‌ها به تنهایی برای این تسک ممکن است خیلی عملکردی مطلوب نداشته باشد.

برای رفع چالش مذکور از ترکیب مدل‌های بازگشتی با الگوریتم Viterbi استفاده می‌شود به این ترتیب که از میان توالی‌های خروجی مدل بازگشتی با کمک الگوریتم Viterbi توالی مجاز با بیشترین احتمال انتخاب می‌شود؛ معمولاً برای پیاده‌سازی چنین سیستمی از یک لایه CRF به عنوان آخرین لایه شبکه بازگشتی استفاده می‌شود تا به این ترتیب عملکرد خوب شبکه‌های عصبی را در کنار تضمین توالی معتبر در کنار هم داشته باشیم زیرا همانطور که مشاهده می‌شود تسک NER بسیار پیچیده بوده و الگوریتم Viterbi نمی‌تواند عملکرد خوبی از خود نشان دهد و احتمالاً استفاده از شبکه‌های عصبی عمیق برای چنین تسکی مطلوب‌تر باشد علی الخصوص استفاده از bi-directional LSTMها...

توضیحات پایانی:

**تمام کدهای مربوط به به هرکدام از پیاده‌سازی‌های ذکر شده در گزارش، در فایل نوت‌بوک ضمیمه شده، آمده است.*

توضیحات و نتایج کدها در این گزارش ارائه شده است.

**نوت‌بوک‌ها از کولب گوگل استخراج شده و به دلیل حجم محاسبات مربوط به شبکه‌های عصبی بعضاً زمان اجرای cellها طولانی است.*