به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق وکامپیوتر

درس پردازش زبان های طبیعی

پروژه شماره : 3

نام و نام خانوادگی : محمدرضا بختیاری

شماره دانشجویی : 810197468

اردیبهشت 1401

فهرست سوالات

1- تعیین نقش کلمات

الف)

در این قسمت ابتدا یک بار جملات را با استفاده از تگ ست universal و یک بار به صورت عادی به صورت زیر لود کرده و نتایج را مشاهده می کنیم :

```
treebank_corpus_normally = treebank.tagged_sents()
treebank_corpus_universal = treebank.tagged_sents(tagset='universal')
```

که برای هر قسمت نتایج زیر حاصل می شود :

بارگذاری به صورت عادی :

[[('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'), ('.', '.')], [('Mr.', 'NNP'), ('Vinken', 'NNP'), ('is', 'VBZ'), ('chairman', 'NN'), ('of', 'IN'), ('Elsevier', 'NNP'), ('N.V.', 'NNP'), (',', ','), ('the', 'DT'), ('Dutch', 'NNP'), ('publishing', 'VBG'), ('group', 'NN'), ('.', '.')], ...]

بارگذاری با استفاده از تگ ست 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'), ('.', '.')], [('Mr.', 'NOUN'), ('Vinken', 'NOUN'), ('is', 'VERB'), ('chairman', 'NOUN'), ('of', 'ADP'), ('Elsevier', 'NOUN'), ('N.V.', 'NOUN'), (',', '.'), ('the', 'DET'), ('Dutch', 'NOUN'), ('publishing', 'VERB'), ('group', 'NOUN'), ('.', '.')], ...]

مشاهده می کنیم هنگامی که بارگذاری را با استفاده از تگ ست universal انجام می دهیم , فقط 12 تگ برجسته تر از میان 46 تگ مجموعه PTB[1] را مورد بررسی و استفاده قرار می دهد.

که این 12 کلاس استفاده شده برای مجموعه تگ ها[2] عبارت است از :

Verb, Noun, Pronouns, Adjectives, Adverbs, Adpositions, Conjunctions, Determiners, Cardinal Numbers, Particles, Other/ Foreign words, Punctuations

که به اختصار به صورت زیر نمایش داده می شوند :

{'ADJ', 'PRON', 'VERB', '.', 'ADV', 'NUM', 'CONJ', 'X', 'NOUN', 'DET', 'PRT', 'ADP'}

در نهایت به علت این که استفاده از 12 کلاس برای مجموعه تگ ها به جای استفاده از تمامی تگ ست ها (46 مورد) در قسمت های بعدی منجر به سریع تر شدن الگوریتم Viterbi می شود , جملات را با استفاده از مجموعه تگ های universal استفاده می کنیم.

[1] Penn Treebank
[2] Tagsets

ب)

در این قسمت نیاز داریم داده ها را به سه دسته ی آموزش[1] , آزمون[2] و اعتبارسنجی[3] تقسیم بندی کنیم. با توجه به این که در الگوریتم Viterbi نیاز به محاسبه ی هیچ فراپارامتری[4] نداریم , نیازی به استفاده از داده های اعتبار سنجی نداریم و داده ها را به دو دسته ی آموزش و آزمون به صورت زیر تقسیم بندی می کنیم :

```
random.seed(1234)
train_set, test_set = train_test_split(nltk_data,train_size=0.95)
```

در ابتدا از نسبت 0.8 و 0.2 به ترتیب برای داده های آموزش و آزمون استفاده کردیم , اما به علت زمان بر بودن زمان اجرا , مقدار کمتری به داده های آزمون اختصاص دادیم و در نهایت مقادیر زیر را برای هر دسته اختصاص دادیم :

Train : 95% Test : 5% Validation : 0%

پ)

برای پیاده سازی الگوریتم Viterbi ابتدا نیاز داریم تا ماتریس احتمالات انتقال[5] را تشکیل دهیم تا نشان دهیم هر تگ با چه احتمالی بعد از تگ دیگری می آید. همان طور که مشخص است تعداد سطر ها و ستون های این ماتریس مربعی , به اندازه تعداد مجموعه تگ ها است که در اینجا این ماتریس 12*12 می باشد.

	ADJ	PRON	VERB	.	ADV	NUM	CONJ	X	NOUN	DET	PRT	ADP
ADJ	0.066874	0.000654	0.011772	0.066056	0.004415	0.020929	0.017332	0.020602	0.698659	0.004905	0.010955	0.076848
PRON	0.072547	0.008018	0.486827	0.040855	0.035128	0.007255	0.005346	0.091638	0.208858	0.009927	0.012982	0.020619
VERB	0.065372	0.035510	0.169503	0.035355	0.082160	0.022435	0.005338	0.217701	0.109469	0.133916	0.031719	0.091521
.	0.045013	0.067026	0.088500	0.092812	0.052650	0.080413	0.058580	0.026774	0.220934	0.172956	0.002336	0.091914
ADV	0.127737	0.015594	0.346715	0.134705	0.079297	0.032183	0.006967	0.023225	0.032515	0.069675	0.012608	0.118779
NUM	0.033762	0.001195	0.017927	0.116821	0.002988	0.184942	0.013744	0.213027	0.349866	0.003585	0.027487	0.034658
CONJ	0.118299	0.060998	0.157116	0.036506	0.054529	0.041128	0.000462	0.008318	0.345194	0.120148	0.004621	0.052680
X	0.016828	0.055406	0.203048	0.165106	0.025718	0.002699	0.010478	0.075409	0.060168	0.055247	0.184791	0.145102
NOUN	0.012183	0.004596	0.147578	0.239167	0.016924	0.009192	0.042712	0.029144	0.264152	0.013204	0.044171	0.176977
DET	0.206506	0.003494	0.039398	0.018193	0.012169	0.021807	0.000482	0.045060	0.637590	0.005542	0.000241	0.009518
PRT	0.084906	0.017892	0.400455	0.041965	0.009759	0.057905	0.001952	0.013338	0.247885	0.101171	0.001952	0.020820
ADP	0.106743	0.068925	0.007990	0.039949	0.013103	0.062001	0.000852	0.035262	0.322893	0.323746	0.001491	0.017045

شکل 1 : نمایی از ماتریس Transition probabilities

[1] Train
[2] Test
[3] Validation
[4] Hyperparameter
[5] Transition probabilities

در ادامه نیاز داریم احتمال انتشار[1] را نیز با استفاده از تابع Word_given_tag محاسبه کرده و در نهایت با استفاده از احتمال انتشار و ماتریس احتمال انتقال , الگوریتم Viterbi را پیاده سازی می کنیم و مدل را بر روی دادگان آزمون اجرا می کنیم و دقت را محاسبه می کنیم و خواهیم داشت :

The accuracy using Viterbi algorithm is: 0.9396417445482866 %

توجه : در حالی که فقط 5 درصد دادگان را به دادگان آزمون اختصاص دادیم باز هم اجرای مدل بر روی دادگان تست زمان بر بود و حدود 21 دقیقه ارزیابی مدل و اجرای آن بر روی دادگان آزمون زمان برد ! (پیشاپیش از صبر و شکیبایی شما کمال تشکر را دارم)

ت)

ابتدا در این قسمت تعداد کلماتی را که به اشتباه تگ خورده اند را به تفکیک هر تگ مشاهده می کنیم:

'ADJ': 24, 'ADP': 20, 'ADV': 16, 'CONJ': 1, 'DET': 5, 'NOUN': 211, 'PRT': 7, 'VERB': 26

مشاهده می کنیم بیشترین تگ اشتباه خورده مربوط به تگ ست اسم[2] می باشد که یکی از علل آن اشتباه تگ خوردن کلمات ناشناخته[3] است که در قسمت بعد به آن خواهیم پرداخت.

حال نمونه هایی از کلماتی را که به اشتباه تگ خورده اند را مشاهده می کنیم :

```
[[('a', 'DET'), (('whirling', 'NOUN'), ('whirling', 'ADJ'))],
 [('would', 'VERB'), (('chase', 'NOUN'), ('chase', 'VERB'))],
 [('chase', 'VERB'), (('away', 'PRT'), ('away', 'ADV'))],
 [('two', 'NUM'), (('ancillary', 'NOUN'), ('ancillary', 'ADJ'))],
 [('its', 'PRON'), (('offer', 'VERB'), ('offer', 'NOUN'))]]
```

تمامی کلماتی که به اشتباه تگ خورده اند در خروجی قطعه کد این بخش موجود است , به علت محدودیت فضا و شلوغ نشدن گزارش , در این جا از آوردن تمامی آن ها خودداری می کنیم و فقط بخشی از آن ها را مشاهده می کنیم.

مشاهده می کنیم علت خطا در این قسمت این است که بیشتر کلماتی که در دادگان آموزش[4] حضور ندارند (در واقع به عنوان کلمات ناشناخته تلقی می شوند) , برای سادگی به اولین تگ موجود در تگ ست آموزش[5] اختصاص داده می شوند.

در قسمت بعدی بیشتر به کلمات ناشناخته و راهکار های مقابله با آن می پردازیم.

ث)

به طور کلی راه های گوناگونی برای مقابله با کلمات ناشناخته وجود دارد که در اینجا به دو مورد از مهمترین آن ها که در اینجا استفاده کردیم می پردازیم.

---

[1] Emission probability
[2] NOUN
[3] Unknown words
[4] Training set
[5] Training POS tagset

یکی از کارآمد ترین راهکار ها استفاده از قواعد مبتنی بر قانون[1] می باشد. به این صورت که ابتدا مشاهده می کنیم بیشترین تعداد کلمات ناشناخته متعلق به کدام تگ می باشد (در اینجا NOUN) سپس متناسب با نشانه های مورفولوژیکی[2] برای هر تگ , قواعد مبتنی بر قانون وضع می کنیم که عبارت اند از :

```
(r'.*ing$', 'VERB'),                # gerund
(r'.*ed$', 'VERB'),                 # past tense verbs
(r'.*es$', 'VERB'),                 # singular present verbs
(r'.*ould$', 'VERB'),               # modal verbs
(r'.*\'s$', 'NOUN'),                # possessive nouns
(r'.*s$', 'NOUN'),                  # plural nouns
(r'^-?[0-9]+(.[0-9]+)?$', 'NUM'),   # cardinal numbers
(r'(The|the|A|a|An|an)$', 'DET'),   # articles or determinants
(r'.*able$', 'ADJ'),                # adjectives
(r'.*ness$', 'NOUN'),               # nouns formed from adjectives
(r'.*ly$', 'ADV'),                  # adverbs
(r'.*', 'NOUN')                     # nouns
```

به عنوان یک نمونه , کلماتی که با able ختم می شوند احتمالا صفت هستند.

یکی دیگر از مشکلات الگوریتم Viterbi در مواجه شدن با کلمات ناشناخته ای که در مجموعه دادگان آموزش نیست , قرار دادن احتمال صفر برای آن ها و در نتیجه انتخاب اولین تگ در مجموعه تگ آموزش برای این کلمات است. برای رفع این مشکل الگوریتم Viterbi را به گونه ای اصلاح می کنیم که وقتی احتمالات حالت صفر شد , تگ را بر اساس برچسبی که در قسمت قبل ایجاد کردیم (rule-based) به روز کند. در واقع برای رویارویی با کلمات ناشناخته الگوریتم را به گونه ای تغییر می دهیم که در این مواقع فقط یکی از احتمالات انتقال[3] یا انتشار[4] را در نظر بگیرد.

پس از اصلاح کردن الگوریتم , مجددا آن را بر روی دادگان آزمون اجرا می کنیم و دقت جدید را محاسبه می کنیم که خواهیم داشت :

The accuracy using modified Viterbi algorithm is: 0.9524922118380063 %

مشاهده می کنیم که پس از اصلاح الگوریتم , دقت بر روی دادگان آزمون بهتر شده و حدود 1.3 درصد افزایش میابد.

توجه : همانند قسمت پ در این قسمت باز هم اجرای مدل بر روی دادگان تست زمان بر بود و حدود 17 دقیقه ارزیابی مدل و اجرای آن بر روی دادگان آزمون زمان برد ! (پیشاپیش از صبر و شکیبایی شما مجددا سپاسگزارم)

[1] Rule-based
[2] Morphological cues
[3] Transition
[4] Emission

ج)

در این قسمت , برچسب زدن را با استفاده از مدل های بازگشتی همچون RNN[1] پیاده سازی می کنیم. در اینجا برخلاف قسمت قبل نیاز داریم تا پارامتر ها و فراپارامتر[2] های مدل خود را با استفاده از دادگان ارزیابی[3] به دست آوریم.

در اینجا 15 درصد دادگان را به داده های آزمون اختصاص می دهیم. سپس 15 درصد مابقی دادگان را نیز به دادگان ارزیابی اختصاص می دهیم و باقی را به دادگان آموزش اختصاص می دهیم و خواهیم داشت :

Train : 72.25% Test : 15% Validation : 12.75%

در واقع مزیت استفاده از دادگان ارزیابی و اهمیت آن نسبت به دادگان آزمون و آموزش در این است که می توانیم با استفاده از دادگانی که هنوز ندیده ایم[4] و از آن ها در آموزش مدل استفاده نکرده ایم , قبل از ارزیابی نهایی مدل بر روی دادگان آزمون , ارزیابی اولیه ای جهت پیدا کردن و بهینه کردن پارامتر های مدل خود انجام دهیم.

چ)

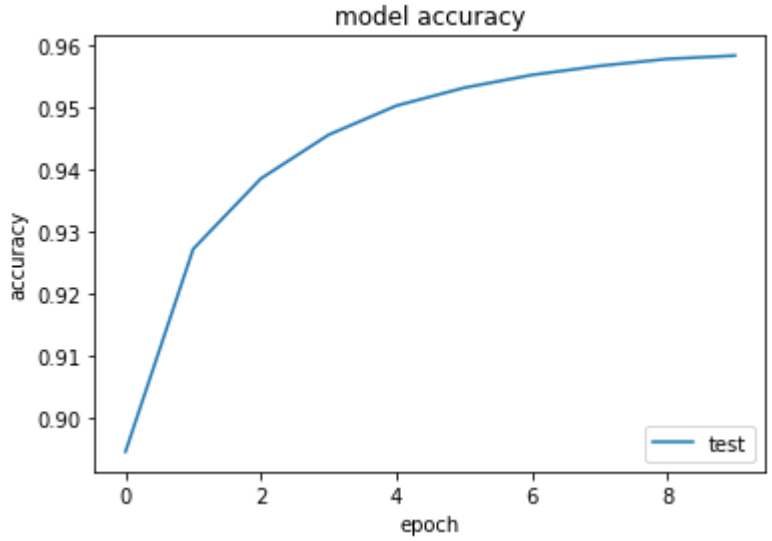
در ادامه مدل به دست آمده را بر روی دادگان آزمون در طی 10 دوره[5] ارزیابی کرده و در هر دوره , مقادیر دقت و ضرر[6] را گزارش می کنیم.

```
Epoch 1/10
408/408 [==============================] - 39s 89ms/step - loss: 0.5065 - acc: 0.8556 - val_loss: 0.3372 - val_acc: 0.8947
Epoch 2/10
408/408 [==============================] - 36s 88ms/step - loss: 0.2829 - acc: 0.9127 - val_loss: 0.2377 - val_acc: 0.9273
Epoch 3/10
408/408 [==============================] - 36s 89ms/step - loss: 0.2149 - acc: 0.9332 - val_loss: 0.1937 - val_acc: 0.9386
Epoch 4/10
408/408 [==============================] - 37s 90ms/step - loss: 0.1822 - acc: 0.9418 - val_loss: 0.1691 - val_acc: 0.9456
Epoch 5/10
408/408 [==============================] - 36s 88ms/step - loss: 0.1622 - acc: 0.9474 - val_loss: 0.1532 - val_acc: 0.9503
Epoch 6/10
408/408 [==============================] - 36s 88ms/step - loss: 0.1488 - acc: 0.9515 - val_loss: 0.1432 - val_acc: 0.9532
Epoch 7/10
408/408 [==============================] - 36s 89ms/step - loss: 0.1398 - acc: 0.9539 - val_loss: 0.1353 - val_acc: 0.9552
Epoch 8/10
408/408 [==============================] - 36s 89ms/step - loss: 0.1333 - acc: 0.9556 - val_loss: 0.1297 - val_acc: 0.9567
Epoch 9/10
408/408 [==============================] - 36s 89ms/step - loss: 0.1285 - acc: 0.9569 - val_loss: 0.1258 - val_acc: 0.9578
Epoch 10/10
408/408 [==============================] - 36s 89ms/step - loss: 0.1249 - acc: 0.9579 - val_loss: 0.1230 - val_acc: 0.9584
```

مشاهده می کنیم دقت مدل در پایان اولین دوره از 85% شروع شده و در پایان دوره دهم به چیزی نزدیک به 95% رسیده است. نزدیک به دقتی که با الگوریتم Viterbi اصلاح شده به دست آورده بودیم.

[1] Recurrent neural network
[2] Hyperparameter
[3] Validation data
[4] Unseen data
[5] Epoch
[6] Loss

در ادامه نمودار دقت بر حسب دوره به دست آمده توسط RNN را مشاهده می کنیم.



شکل 2 : نمودار دقت بر حسب دوره برای RNN

ج)

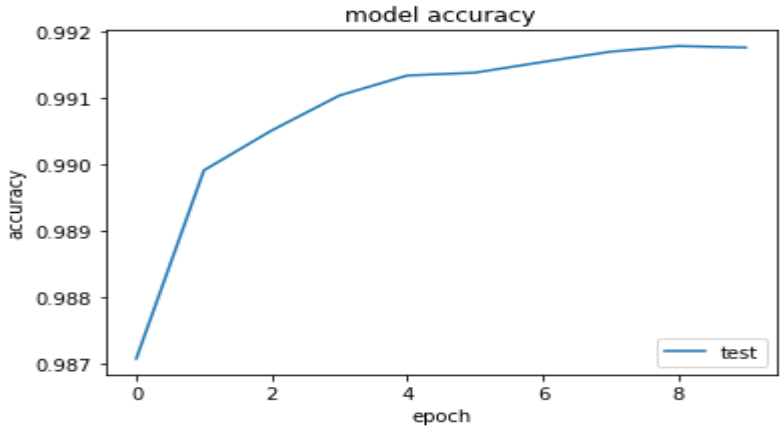
در این قسمت نیز همانند قسمت قبل الگوریتم را این بار با LSTM و GRU پیاده سازی می کنیم و دقت را گزارش می کنیم. (تقسیم بندی دادگان آموزش , آزمون و ارزیابی نیز دقیقا همانند قسمت ج می باشد و بدون تغییر باقی می ماند)

LSTM

دقت و خطا را برای هر دوره با استفاده از به کارگیری LSTM مشاهده می کنیم :

```
Epoch 1/10
408/408 [==============================] - 194s 470ms/step - loss: 0.3111 - acc: 0.9295 - val_loss: 0.0497 - val_acc: 0.9871
Epoch 2/10
408/408 [==============================] - 191s 468ms/step - loss: 0.0351 - acc: 0.9895 - val_loss: 0.0304 - val_acc: 0.9899
Epoch 3/10
408/408 [==============================] - 191s 468ms/step - loss: 0.0243 - acc: 0.9915 - val_loss: 0.0268 - val_acc: 0.9905
Epoch 4/10
408/408 [==============================] - 192s 470ms/step - loss: 0.0205 - acc: 0.9925 - val_loss: 0.0251 - val_acc: 0.9910
Epoch 5/10
408/408 [==============================] - 192s 470ms/step - loss: 0.0183 - acc: 0.9933 - val_loss: 0.0243 - val_acc: 0.9913
Epoch 6/10
408/408 [==============================] - 192s 471ms/step - loss: 0.0167 - acc: 0.9938 - val_loss: 0.0240 - val_acc: 0.9914
Epoch 7/10
408/408 [==============================] - 192s 470ms/step - loss: 0.0154 - acc: 0.9943 - val_loss: 0.0239 - val_acc: 0.9915
Epoch 8/10
408/408 [==============================] - 191s 469ms/step - loss: 0.0141 - acc: 0.9948 - val_loss: 0.0238 - val_acc: 0.9917
Epoch 9/10
408/408 [==============================] - 191s 469ms/step - loss: 0.0128 - acc: 0.9954 - val_loss: 0.0239 - val_acc: 0.9918
Epoch 10/10
408/408 [==============================] - 192s 470ms/step - loss: 0.0114 - acc: 0.9960 - val_loss: 0.0242 - val_acc: 0.9918
```

در ادامه نمودار دقت بر حسب دوره به دست آمده توسط LSTM را مشاهده می کنیم.



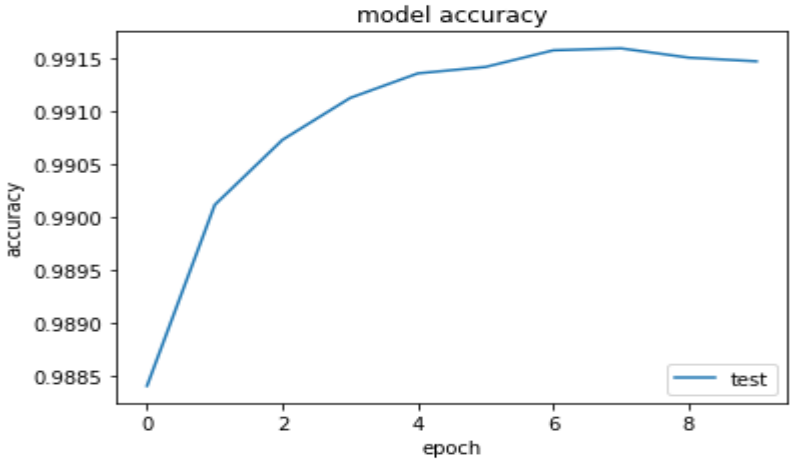
شکل 3 : نمودار دقت بر حسب دوره برای LSTM

GRU

دقت و خطا را برای هر دوره با استفاده از به کارگیری GRU مشاهده می کنیم :

```
Epoch 1/10
408/408 [==============================] - 182s 439ms/step - loss: 0.2224 - acc: 0.9574 - val_loss: 0.0356 - val_acc: 0.9884
Epoch 2/10
408/408 [==============================] - 179s 438ms/step - loss: 0.0278 - acc: 0.9903 - val_loss: 0.0272 - val_acc: 0.9901
Epoch 3/10
408/408 [==============================] - 179s 438ms/step - loss: 0.0210 - acc: 0.9922 - val_loss: 0.0251 - val_acc: 0.9907
Epoch 4/10
408/408 [==============================] - 180s 441ms/step - loss: 0.0182 - acc: 0.9932 - val_loss: 0.0242 - val_acc: 0.9911
Epoch 5/10
408/408 [==============================] - 178s 436ms/step - loss: 0.0165 - acc: 0.9938 - val_loss: 0.0238 - val_acc: 0.9914
Epoch 6/10
408/408 [==============================] - 178s 437ms/step - loss: 0.0152 - acc: 0.9943 - val_loss: 0.0236 - val_acc: 0.9914
Epoch 7/10
408/408 [==============================] - 183s 449ms/step - loss: 0.0141 - acc: 0.9947 - val_loss: 0.0238 - val_acc: 0.9916
Epoch 8/10
408/408 [==============================] - 178s 435ms/step - loss: 0.0131 - acc: 0.9951 - val_loss: 0.0242 - val_acc: 0.9916
Epoch 9/10
408/408 [==============================] - 177s 434ms/step - loss: 0.0121 - acc: 0.9955 - val_loss: 0.0247 - val_acc: 0.9915
Epoch 10/10
408/408 [==============================] - 178s 436ms/step - loss: 0.0111 - acc: 0.9959 - val_loss: 0.0255 - val_acc: 0.9915
```

در ادامه نمودار دقت بر حسب دوره به دست آمده توسط GRU را مشاهده می کنیم.



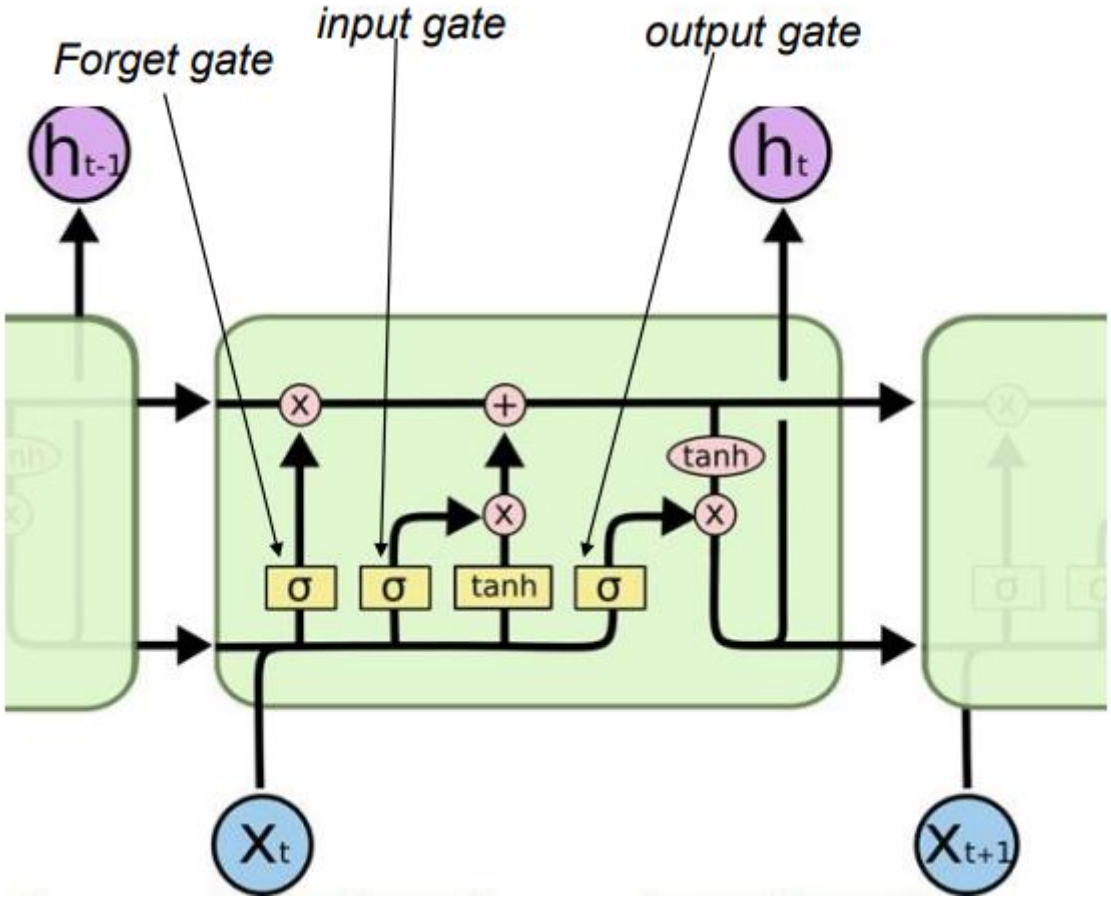
شکل 4 : نمودار دقت برحسب دوره برای GRU

همان طور که مشاهده می کنیم LSTM بهتر از RNN و GRU بهتر از LSTM عمل کرده است. به طور کلی در شبکه های RNN فقط از ورودی و لایه ی مخفی[1] به دست آمده در لایه ی قبل استفاده می کنیم اما در LSTM علاوه بر این از متن نوشته[2] مرحله قبل نیز استفاده می کنیم. به این معنی که در هر مرحله می توانیم بسته به نیاز اطلاعاتی از مراحل قبل را اضافه یا حذف کنیم.

پس در واقع LSTM و GRU دو مشکل بزرگ RNN را که یکی فاصله بین کلمات مرتبط است[3] و دیگری صفر یا بینهایت شدن احتمال کل ناشی از ضرب احتمالات را برطرف می کنند که منجر به افزایش دقت خروجی می شود.

خ)

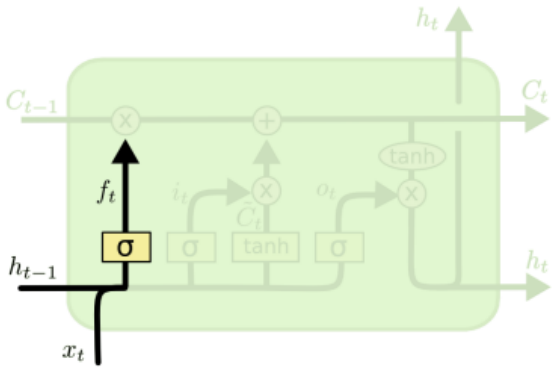
به طور کلی LSTM دارای 3 گیت حذف (Forget gate) , اضافه (Input gate) و خروجی (Outputgate) می باشد. ابتدا نمای کلی از گیت های LSTM را مشاهده می کنیم و سپس به توضیح هر یک از آن ها و مقایسه ی کلی آن با گیت های GRU می پردازیم.



شکل 5 : نمای کلی از گیت های LSTM

[1] Hidden layer
[2] Context
[3] Gap between relevant words

Forget gate : در این گیت با استفاده از تابع سیگموییدکه می تواند مقادیر بین 0 و 1 اختیار کند , می توانیم تصمیم بگیریم که آیا اطلاعاتی که از قبل داریم و دیگر مورد استفاده قرار نمی گیرد یا بخشی از آن را حذف کنیم یا خیر.

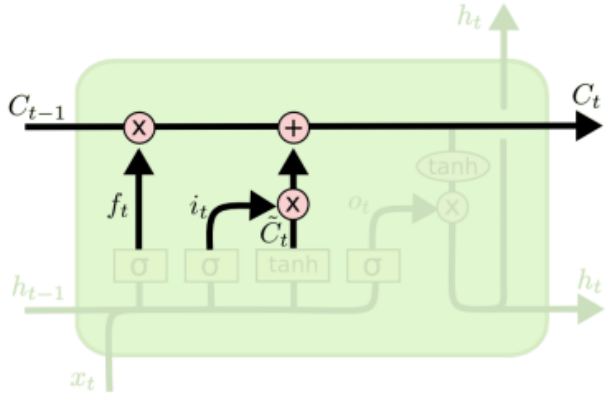


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

شکل 6 : نمایی از Forget gate در LSTM

در صورتی که تابع سیگموید مقدار صفر اختیار کند اطلاعات لایه ی قبل را حذف می کنیم و اگر مقدار یک را اختیار کند تمامی اطلاعات لایه ی قبل را حفظ خواهیم کرد.

Input gate : در این گیت تصمیم می گیریم آیا اطلاعات جدیدی را می خواهیم به لایه ی متن[1] اضافه کنیم یا خیر و این کار را نیز مشابه با قسمت قبل با استفاده از تابع سیگموید موجود انجام می دهیم.

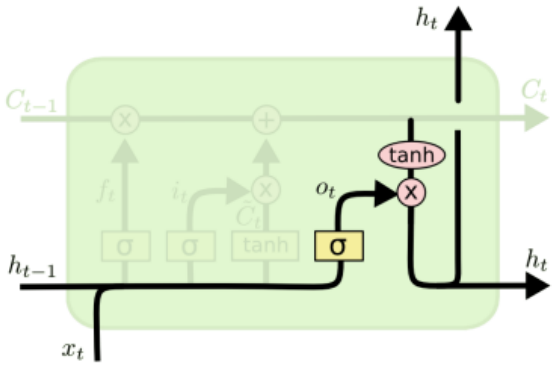


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

شکل 7 : نمایی از Input gate در LSTM

Output gate : در این گیت نیز در نهایت تصمیم می گیریم که چه چیزی را به عنوان خروجی این لایه انتخاب می کنیم.

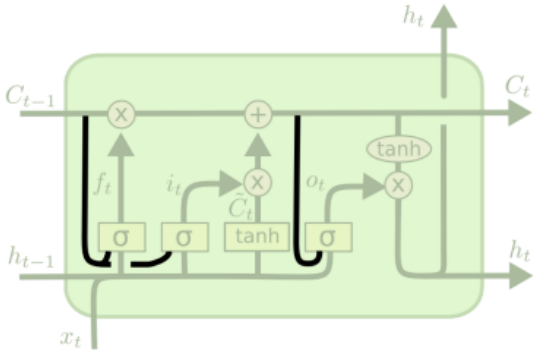
[1] Context layer



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

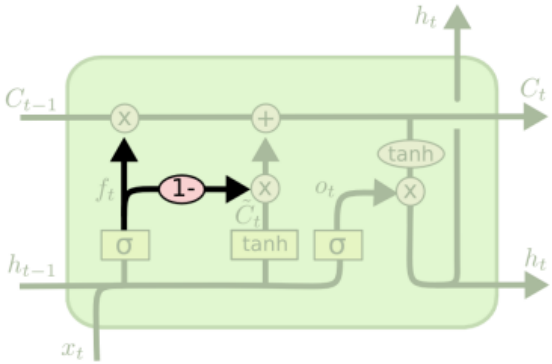
شکل 8 : نمایی از Output gate در LSTM

البته LSTM محدود به این ها نمی شود و انواع دیگری نیز همانند Peephole connections و یا Coupled forget and input gates دارد که شامل تغییراتی در گیت ها و اتصال آن ها به یکدیگر می باشد.



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t]\ +\ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t]\ +\ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t]\ +\ b_o\right)$$

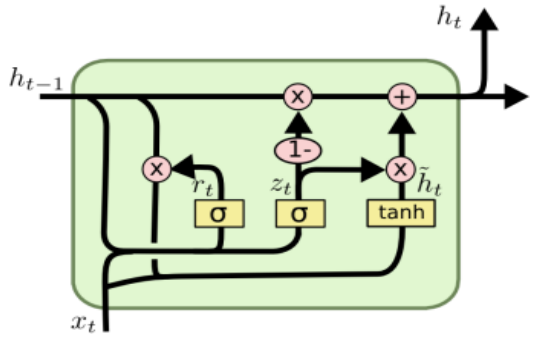
شکل 9 : Peephole connections



$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

شکل 10 : Coupled forget and input gates

حال برای این که متوجه شباهت ها و تفاوت های LSTM با GRU شویم ابتدا نمای کلی از گیت های GRU را مشاهده می کنیم.

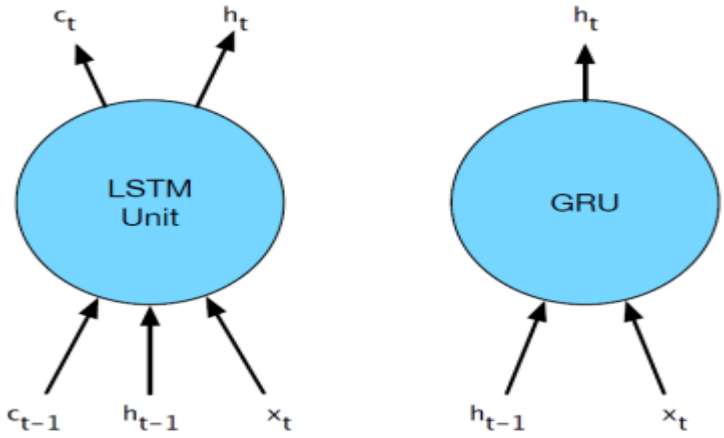


$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

شکل 11 : نمای کلی از گیت های GRU

مشاهده می کنیم در GRU بر خلاف LSTM فقط یک گیت داریم و به نوعی Cell state و Hidden state با هم ترکیب شده بعلاوه این که دو گیت Forget , Input نیز با هم ترکیب شده و Update gate را تشکیل داده اند.

در نهایت نیز ورودی ها و خروجی های هر لایه از LSTM و GRU را مشاهده می کنیم :



شکل 12 : نمای کلی از ورودی ها و خروجی های LSTM و GRU

د)

همان طور که مشاهده کردیم LSTM و GRU با اختلاف قابل ملاحظه ای از RNN بهتر عملکردند. همچنین GRU نیز در تعداد دوره کمتری به دقت بهتری نسبت به LSTM رسید.

در نهایت از میان مدل های بازگشتی GRU بهترین عملکرد را داشت و در نهایت تا دقت حدود 99.6 درصد دست پیدا کرد , درصورتی که دقت الگوریتم Viterbi اصلاح شده نهایتا تا 95 درصد می رسید.

دریافتیم که در مسائل پیش بینی سری زمانی[1] که بعضا ورودی یا خروجی طول غیرثابت[2] دارند استفاده از مدل های بازگشتی می تواند به شدت راهگشا باشد.

[1] Time-series Prediction
[2] Unfixed length

2- تشخیص گروه های اسمی

الف)

در این قسمت با استفاده از کتابخانه nltk و دستور nltk.tree2conlltags(nltk.ne_chunk(sent)) برای هر کلمه در سیستم BIO تگ مربوط به named entity را مشخص می کنیم.

به عنوان نمونه جمله ی اول را به همراه برچسب های آن مشاهده می کنیم :

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

ب)

ابتدا تقسیم بندی دادگان به داده های آزمون و آموزش را همانند بخش اول به نسبت به ترتیب 95 و 5 درصد انجام می دهیم. (در این قسمت نیز به علت این که نیازی به محاسبه ی پارامتر و یا فراپارامتری نداریم در نتیجه نیازی به استفاده از داده های ارزیابی نخواهیم داشت)

در نیاز به تشکیل ماتریس انتقال[1] خواهیم داشت که یک ماتریس مربعی به صورت زیر خواهد بود :

	B-GSP	I-GPE	I-PERSON	O	B-PERSON	I-ORGANIZATION	B-FACILITY	B-ORGANIZATION	B-LOCATION	I-GSP	B-GPE	I-LOCATION	I-FACILITY
B-GSP	0.000000	0.000000	0.000000	0.948718	0.000000	0.000000	0.000000	0.025641	0.000000	0.025641	0.000000	0.000000	0.000000
I-GPE	0.000000	0.036000	0.000000	0.948000	0.008000	0.000000	0.000000	0.008000	0.000000	0.000000	0.000000	0.000000	0.000000
I-PERSON	0.000000	0.000000	0.157248	0.839476	0.003276	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
O	0.000445	0.000000	0.000000	0.943183	0.020551	0.000000	0.000582	0.015168	0.000365	0.000000	0.019696	0.000000	0.000000
B-PERSON	0.000000	0.000000	0.526074	0.354294	0.062883	0.000000	0.000000	0.043967	0.000000	0.000000	0.012781	0.000000	0.000000
I-ORGANIZATION	0.000000	0.000000	0.000000	0.744934	0.000000	0.251490	0.000000	0.002384	0.000000	0.000000	0.001192	0.000000	0.000000
B-FACILITY	0.000000	0.000000	0.000000	0.019608	0.058824	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.921569
B-ORGANIZATION	0.000000	0.000000	0.000000	0.548452	0.004711	0.422611	0.000000	0.022207	0.000000	0.000000	0.002019	0.000000	0.000000
B-LOCATION	0.000000	0.000000	0.000000	0.060606	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.030303	0.909091	0.000000
I-GSP	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
B-GPE	0.000000	0.136699	0.000000	0.833239	0.008508	0.000000	0.000000	0.017584	0.000567	0.000000	0.003403	0.000000	0.000000
I-LOCATION	0.000000	0.000000	0.000000	0.967742	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.032258	0.000000
I-FACILITY	0.000000	0.000000	0.000000	0.851852	0.000000	0.000000	0.000000	0.018519	0.000000	0.000000	0.000000	0.000000	0.129630

شکل 13 : ماتریس انتقال

سایر مراحل را نیز همانند بخش اول انجام می دهیم.

[1] Transition

پ)

در نهایت پس از پیاده سازی مدل و ارزیابی آن بر روی دادگان آزمون , مقادیر خواسته شده را محاسبه می کنیم.

توجه : طراحی مدل و پیاده سازی و ارزیابی آن بر روی دادگان آزمون در هنگام اجرا در google colab چیزی نزدیک به 40 دقیقه زمان برد , لذا پیشاپیش از صبر و شکیبایی شما سپاسگزارم.

در نهایت مقادیر خواسته شده به صورت زیر خواهد بود :

The recall using Viterbi algorithm is: 0.4334967325520667 %
The precision using Viterbi algorithm is: 0.392953100798435 %
The F1 using Viterbi algorithm is: 0.4032764591143296 %

ت)

استفاده از مدل های بازگشتی همانند RNN برای حل مسائل NER پیشنهاد نمی شود. یکی از مشکلاتی که ممکن است به وجود آید رخ داد توالی های غیر ممکن است.

به عنوان مثال RNN ضمانت نمی کند که قبل از I (inside) لزوما O (outside) نیاید و به طور کلی همان طور که اشاره شد تضمینی برای مقابله با رخ دادن توالی های غیر ممکن ندارد.

راه حل موجود استفاده از CRFs[1] ها است. به این صورت که بعد از RNN یک لایه ی CRF اضافه می کنیم که در این صورت خروجی دنباله ای از بهترین خروجی ها است نه یک تگ. در نهایت بعد از لایه ی CRF از Viterbi استفاده می کنیم تا محتمل ترین دنباله را انتخاب کند , در این صورت توانسته ایم با رخ دادن توالی های غیر ممکن مقابله کنیم.

[1] Conditional Random Fields

پیوست :

تمامی فایل های ipynb در پوشه ی Codes موجود می باشد. همچنین تمامی کد ها در محیط colab اجرا و تست شده اند.

با توجه به شرایط و پیاده سازی های موجود در این پروژه و بعضا سنگین بودن حجم پردازشات , اجرای برخی سل ها زمان بر خواهد بود (زمان تقریبی اجرای این سل ها بعضا در گزارش قید شده است) , لذا پیشاپیش از صبر و تحمل شما برای اجرا شدن کدها سپاسگزارم.