گزارش کار تمرین اول مبانی پردازش زبان طبیعی

استاد قاسمی

سید مجتبی ابطحی

۲۲۰۷۹۸۰۲۴

در ابتدا چون من آشنایی چندانی با شبکه های عصبی نداشتم شروع به یادگیری آن کردم و به دیدن چندین آموزش و فیلم مشغول شدم و در نهایت برای اینکه بهتر یادگیری انجام شود این تمرین را ابتدا در قالب پایتون خالص و بدون استفاده از کتابخانه ای مانند تنسورفلو و پای تورچ شروع به انجام تمرین کردم

مدل اول (استفاده از پایتون خالص):

در ابتدا من کلمه های stop زبان فارسی را import کردم تا بتوانم از متن متن که به من داده شده آنها را حذف کنم و همچنین از ویژگی ریشه یابی و نرمال سازی کتابخانه هضم استفاده کردم تا بهتر مدل عصبی کار بکند و جواب های بهتری بدهد

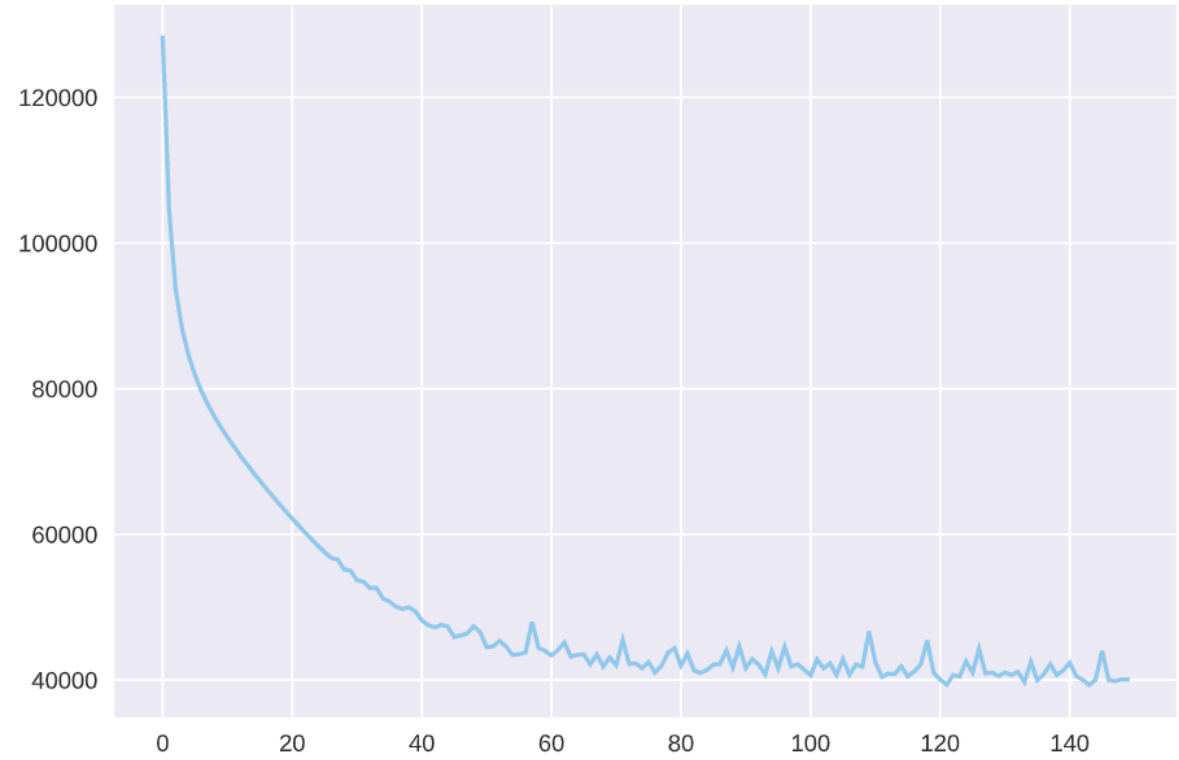
و سپس به اندازه window ما دیتا x , y مان را تولید می کنیم به صورت one-hot شده

و سپس بعد از tokenizer کتابخانه keras استفاده کردم برای این که بتوانم کلمات یکتای متن خودم را پیدا کنم و به آنها شناسه بدهم(شناسه به کلمه و کلمه به شناسه)

در مرحله بعدی ما مدلمان را درست می کنیم ساختار آن به این شکل است که کلمات مرکز را به صورت one-hotشده به عنوان ورودی به شبکه می دهیم و یک لایه hidden داریم که تعداد عصب های آن به تعداد ابعاد کلمه های ماست که همان word-embedding میشود و در آخر لایه خروجی ما که تعداد عصب های آن به اندازه تعداد کلمات یکتای ما است درست می کنیم و در آخر آن را از تابع softmax عبور میدهیم که یک خروجی احتمالاتی داشته باشیم و بتوانیم آن را با کلمه همسایه همان کلمه ای که به عنوان کلمه مرکز به شبکه دادیم مقایسه کنیم و تفاوت آن را با استفاده از تابع cross_entropy به دست می آوریمو با استفاده از روش بهینه سازی gradient decent پارامتر های مان را به روز رسانی می کنیم

که در آن مرحله های مختلف برای ساخت یک شبکه عصبی را پیاده سازی کردم مثل foward backward و محاسبه مشتق جزئی هر پارامتر که همان وزن های ما می شود که همان جزو مرحله gradient decent را انجام دادم

نمودار کاهش loss مدل من بعد از ۱۵۰ ایپاک بدین شکل است:



در آخر برای گرفتن نتیجه کار تابعی درست کردم که به آن کلمه ورودی را می‌دهیم و تعداد کلمه های مشابه را هم به عنوان پارامتر به تابع میدهیم و در خروجی به ما کلمه های مشابه به آن کلمه ورودی را خروجی میدهد نتایج بدین شکل است:

```
get_word_similarities('بشر', model, 10)
```

```
محمود
حضرته
صدقه
بذاک
باللقاء
بشر
الکریم
یجعلنا
قد
حمدنا
```

```
get_word_similarities('ویرانه', model, 10)
```

```
باد
دید
وجود
عالم
جوید
جغدان
حلال
صد
ویرانه
باغ
```

```
get_word_similarities('انگور', model, 20)
```

```
صافی
میخانه
فسرد
چونک
خوشه
انگور
خورشید
اجل
دوید
غرقه
ربود
کوی
دگران
بنگرید
پرتو
همگی
درد
اوفتاد
پر
صحرا
```

همانطور که مشاهده می کنید نتایج خوبی را این مدل پس از یادگیری به ما ارائه می دهد

پس از آن به سمت پیاده سازی شبکه عصبی به کمک کتابخانه های موجود keras رفتم

مدل دوم (استفاده از کتابخانه keras):

که بسیار شبیه به پیاده سازی اولم با پایتون خالص بود با این تفاوت که به جای اینکه کل متن ام را به هم بچسبانم بر روی آن کلمه های همسایه را پیدا کنم آمدم و هر بیت را به صورت جداگانه روی آن کلمه های همسایه را پیدا کردم چون اگر بیت ها را به هم می چسباندم باعث می‌شد که به طور مثال کلمه آخر یک بیتی به کلمه های اول بیت دیگر مربوط می‌شود که این اشتباه است

یکی از مشکلات مهم و وقت گیری که در حین پیاده سازی این مدل ها داشتم این بود که به محدودیت مموری میخوردم و مجبور بودم هر متغیری که حجم زیادی از معماری را می گرفت آن را پاک کنم تا مموری خالی شود ولی باز هم جوابگو نبود و مجبور شدم که کلمه هایی که کمتر از عددی تکرار شده اند را از token هایم حذف کنم

تغییر دیگری که داده ام این بود که برای one-hot کردن کلمه هایم دیگر از تابعی که خودم در مدل قبلی زده بودم استفاده نکردم و از OneHotEncoder sklearn استفاده کردم که این باعث سرعت بخشیدن به این مرحله می شد

قالب این شبکه همانند شبکه قبلی است با این تفاوت که به جای روش بهینه سازی GD از روش بهینه سازی adam استفاده کردم

ساختار مدل:

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 2575)]            0

 dense (Dense)               (None, 100)               257600

 dense_1 (Dense)             (None, 2575)              260075

=================================================================
Total params: 517,675
Trainable params: 517,675
Non-trainable params: 0
_________________________________________________________________
```

و همانطور که در تصویر پایین میبینید در هنگام train کردن پس از ۱۰۰ ایپاک مدل من مقدار loss بسیار زیاد می باشد و همچنین مقدار دقت آن نیز بسیار کم می باشد

```
940/940 [==============================] - 5s 5ms/step - loss: 4.4726 - accuracy: 0.0561 - mse: 3.8033e-04
Epoch 97/100
940/940 [==============================] - 5s 5ms/step - loss: 4.4717 - accuracy: 0.0571 - mse: 3.8035e-04
Epoch 98/100
940/940 [==============================] - 5s 5ms/step - loss: 4.4709 - accuracy: 0.0564 - mse: 3.8035e-04
Epoch 99/100
940/940 [==============================] - 5s 5ms/step - loss: 4.4700 - accuracy: 0.0568 - mse: 3.8036e-04
Epoch 100/100
940/940 [==============================] - 5s 5ms/step - loss: 4.4691 - accuracy: 0.0571 - mse: 3.8035e-04
```

ولی با این حال نیز نتایجی که به عنوان کلمه های مشابه این مدل به ما میدهد قابل قبول می باشد به مانند تصویر زیر

```
n_similar(['خلق', 'آتش', 'شادی'], model, n=10)
```

```
خلق = ['رنگ', 'گل', 'دم', 'دانست#دان', 'جان', 'جگر', 'بو', 'یار', 'سفر', 'یافت#یاب']
آتش = ['شکر', 'دل', 'لب', 'تنگ', 'عشق', 'جان', 'شهد', 'کان', 'پر', 'خوش']
شادی = ['مست', 'دل', 'میان', 'خویش', 'حقست', 'زیر', 'موسی', 'آفتاب', 'قسمت', 'التیه']
```

مدل سوم و نهایی (استفاده از روش بهینه سازی negative sampling):

پس از دیدن نتایج نامطلوبی که از آخرین مدل گرفتم سعی کردم یکی از روش های بهینه سازی word2vec آقای milkolov پیشنهاد داده بود را پیاده سازی بکنم که همان negative-sampling می باشد

که در پیاده سازی این مدل من از پیاده سازی یکی از وب سایت ها کمک گرفتم

که تفاوت این مدل با مدل قبلی در تولید کردن دیتای ورودی است که مدل این گونه کار می کند که ما کلمات همسایه را با کمک متد skipgrams کتابخانه keras به عنوان آیدی میگیریم و آن را به عنوان نمونه مثبت نامگذاری میکنیم و به آن برچسب یک می دهیم و سپس برای اینکه کلمه های غیر از کلمه های همسایه آن کلمه مرکزی را اثرش را کم کنیم می آییم و چند نمونه کلمه از کل متن جدا می کنیم با استفاده از روش unigram distribution که این کلمات را به عنوان کلمات منفی نام گذاری می کنیم و به آن برچسب صفر می دهیم و این بدین معناست که این کلمه مرکزی با این کلمات منفی نمی آید و باید اثر آن را کم کنیم و برعکس آن این کلمه مرکزی با این کلمه مثبت کلمه همسایه آن می باشد را باید اثر آن را زیاد کنیم

برای مقدار دهی اولیه ابعاد کلمات ما از لایه Embedding کتابخانه keras استفاده می کنیم که کار آن به این صورت است که می آید برای هر کلمه به صورت رندم ابعادی به اندازه ابعادی که ما می خواهیم مقدار می دهد

ساختار این مدل بدین گونه است که ما یک لایه از ابعاد کلمات مرکزی و یک لایه از ابعاد کلمات همسایه داریم که ما به عنوان ورودی مدل لیستی از کلمات مرکز به علاوه لیستی از کلمه همسایه هر کلمه به علاوه تعداد کلمات منفی نمونه برداری شده از کل متن به عنوان ورودی به شبکه می دهیم و در خروجی آن را با برچسب هایی که به کلمه های منفی و کلمه مثبت زده ایم مقایسه می کنیم و بدین گونه به شبکه میفهمانیم که باید کلمه همسایه اهمیت داده بشود و به کلمه ای که در محدوده همسایه کلمه مرکزی نیست اهمیت کمتری داده بشود

ساختار مدل:

```
Model: "word2_vec"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 center_embedding (Embedding  multiple                 1784200
 )

 context_embedding (Embeddin  multiple                 1784200
 g)

=================================================================
Total params: 3,568,400
Trainable params: 3,568,400
Non-trainable params: 0
_________________________________________________________________
```

و این نمونه برداری به این دلیل است که به جای اینکه در مدل قبل همه کلمه های غیر همسایه را برداریم و تاثیر همه آنها را کم کنیم می آییم و چند مقدار کمرا نمونه برداری می کنید و آنها را کم میکنیم و این به سرعت train شبکه بسیار کمک می کند و دیگر لازم نیست که از لایه softmax عبور کند و باعث محاسبات سنگین زیادی شود

و برای اینکه وزن های کلمات را بگیریم از لایه وزن های کلمات مرکزی مقادیر آن را می گیریم با سپس بعد از آن با استفاده از روش مشابهت کسینوسی میتوانیم بفهمیم که یک کلمه کدام یک از کلمات بیشترین نزدیکی را به آن دارند و این کار را توانستم با استفاده از متد cosine_similarity کتابخانه sklearn انجام بدهم

نتیجه در هنگام train بسیار خوب میباشد :

```
Epoch 43/50
67/67 [==============================] - 1s 11ms/step - loss: 0.6619 - accuracy: 0.7824
Epoch 44/50
67/67 [==============================] - 1s 11ms/step - loss: 0.6582 - accuracy: 0.7824
Epoch 45/50
67/67 [==============================] - 1s 11ms/step - loss: 0.6548 - accuracy: 0.7824
Epoch 46/50
67/67 [==============================] - 1s 11ms/step - loss: 0.6516 - accuracy: 0.7824
Epoch 47/50
67/67 [==============================] - 1s 11ms/step - loss: 0.6487 - accuracy: 0.7824
Epoch 48/50
67/67 [==============================] - 1s 11ms/step - loss: 0.6459 - accuracy: 0.7823
Epoch 49/50
67/67 [==============================] - 1s 11ms/step - loss: 0.6433 - accuracy: 0.7822
Epoch 50/50
67/67 [==============================] - 1s 11ms/step - loss: 0.6409 - accuracy: 0.7822
```

و اگر تعداد سمپل های منفی را کمتر کنیم دقت ازین هم بیشتر میشود و loss هم کمتر میشود ولی کلمه های پیشنهادی بدتر میشوند چون کلمه های منفی کمتر میشوند و تاثیر انها هم کم میشود به این دلیل مدل ما کمتر یاد میگیرد نسبت به همه متن مان

که در عکس پایین نتایج به دست آمده از مدل را مشاهده می کنیم که بسیار قابل قبول می باشد

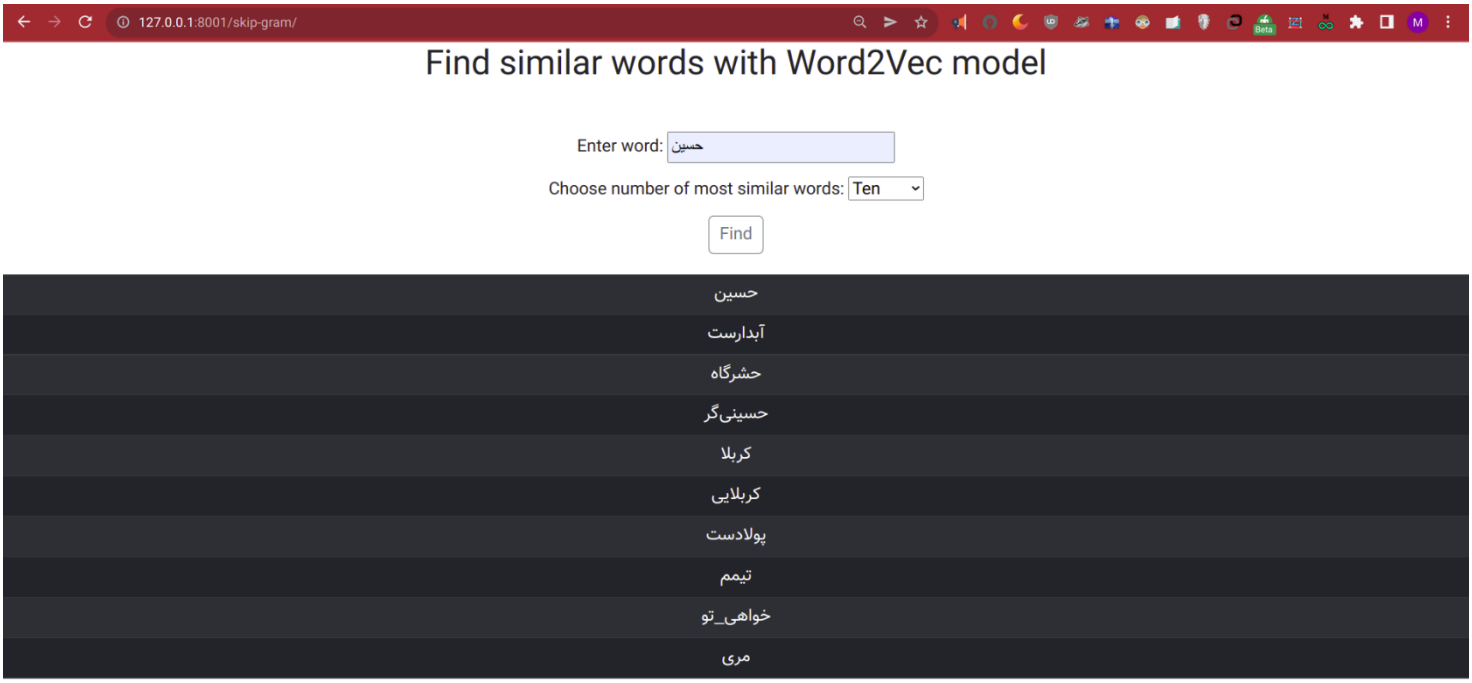
```
In 163  1  cosine_similarity_word(['حسین', 'یوسف', 'خسرو', 'گل'], cosine_matrix, 10)

    ['حسین', 'آبدارست', 'کربلایی', 'شهید', 'دلست', 'مرتضی', 'سزا', 'گفتا', 'پرفنی', 'کاثر'] = حسین

    ['یوسف', 'شی', 'میu200cرا', 'بدیشان', 'امراه', 'زلیخا', 'عورت', 'تواند', 'خوبروی', 'بنمایمش'] = یوسف

    ['خسرو', 'فرهاد', 'فرمانست', 'زیر', 'بشکافته', 'نهلد', 'حاصل', 'گلu200cگر', 'رعنا', 'ایمنست'] = خسرو

    ['گل', 'کلوخ', 'منحنی', 'اندازیست', 'خند', 'بدریدهu200cای', 'شکنست', 'درآری', 'بنفشه', 'گردیu200cگر'] = گل
```

و پس از آن دیتاهای مورد نیاز برای وب اپ را در قالب فایل pickle ذخیره می کنم که تا در نشان دادن نمودار پراکندگی سه بعدی و دو بعدی بردار کلمات از آن استفاده بکنیم با کمک از کتابخانه plotly

وب اپ (فریم ورک Django):

برای سمت وب من از فریم ورک django استفاده کردم که یک سمت وب برای زبان پایتون است و از قبلاً من از این استفاده کرده بودم و با آن آشنا بودند به خاطر همین از این فریم ورک استفاده کردند

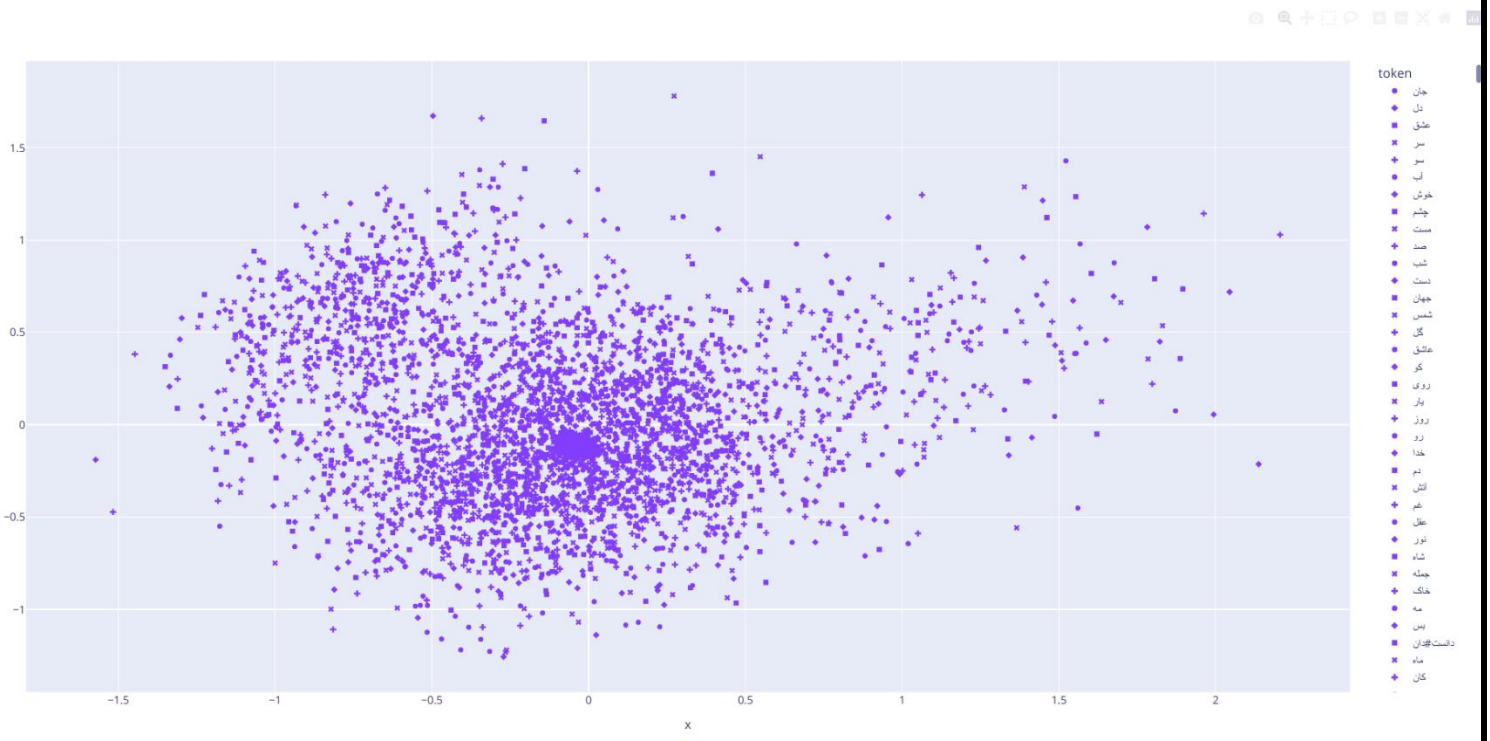
کارایی آن به این صورت است که قسمت backend می آید و آبجکت numpy که ذخیره کرده بودم محتویات آن شامل ابعاد train شده کلمات می باشد و همچنین لیست کلمات یکتای متن train شده را میگیرم و بر اساس شباهت کسینوسی به کاربر در قالب Html و css کلمات مشابه ای که کاربر می‌خواهد را به آن می دهم به مانند شکل زیر:



نشان دادن مختصات کلمات به صورت سه بعدی و دو بعدی به صورت تعاملی و زنده با امکان بزرگنمایی و عکس گرفتن و امکانات دیگر:

برای نشان دادن گرافیکی مختصات کلمات به صورت دو بعدی و سه بعدی من یک اسکریپت پایتون نوشتم نیاز داشتم که ابعاد کلمات را کم کنم من از متد PCA کتابخانه sklearn که همان کاهش ابعاد ماتریکس میباشد استفاده کردم و به طور مثال کلمات من ابعادش ۲۰۰ باشد آن را به دو بعدی تبدیل می کند و یا سه بعدی برای اینکه بتوانم با استفاده از کتابخانه plotly آن را به صورت تعاملی و زنده نمودار آن را نشان بدهم

برای نمودار پراکندگی کلمات به صورت دوبعدی تعاملی این خروجی را می دهد:



برای نمودار پراکندگی کلمات به صورت سه بعدی تعاملی این خروجی را می دهد: