به نام خدا

موضوع: گزارشکار تمرین اول درس پردازش زبان و گفتار

نام استاد: استاد قاسمی

نام و نام خانوادگی: حسین شعله رسا

شماره دانشجویی: 220798066

خلاصه:

این تمرین با روش skip-gram و negative-sampling پیاده سازی شده و مقدار وکتور هر کلمه پس از ترین شدن شبکه عصبی قابل محاسبه می باشد. از تنسرفلو برای ساخت شبکه عصبی و preprocessing استفاده شده و سپس با استفاده از فریمورک جنگو قسمت وب اپ ساده پیاده سازی شده است.

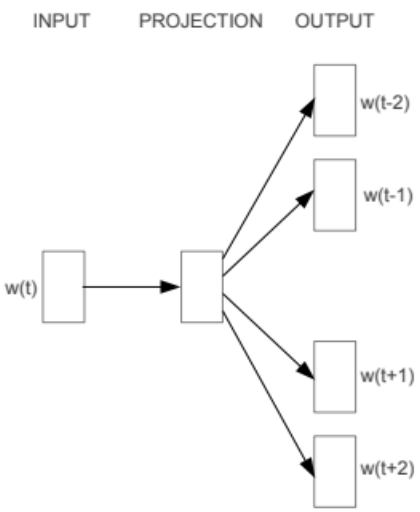
لینک گوگل کولب:

https://colab.research.google.com/drive/1fNQee1RZhm2JDHOqszbaSWn6qnFXuGaQ?usp=sharing

وب اپ در فایل ضمیمه موجود است. برای اجرای آن نیاز به نصب پایتون و فریمورک جنگو میباشد. باقی پکیج ها در فایل requirements.txt است.

Skip-gram:

در این روش کلمات مرکز را به مدل داده و سپس برای ما کلمات اطراف را پیش بینی میکند. همانطور که در تصویر مشاهده میکنید کلمات مرکزی را بعنوان ورودی به شبکه میدهیم و همچنین hidden layer و سپس خروجی ما پیش بینی کلمات نزدیک به کلمه مرکز می باشد.



Skip-gram

در این روش از negative sampling استفاده شده است. به این صورت که کلمات اطراف کلمه مرکز با window مورد نظر انتخاب شده و لیبل 1 به آن ها داده شده است. حال به صورت رندوم کلماتی که در window ما در کلمه اطراف کلمه مرکز امان نیستند را لیبل 0 داده شده است.

دیتاست ما به این صورت باید باشه **(target, context), relevancy**

Preprocessing:

ابتدا دیتا امان را آماده میکنیم. در قسمت preprocessing هر بیت یک جمله در نظر گرفته شده است. سپس با کتابخانه hazm هر جمله به صورت کلمه کلمه در آمده و نورمالایز طبق زبان فارسی شده است. با استفاده از

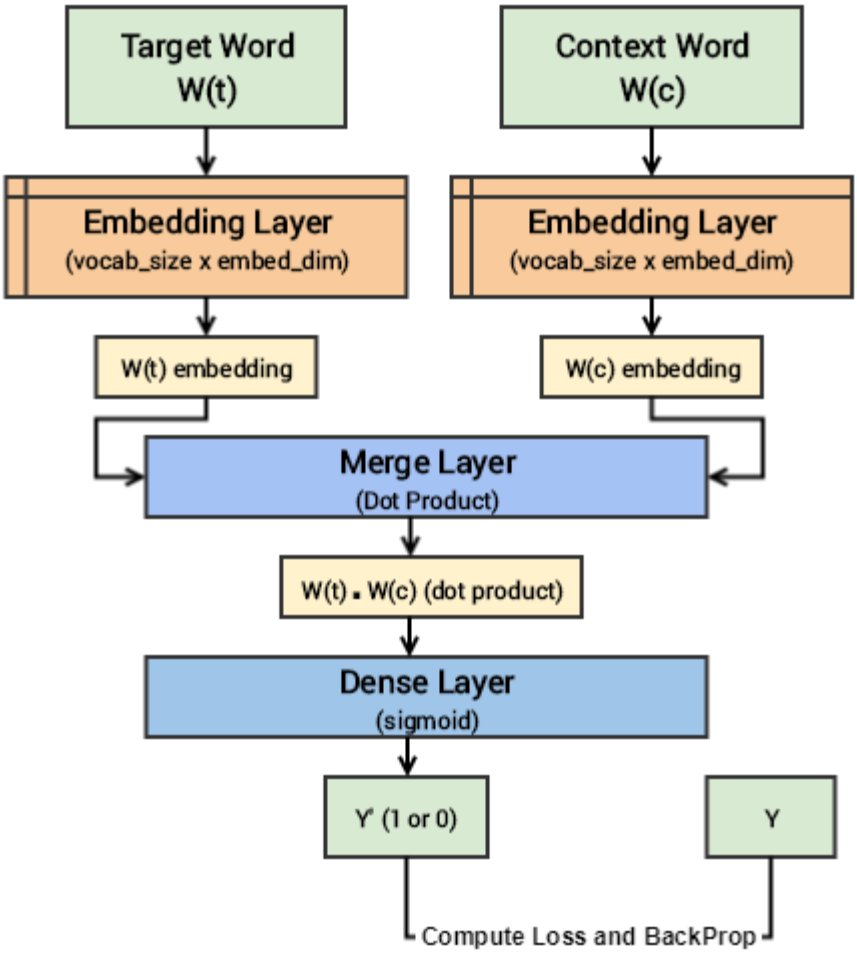
stopwords های فارسی که در اینترنت موجود بود و اضافه و کم کردن این کلمات stopwords مورد نظر در فایل آن تولید شد. با استفاده از این فایل کلمات اضافی را حذف کنیم. در انتها با استفاده از preprocessing کراس کلمات توکنایز شده اند.

با استفاده از preprocessing کراس از تابع skipgram استفاده شده است تا با دادن corpes برایمان positive samples و negative samples را بسازد. با توجه به سایت تنسرفلو تعداد negative samples به ازای هر جمله چون دیتا کمی داریم بهتر است عددی بین 5 تا 20 باشد. که عدد 15 انتخاب شده است. Window_size هم عدد 2 انتخاب شده است. در آخر دیتا به صورت زیر در آمده است.

```
((2118) تاخیر ,(3278) مهلتی) -> 1
((31) شیر ,(39) خون) -> 1
((1565) مثنوی ,(3278) مهلتی) -> 1
((39) خون ,(3278) مهلتی) -> 1
((1199) بایست ,(31) شیر) -> 1
((3406) روییده ,(39) خون) -> 0
((1565) مثنوی ,(2118) تاخیر) -> 1
((4643) ساو ,(1564) مدتی) -> 0
((31) شیر ,(1199) بایست) -> 1
((3278) مهلتی ,(1199) بایست) -> 1
```

با توجه به پیاده سازی هایی که برای قسمت شبکه عصبی آن شده بود و آزمون خطاها، شبکه پیاده سازی را طبق سایت تنسرفلو که برای word2vec skipgram درست کرده بود پیاده سازی شده است. روش قبلی که استفاده شده بود به علت loss بالا جایگزین شد.

مدل قبلی به صورت زیر بود:



توی این مدل دو تا embedding layer که یکی برای کلمات مرکز و دیگری برای کلمات اطراف انتخاب شده و چون این لایه ها به صورت 2D هستند باید اول reshape بشه به یک بعد. سپس بعد از کانکت کردن این دو لایه به لایه sigmoid میدهیم که تشخیص دهد کلمه ما مشابه هست یا خیر. این مدل بعد از 50 ایپاک به لاس 4 میرسید.

در مدل جدید با استفاده از Model subclassing دو لایه embedding مانند شکل بالا درست شده است. سپس در قسمت call این مدل دیتا مرکزمان را به embedding اول(به اسم w2v_embedding) و دیتا اطراف به embedding دوم می دهیم.

لایه embedding کراس هم میتواند به تنهایی ترین شود و هم میتواند مانند مدل بالا همراه با dense layer های دیگر استفاده شود.

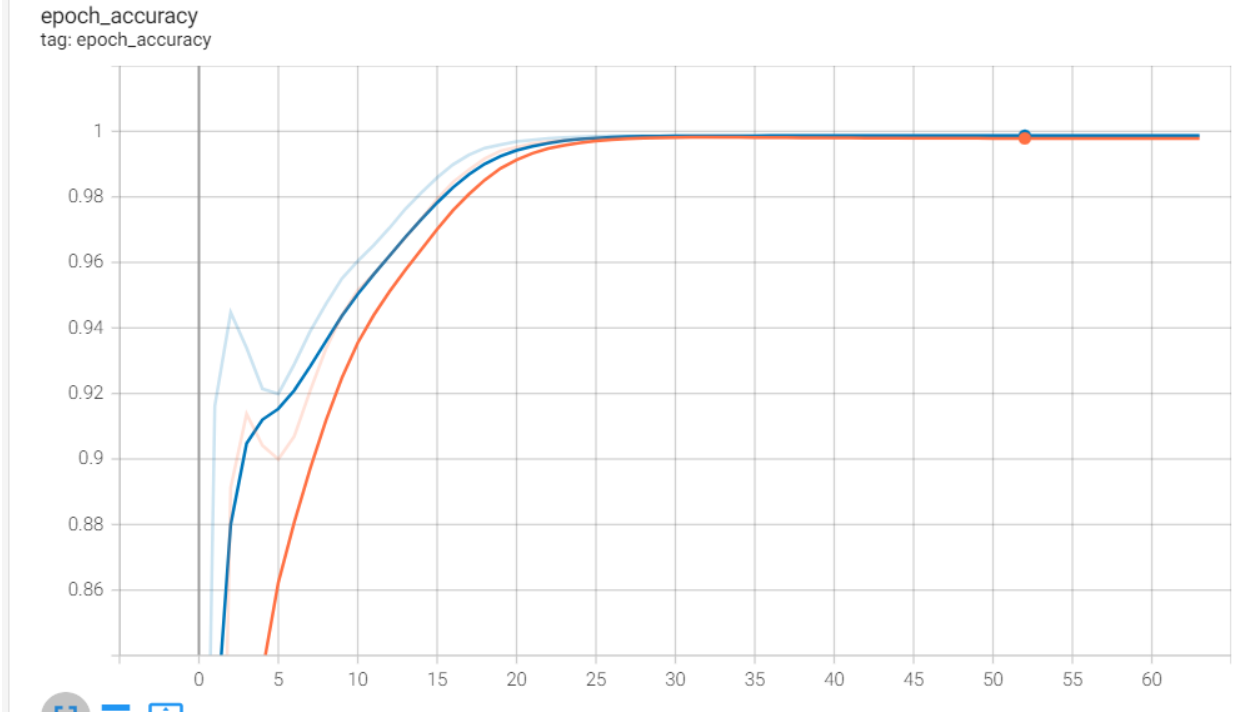
در ادامه با استفاده از Dataset در تنسرفلو دیتا را آماده کرده و از کش کردن برای سرعت بیشتر و کنترل مموری استفاده شده.

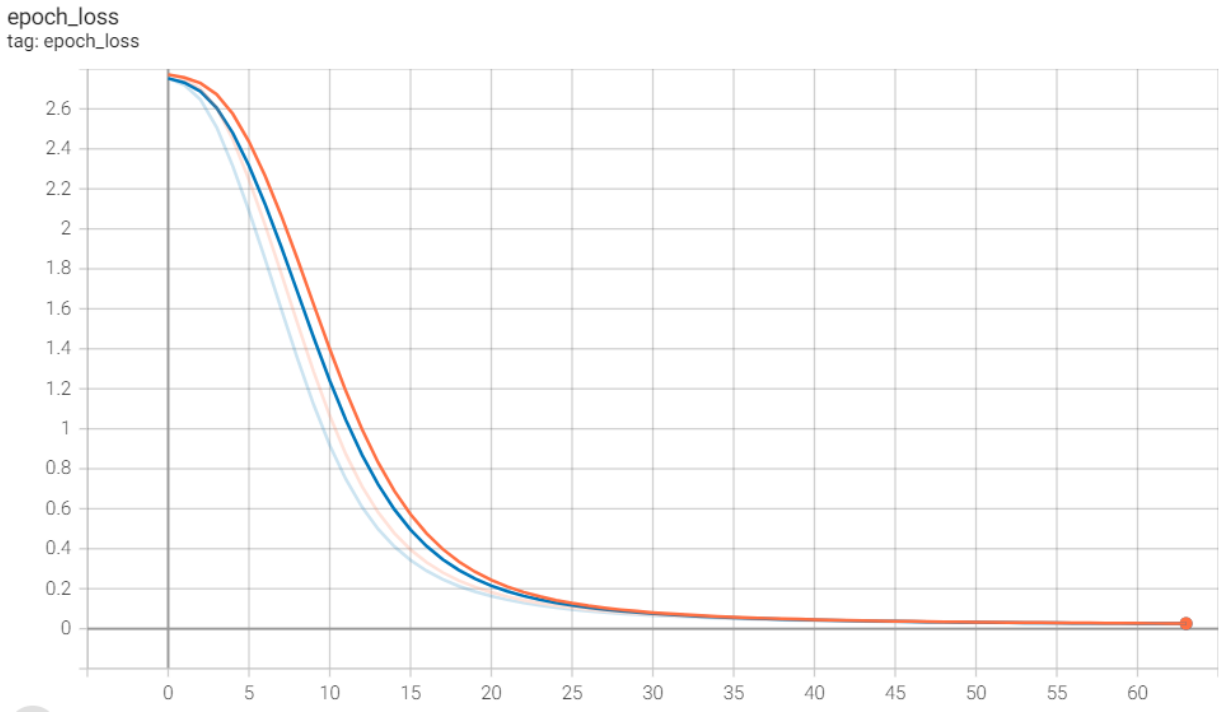
برای لاس از CategoricalCrossentropy استفاده شده که در این مسیله بهتر از sigmoid_cross_entropy_with_logits عمل کرده است. Embed size های مختلفی تست شده است و عدد 100 انتخاب شده است. Embed size های کوچکتر دقت کمتری بعد از ترین داشت. از دو کالبک یکی برای کنترل learning rate اپتیمایزر Adam و دیگری برای لاگ گرفتن شبکه استفاده شده تا بتوان با tensorboard اطلاعاتی از نحوه علمکرد شبکه داشت.

دیتا به صورت 80 درصد ترین درامده و 20 درصد ولیدیشن برای نحوه عملکرد شبکه و بعد از 64 ایپاک شبکه عملکرد زیر را داشته است. Loss=0.024, acc=0.998

```
82/82 - 1s - loss: 0.0265 - accuracy: 0.9981 - val_loss: 0.0258 - val_accuracy: 0.9986 - lr: 0.0010 - 1s/epoch - 16ms/step
Epoch 59/64
82/82 - 1s - loss: 0.0261 - accuracy: 0.9981 - val_loss: 0.0255 - val_accuracy: 0.9986 - lr: 0.0010 - 1s/epoch - 16ms/step
Epoch 60/64
82/82 - 1s - loss: 0.0257 - accuracy: 0.9981 - val_loss: 0.0251 - val_accuracy: 0.9986 - lr: 0.0010 - 1s/epoch - 16ms/step
Epoch 61/64
82/82 - 1s - loss: 0.0254 - accuracy: 0.9981 - val_loss: 0.0247 - val_accuracy: 0.9986 - lr: 0.0010 - 1s/epoch - 16ms/step
Epoch 62/64
82/82 - 1s - loss: 0.0250 - accuracy: 0.9981 - val_loss: 0.0244 - val_accuracy: 0.9986 - lr: 0.0010 - 1s/epoch - 15ms/step
Epoch 63/64
82/82 - 2s - loss: 0.0247 - accuracy: 0.9981 - val_loss: 0.0241 - val_accuracy: 0.9986 - lr: 0.0010 - 2s/epoch - 23ms/step
Epoch 64/64
82/82 - 1s - loss: 0.0244 - accuracy: 0.9981 - val_loss: 0.0238 - val_accuracy: 0.9986 - lr: 0.0010 - 1s/epoch - 15ms/step
```

epoch_accuracy



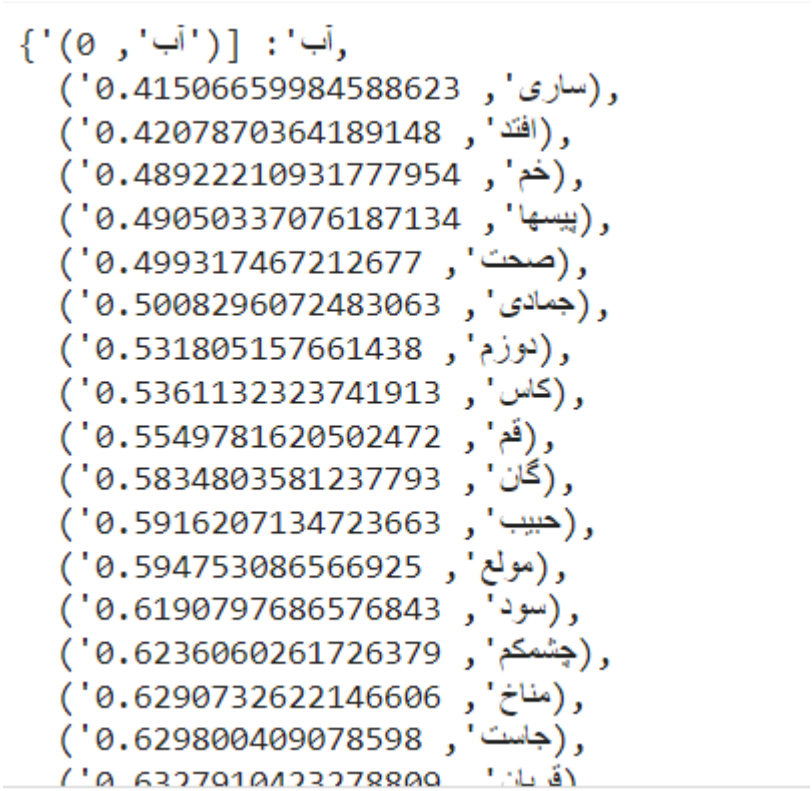


سپس از لایه embedding وزن کلمات استخراج میکنیم و یک نمونه نمایش داده میشود.

```
words_vector('جان')
```

```
array([ 0.00510174, -0.04569227, -0.0358027 ,  0.04402636, -0.00212815,
       -0.01045753, -0.01103012, -0.0317234 ,  0.00127383,  0.02015941,
        0.04320139,  0.03998394,  0.0138208 ,  0.04793457, -0.03604396,
       -0.02868651,  0.02054424,  0.03434862, -0.01135812,  0.02745606,
        0.03492323,  0.03764881,  0.00563232, -0.04322922,  0.03492751,
        0.04287704,  0.02478992, -0.04784408, -0.04725004,  0.01214479,
       -0.04733144,  0.0341861 , -0.03061377, -0.03453442, -0.0317362 ,
        0.04029289,  0.0383371 ,  0.03701638,  0.02715004, -0.00542282,
       -0.00909925, -0.00446457, -0.02823055, -0.0067502 , -0.04722176,
        0.02397462,  0.01368773,  0.02769938,  0.03425359,  0.04772789,
        0.00987557, -0.01065553,  0.02257908, -0.03776776, -0.02041615,
        0.02071064, -0.01625574, -0.03851831,  0.02720938, -0.02732415,
       -0.04569812,  0.02334676,  0.02842505,  0.00550824, -0.00999757,
        0.01930877,  0.01273752,  0.01173627,  0.01340861, -0.03150704,
        0.04315033, -0.0085009 , -0.01431683,  0.03844912,  0.04348614,
        0.04812291,  0.03220476, -0.02077179,  0.04291448, -0.00727718,
        0.04151708,  0.03730312,  0.01526808, -0.01866696,  0.02087865,
        0.00374015, -0.02724274,  0.03026188,  0.00102326,  0.01298115,
        0.01973959,  0.00776633, -0.03596501, -0.00147633, -0.01718669,
        0.0037439 ,  0.00245715, -0.00307816,  0.03377279, -0.00709101],
      dtype=float32)
```

10 کلمه که 20 همسایه نزدیک آن نمایش داده شود:



تعداد کلمات کل دیتاست پس از حذف استاپ ورد ها حدود 42 هزار کلمه است که تقریبا این مقدار کمی برای انجام این تسک می باشد.

Web APP:

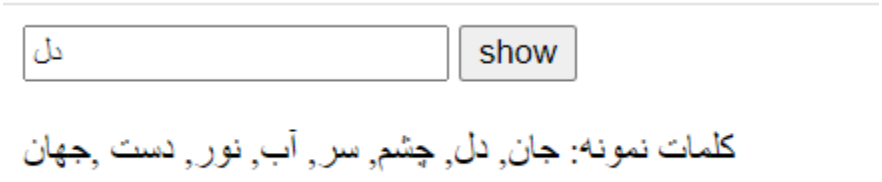
برای اجرای آن بعد از نصب نیازمندی ها در فایل requirements.txt باید با دستور

Python manage.py runserver سرور را اجرا کرد. روت اول برای نمایش وکتور هر کلمه میباشد. در آدرس

http://127.0.0.1:8000/words/vector/

در قسمت بعدی برای نمایش 15 کلمه مشابه از روت زیر استفاده میکنیم.

http://127.0.0.1:8000/words/similarity/



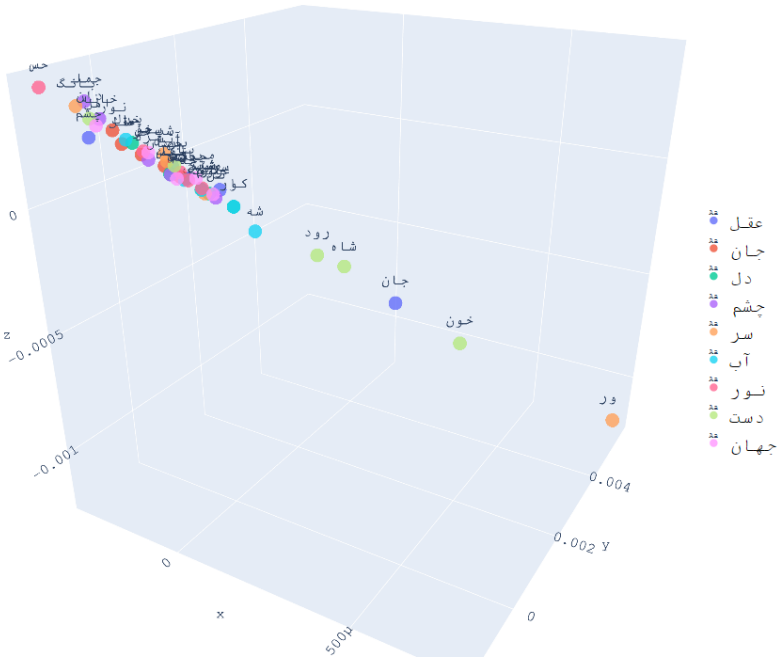
کلمات مشابه برای کلمه دل

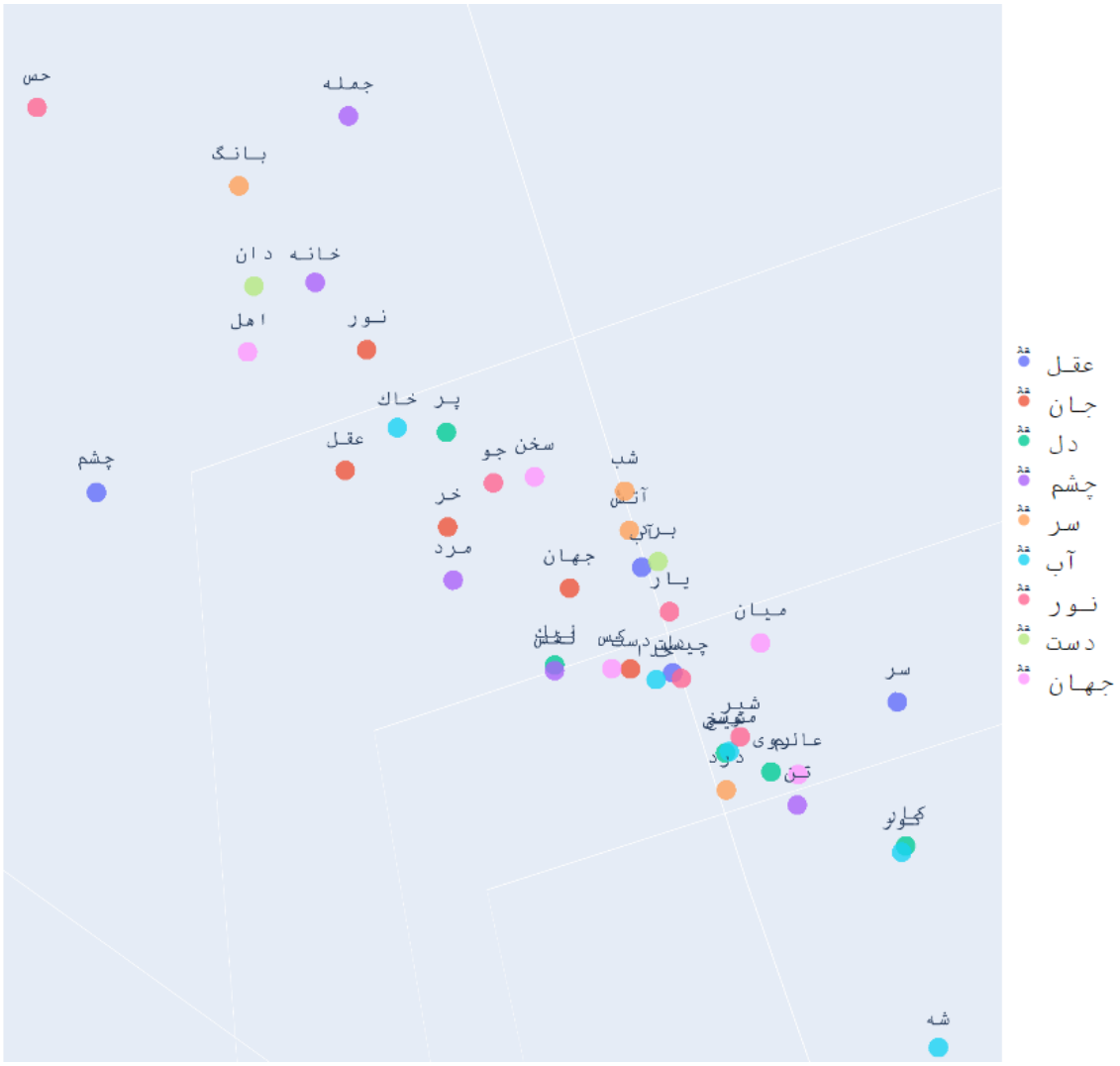
#	کلمه
0	دل
1	باختم
2	عدوی
3	عرش
4	پیل
5	کژبینش
6	خنبك
7	نطعش
8	نشست
9	تراشد
10	ناپیدا
11	حاتمی
12	واکشیدن
13	تدبیرها
14	دریابدش

برای نمایش کلمات در 3 بعد نیازمند آن است که از روش های کاهش بعد استفاده کنیم. یکی از مرسوم ترین روش ها T-SNE است که در بعد زیاد و بعد کم شباهت می گیرد. سپس با استفاده از اپتیمایزر ها اختلاف شباهت را به کمترین مقدار کاهش دهد. مثلا با روش gradient decent

برای محاسبات ساده تر از pca استفاده می شود به این صورت است که به صورت خطی بعد ها را کاهش میدهد. برای نقشه لغات از T-SNE استفاده شده است و برای نمایش همه کلمات از PCA

کشیدن نقشه ده کلمه با پیدا کردن 20 شباهت در نقشه

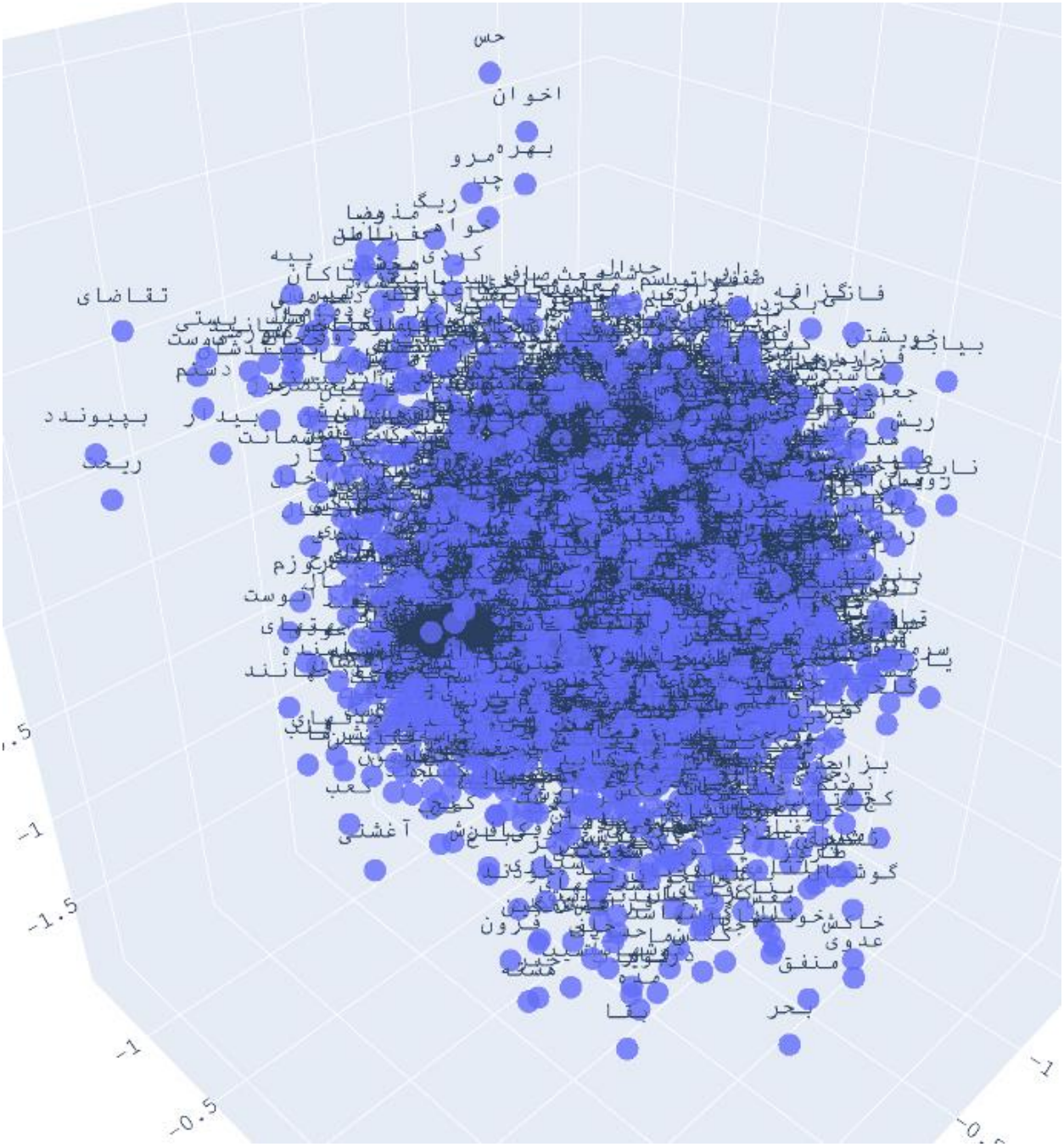




به دلیل نیاز اپتیمایز کردن از iterate 300 و learning rate برابر 10 برای نمایش استفاده شده است. این عمل مدتی طول میکشد تا ساخته شود سپس بدون کندی قابل بزرگنمایی است. نمایش این نقشه به صورت دو بعدی نیز ممکن است.

منبع این پیاده سازی: https://towardsdatascience.com/visualizing-word-embedding-with-pca-and-t-sne-961a692509f5

تمام لغات:



فیلم از این بخش:

https://drive.google.com/file/d/16vvOOhYsf6esPvbsQEHDTYHxY-8qnAJx/view?usp=sharing

برای خروجی گرفتن از نقشه روی ایکون دوربین ضربه بزنید و خروجی گرفته میشود

نمایش کل لغات با روش pca

http://127.0.0.1:8000/words/3d/pca/

روت برای نمایش چند کلمه برتر با روش tsne به دلیل اپتیمایز کردن کمی زمان میکشد تا لود شود

http://127.0.0.1:8000/words/3d/tsne/

نمایش مپ برای 15 کلمه مشابه یک لغت

http://127.0.0.1:8000/words/similarity/3d/