



به نام خدا

عنوان: پیاده‌سازی word2vec و visualization

درس: NLP

استاد درس: مهندس قاسمی

مهدی دهقانی

۲۲۰۷۹۷۰۴۸

mehdi.dehghani@ut.ac.ir

mahdiazadi18@yahoo.com

داده‌های ورودی

داده‌های ورودی درون پوشه Data قرار می‌گیرند. هر چقدر داده‌های ورودی بزرگتر باشند، خروجی بهتری حاصل خواهد شد.

برای این قسمت، دو داده fa.fooladvand.txt و voa_fa_۲۰۰۳-۲۰۰۸_orig.txt در نظر گرفته شده است که اولی بخاطر کوچک بودن خروجی بسیار خوبی را فراهم نمی‌کند اما دومی به نسبت اندازه داده‌ها، خروجی بسیار خوبی خواهد داد. اما فعلاً تمام کارهای انجام شده و فایل‌های قرار داده شده از آموزش و تست برای فایل fa.fooladvand.txt است. اما طبق اجرایی که به صورت شخصی داشتم، فایل دوم (voa) خروجی بسیار خوبی را ارائه می‌دهد.

پیش پردازش داده‌ها

برای پیش پردازش داده‌های ورودی، ساده‌ترین راه حذف تمام حروف‌ها به جز حروف الفبای زبان مربوطه است. که می‌توان با استفاده از قسمت کامنت شده برای just words و استفاده نکردن از بقیه قسمت‌ها تنها کلمات فارسی را نگه داشت.

اما می‌توان از این جلوتر رفت و به جای اینکه تنها کلمات را نگه داشت از راه بهتری استفاده کرد و بعضی از علائم نگارشی و مواردی نظیر آن که به پردازش کمک می‌کند را نگه داشت و از کتابخانه‌هایی که در این زمینه وجود دارد استفاده کرد تا داده‌های ورودی برای اعمال word2vec شکل بهتری داشته باشند. در این قسمت از کتابخانه hazm یا همان هضم استفاده شده است که بنظر برای زبان فارسی نسبتاً مناسب است. می‌توان با استفاده از هضم و متدهایی مثل Normalizer یا InformalNor-malizer متن ورودی را نسبت به زبان فارسی نرمال‌سازی کرد (به طور مثال فعلی مانند داده خواهد شد به داده_خواهد شد تبدیل می‌شود که این یعنی به جای ۳ توکن به ۱ توکن تبدیل می‌شود که بسیار به ما کمک خواهد کرد)

در ادامه می‌توان stop words زبان مربوطه را از داده‌های نرمال‌سازی شده حذف کرد که این قسمت کامنت شده است ولی قابل استفاده است و حتی می‌توان از stop words های ارائه شده دیگر استفاده کرد ولی در حال حاضر stop words ای که خود هضم ارائه داده است در این قسمت قرار داده شده است.

در ادامه می‌توان از stem یا lemmatize استفاده کرد تا کلماتی مثل کتاب‌ها، کتاب‌های و کتاب به عنوان سه کلمه جدا در نظر گرفته نشوند و همه به عنوان یک کلمه کتاب در نظر گرفته شوند. در این قسمت نیز از هضم استفاده شده است اما به دلیل اینکه در بعضی از موارد عملکرد خوبی از این دو متد گرفته نشد (به طور مثال کلمه داشت به دا تبدیل می‌شد یا رسمی به رسم) تصمیم بر این شد که روی داده کوچک‌تر استفاده نشود اما این دو متد در نظر گرفته شده و فقط در قسمت اجرایی کامنت شده است و می‌توان از آن‌ها نیز استفاده نمود.

در نهایت داده‌هایی که پیش پردازش شده‌اند و به صورت لیستی از توکن‌ها در اختیار ما هستند را در فایل جدیدی با پسوند preprocessed_ می‌نویسیم. به این صورت که بین هر توکن یک فاصله قرار می‌دهیم تا برای بعد به سادگی قابل خواندن از فایل باشند. همچنین خطوطی که خالی از کلمات فارسی هستند نیز قبل از این حذف شده و در نظر گرفته نشده‌اند تا مشکلی به وجود نیاید.

برای پیاده‌سازی word2vec از tensorflow و pytorch استفاده کردیم که در pytorch تنها از روش skip gram و negative sampling استفاده شده است اما در tensorflow هر ۴ روش مربوط به word2vec شامل موارد زیر پیاده‌سازی شده است:

- skip gram with negative sampling
- skip gram with hierarchical softmax
- cbow with negative sampling
- cbow with hierarchical softmax

ساخت dataset pipeline

در ادامه یک مثال عینی از تبدیل یک جمله خام به ماتریس‌هایی که داده‌ها را برای آموزش مدل word2vec با معماری skip_gram یا cbow می‌دارند را می‌بینیم.

فرض کنید یک جمله در corpus داریم: the quick brown fox jumps over the lazy dog. با اندازه پنجره (حداکثر تعداد کلمات در سمت چپ یا راست کلمه مورد نظر) در زیر کلمات. فرض کنید جمله قبلاً subsample شده است و کلمات به شاخص‌ها نگاشت شده‌اند.

هر یک از کلمات در جمله را target word می‌نامیم و کلمات درون پنجره که در مرکز آن‌ها target word قرار دارد را context words می‌نامیم. به عنوان مثال، 'quick' و 'brown'، context words با target word 'the' هستند و 'the'، 'brown' و 'fox'، context words با target word 'quick' هستند.

	the	quick	brown	fox	jumps	over	the	lazy	dog
window size	2	2	1	1	2	2	1	2	1

برای skip_gram، وظیفه پیش‌بینی context words با توجه به target word است. شاخص هر target word به سادگی تکرار می‌شود تا با تعداد context words آن مطابقت داشته باشد. این ماتریس ورودی ما خواهد بود.



Skip gram with negative sampling

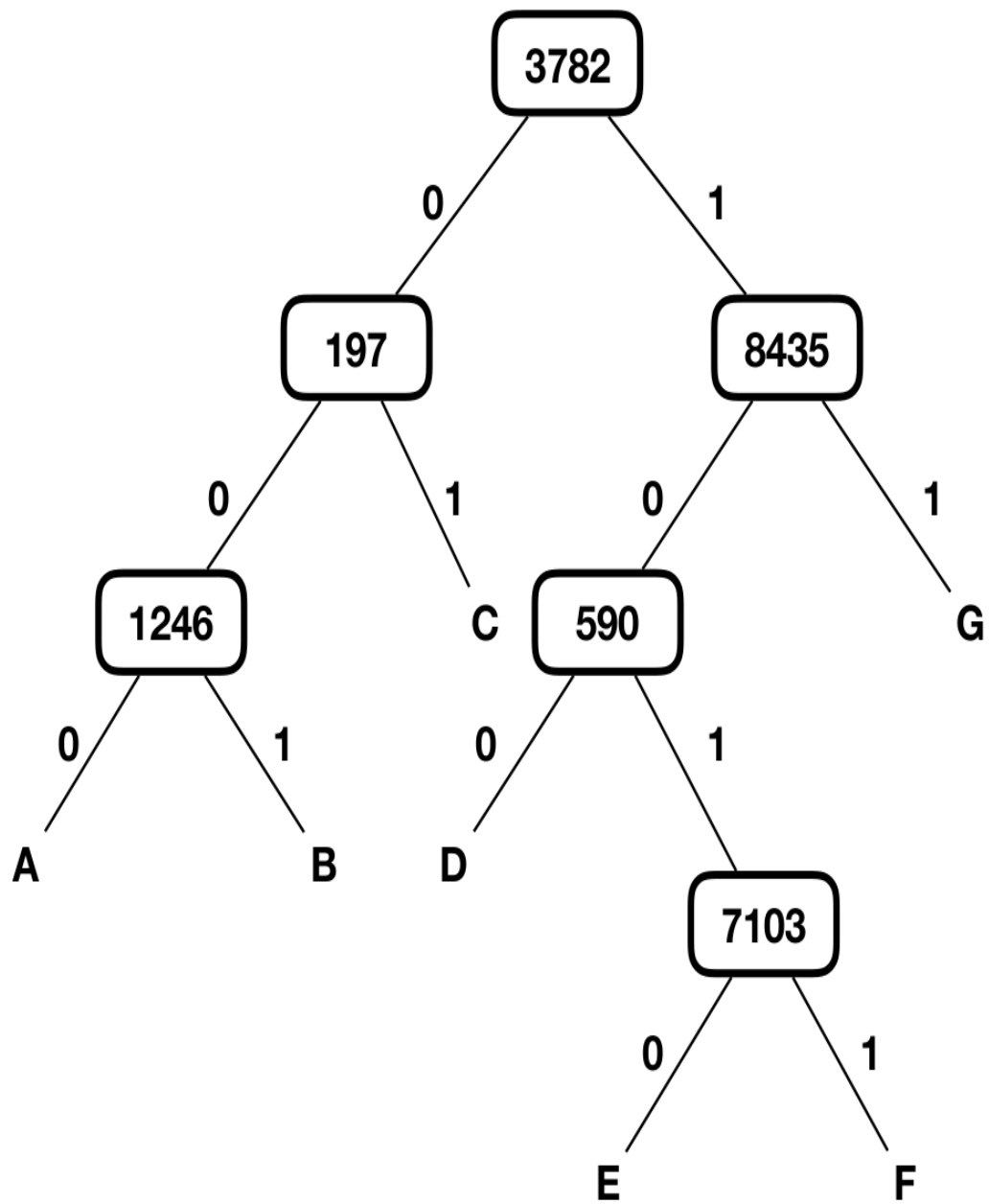
برای cbow، وظیفه پیش‌بینی target word با توجه به context words است. از آنجایی که هر target word ممکن است دارای تعداد متغیری از context words باشد، لیست context words را به حداکثر اندازه ممکن (اندازه پنجره * ۲) می‌رسانیم (pad می‌کنیم) و اندازه واقعی target word را اضافه می‌کنیم.

$2 * \text{window_size} + 1$					1
quick	brown	-	-	2	the
the	brown	fox	-	3	quick
quick	fox	-	-	2	brown
brown	jumps	-	-	2	fox
brown	fox	over	the	4	jumps
fox	jumps	the	lazy	4	over
over	lazy	-	-	2	the
over	the	dog	-	3	lazy
lazy	-	-	-	1	dog
input matrix					label matrix

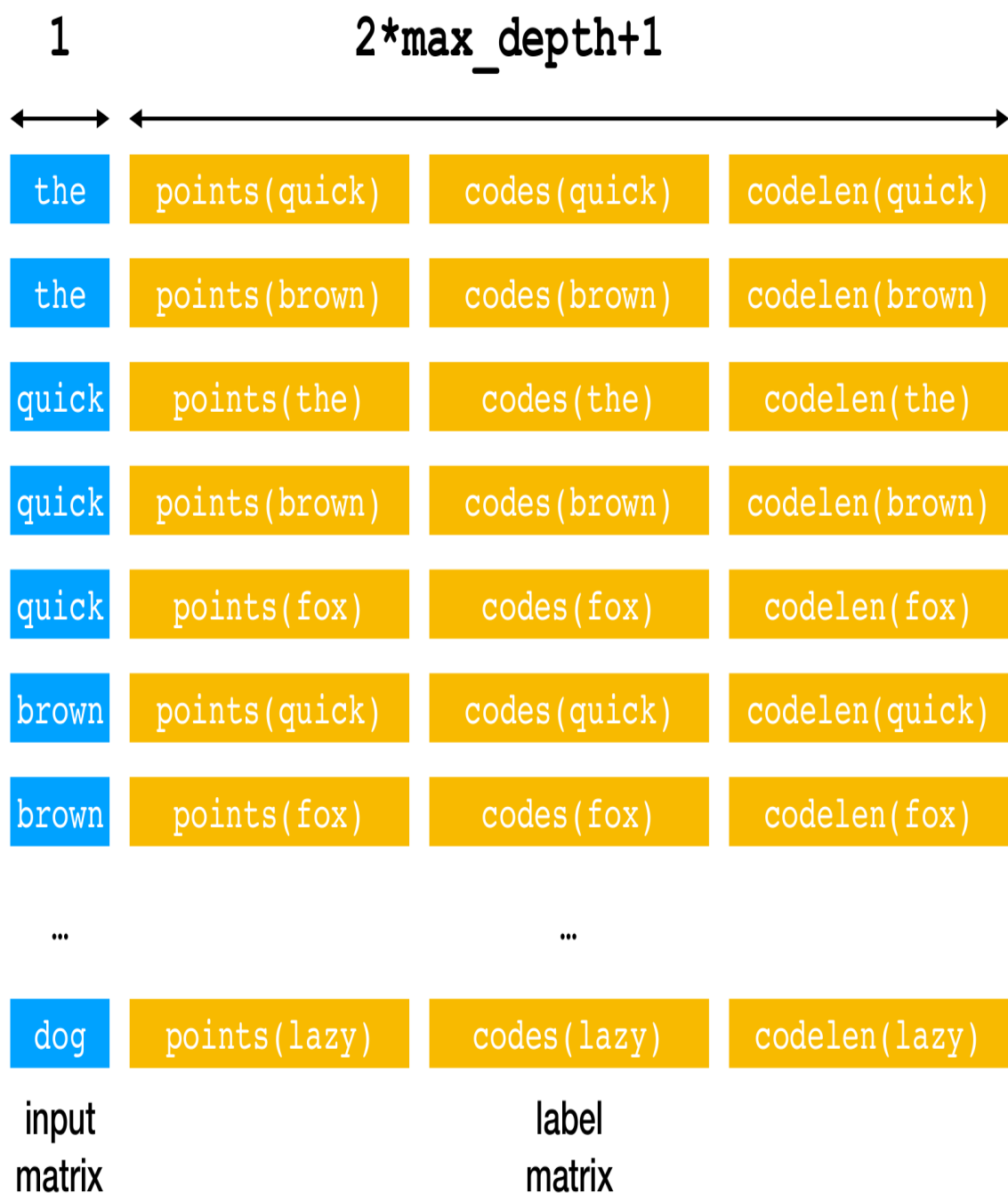
Continuous bag of words with negative sampling

اگر الگوریتم آموزشی `negative_sampling` باشد، ما به سادگی ماتریس برچسب را با شاخص‌های کلماتی که باید پیش‌بینی شوند پر می‌کنیم: `skip_gram` برای context words یا `cbow` برای target words.

اگر الگوریتم آموزشی `hierarchical_softmax` باشد، یک درخت هافمن برای مجموعه کلمات واژگان ساخته می‌شود. هر کلمه واژگان دقیقاً با یک گره برگ مرتبط است و کلماتی که در `negative_sampling` باید پیش‌بینی شوند، با دنباله‌ای از کدها و point ها جایگزین می‌شوند که توسط گره‌های داخلی در مسیر ریشه به برگ تعیین می‌شوند. به عنوان مثال، کدها و point های E عبارتند از ۳۷۸۲، ۸۴۳۵، ۵۹۰، ۷۱۰۳ و ۱، ۰، ۱، ۰. ماتریس برچسب را با کدها و point های پر (pad) شده (تا حداکثر عمق)، همراه با طول واقعی کدها یا point ها پر می‌کنیم.



Huffman Tree



Skip gram with hierarchical softmax

$2 * \text{window_size} + 1$					$2 * \text{max_depth} + 1$		
quick	brown	-	-	2	points(the)	codes(the)	codelen(the)
the	brown	fox	-	3	points(quick)	codes(quick)	codelen(quick)
quick	fox	-	-	2	points(brown)	codes(brown)	codelen(brown)
brown	jumps	-	-	2	points(fox)	codes(fox)	codelen(fox)
brown	fox	over	the	4	points(jumps)	codes(jumps)	codelen(jumps)
fox	jumps	the	lazy	4	points(over)	codes(over)	codelen(over)
over	lazy	-	-	2	points(the)	codes(the)	codelen(the)
over	the	dog	-	3	points(lazy)	codes(lazy)	codelen(lazy)
lazy	-	-	-	1	points(dog)	codes(dog)	codelen(dog)
input matrix					label matrix		

Continuous bag of words with hierarchical softmax

به طور خلاصه، یک ماتریس ورودی و یک ماتریس برچسب از یک جمله ورودی خام ایجاد می‌شود که اطلاعات ورودی و برچسب را برای کار پیش‌بینی فراهم می‌کند.

تست و خروجی ها

برای بعضی از توابع و متدها قسمتی به عنوان test نوشته شده است که بر روی مثال خاصی، آن توابع یا متدها را تست می کند و همچنین بعد از مقداردهی پارامترهای مربوطه برای هرکدام از tensorflow یا pytorch و اجرای آن ها، به ازای هر چند مرحله، از مقدار فعلی لرنینگ ریت و لاس (Average یا Total) در آن مرحله خروجی گرفته شده و نمایش داده می شود. همچنین برای اجرای pytorch از اسکریپت word2vec.py و برای اجرای tensorflow از اسکریپت run_training.py استفاده می شود. بدیهی است که به ازای مقادیر مختلف پارامترهای در نظر گرفته شده برای هرکدام، خروجی متفاوت خواهد بود.

ساخت فایل های خروجی

در هر دو tensorflow و pytorch، پس از اجرای اسکریپت های گفته شده در قسمت قبلی که روی corpus داده شده، آموزش انجام می دهند، از اسکریپت create_files.py استفاده می کنیم تا فایل های مورد نیاز برای تست یا شبیه سازی های مورد نظر، نظیر فایل های tsv برای projector و words-vectors و برای ساخت گراف word2vec و در نهایت اجرای webapp، ساخته شوند.

فایل های خروجی درون پوشه های out_word2vec_pytorch و out_word2vec_tensorflow برای tensorflow، قرار می گیرند.

نزدیک ترین همسایه ها

در هر دو tensorflow و pytorch، اسکریپتی با عنوان nearest_neighbors.py وجود دارد که پس از انجام دادن عملیات های قبلی و ساخت فایل های خروجی، می توان این اسکریپت را اجرا کرد تا برای ۱۰ کلمه از واژگان به صورت تصادفی، برای هر کلمه، ۲۰ کلمه ای که از بقیه به آن نزدیک تر هستند نمایش داده شود.

مثال برای کلمه 'عذاب':

```
nearest neighbors to عذاب:
(0.4668243 , 'عذابشان')
(0.4197559 , 'بجشد')
(0.37538737 , 'حقت\۲00cآور')
(0.36143243 , 'دچار')
(0.36032143 , 'شکنجه')
(0.35574484 , 'عذابی')
(0.3442301 , 'مری')
(0.3435179 , 'دردناک')
(0.3403393 , 'رسوایی')
(0.3319929 , 'می\۲00cترسم')
(0.3288855 , 'تهدید')
(0.31565815 , 'مراقب')
(0.31492293 , 'کیفر')
(0.31404504 , 'می\۲00cشوند')
(0.31128582 , 'همانند')
(0.3107616 , 'واجب')
(0.3062084 , 'عذابش')
(0.3027309 , 'اعمال')
(0.30270836 , 'گرفتار')
(0.30194914 , 'منافقان')
-----
```

شبیه‌سازی در projector

برای شبیه‌سازی در projector که tensorflow در نظر گرفته است، می‌توان فایل‌های خروجی tsv که در قسمت‌های قبل گفته شد را در لینک زیر، آپلود کرد.

<https://projector.tensorflow.org/>

شبیه‌سازی برای چند نمونه تست که در فایل‌های پروژه نیز موجود است:

word2vec_tensorflow:

```
arch = 'skip_gram' algm = 'negative_sampling' epochs = 1000 batch_size =  
1024 max_vocab_size = 0 min_count = 3 sample = 1e-3 window_size = 2  
hidden_size = 100 negatives = 2 power = 0.75 alpha = 0.025 min_alpha  
= 0.0001 add_bias = True log_per_steps = 10000
```

```
https://projector.tensorflow.org/?config=https://gist.  
githubusercontent.com/mehdidn/dfb8ed30d60b0862c8a9c1cc6b4a8c6e/  
raw/443c8faf04d924f884bfcf1e29af14800a240cc9/word2vec\_tensorflow\_  
projector
```

word2vec_pytorch:

```
emb_dimension=100 batch_size=1024 window_size=2 negatives=2  
iteration=1000 initial_lr=0.025 min_count=3
```

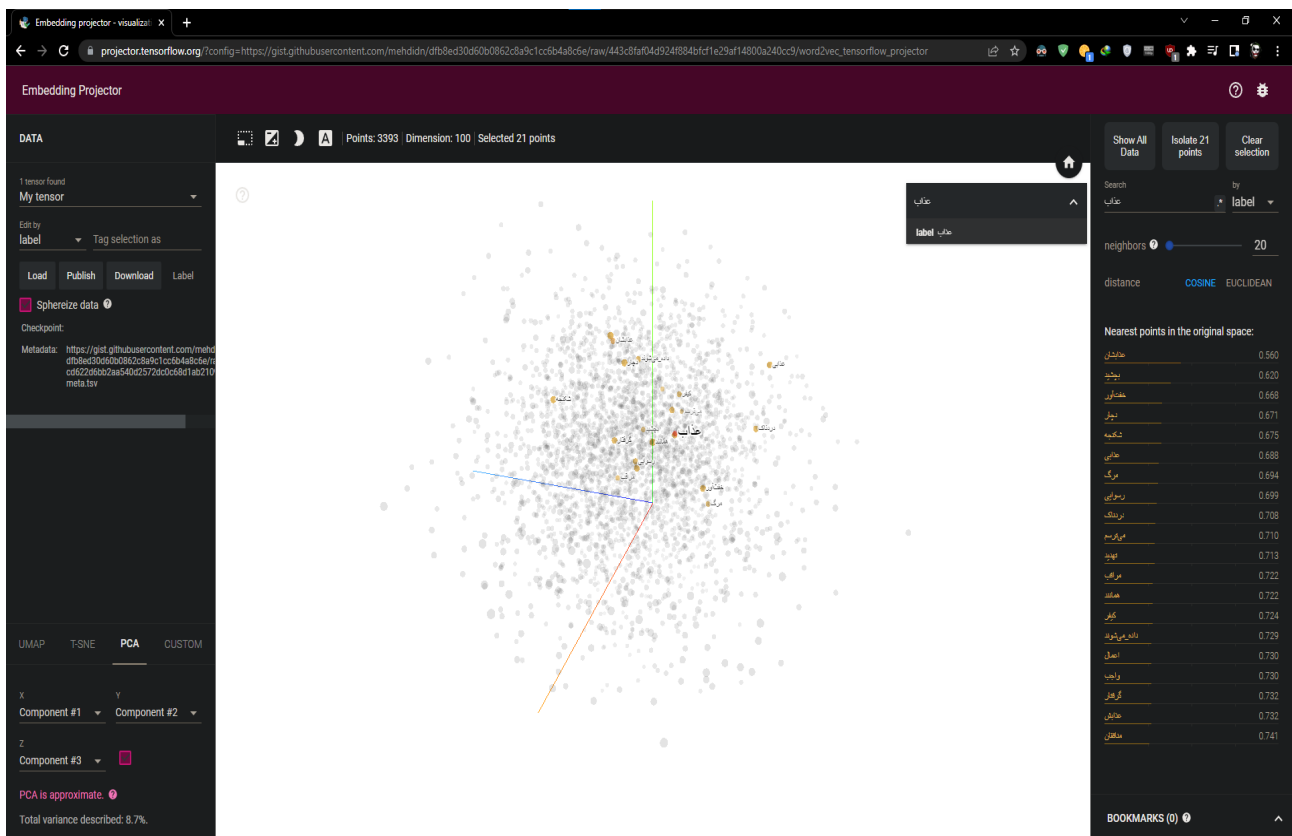
```
https://projector.tensorflow.org/?config=https://gist.  
githubusercontent.com/mehdidn/fbbf5cb443eb50751c6a1bea9306bead/  
raw/805a17f300285aa454f408a853359ccbeff67054/word2vec\_pytorch\_  
projector
```

word2vec_pytorch:

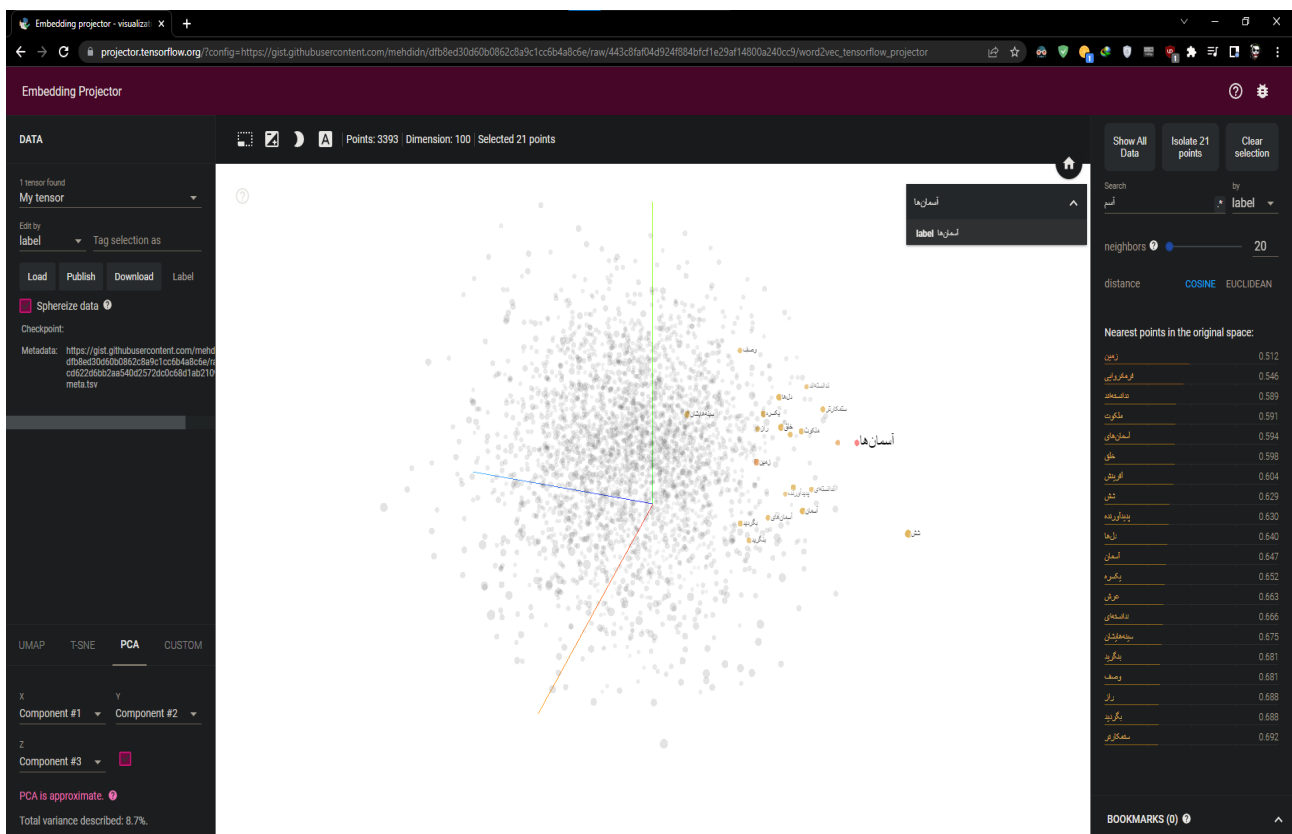
```
emb_dimension=100 batch_size=256 window_size=5 negatives=5  
iteration=100 initial_lr=0.025 min_count=5
```

```
https://projector.tensorflow.org/?config=https://gist.  
githubusercontent.com/mehdidn/8bdf6eb3d228bd521cd8b7e1bd2f8729/  
raw/bd8d491a24abdc0a55a016f2adaeb29473d2b8ea/word2vec\_pytorch\_  
projector\_min\_count\_5
```

مثال برای کلمه 'عذاب' در لینک اول:



مثال برای کلمه 'آسمان‌ها' در لینک اول:



ساخت گراف

فایل words-vectors بدست آمده از قسمت word2vec را در پوشه graph-data موجود در پوشه word2vec_graph قرار می دهیم سپس با تنظیم پارامترهای موجود در اسکریپت words-vectors_to_edges_txt.py نسبت به موارد مورد نظر مشخص می کنیم که برای هر کلمه که به عنوان یک گره در گراف شناخته می شود، نزدیکترین کلمات را تا چه حدی در نظر بگیریم تا بین این کلمات، یال در نظر بگیریم.

به عنوان مثال اگر از threshold صرف نظر کنیم و num_nearest_neighbors را ۱۵ مقداردهی کنیم آنگاه برای هر کلمه موجود در واژگان، بین آن کلمه و ۱۵ نزدیک ترین همسایه آن یال در نظر گرفته می شود، یعنی از کلمه فعلی به ۱۵ نزدیک ترین همسایه آن out-degree متصل می شود یا به عبارت دیگر هر کلمه که به عنوان گره در گراف است، ۱۵ تا out-degree دارد که نشان دهنده ۱۵ نزدیک ترین همسایه آن کلمه است و به هر یک از این ۱۵ کلمه، ۱ عدد in-degree اضافه خواهد شد که همین کلمه یا همان گره فعلی است.

همچنین اگر به عنوان مثال از num_nearest_neighbors صرف نظر کنیم و threshold را ۰.۹ مقداردهی کنیم، برای هر کلمه، تنها به کلمه ای یال خواهیم داشت که شباهت آن به کلمه یا همان گره فعلی بیشتر از ۰.۹ باشد و آنهایی که شباهتشان کمتر از این مقدار باشد یا به عبارتی فاصله آنها بیشتر از مقدار در نظر گرفته شده باشد را نادیده می گیرد.

بنابراین پس از اجرای اسکریپت words-vectors_to_edges_txt.py، در پوشه graph-data فایلی با نام edges.txt به وجود خواهد آمد که نشان دهنده یال های گراف است. به این صورت که یک گره در سمت چپ نوشته شده و سپس نزدیکترین همسایه های آن در سمت راست نوشته شده اند و به این معناست که بین گره سمت چپ و هر گره سمت راست، که هر کدام کلمه هستند، یال وجود دارد. بنابراین گره ها یا همان کلمات سمت راست برای گره یا همان کلمه سمت چپ، out-degree محسوب می شوند و همچنین گره سمت چپ برای هر یک از گره های سمت راست، یک in-degree محسوب می شود.

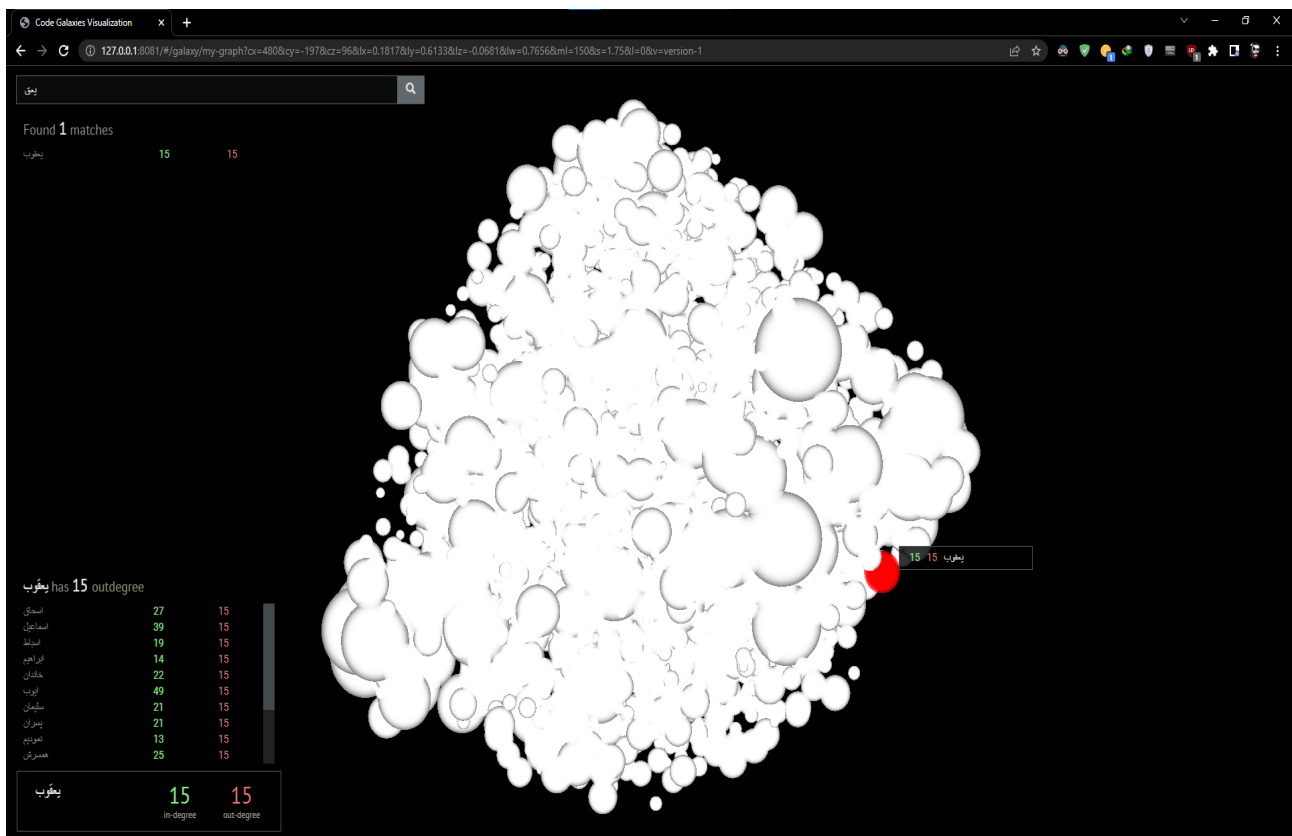
نحوه اجرای وب اپ

دستورالعمل کامل اجرای وب اپ در فایلی با نام how to run web app.txt آمده است. بدیهی است که ابتدا باید پیش نیازهایی مثل node.js، کتابخانه annoy و موارد دیگر نصب شده باشند.

لینک فایل how to run web app.txt:

https://github.com/mehdidn/persian_word2vec/blob/main/how%20to%20run%20web%20app.txt

مثال برای کلمه 'یعقوب' در وب اپ:



مثال برای کلمه 'عذاب' در وب اپ:

