به نام خدا

گزارش تمرین اول

Word2Vec

NLPDL4001

محمد رشیدخان

[m.rashidkhan99@gmail.com](mailto:m.rashidkhan99@gmail.com)

استاد: دکتر قاسمی

پردیس فارابی دانشگاه تهران

بهار 1401

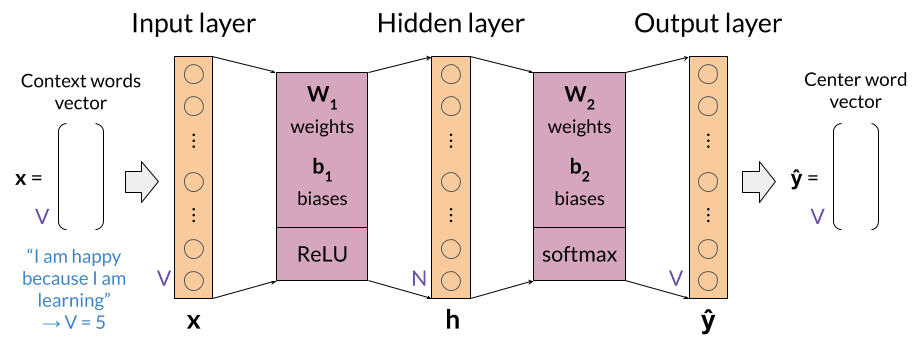
در این تمرین میخواهیم الگوریتم Word2Vec که بردار کلمات را در یک فضای n بعدی به ما میدهد پیاده­سازی کنیم.

طبق مقاله Mikolov et al 2013 ، 2 روش برای این کار پیشنهاد شده است.

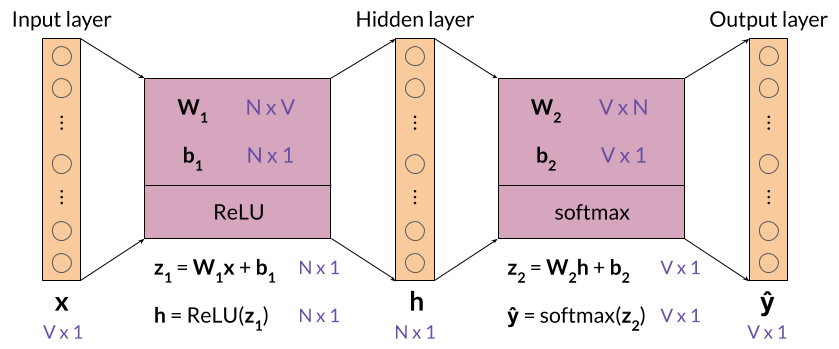
روش اول CBOW (Continuous Bag Of Words) است. روش دوم Skipgram – Negative sampling است که ما با هر 2 روش مدل train کردیم. (البته با کتابخانه­های مختلف مثل Tensorflow – pytorch )

**CBOW (Continuous Bag Of Words)**

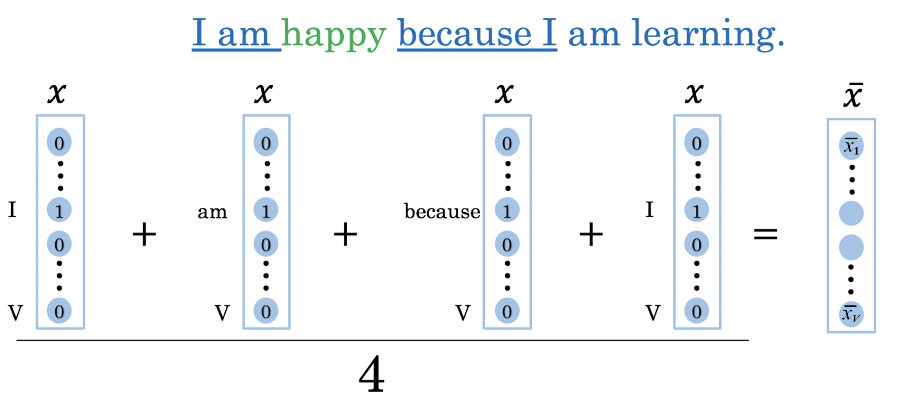
در این روش هدف ما این است که با داشتن تعدادی کلمه در جمله که به انها context words گفته میشود کلمه بینشان را که center word می­گوییم پیش­بینی کنیم. ورودی­ها طبق شکل زیر به شبکه عصبی داده میشوند و خروجی احتمال کلمات است که مدل پیش­بینی کرده است و باید به این نکته توجه داشت که در اینجا خروجی ما word embeddings نیست و در واقع محصول جانبی ما در این شبکه عصبی بردار کلمات میباشد که همان وزن لایه­ها میباشد.

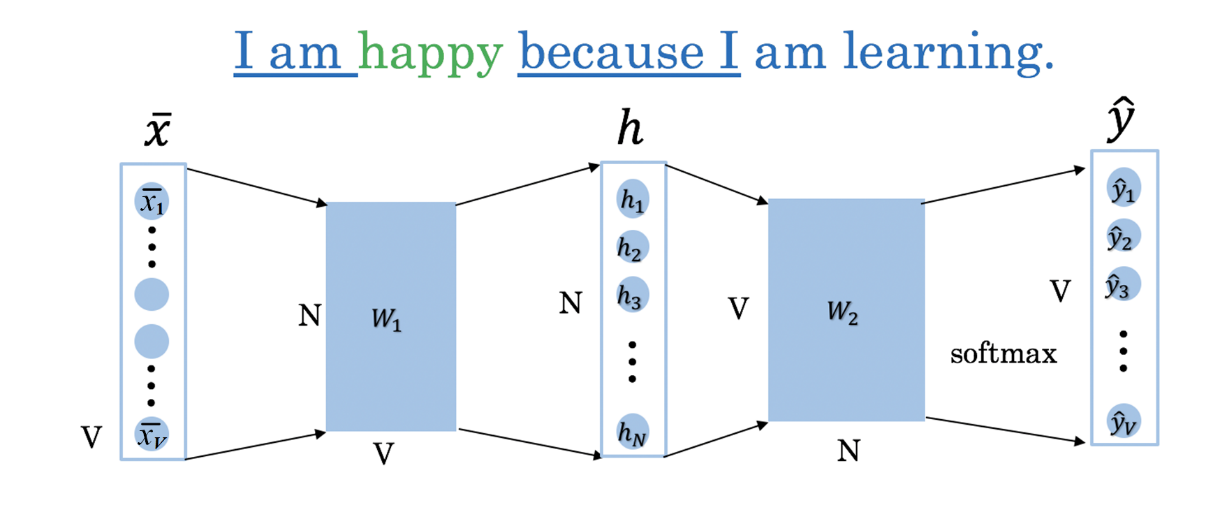


همانطور که در تصویر بالا مشخص است ما سه لایه ورودی، پنهان و خروجی داریم و 2 ماتریس برای وزن لایه­ها داریم که به ترتیب W1 و W2 هستند و همینطور بایاس برای هر لایه و در ورودی برداری از context word ها را داریم که طول بردار اندازه vocab است چون بردارها به صورت one-hot هستند. در لایه پنهان از تابع فعال­سازی Relu استفاده کردیم و در لایه خروجی نیز از تابع softmax برای احتمال­ها استفاده کردیم. خروجی لایه اخر نیز مانند لایه ورودی لایه اول میباشد و هم سایز vocab باید باشد. در شکل زیر ابعاد ورودی و خروجی لایه­ها را به خوبی میتوان مشاهده کرد.



در این معماری ما بردار context words را به صورت میانگین بردارهای تشکیل دهنده آن context در نظر گرفتیم که برای مثال شکل زیر بردار context را برای کلمات {I, am, because, I} نشان میدهد که هدف در اینجا تشخیص کلمه وسط یعنی happy میباشد.





پس ما در این معماری 2 نوع بردار داریم یکی برای center word و یکی هم برای context words که هر 2 به صورت جفتی به عنوان یک نمونه اموزش به مدل داده میشوند.

حال که یک کلیتی از ماجرا را گفتیم به سراغ بررسی معماری اخذ شده میرویم.

به صورت کلی مراحل زیر برای آموزش مدل باید طی شود:

* آماده سازی داده
  + پاکسازی و توکنایز کردن
  + بدست اوردن پنجره­های center word, context word
  + تبدیل کلمات به بردارها
  + ساخت مجموعه آموزش
* ساخت مدل
  + Forward propagation
  + Compute cost
  + Backpropagation
  + Gradient descent
* استخراج word embedding ها
* Visualization and evaluation

**آماده سازی داده**

در مرحله نخست باید دیتایی که به مدل میدهیم را اماده­سازی بکنیم و برای این منظور متن را اول پاکسازی میکنیم و سپس توکنایز میکنیم و بعد پنجره­های کلمات را با توجه به هایپرپارامترها میسازیم. حال این پنجره­ها را تبدیل به بردار میکنیم و بعد به صورت جفت (center word, context words) به عنوان یک نمونه اموزش ذخیره میکنیم. برای ساخت مجموعه آموزش نیز میتوانیم بر حسب مقدار پارامتر batch\_size که مثلا 1024 است batch هایی شامل 1024 نمونه تصادفی از کل جفت­ها انتخاب میکنیم و در هر epoch برای شبکه عصبی یک batch را میفرستیم.

**ساخت مدل**

در مرحله اول از این بخش forward propagation را طراحی میکنیم طبق شبکه مورد نظرمان که shallow هست و در این لایه ماتریس­های موردنظر را بدست میاوریم و خروجی مدل را بعد از اعمال softmax بدست میاوریم. سپس یک تابع هزینه بر اساس cross-entropy پیاده میکنیم که پیش­بینی مدل را بسنجیم و مشتق تابع هزینه را نسبت به پارامترهای شبکه عصبی حساب میکنیم و به وسیله backpropagation مقادیر پارامترها را به روش گرادیان کاهشی آپدیت میکنیم.

**استخراج word embedding ها**

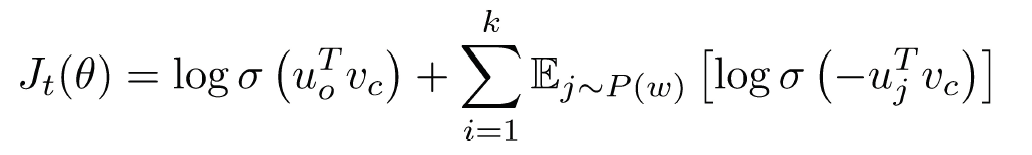
کلا به 3 روش ما میتوانیم بردار کلمات را استخراج کنیم. در روش اول میتوانیم ماتریس W1 را به عنوان بردار کلمات در نظر بگیریم. در روش دوم به طور مشابه میتوانیم وزن لایه 2 را در نظر بگیریم یعنی W2 و در روش سوم میتوانیم میانگین این دو ماتریس را به عنوان بردار کلمات خود در نظر بگیریم.

**شبیه­سازی و ارزیابی**

در این بخش به ارزیابی و نمایش بردار کلمات در فضای 3 یا 2 بعدی میپردازیم که از الگوریتم PCA کمک میگیریم تا این بردارها را در 2 یا 3 بعد نمایش بدیم. برای نمایش 3 بعدی بردار کلمات از Tensorboard استفاده میکنیم همچنین از کتابخانه whatlies نیز برای شبیه­سازی استفاده میکنیم. برای نزدیکترین بردارهای همسایه نیز این قابلیت در Embedder projector وجود دارد با این حال تابع را از اول خودمان بر حسب فاصله اقلیدسی یا کسینوسی پیاده­سازی میکنیم.

**Skipgram (Negative Sampling)**

در این روش ترتیب مراحل تقریبا همان مثل قبل است ولی در ساخت دیتای آموزش کار کمی فرق دارد. در این متد در واقع ما جفت­هایی از کلمه مرکزی با تک­تک کلمات context را در نظر میگیریم و برچسب مثبت یا 1 به آنها میزنیم به معنی این که چنین جفت­هایی درون دیتاست ما کنار هم در یک فاصله مشخص آمده­اند. اما این تمام ماجرا نیست در واقع اگر با این متد بخواهیم پیش برویم ،که به آن naïve softmax گفته میشود، هزینه محاسباتی زیادی را باید بدهیم. در نتیجه ما از روش negative sampling استفاده میکنیم و یک سری داده یا جفت­های جعلی درست میکنیم و به آنها برچسب منفی یا 0 میزنیم و با تابع زیر loss را محاسبه میکنیم که باعث بهینه شدن هزینه محاسباتی­مان میشود.



برای آموزش این دیتاست از کتابخانه pytorch استفاده شده است.

**مراحل کار:**

**Data Loading and Preprocessing**

دیتا را از فایل میخوانیم. البته قبل از آن دیتا را پراسس کرده­ایم و در فایل alimzan-preprocessed ذخیره کرده­ایم که ان را جهت توکنایز کردن استفاده میکنیم.

**Remove Sparse words**

در واقع ما کلماتی که کمتر از حد مشخصی (مثلا 3 بار) تکرار شده­باشند را درون واژه­نامه خود نمی­آوریم و آنها را درون مجموعه­ای نگه­داری میکنیم. (exclude)

**Prepare Training data**

حال باید داده­ها یا همان جفت­هایمان را آماده­ کنیم. پنجره­ها را با سایز مشخص درست میکنیم و سپس جفت­های (center,context) را درست مینماییم.

**Build Unigram Distribution**

همانطور که گفته شد باید جفت­های منفی نیز درست کنیم پس ما یک توزیع نیاز داریم تا کلمات تصادفی را از ان انتخاب کنیم تا جفت­های با برچسب 0 را ایجاد کنیم. ما از توزیع استفاده میکنیم برای انتخاب کلمات نمونه زیرا این توزیع باعث میشود کلماتی که frequency کمتری دارند نیز شانس انتخاب شدن داشته باشند. تعداد کلمات نمونه هم باید مقداری مشخص باشد نه خیلی کم و نه خیلی زیاد.

**Modelling**

حال که داده آموزش آماده شده است به مرحله train میرسیم. ابتدا معماری مدل خود را باید مشخص کنیم برای این کار ما دو لایه embedding در نظر گرفتیم یکی برای کلمات وقتی center هستند و یکی نیز برای همان کلمات هنگامی که به عنوان context یک کلمه دیگر هستند. ابعاد این لایه­ها (vocab\_size, projection\_dim) میباشد. در قسمت forward propagation ما تابع هزینه را بر اساس فرمولی که در ابتدا آورده شد به کمک این دو لایه حساب میکنیم و مقدار هزینه را برمی­گردانیم.

**Train**

باید پارامترهای خود از جمله تعداد dimension کلمات و تعداد epoch و batch size و تعداد نمونه­های منفی را مشخص کنیم. سپس optimizer ,learning rate را مشخص میکنیم. حال یک حلقه به تعداد epoch هایمان میزنیم و درون آن روی تابع getBatch پیمایش میزنیم تا در هر بار پیمایش یک batch از داده­های آموزش را به ما بدهد. سپس آن­ها را برای مدل خود میفرستیم و در متغغیر loss ذخیره میکنیم و از متد backward استفاده میکنیم تا backprop انجام شود و به دنبال آن بهینه­ساز نیز وزن­ها را آپدیت میکند. بعد از اتمام پیمایش­ها اکنون وزن کلمات را میتوانیم از آن دو لایه دریافت نماییم که نشان دهنده word embedding های ما هستند.

**Test**

طبق روال کلی حال باید از صحت word embedding های خود مطئن بشویم که 2 راه برای این کار وجود دارد: Euclidean distance, Cosine similarity

همینطور مقدار میانگین loss مدل در هر 10 پیمایش را نیز نشان داده­ایم.

Epoch : 0, mean\_loss : 0.70

Epoch : 10, mean\_loss : 0.52

Epoch : 20, mean\_loss : 0.45

Epoch : 30, mean\_loss : 0.43

Epoch : 40, mean\_loss : 0.42

Epoch : 50, mean\_loss : 0.41

Epoch : 60, mean\_loss : 0.41

Epoch : 70, mean\_loss : 0.40

Epoch : 80, mean\_loss : 0.40

Epoch : 90, mean\_loss : 0.40

**Web application**

ما از کتابخونه streamlit جهت ایجاد یک وب اپلیکیشن ساده کمک گرفته­ایم که کاربر میتواند در آن یک کلمه را از واژه­نامه انتخاب کند و تعداد k همسایه­های نزدیکش را نیز خودش انتخاب کند و بعد همسایه­های آن را با فاصله­ی اقلیدسی یا کسینوسی بسته به انتخاب کاربر به آن نمایش می­دهیم. جهت نمایش نمایش نقشه لغات نیز از Tensorboard استفاده کردیم که بردارهای کلمات را برایش میفرستیم و نقشه کلمات را به خوبی و بدون کندی نمایش میدهد. همینطور از کتابخونه whatlies نیز جهت شبیه­سازی نقشه لغات استفاده کردیم و به دلیل محدودیت نمیتواند تمام کلمات را به یکباره نمایش دهد و محدودیت 5000 کلمه را دارد تا بدون کندی بتواند کار کند و همینطور قابلیت زوم نیز دارد. مزیت این کتابخانه این است که به ما خروجی از نقشه لغات میتواند بدهد.