Word Vectors

- CSE538: Natural Language Processing -

Report : HW-1

Name:- Akshay Mallipeddi
SBU ID: 112078311

Stony Brook University
Masters in Computer Science

Contents

RESULTS ON ANALOGY TASK WITH HYPER-PARAMETERS TUNING

Python Version:- 2.7.15

TensorFlow Version:- 1.10.1

Baseline Accuracy:- 33.7%

Note:- Got the value for baseline accuracy by taking **max_num_steps=0**

Most important observations:

1. The combination of batch_size = 128, skip_window = 4, num_skips = 8, negative_sample = 256, learning_rate = 1.0, the learning rate was high and the cost was oscillating between 2.8 and 3.7. But when tried the same combination with alpha =0.01, I could observe that the cost values were strictly decreasing and finally converged at 2.56. Though there was no significant effect on accuracy this was an important observation that I made.

2. Another important observation for combination of batch_size = 128, skip_window =4, num_skips = 8, negative_sample = 256, learning_rate =0.01 was that as learning rate was low, learning was slow and it could not reach the minimum cost or could not converge to a value because of low num_steps which was 200000 and I got accuracy as 33.8. Then I increased the learning rate to 0.5 the cost could converge to a point and the accuracy was 34.3 which is better than previous case. **So, small learning rate or large learning rates is a problem.**

3. I even commented the pretrained model part in the code and tried with my own embedding size of 200 there wasn't much of effect on the accuracy.

4. I observed for a single configuration the accuracies were varying a lot and the main reason could be because the initialization of the embedding which is happening randomly from normal or uniform distribution. For example when batch_size = 32, skip_window = 4, num_skips =8, negative_sample = 64 and learning_rate = 1.0 I got (35.1,34.7,35.3,34.2,34.4). The same is mentioned in my README file as well.

5. I have tried many possibilities for batch_size, skip_window, num_skips, negative_samples, max_num_steps, while all these combinations had very less effect on accuracy while tuning learning_rate I could find a good change in the accuracy.

6. My best cross entropy model has a configuration of **batch_size = 128, embedding_size = 128, skip_window = 4, num_skips = 8, num_sampled = 64, Learning_rate = 1.0.**

7. My best NCE model has a configuration of **batch_size = 256, embedding_size = 128, skip_window = 4, num_skips = 8, num_sampled = 128, Learning_rate = 1.0.**

Table 1: Multiple Configurations by keeping **batchsize**=128 constant

Configuration (Batch_size, skip_window, num_skips)	Noise Contrastive Estimation	Cross Entropy
128, 2, 2	34.1%	33.2%
128,2,4	33.2%	33.3%
128, 4 , 8	**34.4%**	**35.3%**
128, 5, 8	33.9%	33.8%
128 ,6, 8	34.3%	33.8%
128,8,16	33.9%	33.9%
128,16,32	34.4%	33.9%
128,32,64	32.8%	33.7%

Referring to Table 1, I have kept batch_size = 128 constant and tried many possibilities for skip_window and num_skips. The best possible combinations was (128, 4, 8). This combination has good accuracies for both CE and NCE. As I increased the skip_window or num_skips the accuracy was decreasing for NCE and CE.

Table 2: Multiple Configurations by keeping **batchsize**=256 constant

Configuration (Batch_size, skip_window, num_skips)	Noise Contrastive Estimation	Cross Entropy
256, 4 , 8	**34.3%**	33.6%
256, 5, 8	33.4%	33.5%
256 ,6, 8	34%	33.9%
256,8,16	**34.3%**	33.8%
256,16,32	33.9%	33.9%
256,32,64	32.5%	**34.0%**

Referring to Table 2, I have kept batch_size = 256 constant and tried many possibilities for skip_window and num_skips. The best possible combinations was (256, 4, 8) and (256, 8, 16) for NCE and (256, 32, 64) for CE. This combination has good accuracies for both CE and NCE. As I increased the skip_window or num_skips the accuracy was decreasing.

Table 3: Multiple Configurations by keeping **batchsize=256** constant by tuning many other parameters

Configuration (Batch_size, skip_window, num_skips, num_samples, learning_rate)	Noise Contrastive Estimation	Cross Entropy
256,4,8,128,1.0	**35.3%**	33.6%
256,8,16, 128, 1.0	33.8%	33.8%
256,4,8, 128, 0.05	33.8%	**33.9%**
256,4,8, 128, 0.01	34.5%	33.8%
256,8,16,128,0.01	34.4%	33.8%
256,4,8,128,2.0	34.7%	**33.9%**

Referring to Table 3, I have kept batch_size = 256 constant and tried many possibilities for skip_window, num_skips and learning_rate. The best possible combinations was (256,4,8,128,1.0) for NCE and (256,4,8, 128, 0.05), (256,4,8,128,2.0) for CE.

Table 4: Multiple Configurations by keeping **batchsize=128** constant by tuning many other parameters

Configuration (Batch_size, skip_window, num_skips, learning rate)	Noise Contrastive Estimation	Cross Entropy
128,4,8, 0.05	34.2%	33.9%
128,4,8,0.01	**34.4%**	33.8%
128,5,8,0.1	34%	**34.1%**

Referring to Table 4, I have kept batch_size = 128 constant and tried many possibilities for skip_window, num_skips and learning_rate. The best possible combinations was (128,4,8,0.01) for NCE and (128,5,8,0.1) for CE.

Table 5: Keeping **batch_size = 128, skip_window = 4, num_skips=8** constantand tuning max_num_steps

Configuration (Batch_size, skip_window, num_skips, max_num_steps)	Noise Contrastive Estimation	Cross Entropy
128,4,8,250000	34%	33.9%
128,4,8,300000	34%	33.5%

Referring to Table 5, I have kept batch_size = 128, skip_window = 4, num_skips=8, and tried possibilities for max_num_steps. Observation was that increasing the value for max_num_steps did not effect the accuracy for both CE and NCE. We should not increase the max_num_steps because it is possible that the model becomes overfit and fails on test data.

Table 6: Keeping **batch_size = 32, skip_window = 4, num_skips=8** constant and tuning learning_rate

Configuration (Batch_size, skip_window, num_skips, learning_rate)	Noise Contrastive Estimation	Cross Entropy
32,4,8,1.0	**35.3%**	33.3%
32,4,8, 0.05	33.5%	**33.8%**
32,4,8,1.5	33.6%	33%

Referring to Table 6, I have kept batch_size = 32, skip_window = 4, num_skips=8, and tried possibilities for learning_rate.

TOP 20 SIMILAR WORDS

From below tables we can see that few of synonyms and antonyms were captured by the models. For example, first->last (**antonym**); first->original (**synonym**). And more than 50% of the words in top 20 were somewhat related. [Could be antonyms, synonyms]. Few words like ismailis, afro, t, s, rabukawaqa etc were noisy and not at all related.

Table 7: Top 20 words for **"first"** [Cross_Entropy v/s NCE]

Cross Entropy	NCE
last ,second ,same ,original ,next ,main , final ,most ,best , largest ,following ,another , entire ,greatest ,latter , actual ,own , ismailis ,directly ,afro ,	most ,during ,name , after ,at ,best ,was , and ,war ,following , in ,of ,one ,to , when ,book ,on , he ,which,last

Table 8: Top 20 words for **"would"** [Cross_Entropy v/s NCE]

Cross Entropy	NCE
will ,could ,can , did ,must , should , might ,may ,does , was , began ,had , came ,is ,cannot , seems ,were ,do ,t , became	been ,could ,will , not ,they ,we ,who , that ,does ,did ,might ,only , if ,do ,but ,these , should ,so ,must,even

Table 9: Top 20 words for **"american"** [Cross_Entropy v/s NCE]

Cross Entropy	NCE
british ,german ,french ,russian ,english , canadian ,italian ,usurped , european , transpiration ,reject , oncifelis ,kaunas , anatomical ,projectiles ,relinquished , presiding ,hobbled ,ceiling , rabukawaqa	french ,nine ,war , of ,three ,four , eight ,five ,six , in ,two ,by ,one ,zero , s ,its ,that ,seven ,from ,UNK

My observation is that the words for CE are better than NCE the main reason could be that CE looks into the whole vocabulary while NCE looks only only on K negative samples and my inference is that negative samples are to be generated using semantics and not randomly. [1]. If the negative

[1] `http://demo.clab.cs.cmu.edu/cdyer/nce_notes.pdf`

sampling is done properly then the word representations would possibly be better.

SUMMARY FOR NCE

Noise contrastive estimation was mainly introduced to reduce the computation that was involved in the denominator in the soft-max formula that involved the whole vocabulary.

1. The main idea behind NCE is to use positive and negative samples. So, it reduces the idea to estimating parameters for a binary classification.
2. We train a logistic classifier to identify true samples from a set of positive and negative samples described below.
3. By learning to distinguish the true pairs from corrupted ones, the classifier will ultimately learn the word vectors
4. For NCE, we have two kinds of distributions one for positive samples and other for negative samples. We pick one sample from positive distribution and 'k' samples from the negative distribution. So, for one context word we will have one true target (label) and 'k' noisy or negative targets (labels).
5. Now we have reduced the problem to the vocabulary size of (1+k) samples, which makes NCE much faster than softmax.
6. We have NCE loss function for given batch of instances as:-

$$J(\theta, Batch) = \sum_{w_o, w_c \in Batch} -\left[log\ P_r(D=1, w_o|w_c) + \sum_{x \in V^k} log\ (1 - P_r(D=1, w_x|w_c)) \right]$$

7. The first term in the formula tells us the log of the probability for the true sample that is considered for one single instance. The second term tells the log of the probability for 'k' negative samples for one single instance.
8. We now try to maximize the formula which is a log-likelihood function to learn for the parameters, in our case word embedding.
9. So, as we saw the above points NCE reduces the computation complexity and it takes 'k' noisy samples converting the original problem into a binary classification task. The main intuition behind this negative sampling is that predicting the target is difficult where as gathering the wrong(negative) target words can be easily done. Implementing this is easy and helpful for the classifier to predict the positive and negative samples with enough training data.

OBSERVATIONS AND LEARNINGS

1. Most time consuming task for me was calculating the accuracy with hyper-parameters using word_analogy.py. My intuition for cosine similarity was totally wrong initially. I was calculating the average cosine similarity of the examples and then comparing the the cosine similarity of each of the choice with this average cosine similarity and taking the maximum and minimum values. After going through few examples of my own, I realized this is the wrong way.

2. I then took the difference vectors of each pair from the examples and took the average of these differences. Then similarly I took the difference vector of each choice and took cosine similarity of this difference vector with the average that we got from the examples. I calculated this for all the choices and selected maximum and minimum from this combination.

3. One more observation I made was that the inbuilt spatial.distance.cosine()[2] was returning the cosine distance instead of cosine similarity. For this to work I had to subtract **1** from the return value of the function.

4. I read few papers in which they got very good accuracy for **embedding size = 300**

[2] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cosine.html`