ASSIGNMENT REPORT

Hyper-Parameters Tuning:

Num Steps: By taking different values of num steps we can have different number of iterations(epoch) over the corpus(vocabulary) or the training data. The value of these iterations should not be very large or very small. Small iterations might not give the true results while many iterations might result in overfitting.

Batch size: Batch size is the number of instances generated in one batch. The batch size selected should be optimum and in general lie between 100-300. Taking a batch size greater than this might result in bad results.

Number of negative samples: I tried testing the model by taking negative samples on both sides i.e. by taking larger samples and by taking smaller samples. Negative samples are the samples that are drawn from noise distribution to indicate that noise samples are more frequent than data samples in a distribution. This estimate approaches the likelihood gradient of the normalized model, allowing us to trade off computation cost against the accuracy

Num Skips and Skip Window: Num Skips are the number of samples you want to draw in a window and skip window decides how many words to consider left and right from a center word. This in a way helps to find the meaning of each center word based on its surrounding words. The value selected for this should be at an optimum level as too close will not produce significant results and too far off words might give us incorrect meanings.

Embedding Size: The embedding size basically denotes the dimensions of the vector. The size determines the information it holds about the input vector. Here also too big a size won't meet the purpose of word embeddings and too small will not give us good results.

Experimentation Details:

NCE Loss Model Test:

Test 1: I initialized the Num steps(epochs) as zero. The only parameter change was the num of steps, keeping all other parameters same. With this model I got an accuracy of 30.6%.

Test 2: By taking the max steps as 200000. I got an accuracy of 31.6%, which was about 1% increase from the baseline where I got 30.6%

Test 3: I noticed that by increasing the batch size from 128 to 180, the accuracy of the model did not have much of an impact. I got the value as 31.9%. So, from this I inferred that the optimum batch size should lie between 100-300.

Test 4: Changing the batch size to 256, num skips to 2 samples to 96 and taking 100000 steps, I got an accuracy of 31.7%.

Test 5: When the num skips were increased from 4 to 6 and the samples were increased from 64 to 128, I saw an increase in accuracy to 32.1%. This was by far the best accuracy that I achieved till now.

Test 6: When the sample was increased from 64 to 128, I saw a change in accuracy to 31.8%. So, in larger values of k, the likelihood gradient of the normalized model was allowing us to trade off computation cost against the accuracy.

Test 7: When the sample was decreased from 64 to 32, the accuracy decreased to 31.2%. So, we need to find an optimum number of sample for prediction.

Test 8: When the embedding size was changed to 200 from 128 and the value of k was also changed from 64 to 128, I got an accuracy of 34.8%. Though this was the best prediction that I got but surely there was issues in predicting similar words. I was getting noisy data, may be because it was not run on the pretrained model.

Cross Entropy Loss Model Test:

Test 1: I initialized the Num steps(epochs) as zero. The only parameter change was the num of steps, keeping all other parameters same. With this model I got an accuracy of 30.6%.

Test 2: By taking the max steps as 200000. I got an accuracy of 30.9%, which was about 0.3% increase from the baseline where I got 30.9%

Test 3: I noticed that by increasing the batch size from 128 to 200, the accuracy of the model did not have much of an impact. I got the value as 30.9%.

Test 4: When the num skips were changed to 6 and skip window was changed to 3 and batch size was changed to 132, my accuracy was 30.9%. So, there was a slight increase in the value from the baseline accuracy.

Test 5: When the skip window was increased from 4 to 6, I saw a change in accuracy to 30.6%.

<u>TOP 20 SIMILAR WORDS</u>

CROSS ENTROPY

First	**American**	**Would**
First	American	Would
Last	German	Could
Name	British	Said
Following	French	Will
During	English	We
Original	Italian	India
Most	Russian	Must
Same	War	Been
End	European	Not
Until	Understood	Does
Second	Borges	Do
Best	International	They
Book	Autres	Did
After	Terminal	You
Beginning	Irish	Families
Next	Canadian	Who
City	Intending	Should
Title	Trade	Believed
Before	Contributors	If
Main	Composer	May

NOISE CONTRASTIVE ESTIMATION

First	**American**	**Would**
First	American	Would
During	English	Could
Until	French	So
Word	International	Will
Most	between	only
Time	d	not
Early	society	they
End	property	however
After	political	althoughw
Use	russian	what
Century	proudhon	but
State	modern	property
From	European	international
Movement	State	warren
Before	Including	should
Book	Word	t
Against	Rights	no
Society	Warren	some
Term	Social	considered
Only	Self	him

<u>NCE (Noise Contrastive Estimation):</u>

In our model Word2vec, our goal was to find the distribution of target words w based on the given sequence of context or center words h. Initially, the cross entropy model was used to predict the target words based on the context words by using a scoring function. The cross entropy worked on the concept of softmax which calculates the probabilities of each target word over all possible context words. However, this turned out to be very expensive, because it required the computation and normalization of each probability using the score for all other V(vocabulary) words w' in the current context h, at every training step.

To deal with the expensive computation of softmax, Word2Vec uses a technique called Noise Contrastive Estimation. With the help of NCE, we can train billion-word datasets with vocabularies of hundreds of thousands of words without turning out to be expensive. NCE is based on the concept of LBL (log bilinear model) which works by performing linear predictions in the word feature vector space. Given a sequence of words, the model computes the predicted representation for the target word by taking a linear combination of the context word feature vectors. Also, to reduce the high cost involved in matrix-vector multiplication the model uses vectors.

The NCE model computes expensive learning problem into a binary classification problem that uses the same parameters but are easier to compute. To make the softmax soft in terms of computational cost, sampling methods are introduced, which instead of considering the entire vocabulary space, takes just few samples to mimic the true distribution and obtain the desired results. The working principle here is to train a classifier to find the difference between samples from data distribution and samples from noise distribution. The idea of taking k samples (words that are not present in the context window of the context) is to show that noise samples are k times more frequent than data samples. The goal is to have a high probability for data samples and low probability for negative samples. In this way we avoid the summation over the entire vocabulary as done in softmax and just consider the sum over k samples for the estimation of the target value and see how similar it is to the context word.

The formula is:

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

In my model, I implemented both the models, cross entropy and NCE and I could see the difference in computation and accuracy of the result between the two. While running the cross entropy mode, it gave me a loss of 4.8x. NCE on the other hand gave me a loss of 1.3x, which is very less and this indicates how close words are mapped to each other. We can have a better representation of words in this way.