1 Hyper-parameter Exploration

1.1 batch_size

In Stochastic Gradient Descent optimization, we train our model on batches of data instead of the entire data. *batch_size* corresponds to the number of pairs of (predicting word, context word) in a particular batch i.e. the size of our training batch. Ideally, as we add more data i.e. increase our *batch_size*, the training loss should decrease and the accuracy should go up. For larger datasets, a smaller *batch_size* leads to faster convergence as it processing it is computationally efficient while a large *batch_size* requires more time to converge and takes up more memory.

1.2 max_num_steps

This corresponds to the number of epochs to run before stopping the learning/optimization process. If the number of epochs is low, the loss of our model does not converge to the minima. As the epochs increase, our loss starts to converge towards minima. However, there comes a saddle point in the number of epochs vs loss where an increase in epochs does not decrease the loss significantly. At this point, our model is well-trained.

1.3 learning_rate

The learning rate controls the weight update in the learning process. A small learning rate will lead to smaller updates in the weights of our model which will lead to a slower convergence towards the optimum value. However, it is more likely to lead us to the optimal values resulting in better overall accuracy of the model. A large learning rate will lead to larger updates in the weights and faster convergence but it is less likely to find the minima and the accuracy of our model will suffer.

1.4 window_size

The window size determines the number of words to the left and to the right of a target word that we have include as context words. A smaller window helps us in predicting words that are more related. Therefore, a smaller window size gives more syntactically similar results and a larger window makes more semantically similar predictions. In our implementation, the window_size is controlled by skip_window which is defined as "how many words to consider left and right of a centre word".

1.5 num_sampled

In negative sampling, num_sampled represents the number of negative examples to sample. Here, we randomly select a small number of "negative" words to update the weights for instead of updating all the weights of our neural network.

2 Experimental Analysis

2.1 Epochs

Parameters	batch_size	learning_rate	skip_window	num_skips	embedding_size
Values	128	1.0	4	8	128

Epochs	**NCE Loss**	**Overall Accuracy**
100001	1.5258	33.4%
200001	1.4969	33.9%
300001	1.4866	33.5%
400001	1.4288	33.9%

From the above empirical results, as the number of epochs increase, the loss decreases and the accuracy generally improves. However, after 400001 epochs, the improve in accuracy is marginal when compared to the time required for computation indicating that the model is already well-trained and has reached a saddle point.

2.2 Batch Size

Parameters	Epochs	learning_rate	skip_window	num_skips	embedding_size
Values	200001	1.0	4	8	128

Batch Size	**NCE Loss**	**Overall Accuracy**
64	1.7318	33.4%
128	1.4969	33.9%
256	1.4119	34.5%
512	1.4916	33.5%
1024	1.4567	33.8%

As the batch_size grew, the accuracy also became better but it dropped after batch_size=256. However the computations became relatively slow and batches that were ≥ 512 in size were very slow to train. However, the maximum value for batch_size on which we can train without encountering any problems also depends on the RAM as we should be able to hold the batches and the results of vector multiplications in memory (since I was not using a GPU).

2.3 Learning Rate

Parameters	Epochs	batch_size	skip_window	num_skips	embedding_size
Values	200001	256	4	8	128

As we can see from the table, a lower learning rate lead to a better accuracy of the model and a lower loss. But reaching the optimum value takes time in case of lower learning rates as the weights are updated by a very small amount when compared to the case when learning rate is higher. Also, if the learning rate is very high, the model diverges instead of converging as evident by the lower accuracy (which is even lower than the pre-trained model).

Learning Rate	**NCE Loss**	**Overall Accuracy**
0.05	1.3539	34.2%
0.1	1.5977	34.1%
1	1.23548	33.6%
5	8.6337	32.8%
10	9.13567	29.4%

2.4 Window Size (skip_window)

Parameters	Epochs	batch_size	skip_window	num_skips	embedding_size
Values	200001	256	4	8	128

skip_window	**NCE Loss**	**Overall Accuracy**
2	1.2646	34.4%
4	1.6054	33.8%
8	2.8383	33.6%
16	8.3620	31.9%
32	10.5408	33.6%

I observed that as we increase the size of the window, the accuracy of the model goes down a bit and the loss increases drastically. One possible explanation could be that as we consider longer windows, we start modelling relations between words that are far apart and might not necessarily be related. Also, longer windows weaken the relationship between syntactically similar words. Therefore, having a small window seems to be better for our task.

2.5 num_sampled

Parameters	Epochs	batch_size	skip_window	num_skips	learning_rate
Values	200001	128	4	8	1

num_sampled	**NCE Loss**	**Overall Accuracy**
16	0.4367	34.0%
32	1.0629	34.6%
64	1.3168	34.5%
128	4.3657	33.8%

From the above table, we can see that as num_sampled decreases, the loss decreases too. This correlates with the fact that we update the weights of *num_samples* negative samples and this also forms a part of the equation of cost function. Therefore, higher the *num_samples*, higher the loss. However, a very low and a very high number of negative samples do not correlate well with a good accuracy score.

3 Best Accuracy

3.1 NCE

The best accuracy that I achieved using NCE loss was 35.7% but it was with a high learning rate (~ 6). I believe that this is because the mturk data is very noisy as indicated by Professor Niranjan. The second best accuracy was around 34.6% with learning rate = 0.1.

3.2 Cross Entropy

The best accuracy that I achieved using Cross Entropy loss was 33.9%.

Note: Please see the README.md for more details about configuration used to achieve this.

4 Top 20 Similar Words

4.1 NCE

Word	**Top 20 similar word**
first	american, william, word, law, armand, bourgeois, nine, eight, known, proudhon, founder, dans, exploiting, had, wasn, radical, levellers, operative, for, moment
would	could, using, called, can, together, set, nevertheless, theory, notion, do, right, you, so, only, general, myself, which, civil, over, european, words
american	william, first, nine, law, armand, dans, word, eight, bourgeois, founder, proudhon, known, had, exploiting, wasn, seven, radical, individual, communistic, four

From the above table I observed that first and american are highly similar. Their cosine similarity was as high as 0.98. This means that the two words appear together frequently in the window size that we chose ($num\_skips = 10$). Thus it makes sense that they share many words in the top 20 most similar words.

4.2 Cross Entropy

Word	**Top 20 similar word**
first	last, most, same, original, main, relocations, river, largest, best, protect, west, entire, pending, dweezil, actual, liberally, vanuatu, parking, end
would	could, will, must, did, can, should, does, cannot, was, do, had, began, families, may, is, said, we, believed, made
american	german, british, french, russian, canadian, italian, english, its, sholay, lisa, unofficial, hiatus, nagai, weren, borges, condoms, reckless, denham, martina

Personally, I liked the similar words predicted using cross-entropy model even though the cosine similarity of top 20 most similar words (0.5-0.6) was not as high as in the case of NCE model (0.98). For example, the words similar to *american* are *russian, british, french, canadian* etc which makes absolute sense as they all are nationalities. Also, the word *would* has better similar words than NCE such as *will, must, cannot, was, do* etc

5 NCE loss: A Summary

The objective of the NCE loss function is to learn the weight matrix aka the word embeddings by following a different learning objective than cross entropy. As opposed to the expensive vocabulary sized softmax operation used at the final layer of neural networks, NCE is a cheaper sampling based operation which leads to faster training.

NCE does not try to estimate the probability directly, instead it uses an auxiliary loss that also optimizes the goal of maximizing the probability of words. In other words, NCE learns a model that differentiates the target word from noise by randomly selecting noise/negative samples from the vocabulary. NCE basically reduces the language modelling problem to the problem of estimating the parameters of a probabilistic binary classifier where this binary classifier has to differentiate between a true sample and a noisy/negative sample.

For every positive sample, we will draw k noise samples $y_{1...k} \sim q(y)$ and add it to our dataset. We can label each real sample with $y = 1$ and each noise sample with $y = 0$ to train the binary classifier.

The conditional probability of the label given the sample is:

$$p(y_i^0 = 1|x_i, \theta) = \frac{\exp(y_i^0, h_\theta)}{\exp(y_i^0 h_\theta) + k * q(y_i^0)}$$

$$p(y_i^t = 0|x_i, \theta) = \frac{k * q(y_i^t)}{\exp(y_i^t h_\theta) + k * q(y_i^t)}, t = 1, \cdots, k$$

The scaling factor k in front of $q(y_i)$ accounts for the fact that noise samples are k times more frequent than data samples.

$$\Delta s_\theta(w, h) = s_\theta(w, h) - \log(kP_n(w))$$

Here, first term is the un-normalized score of the word w under the model and the second term is the log probability of word sampling from the noise distribution.Thus, to maximize the probability of a word comes from the true distribution we need to maximize delta.

$$\begin{aligned} J^h(\theta) =& E_{P_d^h}\left[\log P^h(D = 1|w, \theta)\right] + kE_{P_n}\left[\log P^h(D = 0|w, \theta)\right] \\ =& E_{P_d^h}\left[\log \sigma\left(\Delta s_\theta(w, h)\right)\right] + kE_{P_n}\left[\log\left(1 - \sigma\left(\Delta s_\theta(w, h)\right)\right)\right], \end{aligned}$$

In the above equation, the first term corresponds to D=1 case i.e. when the data belongs to true class and second term corresponds to D=0 case i.e. when the sample is noise. This is like a binary classifier. The classifier thus learns the word vectors by learning to discriminate the true pairs from the noisy ones.

References:

1. https://dl.acm.org/citation.cfm?id=2999865
2. https://arxiv.org/pdf/1410.8251.pdf
3. www.linkedin.com/pulse/heavy-softmax-use-nce-loss-shamane-siriwardhana/
4. http://demo.clab.cs.cmu.edu/cdyer/nce_notes.pdf