NLP Assignment 1: Report

Aditya Deshpande [SBU ID: 112101105]

Implementation:

1. Generate Batch:

This generates a batch of words; the variable 'batch' contains context words and 'labels' contains label words [Label words are the words surrounding context words]. The window_size is first calculated as suggested in the hints(comments). While the batch we are generating does not exceed the total data, we first move to the context word with the condition that data_index is at the context word. Add context word to 'batch', then add one word that is to the left as well as to the right of the context word into the 'labels' numpy array. The context word changes after num_skips times. We do the above operation until we have reached the batch_size, upon which we return the batch and labels.

2. Cross Entropy:

Cross entropy is the argmin[negative log likelihood] as taught during the lecture. The formula for cross entropy was done, and is also shown as a hint in the comments for variables. Using the exact same formula for variables 'a' and 'b', the cross entropy is calculated.
A = log(exp((transpose(Uo).Vc)))
B = log(sum{exp(transpose{Uw}Vc)})
For term A, first we take transpose of Uo, multiply it with Vc. A small value[1e-10] is added to avoid error when tensorflow tries to calculate tf.log(0) and generates NaN
For term B, we take transpose of Uw, then use matrix multiplication and take the exponential value of this. We then calculate the sum by reducing the dimension to [batch_size, 1].
Having calculated both terms, we now return their difference as the cross entropy loss.

Note: For most of the models, my cross_entropy_loss is around 4.8

3. NCE Loss:

The paper[1] given in the assignment was referred and the exact formula was used to calculate NCE Loss.
NCE is an approximation technique based on negative sampling. According to the formula, we calculate sigmoid of a positive and a negative probability and perform a few more operations.
Firstly, I extracted word embeddings for predicting words[labels] and reshaped them.
Then transpose followed by matrix multiplication.
After this, the word embeddings for negative samples is calculated and similarly matrix multiplication is done.
As unigram_prob is a vector, it needs to be converted to a tensor, which is done by the tf.convert_to_tensor() method. Now, positive bias, positive score, negative bias, negative scores are calculated, followed by the calculation of positive and negative unigram probabilities.
After all this, we calculate sigmoid1 and sigmoid2[the 2 terms]. Correction value of 1e-10 is added. Sigmoid1 is a positive probability, and sigmoid2 is negative probability.
Finally, negative scalar multiplication of - the log of sigmoid1, and sum of log of sigmoid 2 is calculated according to the formula and this value is returned as NCE loss.

Note: NCE values for most of the models are in coming in the range of 1.3-1.8

4. Word analogy

Initially a single line is taken from all the lines, this line is split over '||' to create 2 partitions – a set of 3-tuples that we call examples and a set of 4-tuples that we call choices. By a set of operations involving stripping and splitting, we get the word pairs A:B C:D E:F from the 'example' array and A:B C:D E:F G:H from the 'choice' array.
Now, we calculate the cosine similarity between the elements of 'example' array and then take average of cosine similarity.
For the key,value pairs in 'choice' array, we calculate maximum and minimum distance from the relation[i.e. from the average distance]
Finally we add and write the results to a separate file according to the given specification.

Set of Hyperparameters explored:

1. Batch Size: This is the total words in training per batch. Loss functions were applied on words belonging to a batch of a certain batch size. A batch size of 64 gave a lower NCE loss and better accuracy.

2. Max_steps: The default value was 200001 and this is the total number of steps the model trains for. This parameter was tweaked to 100001, 50001 and 500001. A higher number of training steps usually indicates that the model is trained extensively.

3. Learning Rate: This is a measure of how fast the model parameters of an SGD are updated to decrease the cost function. Quick googling showed that learning rates are usually a small value such as 0.1, 0.001 and so on so that the model converges better, but my best model had a learning rate of 2.0.

4. Number of samples: The default is 64 for this assignment and this is the number of negative samples k over which V^k is created. I experimented with this by setting the values as 32, 64, 96 for different models.

5. Number of skips: This is the number of words to the left and right of the context word.

6. Skip window: This is another hyperparameter explored and is used to alter the window size.

Results on Analogy Task for 5 different configurations:

A] Cross Entropy:

Max Steps	Batch Size	No. of Skips	Skip Window	No. of Samples	Learning Rate	Accuracy of Least Illustrative Guess	Accuracy of Most Illustrative Guess	Overall Accuracy
200001	128	4	4	64	1.0	34.4%	31.4%	32.9%
100001	64	16	8	64	1.0	27.2%	31.7%	29.5%
100001	200	4	8	96	1.0	31.7%	34.3%	33.0%
50001	64	4	8	64	1.5	30.5%	29.4%	30.0%
200001	64	2	4	64	0.8	33.4%	21.4%	27.4%

B] NCE:

Max Steps	Batch size	No.of skips	Skip Window	No. of samples	Learning Rate	Least Illustrative Guess	Most Illustrative Guess	Overall Accuracy
200001	128	4	4	64	1.0	**32.8%**	**30.4%**	**31.6%**
100001	64	16	8	64	1.0	**30.3%**	**29.3%**	**29.8%**
100001	200	4	8	96	1.0	**33.7%**	**30.3%**	**32.0%**
50001	64	4	8	64	1.5	**35.6%**	**32.6%**	**34.1%**
200001	64	2	4	64	0.8	**29.8%**	**30.4%**	**30.1%**

The observations I made were –

1. The loss converges at around 100000 steps for cross entropy and for NCE, it is very low (<50000).

2. For cross entropy, a higher batch size yielded better accuracy whereas for NCE, a low batch size yielded the most accuracy.

3. A lower learning rate is giving bad accuracy in my case and is also taking a lot of time to compute.

4. When number of skips were lower, an increase in accuracy was found.

Top 20 Words nearest to {first, American, would}

A] Cross Entropy

Nearest to **first**: last, name, following, most, original, during, same, end, until, second, best, largest, main, next, city, rest, beginning, book, th, united,
Nearest to **american**: german, british, italian, english, french, herald, ganda, understood, european, russian, manek, virulence, inhaled, westerosi, orestes, diagnostic, laminin, realise, motherhood, greektown,
Nearest to **would**: said, india, we, could, will, must, been, does, do, did, seems, you, should, they, believed, who, may, become, argued, seen,

B] NCE Loss

Nearest to **first**: during, th, time, since, before, century, act, term, all, revolution, until, later, english, s, king, law, he, work, however, french,
Nearest to **american**: german, british, italian, war, composer, russian, d, century, collages, nanyang, irish, eu, civil, canadian, first, auteurist, actor, spanish, attorneys, hazmi,
Nearest to **would**: could, will, said, we, must, india, did, do, if, who, seems, you, become, so, made, even, she, believed, only, never,

Observations –
1. A different set of words appears in the Top 20 list for cross entropy and NCE.
2. The closest words to "first" are nouns although first can either be a noun, pronoun, adjective or an adverb.
3. The closest words to "American" are names of countries [Proper Nouns] and cases where American is used as an adverb or an adjective.
4. The closest words to "would" are similar sounding interrogations such as could. However, one anomaly is spotted – India, and the reasoning for that could be the presence of a similar piece of text in the training corpus [e.g. - Would India … ?].

Noise Contrastive Estimation Summary:

- Noise Contrastive Estimation is a method for **fitting unnormalized models**, which is a very big advantage, and is based on the reduction of density estimation to binary classification [positive or negative]. The idea is to convert a multinomial classification problem (as it is the problem of predicting the next word) to a binary classification problem. That is, instead of using softmax to estimate a true probability distribution of the output word, a binary logistic regression (binary classification) is used instead. [2]
- NCE uses **logistic function**, and the basic idea is to train a logistic regressor to separate "data" from "noise".
- It has been proven to be at par with some other Language Models while requiring much **less computation time** than most others.
- NCE works because **instead of predicting** the next word (the "standard" training technique), the optimized classifier simply **predicts whether a pair of words is good or bad**, thus giving us a good measure of accuracy.
- The gradient in the equation formulated in NCE involves a sum over k noise samples instead of a sum over the entire vocabulary, making the NCE **training time linear** in the number of noise samples and independent of the vocabulary size. As we increase the number of noise samples k, this estimate approaches the likelihood gradient of the normalized model, allowing us to trade off computation cost against estimation accuracy.

Citations –
[1] – Andriy Mnih, Kavukcuoglu https://www.cs.toronto.edu/~amnih/papers/wordreps.pdf
[2] - https://datascience.stackexchange.com/questions/13216/intuitive-explanation-of-noise-contrastive-estimation-nce-loss
[3] – Rushil Sharma, Anagh Agrawal [classmates] helped me in understanding of the NCE paper.