CSE538: Natural Language Processing (Assignment -1)

Word2Vec Assignment Report

-Utkarsh Garg (SBU#: 112672834, Fall'19)

HyperParameters explored:

- **Batch_Size:** Directly proportional to average loss value. Since we are decreasing words/batch with increasing batch, the model undergoes more fine tuning at each checkpoint step. Increasing batch size ***increases*** loss value.

- **Skip Windows** : Learning the embeddings from left and right sides of pivot word. Increasing the no of context words on either side of a batch results in better training of the center word, and thus affects loss directly.

- **Num_skips:** These are number of word pairs that are trained for each batch. Increasing num_skips increases the loss.

- **Max_Steps:** Increasing steps ***decreases*** the loss value at each iteration.

- **No. of Samples(k)** : As k increases, the training of model improves, resulting in increase in average loss value .

Baseline Models :

Params:

- Batch_Size=128
- Embedding_Size=128
- Skip_windows=4
- Num_skips=8
- Max_Steps=200001
- No of Samples=64

Cross Entropy Loss Function:

Average Loss value at last step : 4.09

Overall Accuracy : 31.3%

Noise Contrastive Estimation (NCE) Loss Function:

Average Loss value at last step : 1.65

Overall Accuracy : 31.6%

Note: NCE has a better accuracy on baseline as compared to Cross Entropy.

Ideal Configuration:
batch_size = 64, skip_window = 2, num_skips = 4, max_num_steps = 400001, num_samples=64

=>Cross Entropy - Average loss at step 400000 : 4.032

Best accuracy achieved for Dev File on Cross Entropy model :

```
Generated by:                          score_maxdiff.pl
Mechanical Turk File:                  word_analogy_dev_mturk_answers.txt
Test File:                             word_analogy_dev_predictions_cross_entropy.txt
Number of MaxDiff Questions:           914
Number of Least Illustrative Guessed Correctly:   299
Number of Least Illustrative Guessed Incorrectly: 615
Accuracy of Least Illustrative Guesses:           32.7%
Number of Most Illustrative Guessed Correctly:    276
Number of Most Illustrative Guessed Incorrectly:  638
Accuracy of Most Illustrative Guesses:            30.2%
Overall Accuracy:                      31.5%
```

=>NCE - Average loss at step 400000 : 1.36

Best accuracy achieved for Dev File on NCE model :

```
Generated by:                          score_maxdiff.pl
Mechanical Turk File:                  word_analogy_dev_mturk_answers.txt
Test File:                             word_analogy_dev_predictions_nce.txt
Number of MaxDiff Questions:           914
Number of Least Illustrative Guessed Correctly:   297
Number of Least Illustrative Guessed Incorrectly: 617
Accuracy of Least Illustrative Guesses:           32.5%
Number of Most Illustrative Guessed Correctly:    280
Number of Most Illustrative Guessed Incorrectly:  634
Accuracy of Most Illustrative Guesses:            30.6%
Overall Accuracy:                      31.6%
```

Cross_Entropy:

	Baseline	Run1	Run2	Run3	Run4	Run5
Batch Size	128	256	64	256	512	64
Embedding Size	128	128	128	128	128	128
Skip Windows	4	4	2	2	4	2
Num Skips	8	8	4	4	8	4
Max Steps	200001	200001	200001	200001	200001	400001
No of Samples	64	64	64	128	64	64
Accuracy of Least Illustrative Guesses	32.4%	32.4%	32.6%	32.6%	32.5%	32.7%
Accuracy of Most Illustrative Guesses	30.3%	30.3%	30.3%	30.4%	30.3%	30.2%
Average Loss at Last Step	4.09	4.67	2.78	4.03	4.81	4.04
Overall Accuracy	31.3%	31.3%	31.4%	31.2%	31.4%	31.5%

Noise Contrastive Estimation:

	Baseline	Run1	Run2	Run3	Run4	Run5
Batch Size	128	128	256	128	128	64
Embedding Size	128	128	128	128	128	128
Skip Windows	4	4	4	1	2	2
Num Skips	8	8	8	2	4	4
Max Steps	200001	200001	200001	200001	200001	400001
No of Samples	64	64	64	64	32	64
Accuracy of Least Illustrative Guesses	32.5%	32.5%	32.4%	32.2%	32.4%	31.4%
Accuracy of Most Illustrative Guesses	30.6%	30.3%	30.3%	30.4%	30.3%	30.7%
Average Loss at Last Step	1.65	0.94	1.39	0.893	1.098	1.36
Overall Accuracy	31.6%	31.4%	31.3%	31.5%	31.3%	31.6%

Cross Entropy Similar Words: Best Model

Similar to **first**: th, rest, third, after, abuse, kingdom, before, until, book, next, best, end, second, during, original, most, name, following, last, first,

Similar to **american**: next, russian, war, its, best, french, end, english, second, italian, during, original, most, name, following, british, last, german, american, american,

Similar to **would**: did, who, following, does, do, been, they, india, british, must, last, german, said, we, will, could, not, would, would, would,

NCE Similar Words: Best Model

Similar to **first**: intelligent, working, class, and, is, visions, english, curtin, whilst, french, term, including, as, a, abuse, still, against, used, early, first,

Similar to **american**: and, is, visions, english, british, curtin, german, whilst, french, term, including, as, a, abuse, still, against, used, early, american, american,

Similar to **would**: we, been, will, could, that, not, whilst, french, term, including, as, a, abuse, still, against, used, early, would, would, would,

Noise Contrastive Estimation:

It's a type of loss estimation method, that works when the size of the training data(vocabulary size in this case) is huge. Cross Entropy cost function , when used with standard Neural Networks yields values of output neurons representing Probability. Further up, the output probabilistic scores are then normalised, using Softmax function for each input neuron fed to the Neural net. Applying Softmax over large input set is tremendously costly , and takes a huge amount of CPU processing time.

NCE comes to the play in this case. The logic behind this cost function is that instead of using Probability density function of the output (next unknown word, in the case of Word2Vec problem) , we use Binary Logistic Regression instead.

i.e, Multinomial Classification is reduced to *Binary Classification problem*.

Here's the Softmax equation for Cross Entropy:

$$p_\theta(w \mid c) = \frac{u_\theta(w, c)}{\sum_{w' \in V} u_\theta(w', c)}$$

In NCE, **we simplify this expensive cost function by elimination of the normalising part.**

Objective: Train a Logistic Regression classifier, which can separate samples from true distribution and samples from noise distribution, based on ratio of probabilities of the sample under the model and noise distribution.

Implementation:

- *Learning/Training Steps :* For each training sample, out classifier is fed a true pair(label) and some randomly corrupted pairs (say 'k' samples) .
- *Prediction Step* : The classifier simply predicts whether a given word pair (output or test-set) is good or bad.

Negative Sampling: For corrupted pairs, less frequent words are drawn more often.
The effect of applying negative sampling to model is that it approximates Maximum Likelihood Estimation , when the ratio of noise to real data is high.

We calculate a special version of the digits for each of the classes (1 from data + k from noise distributions) using equation:

$$\Delta \square_{\square 0}(\square, h) = \square_{\square 0}(\square, h) - \log \square_{\square \square}(\square)$$

Here, the first term is the unnormalized score. And the next term is the log probability of k words sampled from the noise distribution

For the negative samples selected, we then take the *Negative Log Like*lihood while calculating loss on predicting context words given center word. This gives us the overall Loss function, while training word2vec model.

Conclusion: The sum over k noise samples instead of a sum over the entire vocabulary, makes 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 the accuracy