Assignment 1: Report

Sabir Ismail, SBU ID: 111734933

CSE 538 Fall 2018

1. Word Embedding:

In this project I implement the Skip Gram model, two loss function (1) Cross Entropy and (2) Noise Constructive Estimation and Word Analogy and used these two-loss function to generate the Word Embedding. I tried with different configuration of the hyper-parameters and evaluate the performance in the word analogy data-set and top 20 words for some given words.

2. Hyper-Parameter Exploration & Word Analogy:

2.1 Hyper-Parameter Exploration:

There are five hyper-parameter that I tried to tune, (I) Batch Size, (II) Embedded Size, (III) Skip Window, (IV) Number of Skips, (V) Number of Batches.

1. Batch size: It does not change the results a lot stand alone. If I change some other parameter with it, then changes happen.
2. Embedded size: I run on my own model, not loading the pretrained model, with different embedded size. The accuracy of the analogy increases but the top 20 words are worse.
3. Skip window: It has greater effect of the results on top 20 words. Smaller values give better top words.
4. Number of skip window: It also has same effect as skip window.
5. Batch size: After a certain size, it has no effect, average loss remain same.

2.2 Word Analogy:

To implement the word analogy task, first I calculate the vector difference for each pair(a,b) of words from the given relation. Then take the mean vector of the all vector differences. This give me the vector to which I will measure the, similarity of the query pairs. Next to compute the most illustrative and least illustrative pair, for each pair (c,d), I calculate the difference (d-c), finally the cosine similarity between this vector and mean vector. The most illustrative and least illustrative pair are the maximum and minimum cosine similarity value.

2.3 Cross Entropy:

The following table shows my experimental data with the hyper-parameters.

Meaning of the model: model_batchsize_embeddedsize_skipwindow_numberofskip_numberofbatches

Model	Loss	Pretrained Used	Analogy Accuracy
cross_entropy_256_128_8_16_400001	4.851963997	Yes	29.00%
cross_entropy_256_128_1_2_400001	4.852030279	Yes	29.00%
cross_entropy_128_128_4_8_200001	4.851963997	Yes	29.00%
cross_entropy_64_256_16_32_200001	5.545532227	No	32.40%

cross_entropy_64_128_16_32_200001	4.85196352	No	32.60%
cross_entropy_64_128_16_32_200001	4.85196352	Yes	33.80%
cross_entropy_64_128_16_32_400001	4.85196352	Yes	33.80%
cross_entropy_64_128_8_16_200001	4.851963997	Yes	33.80%
cross_entropy_128_128_32_16_200001	4.85196352	Yes	33.80%
cross_entropy_64_128_32_16_400001	**4.85196352**	**Yes**	**33.80%**

Observation:

- I got better accuracy in the word analogy when using 8-32 skip window and number of skips.
- But the average loss and for the top 20 words, smaller number of skip window and number of skips gives better results.
- I also noticed that number of steps, did not affect a lot, for some very large number of steps, after sometimes, average loss does not reduce.
- When I do not used pre-trained model and try with larger embedded size, I got better results in the analogy but bad in the top 20 words.
- Embedded size works well for the more contextual meaning rather than semantic meaning.

2.4 Noise Construction Entropy:

Meaning of the model: model_batchsize_embeddedsize_skipwindow_numberofskip_numberofbatches

Model	Loss	Pretrained Used	Analogy Accuracy
nce_512_128_1_2_100001	0.60229164	Yes	28.90%
nce_512_128_1_2_400001	0.965992543	Yes	28.80%
nce_512_128_8_16_400001	1.03071159	Yes	29.00%
nce_1024_128_8_16_400001	1.023730828	Yes	29.00%
nce_1024_128_16_32_400001	1.178907914	Yes	33.80%
nce_512_128_1_2_500001	1.078128762	Yes	33.90%
nce_512_128_8_16_500001	1.281818531	Yes	33.80%
nce_128_128_8_16_500001	1.079612234	Yes	33.80%
nce_128_128_4_4_500001	3.238003731	Yes	33.80%
nce_128_128_4_4_500001	0.89891915	Yes	33.80%
nce_128_256_4_4_500001	0.766869608	No	31.20%
nce_512_256_4_4_500001	1.064935726	No	29.20%
nce_512_128_1_2_500001	**1.002989299**	**Yes**	**34.00%**

Observation:

3. TOP 20 words:

4.1 Cross Entropy:

Settings:

Batch Size: **128;** Embedded Size: **128;** Skip Window: **2;** # of Skips: **2** # of Steps: **400001**

First	last, name, following, during, most, original, second, same, until, end, after, best, before, book, city, united, main, next, beginning, title,
America	german, british, english, french, italian, its, war, russian, european, understood, of, international, irish, canadian, borges, united, trade, d, other, autres,
Would	not, that, could, will, been, we, said, must, india, they, do, does, who, did, you, to, families, if, should, may,

4.2 Noise Construction Entropy:

Settings:

Batch Size: **512;** Embedded Size: **128;** Skip Window: **8;** # of Skips: **16;** # of Steps: **400001**

First	last, name, following, during, most, original, second, same, until, end, after, best, before, book, city, united, main, next, beginning, title
America	german, british, italian, its, war, russian, european, understood, international, irish, canadian, borges, united, trade, d, other, autres, player, terminal, writer
Would	not, that, could, will, been, we, said, must, india, they, do, does, who, did, you, families, if, should, may, but

Observation:

- I got better top 20 words when, the average loss is minimum.
- Fewer number of skip window and number of skips give better results.
- Number of steps does not affect the similarity much. But it should be large enough.
- With pre-trained model I got the better results.
- I also tried with not using pre-trained model and increase the embedded size but did not get better results.

4. Summary of NCE

The basic idea for the NCE is, for a given context word, w_c, a target word w_o and several noise words w_x, implement a cost function using Logistic Regression Classifier. NCE, use true distribution for the target word and a noisy distribution for the noisy words, and use logistic regression classifier, which separate those two distributions. Here is the final equation.

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

where,

$$Pr(D = 1, w_o | w_c) = \sigma\left(s(w_o, w_c) - \log\left[kPr(w_o)\right]\right)$$

$$Pr(D = 1, w_x | w_c) = \sigma\left(s(w_x, w_c) - \log\left[kPr(w_x)\right]\right)$$

To compute the probability, Pr(D=1, w_o|w_c), we start with the joint probability Pr(D,w_o|w_c). We have a list of N+1 words, w_o,w_c1,w_c2,…w_cn, and we have 1 out of N+1 chance, to pick up the true words and N out of N+1 chances, to pick up the noisy word denoted as q(w_x).

$$Pr(D=1|w_o,w_c) = \frac{Pr(D=1,w_o|w_c)}{Pr(D=1,w_o|w_c)+Pr(D=0,w_o|w_c)} = \frac{Pr(w_o|w_c)}{Pr(w_o|w_c)+Nq(w_x)}$$

$$Pr(D=0,w_o|w_c) = \frac{Pr(D=0,w_o|w_c)}{Pr(D=1,w_o|w_c)+Pr(D=0,w_o|w_c)} = \frac{Nq(w_x)}{Pr(w_o|w_c)+Nq(w_q)}$$

We are using single sample form the true words and N sample form the noise words, that why there is an N in the lower equation. Here, we have a costly function to compute, Pr(w_o|w_c). To overcome this calculation, Logistic Regression is used. We can use logistic regression with unnormalized probability term as follows.

$$Pr(D=1,w_o|w_c) = \frac{u_\theta(w_o,w_c)}{u_\theta(w_o,w_c)+Nq(w_q)}$$

$$Pr(D=0,w_o|w_c) = \frac{Nq(w_x)}{u_\theta(w_o,w_c)+Nq(w_q)}$$

The u_theta is the exp(w_o,w_c). But this score is unnormalized and to normalized it, a score from the noise distribution is introduced. So, for the above equation, after applying sigmoid function, equation is:

$$Pr(D=1,w_o|w_c) = \frac{u_\theta(w_o,w_c)}{u_\theta(w_o,w_c)+Nq(w_q)} = \sigma(\triangle(s_\theta(w_o,w_c))$$

But now to calculate the delta_s, the difference between the score and the noise distribution (in log scale) is taken.

$$\triangle(s_\theta(w_o,w_c)) = s_\theta(w_o,w_c) - log(kPr(w_o))$$

To maximize the probability of a word comes from the true distribution we need to maximize the delta(s) part. So, the final loss equation is:

$$\mathcal{L}_\theta = -[\log \frac{\exp({w_c}^\top w_o)}{\exp({w_c}^\top w_o)+Nq(w_x)} + \sum_{\substack{i=1 \\ w_x}}^{N} \log \frac{Nq(w_x)}{\exp({w_c}^\top w_o)+Nq(w_x)}]$$