CS 565 ASSIGNMENT 2

PRIYANSHU SINGH 170101049

Colab Link Problem 1
Colab Link Problem 2

Instructions: Go to Runtime. Select run all cells to reproduce the results. It may take very long time(2-3 hrs) to run completely. Output results of previous runs should be visible.

- For sentence segmentation and word tokenization, I have used **NLTK Punkt** segmenter and tokenizer. The corpus is first shuffled and then split into 9:1 ratio to make Part-1 (train+development) and Part-2(test set). Part-1 is again shuffled and then split into 9:1 ratio to make Training set and Dev/Validation set.

N-gram Language Model

* While discounting, on working with whole corpus, Colab crashes due to exceeding RAM. For linear interpolation, the code can run on whole corpus but for consistent comparison, I have run it on a subset of corpus (2000 sentences) for linear interpolation, discounting as well as Laplace smoothing.

Data Preprocessing:

- After the corpus is split into different sentences using Punkt sentence segmentation, each sentence is changed to include two *START* words at the beginning and one *STOP* word at the end. Doing this will semantically correct because there was no token by the name *START* and *STOP* in our corpus.
- Tokens are generated in training set and development set using Punkt word tokenizer. All the tokens with frequency less than 5 are replaced with *UNK* tokens. Doing this will be semantically correct because there was no token by the name *UNK* in our corpus. This will help eliminate the case of assigning zero probabilities to a sentence.
- '\n' in the corpus was replaced with empty string ('').

Execution Steps:

- As mentioned above, the data is split into 3 parts: - Training set, Dev set and Test set. I have made 5 different dev sets and training sets by randomly shuffling the Part-1 of the corpus.

- For each dev set, two different perplexities are calculated. One using linear interpolation and other using discounting method.

- For linear interpolation, we estimate trigram probability $P(w_n|w_{n-2}w_{n-1})$ by mixing together the unigram, bigram and trigram probabilities, each weighted by a λ. In this case, we denote them by λ_1, λ_2, λ_3 for unigram, bigram and trigram respectively. We do a grid search for finding optimal λ's, with each λ ranging from [0,1] and searched incrementally with a step size of 0.1 with the help of held-out corpus(our dev set) by choosing values of λ that maximize the likelihood of the held-out corpus.

- For Discounting, we discount the higher order n-grams to save some probability mass for the lower order n-gram. Here also, we have to do a grid search on two hyperparameters β_1, β_2 which denote the discounting values for bigrams and trigrams. This is also achieved by using a held-out corpus on which the likelihood is calculated.

- Finally, after finding the optimal λ's and β's, the perplexity is calculated for the test set using both linear interpolation and discounting methods.

- Below 2 tables contains the summarization of the results produced by the code.

Iteration	λ_1	λ_2	λ_3	β_1	β_2	Likelihood Lin. Interpolation	Likelihood Discounting
1	0.9	0.1	0.0	0.6	0.7	-6193.28	-20632.58
2	0.9	0.1	0.0	0.6	0.7	-6291.90	-21058.15
3	0.9	0.1	0.0	0.6	0.7	-6550.99	-21470.01
4	0.9	0.1	0.0	0.6	0.7	-6746.48	-22187.19
5	0.9	0.1	0.0	0.6	0.7	-6010.91	-19593.31

Iteration	Perp_Dev_Interpolation	Perp_Dev_Discount	Perp_Dev_Laplace	Perp_Test_Interpolation	Perp_Test_Discount	Perp_Test_Laplace
1	36.42	20.65	339.19	34.27	20.19	354.04
2	36.98	21.44	345.97	33.98	20.17	351.99
3	36.71	21.02	369.62	34.24	19.99	349.52
4	38.76	21.61	347.66	33.65	19.97	350.70
5	36.20	21.23	344.68	34.49	20.08	352.78

Observations:

- In all 5 different train and dev datasets, the value of optimal λ's and β's are same. It may be due to very small size of the corpus used.

- The perplexity values also do not differ by a large amount, which can be explained by the small corpus size used

- Below Table shows the variance obtained by using linear interpolation, discounting and Laplace smoothing. It can be seen from the data that variance on smoothing is very close to zero while Laplace Smoothing has maximum variance. It can be seen from perplexity and variance values that Laplace smoothing is not as good a method as Linear interpolation and discounting

Smoothing Technique	Variance on Validation Set
Linear Interpolation	0.831
Discounting	0.112
Laplace Smoothing	110.03

Vector Semantics: - GLoVE

GLoVE is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. I have used Gradient Descent as an optimization method for implementing GLoVE.

Derivate expressions in Gradient Descent:

Variables Used:

1. $vocab\_size$: - No of different words in the training corpus
2. $vector\_size$: - Dimensions of word vector are (1, $vector\_size$)
3. w: - Denotes word main embeddings, dimension is ($vocab\_size$,$vector\_size$)
4. u :- Denotes word context embeddings, dimension is ($vocab\_size$,$vector\_size$)
5. b :- Bias associated with W_center
6. b' : - Bias associated with $W\_context$

The objective function (cost function) that is to be minimized is: -

$$J = \sum_{i,j=1}^{|V|} f(X_{ij})\left(w_i^{\mathrm{T}} u_j + b_i + b_j' - \log(X_{ij})\right)^2$$

On taking partial derivatives, the expressions are: -

$$\frac{\partial J}{\partial w_i} = \sum_{j=1}^{|V|} 2f(X_{ij}) J_{ij} u_j$$

$$\frac{\partial J}{\partial u_j} = \sum_{i=1}^{|V|} 2f(X_{ij}) J_{ij} w_i$$

$$\frac{\partial J}{\partial b_i} = \sum_{j=1}^{|V|} 2f(X_{ij}) J_{ij}$$

$$\frac{\partial J}{\partial b'_j} = \sum_{i=1}^{|V|} 2f(X_{ij}) J_{ij}$$

where:

$$J_{ij} = \left(w_i^{\mathrm{T}} u_j + b_i + b'_j - \log(X_{ij})\right)^2$$

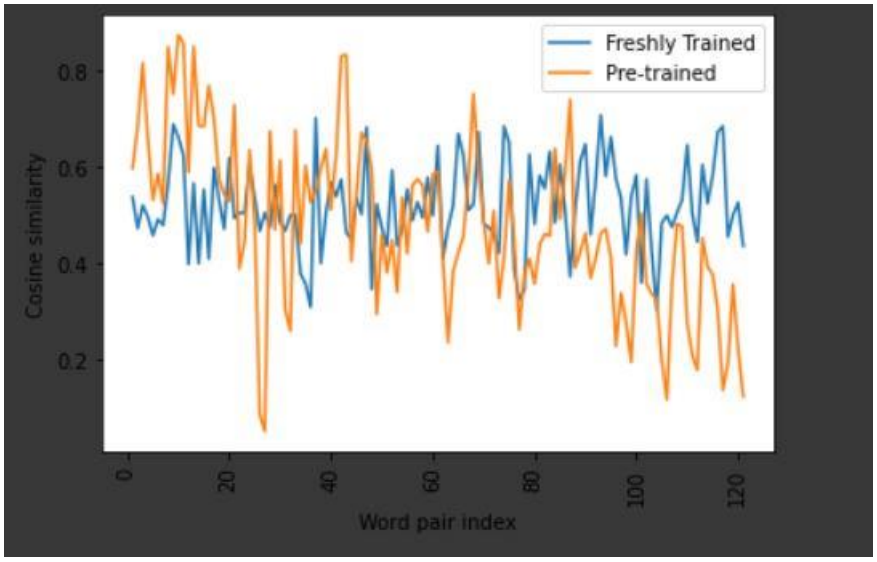
Hyperparameters:

1. Vector dimension = 100
2. No of epochs: -
3. x_{max} = 100
4. α = ¾
5. Window size = 4
6. Learning rate = 0.0005
7. Vocab size = 10,000 (learning rate depends on this)

Comparison with Pre-trained Word Embeddings:

Pre-trained word embeddings for glove were downloaded from Stanford-NLP website. I have used the word pairs from the benchmark Word Similarity 353 dataset which are present in the pre-trained as well as in my model's trained word embeddings. For finding the similarity between the two vectors, I have used the cosine similarity between them which yields answer in the range -1 to 1 with value 0 depicting highly un-correlated words. If we increase vocabulary size and no of epochs, the model results would get more close to the standard pre-trained word embeddings

Below is a plot for reference:



Note: The code was taking a lot of time to run for one epoch. So I was only able to run it for 25 epochs before the assignment deadline. I have continued running my Colab for 50 epochs, which would give a better result. Hopefully, by the time you see it, the plot would look much better. (refer to the last cell of the colab)