CS-565 INTELLIGENT SYSTEM AND INTERFACES

ASSIGNMENT - 2

-MAYANK WADHWANI (170101038)

Link for Google Colab Notebook (contains all the codes): https://bit.ly/3oZ5Xhy
Link for Google Drive that contains all the results obtained (all files): https://bit.ly/3ew2kLc

Note: To allow easy access, the Code has been indexed into sections based on question number and different part of question. So, for instance the code for GloVe implementation can be accessed directly from the corresponding section.

QUESTION 1 N-gram Language Model

DATASET CREATION

We have used **NLTK** library in our program. We use the provided provided `sent_tokenize` function to carry out sentence segmentation in our corpus.

Further we use the `TreebankWordTokenizer` to tokenize the sentences.

We have then randomly shuffled the sentences by using the `Random.shuffle` function and then partitioned in a ratio 9:1 to create the training and the testing dataset. Following this, we again randomly shuffle the training data and partition into 9:1 to create the main training set and the cross validation set. The files obtained can be accessed from the Google Drive link as provided above.

N-Gram Language Model

Linear Interpolation

First, we scan through the training corpus and add tokens **START** and **END** at the start and end of each sentences respectively. We also identify the unknown words or **OOV** (Out of Vocabulary) words in the cross validation set. A token will be identified as an UNK word if either the token is not present in the training corpus or if the frequency is less than some threshold (taken to be 5 in our program).

We have defined our cost function as given in the slides and then used the **Conjugate Gradient** or the CG optimization method to optimize this cost function. This method comes pre-included in the library `scipy`. After running the same, we get the optimal values of lambda which would give the least perplexity.

After running this algorithm on the 5 different training and cross validation sets, we obtain the following results:

Model's Performance on Cross Validation Sets

S. NO (Iteration No.)	$\lambda 1$	$\lambda 2$	$\lambda 3$	Log-Likelihood Value	Perplexity
1	0.600969	0.2106502	0.1883806	-13484491.3119	116.99939
2	0.554246	0.2872971	0.1584560	-13360820.94598	111.87576
3	0.580581	0.1957880	0.2236306	-13463741.79246	116.14516
4	0.569760	0.2532776	0.1769616	-13394025.62128	113.32048
5	0.653749	0.1982521	0.1479982	-13567823.04274	120.49377

Since the training sets may contain different frequencies of unigrams, bigrams and trigrams, the markov probability will vary. Ie. λ1 * q_ml(trigram) + λ2 * q_ml(bigram) + λ3 * q_ml(unigram) will vary. So it was expected that slight variations will occur for the values of lambda parameters. Moreover, the average perplexity came out to be 115.766912 which is quite close to the expected perplexity of 74 (which is the perplexity expected for the trigram model).

Model's Performance on Testing Sets

S. NO (Iteration No.)	$\lambda 1$	$\lambda 2$	$\lambda 3$	Log-Likelihood Value	Perplexity
1	0.600969	0.2106502	0.1883806	-13479033.63042	116.774099
2	0.554246	0.2872971	0.1584560	-13360560.99128	111.989109
3	0.580581	0.1957880	0.2236306	-13474511.87031	116.587772
4	0.569760	0.2532776	0.1769616	-13396359.82613	113.413939
5	0.653749	0.1982521	0.1479982	-13577777.07049	120.918096

As expected, the average perplexity came out to be 115.93603 ie. greater than the perplexity observed for the cross validation sets. This is due to the fact that the optimization algorithm we run (CG in our case) fits the data well for the cross validation set. The generalization error is obtained by calculating the cost on the same for the test set.

Discounting Method

Again, we add tokens START, END and UNK in a similar fashion as we did earlier.

We then apply the **Katz Back Off** Discounting method by finding values for the different alpha sets and try to minimize the beta value in the process. We iterate over different beta values from 0.1 to 0.9 and report the beta value which gives the lowest perplexity on the cross validation set.

After running this algorithm on the 5 different training and cross validation sets, we obtain the following results:

Model's Performance on Cross Validation Sets

S.No. (Iteration No.)	B	Log Likelihood Value	Perplexity
1	0.7	-11888693.63976	106.450378
2	0.7	-11907299.66852	106.755886
3	0.7	-11887430.88948	105.699969
4	0.7	-11878912.42581	105.246599
5	0.7	-11861629.50214	104.769187

Similar to the last case, the observed values for log likelihood and perplexity varies for the different cross validation sets. But it was observed for this case that the value of beta came out to be same (0.7) for all the iterations. The average perplexity came out to be 105.7844 which was less than the linear interpolation one.

Model's Performance on Testing Sets

S.No. (Iteration No.)	B	Log Likelihood Value	Perplexity
1	0.7	-11200324.93771	105.827756
2	0.7	-11196783.85228	105.695494
3	0.7	-11195150.23519	105.634533
4	0.7	-11197109.64697	105.707656
5	0.7	-11197167.63692	105.709820

As expected the average value of the perplexity (105.8151) on the independent test set came out to be slightly greater than that of in the cross validation set since the model fits the cross validation set more perfectly in that case.

DISCUSSION

Hence, it can be observed that a higher variance can be seen in the former case. Since changing the cross validation set has a slightly greater affect on the linear interpolation method as compared to the discounting one since the lambda values are seen to change much more than the beta values which in our case came out to be the same for all the different partitions.

Hence, we can conclude that the Discounting method had the least variance in our case.

Moreover, had we used the Laplace method to fit our model, we would have got a higher value for the perplexity. This is because the Laplace method doesn't take into consideration any frequencies of the context. We simply add 1 to the numerator so that it doesn't become zero. So if lets say, we have 2 different trigrams "live in Delhi" and "live in in", here lets assume that both these trigrams have zero frequency in our training set. Then in this case, the Laplace method fails to differentiate between the two trigrams which is clearly not correct since the probability of the former trigram "live in Delhi" must be higher than the latter one. In the linear interpolation example, the former one will have a higher probability since we can expect to find the bigram "in Delhi" in our corpus in contrast to the other bigram "in in".
So, to summarize the Discounting method has a **lower variance** and gave a **better** value of perplexity as compared to the linear interpolation method and if Laplace method is used, we should expect to get a **higher value** for the perplexity.

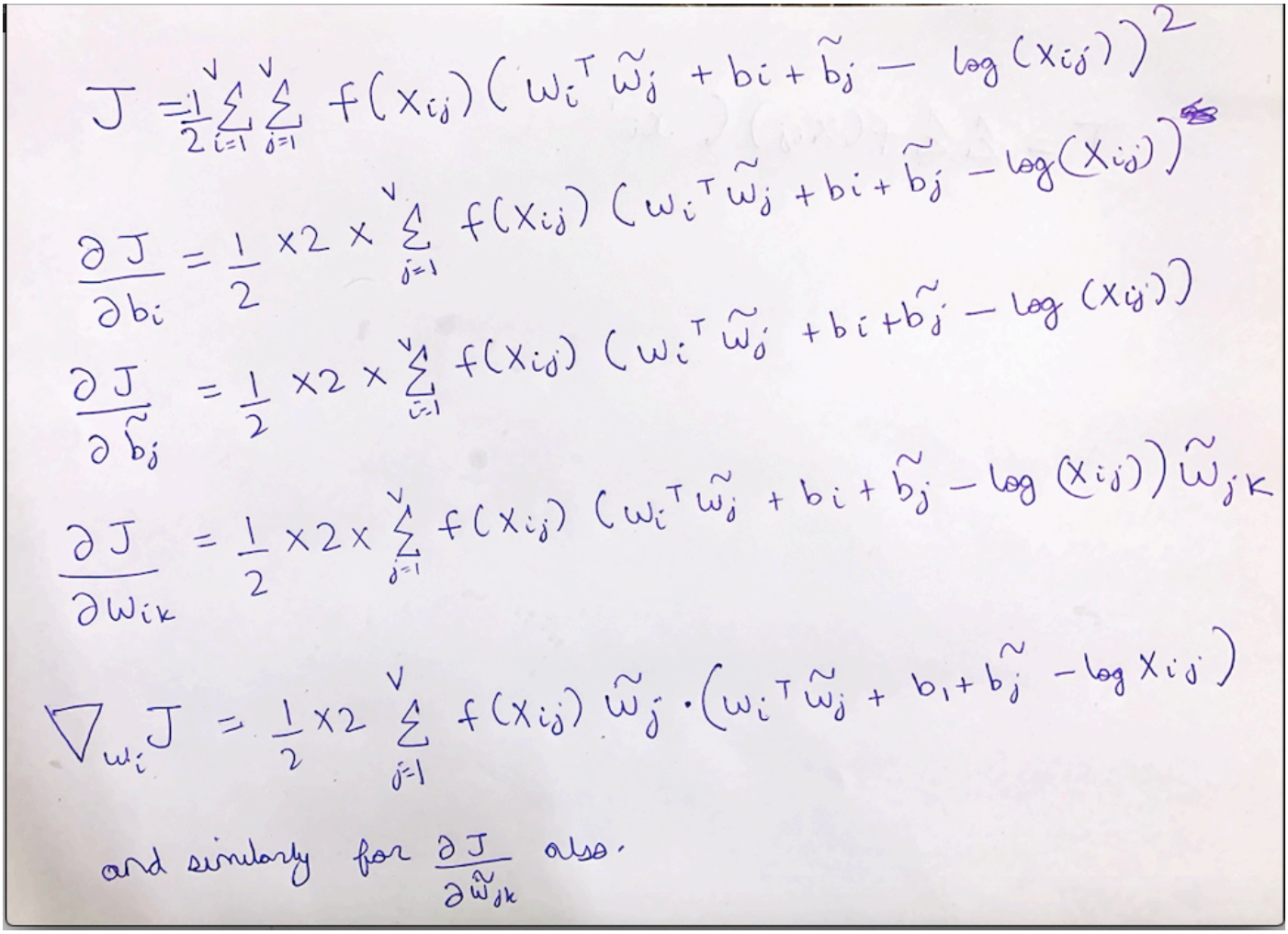
QUESTION 2 GloVe Implementation

IMPLEMENTATION

Since this is a type of unsupervised learning, splitting of the dataset into training and validation was not required. We have used the entire corpus for the implementation. The corpus contains more than 800K sentences and more than 350K tokens in the vocabulary.

To build the co-occurrence matrix, we used a context window of 6. So for each token in a sentence, we will have 6 tokens in its context. To optimize on the storage, we have used a sparse matrix instead of a standard O(V*2) size one (where V is the size of the vocabulary). So if there is an entry (u,v) in our co-occurrence matrix, we will know for sure that the frequency will not be zero (since the matrix is sparse, if it is zero, it will not be added to the dictionary in the first place).

We then initialize the vectors, we have kept the size of each vector to be 100. We initialize all vectors with a random values. This will help as if we fix the value for each vector to be same, it can be shown that gradient descent will yield the same values for the vector. So if initially if we assign all vectors with value 0, then after some time all vectors will have a same value 'x'.



NOTE: We have used **Gradient Descent** algorithm in our program. The necessary derivations as required are shown below:

Here f(Xij) is a function which is used for normalization, if Xij is very high, then it is becomes 1. So even if Xij is very high, it gets normalized as compared to other less frequent values. We have used the values of alpha as 0.75 and x_max equal to 100 to be consistent with the one presented in the original GloVe implementation.

Since we have implemented Gradient Descent from scratch, we had to choose a suitable learning rate for the model. We have chosen different learning rates for different iterations. Since initially all values are random, which suggests that we are very far from the minima, so we take bigger jumps or have a higher learning rate (taken to be **0.01** in our case). After the first 50 iterations, we start getting less and less values of the cost function which suggests that our algorithm is working correctly. We then carry out 150 more iterations with a learning rate 0.001 (10 times less than 0.01). We decrease the learning rate to avoid shooting up of the gradient descent.

After carrying out the process, we observe that the cost function value decreased from a whooping `47236438.94114` to just `76950.70383`. We have used these vectors (after 200 iterations) to carry out the tests as mentioned below.

For the tests,

The Spearman's coefficient method is used as a metric to evaluate word vector embeddings. It gives a measure on how well we can rank the data in a monotonic fashion. We tried our vectors with the Stanford 100-d vectors and obtained the below results for different types of datasets as mentioned below.

The **SIMLEX999** dataset contains 1000 data entries with a similarity score lying between 0-10. Spearman's method takes all these into account to yield a final score.

Spearman correlation on pre-trained GloVe model: 0.37050035710869067

Spearman correlation on my implementation of the GloVe model: 0.10680710798945227

The **MEN** dataset contains 300 data entries with a similarity score ranging between 0-50. On applying Spearman's coefficient, we get the following results:

Spearman correlation on pre-trained GloVe model: 0.7374646969805517

Spearman correlation on my implementation of the GloVe model: 0.43560109095570295

The **WS353** is also a similar state of the art dataset. On applying Spearman's coefficient, we get the following results:

Spearman correlation on pre-trained GloVe model: 0.5433255613304138

Spearman correlation on my implementation of the GloVe model: 0.31462905671095203

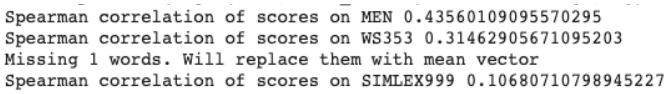


Figure 2 Spearman values obtained on my model

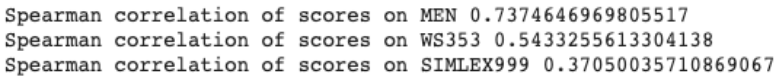


Figure 1 Spearman values obtained on pre trained model

The values are quite close to each other. The difference in the values maybe due to the following reasons:

1) Selection of a larger dataset-> The Stanford Model was trained on a much larger dataset containing more sentences. So evidently, if the number of sentences is large, then the context meaning for each token will be large.
2) Convergence-> Our model after running 200 iterations yielded a cost of around 75K which could still be lower. So if we run more iterations on our model, our cost would become lower fitting the given dataset more properly which will give a better Spearman's score.