CSE 538 NLP Assignment - 1 Report
114707888, Shiva Sankeerth Reddy Yarradla

Question-1: Code Implementation Explanation:

Batch Generation: (data.py)

1. We start from the leftmost center word using the **data_index** variable.
2. To keep track of the number of words in the batch so far, we create a variable **words_in_batch**.
3. We iterate through the data to extract all possible context-words for each center-word while the **words_in_batch** is less than the defined batch_size.
4. For each iteration, we create a temporary virtual window containing context-words using the **skip_window** parameter around the center-word.
5. We randomly sample the context-words within the temporary window using the **num_skips** parameter.
6. For all the context words extracted from the above step, we assign their respective center-word using the **data_index**.
7. We update our data_index to move right using the **stride.**

Loss Functions: (model.py)

1) Negative Log-Likelihood:
   a) The representations of center and context words are gathered using their respective embeddings as U and V respectively.
   b) The Log-Likelihood can be computed using the formula: **sum{(u_w)^T v_c} - log(\sum{exp({u_w}^T v_c)})**.
   c) The Negative Log-Likelihood can be obtained by applying negative to the above formula, effectively converting the equation into **log(sum{exp({u_w}^T v_c)}) - sum{(u_w)^T v_c}**.
   d) I divided the computation into two parts, part_a and part_b.
   e) part_a = **sum{(u_w)^T v_c},** here we element-wise multiply both U and V vectors and compute their sum.
   f) part_b = **log(\sum{exp({u_w}^T v_c)}),** here we take the transpose of the U vector and do an element-wise multiplication between both U.T and V. Then we take the resulting tensor and apply exponential of the elements and compute their sum. In the end, we apply the log function to the result.
   g) The final loss is the mean of (part_b - part_a).

2) Negative Sampling:
   a) The representations of center and context words are gathered using their respective embeddings.
   b) Now, we arbitrarily fix how many **negative samples (K)** to draw and pick them from a unigram distribution with a size of (batch_size, K)
   c) We use these samples to get the representations using the center embeddings for the respective negative sample. Let's call it neg_U.
   d) Here, I vectorized the representation of the center words to make the computation faster.
   e) The loss can be computed using the following formula: **-log{sigmoid(sum{(u_w)^T v_c})} - sum{log(sigmoid(sum{(neg_U)^T u_w}))}.**

f) Here again, I broke down the equation into two parts to ease the computation.
g) part_a = **log{sigmoid(sum{(u_w)^T v_c})}**, here we perform element-wise multiplication of both U and V vectors and compute their sum. Then we apply the sigmoid function over the result and take a logarithm of it.
h) part_b = **sum{log(sigmoid(sum{(neg_U)^T u_w}))}**, here we element-wise multiply the neg_U and U vectors and take their sum. Then the resultant tensor is passed over to sigmoid function and then into an logarithmic function. Finally a sum is calculated on the tensor.
i) The final loss is the mean of (-(part_a + part_b)).

evaluate_pairs: (word_analogy.py)

1. I iterated over each line of the text file indirectly using the embeddings.
2. In each line, we have examples represented as candidate embeddings and choices represented as test embeddings.
3. For each candidate embedding in the line, I calculated the difference between the pair of words and append it to a list, **diff_examples**.
4. For each test embedding in the line, I calculate the difference between the pair of words and append it to a list, **diff_choices**.
5. I calculate the cosine similarity between **diff_examples** and **diff_choices**.
6. I calculate the average similarity and compute the minimum similarity and maximum similarity to get the least illustrative and most illustrative words.
7. Append them to the best_pairs and worst_pairs lists.

Question 2: Hyper-parameters explored and their effect on training

1. **Batch Size**: It is the number of training examples to use in one iteration. I have experimented with batch sizes of 64, 128, and 256. Using 256 as the batch size has been shown to increase the time required to complete the training process.
2. **Vocab Size**: It is the size of vocabulary for the model. I have experimented with the vocabulary sizes of 100000 and 150000. I could not experiment with higher values because the more the vocab size, the more is the amount of memory required to store the embeddings. I could observe a significant difference in training times for both vocab sizes.
3. **Skip Window**: It is the number of words to consider left and right from a context word. I have experimented with the following values, 1 and 2.
4. **Number of Skips**: It is the number of samples to draw from a given window. I have experimented with the values of 2 and 4. Together with the Skip window parameter, this parameter decides the number of context words to pick which in turn affects the time to train the model.
5. **Max Number of Steps**: It is the maximum number of steps to train the model. The more the number of steps, the more time is required to train the model. Together with the large-batch size and embedding size, I have observed that the training time has increased exponentially.
6. **Embedding Size**: It is the vector size for representing the words in the text. The more the embedding size, the more the time required to train the model as each word is represented with larger vectors. This means that the time taken for computing increases internally.
7. **Number of Negative Samples (K)**: It is the number of negative samples to choose from the corpus. This parameter affects the Negative Sampling loss function. I have experimented with the values of 2, 3, 4, and 5. I have observed that for larger values of K, the time required to train the model has significantly increased.

Question 3: Results

1. **Batch Size**: For the model using NLL loss function, increasing the batch size from 64 to 256 has been shown to increase the performance to the value of 128. After that, there was much improvement. I couldn't explore other larger batch size values because of the compute constraints. For the Negative Sampling model, increasing the batch size decreased the performance in terms of loss values. But at other higher values like 256, the model has shown significant improvement.
2. **Vocab Size**: For both NLL and Negative sampling models, an increase or decrease in vocab size hasn't shown any significant change in performance. But I observed an exponential increase in training time for larger vocab sizes.
3. **Skip Window** and **Number of Skips**: When the number of words taken out of the context word window is smaller than the skip window, it did not seem to produce better results. I was not able to experiment much on the Negative sampling model because of the compute constraints. But within the set of experiments I did, I could confirm that varying these parameters has some effect on the performance but to minimal effect.
4. **Max Number of Steps**: I have observed that the number of iterations we are training the model has a significant role in the performance. When observed from the first row, fourth row, and last row of table-1, the lower the number of steps, the higher the loss. It might mean that, if we give enough iterations for the model to learn, it can understand the data better.
5. **Embedding Size**: For both NLL and Negative Sampling models, I have experimented with embedding sizes ranging from 128 to 512. For the NLL model, higher values of this parameter meant huge training times but with no significant improvement. But for the Negative sampling model, a higher embedding size produced one of my best models.
6. **Number of Negative Samples (K)**: I was of the view that increasing the number of negative samples will initially increase the performance in terms of accuracy and then eventually decrease it because more the words that sampled in random to be negative words, it might start choosing more than a few contextual ones as well.

Negative Log-Likelihood:

Batch Size	vocab size	Skip Window	Num_Skips	max_num_steps	embedding_size	Least Illustrative	Most Illustrative	Overall Accuracy	Final Loss
64	100000	1	2	200001	128	37%	33.90%	35.40%	3.719
64	50000	2	4	200001	128	35.60%	32.40%	34%	4.016
128	100000	6	2	200001	128	36.10%	36.50%	36.3	4.676
128	100000	4	2	250001	128	34.70%	36.90%	35.80%	4.669
128	100000	2	4	250001	256	35.80%	34.80%	35.30%	4.662
128	150000	2	2	250001	256	36.10%	32.60%	34.40%	4.634
256	150000	1	2	150001	256	37.1%	33.5%	35.3%	4.994
256	100000	2	4	100001	512	36.3%	34.1%	35.2%	5.352

Table 1: Negative Log-Likelihood

Negative Sampling:

Batch Size	vocab size	Skip Window	Num_Skips	max_num_steps	embedding_size	Number of Negative Samples (K)	Least Illustrative	Most Illustrative	Overall Accuracy	Final Loss
64	100000	1	2	200001	128	5	41.5%	33.5%	37.6%	2.071
64	50000	2	2	150001	128	2	37.2%	33.5%	35.3%	2.326
64	100000	1	4	200001	256	3	34.4%	35.6%	35%	3.775
128	150000	2	4	150001	256	4	39.70%	34.90%	37.30%	2.072
256	150000	2	4	150001	128	4	38.2%	34.8%	36.5%	2.081

Table 2: Negative Sampling

Question - 4: Bias Score Comparison

1. For the EuropeanAmerican_AfricanAmerican_Pleasant_Unpleasant task, the score for NLL model is -0.34246543 while for the Neg Sampling Model it was 0.6782551. This shows the NLL model has a negative correlation between European Americans and Pleasantness, while the Negative Sampling Model does the opposite. The behavior of NLL model comes as a surprise to me, but also proves the naiveness of the model.

2. For the Flowers_Insects_Pleasant_Unpleasant task, the score for NLL model is -0.22720304 while for the Neg Sampling Model it was -0.030644583. This shows that both the models show a negative correlation between the flowers and pleasantness.

3. For the Male_Female_Career_Family task, the score for NLL model is 0.30684277 while for the Neg Sampling Model it was -0.489711. Here, according to the NLL model, it shows that the male are more likely to do career-related tasks while female are more suited for family tasks. The Neg model shows the opposite.

4. For the Math_Arts_Male_Female task, the score for NLL model is 0.59480506 while for the Neg Sampling Model it was -0.13185927. Here the NLL model shows a positive correlation between Male and math while Neg model shows a negative relationship between the attributes given.

5. For the MusicalInstruments_Weapons_Pleasant_Unpleasant task, the score for the NLL model is -0.032272812 while for the Neg Sampling Model it was -0.19972871. Here the NLL model thinks that Musical instruments are less pleasant compared to weapons. This can be attributed to noise in the data as the Neg model also shows a similar correlation.

6. For the Science_Arts_Male_Female task, the score for NLL model is 0.1401295 while for the Neg Sampling Model it was -0.096533515. Here, the NLL model shows a positive correlation between the male and science terms, showing an inherent bias within the model, while the Neg sampling model doesn't show this bias.