Question 1 Linear Embedding - GLoVE.

(a) *Given the vocabulary size V and embedding dimensionality d, how many trainable parameters does the GLoVE model have?*

$$V(d+1)$$

(b) *Write the gradient of the loss function with respect to one parameter vector w_i.*

$$\begin{aligned}\frac{\partial L}{\partial w_i} &= 4(w_i^T w_i + 2b_i - \log X_{ii})w_i + 2\sum_{j=1,j\neq i}^{V} 2(w_i^T w_j + b_i + b_j - \log X_{ij})w_j \\ &= 4(w_i^T w_i + 2b_i - \log X_{ii})w_i + \sum_{j=1,j\neq i}^{V} 4(w_i^T w_j + b_i + b_j - \log X_{ij})w_j \\ &= \sum_{j=1}^{V} 4(w_i^T w_j + b_i + b_j - \log X_{ij})w_j\end{aligned}$$

Or vectorized format,

$$\frac{\partial L}{\partial w} = 4(WW^T + b\mathbb{1}^T + \mathbb{1}b^T - \log X)W$$

(c) *Implement the gradient update of GLoVE in language model.ipynb.*

(d) *Train the model with varying dimensionality d. Which d leads to optimal validation performance? Why does / doesn't larger d always lead to better validation error?*

$d = 12$ (with learning rate 0.2) leads to the optimal validation performance.
Larger d doesn't always lead to better validation error.
As d increases, the model over-fits, causing more validation error.

Question 2 Network architecture.

(a) *As above, assume we have 250 words in the dictionary and use the previous 3 words as inputs. Suppose we use a 16-dimensional word embedding and a hidden layer with 128 units. The trainable parameters of the model consist of 3 weight matrices and 2 sets of biases. What is the total number of trainable parameters in the model? Which part of the model has the largest number of trainable parameters?*

For Word Embedding layer, there are $250 \times 16 = 4,000$ weight parameters. For Hidden layer, there are $48 \times 128 = 6,144$ weight parameters and 128 bias parameters. For Output layer, there are $128 \times 250 = 32,000$ weight parameters and 250 bias parameters. Totally, $42,144$ weight parameters and 378 bias parameters. The Output layer has the largest number of trainable parameters.

(b) *Another method for predicting the next word is an n-gram model, which was mentioned in Lecture 7. If we wanted to use an n-gram model with the same context length as our network, we'd need to store the counts of all possible 4-grams. If we stored all the counts explicitly, how many entries would this table have?*

To store all the counts of 4-grams explicitly, we will get a table with $250^4 = 3,906,250,000$ entries.

Question 3 Training the Neural Network.

Write the output of print_gradients().

```
loss_derivative[2, 5] 0.001112231773782498
loss_derivative[2, 121] -0.9991004720395987
loss_derivative[5, 33] 0.0001903237803173703
loss_derivative[5, 31] -0.7999757709589483

param_gradient.word_embedding_weights[27, 2] -0.27199539981936866
param_gradient.word_embedding_weights[43, 3] 0.8641722267354154
param_gradient.word_embedding_weights[22, 4] -0.2546730202374648
param_gradient.word_embedding_weights[2, 5] 0.0

param_gradient.embed_to_hid_weights[10, 2] -0.6526990313918258
param_gradient.embed_to_hid_weights[15, 3] -0.13106433000472612
param_gradient.embed_to_hid_weights[30, 9] 0.118467746181694
param_gradient.embed_to_hid_weights[35, 21] -0.10004526104604389

param_gradient.hid_bias[10] 0.2537663873815643
param_gradient.hid_bias[20] -0.03326739163635357

param_gradient.output_bias[0] -2.0627596032173052
param_gradient.output_bias[1] 0.0390200857392169
param_gradient.output_bias[2] -0.7561537928318482
param_gradient.output_bias[3] 0.21235172051123635
```

Question 4 Training the Neural Network.

(a) *Use the model to predict the next word. Does the model give sensible predictions? Try to find an example where it makes a plausible prediction even though the 4-gram wasn’t present in the dataset (raw_sentences.txt).*

For ’government of united’, the predictions in the order of probabilities are ’.’, ’life’, ’own’.
For ’city of new’, the predictions in the order of probabilities are ’york’, ’.’, ’home’.
For ’life in the’, the predictions in the order of probabilities are ’world’, ’united’, ’game’.
For ’he is the’, the predictions in the order of probabilities are ’best’, ’same’, ’way’.

I tried a tri-gram, ’where’, ’might’, ’you’, and found "where might you" did not occur in the training set. The predictions made by the model in probability order are ’go’, ’be’, ’come’.
This is a reasonable prediction.

(b) *Plot the 2-dimensional visualization using the method tsne_plot_representation. Look at the plot and find a few clusters of related words. What do the words in each cluster have in common? Plot the 2-dimensional visualization using the method tsne_plot_GLoVE_representation for a 256 dimensional embedding. How do the t-SNE embeddings for both models compare? Plot the 2-dimensional visualization using the method plot_2d_GLoVE_representation. How does this compare to the t-SNE embeddings? (You don’t need to include the plots with your submission.)*

I listed several clusters in **tsne_plot_representation** here:
do, did, does; can, could, will, should, may, might; say, says, said
The words in each cluster here are grouped together according to their perporties, such as numbers, auxiliary verbs.

Question 4 [(b)] continued on next page. . .

I also listed several clusters in **tsne_plot_GLoVE_representation** here:
new, york, city; several times; every day; ...
The adjacent words in each cluster here are grouped together.

Compared with the GLoVE word embedding model, the trained word embedding model uses the properties of words to cluster them, while the GLoVE considers their adjacency.

2d-GLoVE is an underfitting phenomenon. The white space (with no point) in the 2d-GLoVE figure using t-SNE is more than without t-SNE, indicating that t-SNE can effectively aggregate similar/near points in the original representation.

(c) *Are the words 'new' and 'york' close together in the learned representation? Why or why not?*

No.

(d) *Which pair of words is closer together in the learned representation: ('government', 'political'), or ('government', 'university')? Why do you think this is?*