CSC413 Assignment 1

Chen Liang

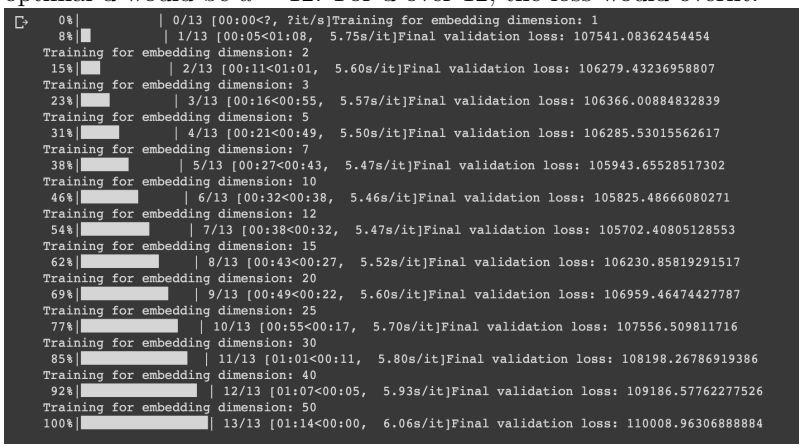
January, 2020

Part 1

1. Given the vocabulary size V and embedding dimensionality d, how many trainable parameters does the GLoVE model have?
   Answer: Vd + V. Reason: There are V words, and for each w_i it has d parameters and for each b_i it has one parameter, so the total trainable parameters equals to $V(d+1) = Vd + V$.

2. Write the gradient of the loss function with respect to one parameter vector w_i.
   Answer: $\frac{\partial L}{\partial w_i} = 4 \cdot (w_i^T w_j + b_i + b_j - logX_{ij})w_j$

3. Train the model with varying dimensionality d. Which d leads wto optimal validation performance? Why does / doesn't larger d always lead to better validation error? When learning rate at 0.2, the optimal d would be $d = 12$. For d over 12, the loss would overfit.



Part 2

1. 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?
   Answer: Input to Embedding Layer: total words = 250, features of each word = 16, so the weights of matrix = $250 \times 16 = 4000$
   Embedding to Hidden Layer: number of embeddings for hidden layer = 3, features of each word = 16, units = 18, so the weights of matrix = $3 \times 16 \times 128 = 6144$
   hidden to output layer: hidden units = 128, words for softmax = 250, weights of matrix = $128 \times 250 = 32000$.
   total trianable parameters = $4000 + 6144 + 128 + 250 + 32000 = 42522$

2. 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?
Answer: We have 250 words for 4-grams, so the entries would be 250^4

Part 3

```
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] -0.6526990313918257
param_gradient.embed_to_hid_weights[15, 3] -0.13106433000472612
param_gradient.embed_to_hid_weights[30, 9] 0.11846774618169399
param_gradient.embed_to_hid_weights[35, 21] -0.10004526104604386

param_gradient.hid_bias[10] 0.25376638738156426
param_gradient.hid_bias[20] -0.03326739163635369

param_gradient.output_bias[0] -2.062759603217304
param_gradient.output_bias[1] 0.03902008573921689
param_gradient.output_bias[2] -0.7561537928318482
param_gradient.output_bias[3] 0.21235172051123635
```

Part 4

1. Pick three words from the vocabulary that go well together (for example, 'government of united', 'city of new', 'life in the', 'he is the' etc.). 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*). To help you out, the function *find_occurrences* lists the words that appear after a given 3-gram in the training set.

```
[ ]  1 trained_model.predict_next_word("he", "is", "the")
```

```
he is the best Prob: 0.30298
he is the same Prob: 0.09604
he is the way Prob: 0.06930
he is the only Prob: 0.04856
he is the man Prob: 0.04572
he is the first Prob: 0.04371
he is the right Prob: 0.02787
he is the president Prob: 0.02374
he is the law Prob: 0.01761
he is the other Prob: 0.01701
```

```
[ ]  1 trained_model.predict_next_word("city", "of", "new")
```

```
city of new york Prob: 0.99444
city of new . Prob: 0.00150
city of new home Prob: 0.00031
city of new world Prob: 0.00026
city of new , Prob: 0.00025
city of new life Prob: 0.00025
city of new place Prob: 0.00024
city of new to Prob: 0.00017
city of new year Prob: 0.00015
city of new ? Prob: 0.00013
```

I think most of the predictions are sensible. (such as "he is the best/same/way/only/man/first/right/president/other".
The below example is a 4-gram wasn't present in the dataset.

```
[30]  1 find_occurrences('i', 'may', 'never')
      2 trained_model.predict_next_word('i', 'may', 'never')
```

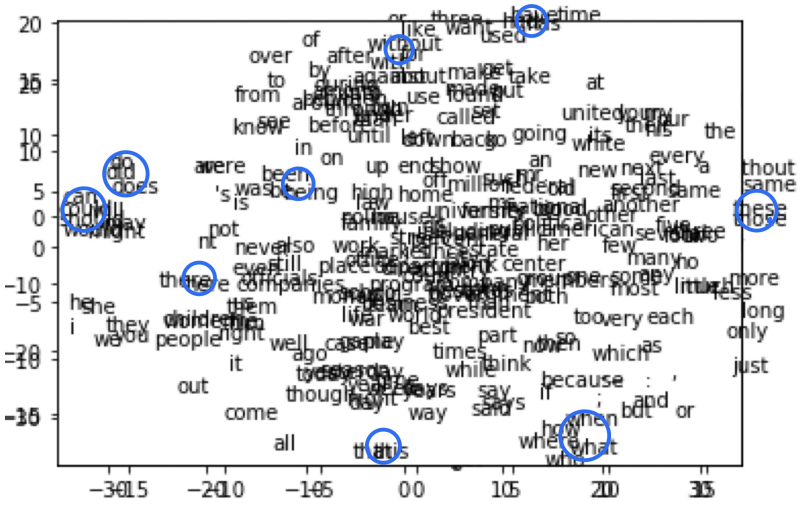
```
The tri-gram "i may never" was followed by the following words in the training set:
    have (1 time)
    get (1 time)
i may never come Prob: 0.14235
i may never say Prob: 0.14058
i may never go Prob: 0.13909
i may never be Prob: 0.09647
i may never been Prob: 0.06991
i may never do Prob: 0.06510
i may never get Prob: 0.03578
i may never have Prob: 0.03258
i may never play Prob: 0.01949
i may never see Prob: 0.01947
```

The predictions "I may never come/say/go/be/been/do/get/have/play/see" are all possible.

2. 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.)

   Answer:

   (a) Examples of clusters: {when, how, where, what who}, these are words to raise questions.{do, does, did}, {has, had, have}, these are different tenses of a verb.



   (b) **tsne_plot_GLoVE_representation** has a non-linear activation function, and it introduces co-occurrence matrix in the word embedding, so it shows more about the co-occurrence of two words. **tsne_plot_representation** has a linear activation function, so it has a stronger pattern showing the grammatical clustering over co-occurrence of two words, so its plot shows a grammatical relationship among different words.

   (c) From the graph plotted by *plot_2d_GLoVE_representation* we could observe that most of the data are crowded at the top right corner, and similar words are not grouped, and the reason could be explained as a problem of underfitting. This could be observed from the loss over embedding dimension graph plotted above, when $d = 2$, the loss is relatively high due to underfitting.

3. Are the words 'new' and 'york' close together in the learned representation? Why or why not?
   Answer: No. The distance between 'new' and 'york' is 4.049. Observe either 'new' and 'york' appeared in each other's the top 10 nearest words. Also, for the word 'new', the nearest word is 'old', which has a distance around 2.827, and for the word 'york', its closest word 'city' only has a distance around 0.830. In this way, 'new' and 'york' are not close to each other.

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

```
[31]  1 trained_model.word_distance ("government", "political")

      1.4759537372356153

[32]  1 trained_model.word_distance ("government", "university")

      1.1895426605899926
```

from the word we observe that goverment and university has a close distance. The reason might be both 'government' and 'university' are nouns while 'political' is an adjective. So the first two words have closer grammatical relationships, and they are closer.