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?

   Solution. □

   Then dimension of weight matrices and biases are

   $$\underset{250\times16}{word\_embedding\_weights} \qquad \underset{128\times48}{embed\_to\_hid\_weights} \qquad \underset{128\times1}{hid\_bias}$$

   $$\underset{250\times128}{hid\_to\_outputs\_weights} \qquad \underset{250\times1}{output\_bias}$$

   Total number of trainable parameter is

   $$250 \times 16 + 128 \times 48 + 128 \times 1 + 250 \times 128 + 250 \times 1 = 42522$$

   The model has largest number of trainable parameters in matrix **hid_to_outputs_weights**

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, wed need to store the counts of all possible 4-grams. If we stored all the counts explicitly, how many entries would this table have?
   Context length is 3. The size of conditional probability table of a 4-grams model is given by

   $$250 \times 250 \times 250 \times 250 = 3906250000$$

3. A printout of your code for the two methods you had to implement: `Model.compute_loss_derivative` and `Model.back_propagate`. Please do not include any other code.

```
def compute_loss_derivative(self, output_activations, expanded_target_batch):
    return output_activations - expanded_target_batch
```

```
############################   YOUR CODE HERE  ##############################
hid_to_output_weights_grad = np.dot(loss_derivative.T, activations.hidden_layer)
embed_to_hid_weights_grad = np.dot(hid_deriv.T, activations.embedding_layer)

output_bias_grad = np.dot(loss_derivative.T, np.ones(input_batch.shape[0]))
hid_bias_grad = np.dot(hid_deriv.T, np.ones(input_batch.shape[0]))
#############################################################################
```

4. The output of the function `checking.print_gradients`.

```
In [23]: checking.print_gradients()

loss_derivative[2, 5] 0.0013789153741
loss_derivative[2, 121] -0.999459885968
loss_derivative[5, 33] 0.000391942483563
loss_derivative[5, 31] -0.708749715825

param_gradient.word_embedding_weights[27, 2] -0.298510438589
param_gradient.word_embedding_weights[43, 3] -1.13004162742
param_gradient.word_embedding_weights[22, 4] -0.211118814492
param_gradient.word_embedding_weights[2, 5] 0.0

param_gradient.embed_to_hid_weights[10, 2] -0.0128399532941
param_gradient.embed_to_hid_weights[15, 3] 0.0937808780803
param_gradient.embed_to_hid_weights[30, 9] -0.16837240452
param_gradient.embed_to_hid_weights[35, 21] 0.0619595914046

param_gradient.hid_bias[10] -0.125907091215
param_gradient.hid_bias[20] -0.389817847348

param_gradient.output_bias[0] -2.23233392034
param_gradient.output_bias[1] 0.0333102255428
param_gradient.output_bias[2] -0.743090094025
param_gradient.output_bias[3] 0.162372657748
```

5. Pick three words from the vocabulary that go well together. Use the model to predict the next word. Does the model give sensible predictions?
   The model does give sensible predictions. Especially with `city of new`, `york` is the most probable output with $p = 0.99$. Even without appearing in the training dataset, `government of united` admits a reasonable prediction where `states` is third most probable prediction, however with low confidence.

6. Plot the 2-dimensional visualization using the method `Model.tsne_plot`. Look at the plot and find a few clusters of related words. What do the words in each cluster have in common?
   As an example, `what, who, where, when, how` and `can, do, does, will, should` form their respective clusters. It seems that each cluster consists of words that function in similar ways grammatically. More specifically, words in a cluster can oftern be substituted with other words in the same cluster without rendering the sentence from being grammatically incorrect.

7. Are the words `new` and `york` close together in the learned representation? Why or why not?

`model.word_distance("new", "york")` outputs 4.14. Calling `display_nearest_words` on both `new` and `york` showed that they are not $k = 10$ nearest words of each other. They are not close together in the learned representation. They dont seem to be close together in tsne plot. It's reasonable since they serve different grammatical function, one being an adjetive and the other a noun.

8. Which pair of words is closer together in the learned representation: (`government, political`), or (`government, university`)? Why do you think this is?
(`government, university`) because they are often substitutable while maintaining grammatical correctness, both are noun and functions similarly in a sentence.