Name Surname: Duygu Serbes
Student Number: 2020700171

CMPE 597 Sp. Tp. Deep Learning
Spring 2021 Project I
Due: April 30 by 11.59pm

Answers

1. (15 pts) Write the mathematical expressions for forward propagation.

   The model consists of one embedding layer, one hidden layer, and one output layer. The model architecture can be seen on Figure 1.

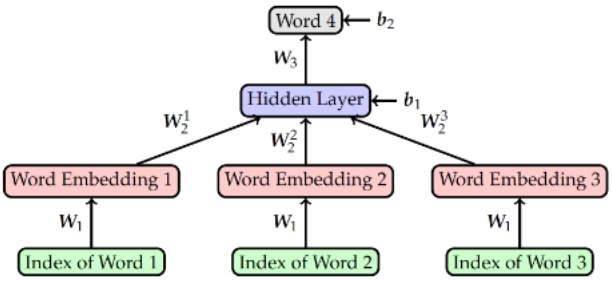

Figure 1: Network arhitecture

Embedding Layer:

The inputs of neural network consist of index of three words which are given in the form of $[batchsize \times 3]$ matrix. Each word index is transformed to one-hot vector in the shape of $[bathsize \times 250]$ represented as I_X before given as an input to the embedding layer which is 16 dimensional. In the embedding layer each 250 dimensional one-hot vector is multiplied with weights, w_1, without adding any bias. E_{preX} represents embedding matrix, here X means order of word in the input sequence. After constructing E_{preX}, the three matrix concatenated horizontally and it results $[batchsize \times 48]$ matrix, E.

$$\underset{batchsize\times 16}{E_{pre1}} = \underset{batchsize\times 250}{I_1} \times \underset{250\times 16}{W_1} \tag{1}$$

$$\underset{batchsize\times 16}{E_{pre2}} = \underset{batchsize\times 250}{I_2} \times \underset{250\times 16}{W_1} \tag{2}$$

$$\underset{batchsize\times 16}{E_{pre3}} = \underset{batchsize\times 250}{I_3} \times \underset{250\times 16}{W_1} \tag{3}$$

$$\underset{batchsize\times 48}{E} = [E_{pre1}\ E_{pre2}\ E_{pre3}] \tag{4}$$

Hidden Layer:

Where the embedding matrix multiplied with W_2 and bias, B_1, added element-wisely which provides broadcasting for matrix which have different dimensions[1]. In that layer sigmoid activation function, σ, is used to get outputs of hidden layer, represented as H.

$$\underset{batchsize\times 128}{H} = \sigma(\underset{batchsize\times 48}{E} \times \underset{48\times 128}{W_2} + \underset{1\times 128}{B_1}) \tag{5}$$

$$\sigma(x) = \text{sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{6}$$

Output Layer:

Where the outputs of hidden layer multiplied with 250 dimensional weight, W_3, and bias, B_2, added element wisely . In that layer, softmax activation function is used to get probabilities, O, of each 250 words found in the vocab.

$$\underset{batchsize\times 250}{O} = \text{softmax}(\underset{batchsize\times 128}{H} \times \underset{128\times 250}{W_3} + \underset{1\times 250}{B_2}) \tag{7}$$

$$\text{softmax}(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{8}$$

2. (15 pts) Write the mathematical expressions of the gradients that you need to compute in backward propagation.

   The formulas 9 and 10 are valid for update of W_1, W_2, W_3. Here, w contains all these weight matrices. We update the weight matrix by taking derivative of loss function with respect to that weight. η means learning rate. For all weight updates chain rule is used while finding error rates of each layer.

$$w(t+1) = w(t) - \eta g_t \tag{9}$$

$$\eta g_t = \nabla J(w(t)) \tag{10}$$

   Backward propagation of output layer (W_3 Update and B_2 Update):

   δ^L means error rate at level L, which is the output layer. Here Y is the matrix of one-hot representation of actual target classes in the mini-batch. $a^{(2)}$ is the weighted input matrix of softmax activation function. Derivation of softmax function can be found in [2].

$$\delta^L = \frac{\partial J}{\partial O}\frac{\partial O}{\partial a^{(2)}} = \frac{1}{n}(O - Y) \tag{11}$$

   W_3 Update:

$$\frac{\partial J}{\partial W_3} = \frac{\partial J}{\partial O}\frac{\partial O}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial W_3} \tag{12}$$

$$\frac{\partial J}{\partial W_3} = H^T \times \delta^L = (\underset{128\times batchsize}{H^T} \times \frac{1}{n}(\underset{batchsize\times 250}{O} - \underset{batchsize\times 250}{Y})) \tag{13}$$

$$W_3 = W_3 - \eta\frac{\partial J}{\partial W_3} \tag{14}$$

B_2 Update: In B_2 update, δ^L columns should be summed up.

$$\frac{\partial J}{\partial B_2} = \frac{\partial J}{\partial O}\frac{\partial O}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial B_2} = \sum_{i=1}^{n}[\underset{1\times 250}{\frac{\partial J}{\partial O}\frac{\partial O}{\partial a^{(2)}}}]_{ij} \tag{15}$$

$$B_2 = B_2 - \eta\frac{\partial J}{\partial B_2} \tag{16}$$

Backward propagation of hidden layer (W_2 Update and B_1 Update): δ^2 represents error rate at layer 2, hidden layer. Here $a^{(1)}$ is weighted input matrix of sigmoid activation function.

$$\delta^2 = \frac{\partial J}{\partial O}\frac{\partial O}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial H}\frac{\partial H}{\partial a^{(1)}} = \delta^L\frac{\partial a^{(2)}}{\partial H}\frac{\partial H}{\partial a^{(1)}} \tag{17}$$

$$\delta^2 = \delta^L \times W_3^T \odot \sigma'(H) = ((\underset{batchsize\times 250}{\delta^L} \times \underset{250\times 128}{W_3^T}) \odot \underset{batchsize\times 128}{\sigma'(H)}) \tag{18}$$

$$\sigma'(H) = (1 - \sigma(H)) \odot \sigma(H) \tag{19}$$

W_2 Update:

$$\frac{\partial J}{\partial W_2} = \frac{\partial J}{\partial O}\frac{\partial O}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial H}\frac{\partial H}{\partial a^{(1)}}\frac{\partial a^{(1)}}{\partial W_2} = \delta^2\frac{\partial a^{(1)}}{\partial W_2} \tag{20}$$

$$\frac{\partial J}{\partial W_2} = E^T \times \delta^2 = \underset{48\times batchsize}{E^T} \times \underset{batchsize\times 128}{\delta^2} \tag{21}$$

$$W_2 = W_2 - \eta \frac{\partial J}{\partial W_2} \tag{22}$$

B_1 Update: In B_1 update, the error rate matrix columns, δ^2, should be summed.

$$\frac{\partial J}{\partial B_1} = \delta^2 \frac{\partial a^{(1)}}{\partial B_2} = \sum_{i=1}^{n} \underset{1\times 128}{[\delta^2]_{ij}} \tag{23}$$

$$B_1 = B_1 - \eta \frac{\partial J}{\partial B_1} \tag{24}$$

Backward propagation of embedding layer (W_1 Update): δ^1 represent the error rate in the embedding layer, it consists of 3 sub-parts, each part belongs to different words. Therefore, we splits the δ^1 vertically and use these sub-parts to calculate gradient of W_1.

$$\delta^1 = \frac{\partial J}{\partial O}\frac{\partial O}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial H}\frac{\partial H}{\partial a^{(1)}}\frac{\partial a^{(1)}}{\partial E} = \delta^L \frac{\partial a^{(2)}}{\partial H}\frac{\partial H}{\partial a^{(1)}}\frac{\partial a^{(1)}}{\partial E} = \delta^2 \frac{\partial a^{(1)}}{\partial E} \tag{25}$$

$$\underset{batchsize\times 48}{\delta^1} = \underset{batchsize\times 128}{\delta^2} \times \underset{128\times 48}{W_2^T} \tag{26}$$

W_1 Update:

$$\frac{\partial J}{\partial W_1} = \delta^1 \frac{\partial E}{\partial W_1} \tag{27}$$

$$\underset{batchsize\times 48}{\delta^1} = [\underset{batchsize\times 16}{\delta_1^1} \quad \underset{batchsize\times 16}{\delta_2^1} \quad \underset{batchsize\times 16}{\delta_3^1}] \tag{28}$$

$$\frac{\partial J}{\partial W_1} = \sum_{i=1}^{3} \underset{250\times batchsize}{I_i^T} \times \underset{batchsize\times 16}{\delta_i^1} \tag{29}$$

$$W_1 = W_1 - \eta \frac{\partial J}{\partial W_1} \tag{30}$$

3. (20 pts) Implement a Network class, `Network.py`, where you have the forward, backward propagation, and the activation functions. Use matrix-vector operations.

   Please visit the code `Network.py`.

Table 1: Training and Validation Accuracy

Learning Rate	Batch Size	Training Accuracy	Training Top 3 Accuracy	Validation Accuracy	Validation Top 3 Accuracy
0.1	32	0.3209	0.518	0.327	0.525
	64	0.3189	0.517	0.326	0.521
	128	0.2092	0.503	0.315	0.513
0.01	32	0.3671	0.580	0.359	0.570
	64	0.3692	0.581	0.363	0.569
	128	0.3690	0.580	0.361	0.568
0.001	32	0.3576	0.561	0.356	0.561
	64	0.3572	0.561	0.353	0.557
	128	0.3574	0.561	0.354	0.557

4. (20 pts) Implement a main function, `main.py`, where you load the dataset, shuffle the training data and divide it into mini-batches, write the loop for the epoch and iterations, and evaluate the model on validation set during training. Report the training and validation accuracy.

   In order to tune the model, some hyper parameters are iterated. The results can be seen on Table 1.The classic accuracy results cannot be improved up to eligible level due to high number of target value. Therefore, besides classic training and validation accuracy, top-3 accuracy also calculated, which indicates the ratio of true class matches with most probable 3 class.

   The experiments done up to 30 epoch because after some point all models performance didn't increase significantly. The listed accuracy numbers belongs to best model produced by these hyper parameters. Best models criteria is minimum validation loss. As you can see the Table 1 highest accuracy rate occurs with 0.01 learning rate. Highest training and validation accuracy is seen on when mini-batch size equals to 64. After that point, this model iterated up to 100 epoch. Training accuracy, training top-3 accuracy, validation accuracy, validation top-3 accuracy, training loss and validation loss graphs are illustrated below.

   Figure 2 demonstrates accuracy of training and validation dataset. Except first 15 epoch, there is no significant improvement for both accuracy.

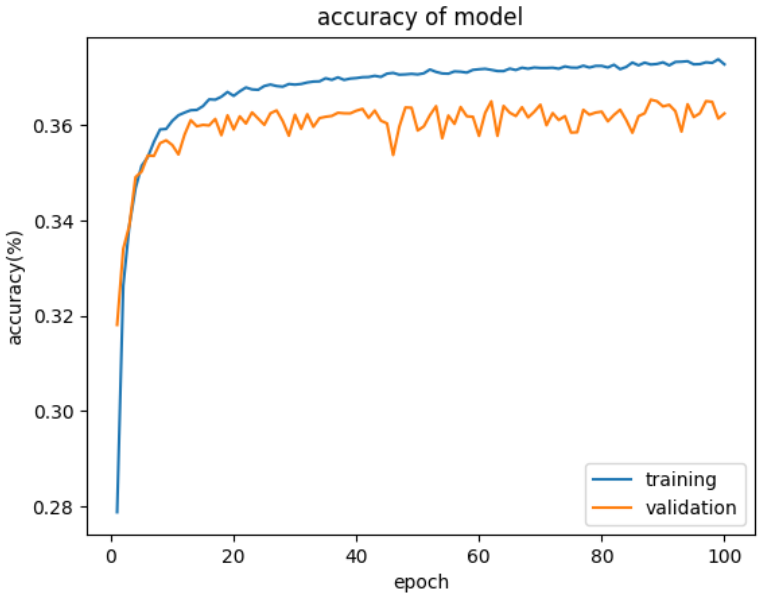

Figure 2: Accuracy of training and validation

Figure 3 illustrates top 3 accuracy, the line graph shapes are similar to classical accuracy plots but accuracy level much higher because here the most probable 3 target words are taken into account to calculate accuracy.

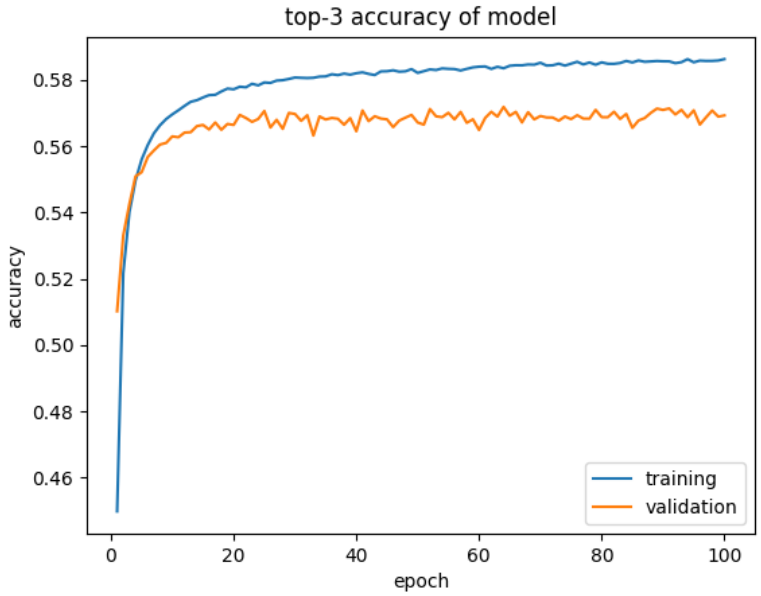

Figure 3: Top-3 accuracy of training and validation

Figure 4 shows the loss values of both training and validation dataset belongs to epoch which reaches up to 100.

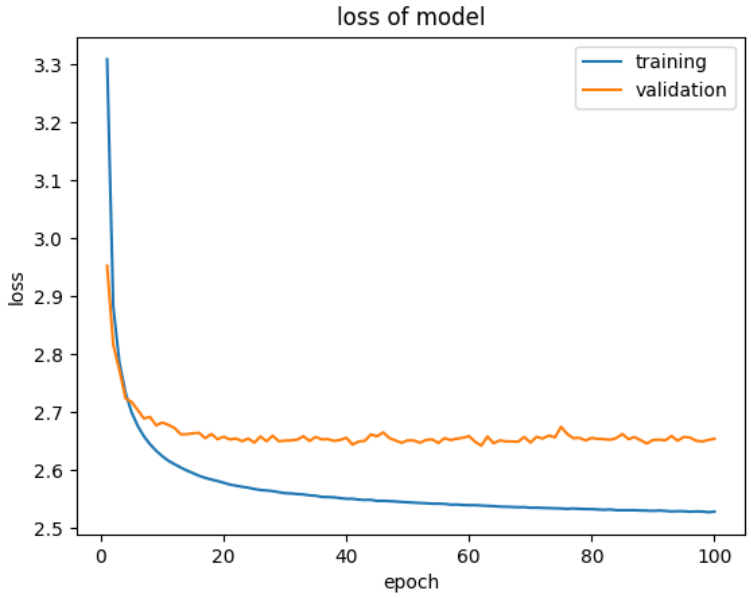

Figure 4: Loss values of training and validation

According to calculation minimum validation seen on epoch 62 and the model was saved for test accuracy results. At epoch 62, training accuracy equals to 0.3717, whereas top 3 accuracy is 0.5833 and training loss is 2.5394 In validation, accuracy equals to 0.3651, while top-3 accuracy is 0.5703 with loss 2.6427.

5. (5 pts) Implement an evaluation function, `eval.py`, where you load the learned network parameters and evaluate the model on test data. Report the test accuracy.

   As a result of saved model, training accuracy equals to 0.3642 and top 3 accuracy equals to 0.5694 where loss value is 2.657.

6. After obtaining 16 dimensional embeddings

   (a) (10 pts) Create a 2-D plot of the embeddings using t-SNE which maps nearby 16 dimensional embeddings close to each other in the 2-D space. You can use of the shelf t-SNE functions. Implement a `tsne.py` file where you load model parameters, return the learned embeddings, and plot t-SNE. Use the words in the `vocab.txt` as the labels in the plot.

   The t-SNE plot can be seen in below Figure. Because 250 words located on the that plot, the words and cluster cannot be readable but when investigated by zooming as seen on Figure 5a and 5b, some clusters can be seen easily.

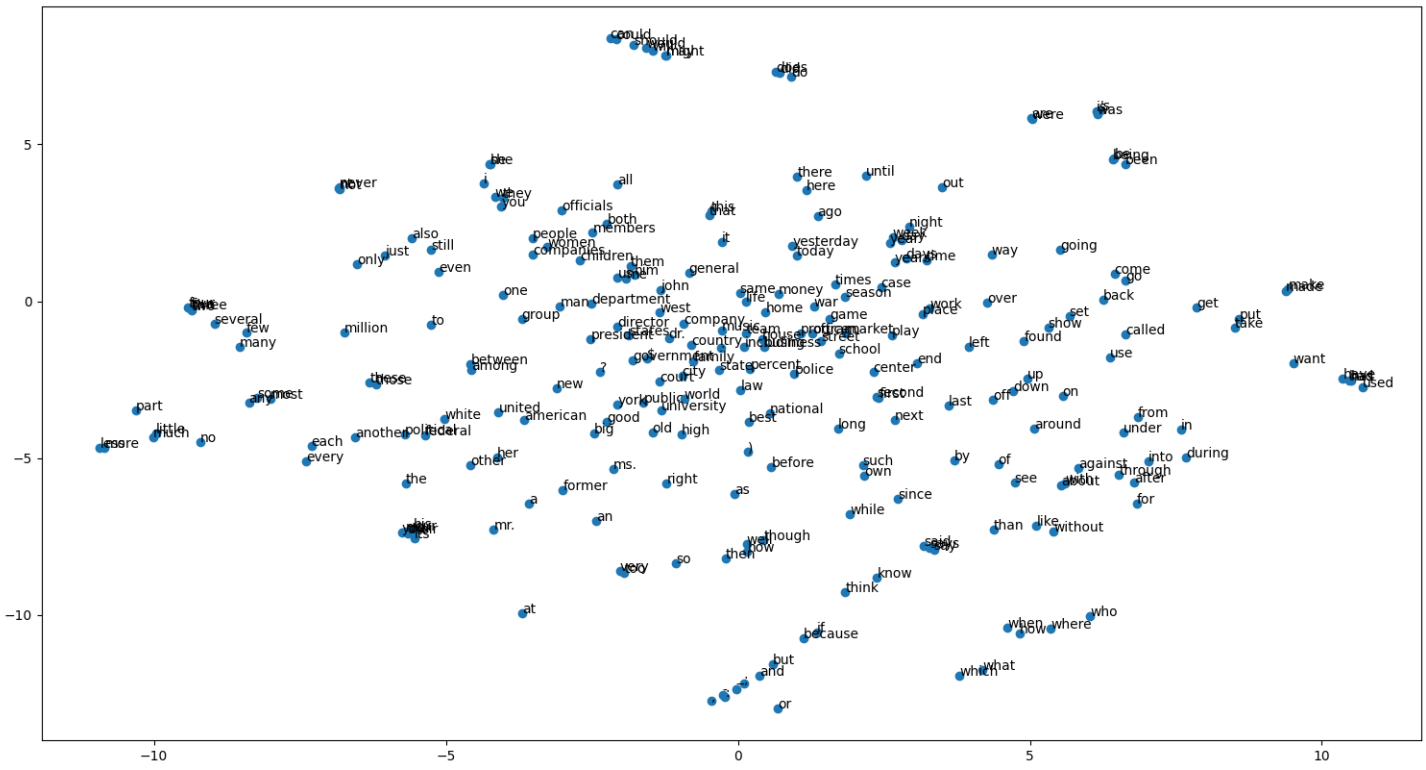

Figure 5: t-SNE plot of vocabulary

(b) (5 pts) Look at the plot and find a few clusters of related words. What do the words in each cluster have in common?

For instance question words in English such as "where, how, when, what" create a cluster. In Figure 6b, plural forms of "be" and singular form of "be" create sub-clusters which are too close to each other. In figure 6c, modal verbs such as "could, would, should" creates a cluster. Also, subject pronounces are grouped. Besides that the numbers forms a group. If you look at Figure 5 in detail, it can be seen that there are many groups other than these.

(a) Questions words (b) Forms of "to be" (c) Modal verbs

Figure 6: Clusters in t-SNE

(c) (5 pts) Pick the following data points; 'city of new', 'life in the', 'he is the'. Use the model to predict the next word. Does the model give sensible predictions?

Yes, the model makes a good prediction even if the accuracy level is low because the input vector can construct meaningful sentence of phrase with several forth word, it means there is no absolute true for the target word.

- The prediction results for "city of new" are "york", "." , "?", "government" and "music".
- The prediction results for "life in the" are "world", "country" , "game", "city" and "street".
- The prediction results for "he is the" are "best", "only" , "president", "same" and "first".

The words are ordered from the highest probability to the lowest probability. As seen on these results, each three words sequence can have several meaningful forth word rather than having only one target word. This situation makes accuracy lower when compared to ordinary neural networks.

7. (5 pts) Provide a README file where I can find the steps to train, load, and evaluate your model.

README file is attached to the .zip file.

References

[1] Vanderplas, J. (2017). *Python data science handbook*. Beijing [etc.]: O'Reilly.

[2] Nielsen, M. (2021). 3.1: The cross-entropy cost function. Retrieved 1 May 2021, from https://eng.libretexts.org/Bookshelves/Computer_Science/Book%3A_Neural_Networks_and_Deep_Learning_(Nielsen)03%A_Improving_the_way_neural_networks_learn3.013A_The_cross - entropy_cost_function.

[3] Derksen, L. (2021). Visualising high-dimensional datasets using PCA and t-SNE in Python. Retrieved 25 April 2021, from https://towardsdatascience.com/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python-8ef87e7915b.

[4] CMPE597 Deep Learning Lecture Notes, Bogazici University.