Name Surname: Pınar Süngü
Student Number: 2019705072

CMPE 597 Sp. Tp. Deep Learning
Spring 2021 Project I
Due: May 2 by 11.59pm

Answers

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

   Forward Propagation

   Input: $X_1, X_2, X_3 \in \mathcal{R}^{250xN}$ where N is number of data points in a batch

   Embedding Layer:

   Parameters: $W_1 \in \mathcal{R}^{16x250}$

   Embedding layer 1 $= W_1X_1 \in \mathcal{R}^{16xN}$
   Embedding layer 2 $= W_1X_2 \in \mathcal{R}^{16xN}$
   Embedding layer 3 $= W_1X_3 \in \mathcal{R}^{16xN}$

   Embedding layer = concatenation of $W_1X_1, W_1X_2, W_1X_3 \in \mathcal{R}^{48xN}$

   Hidden Layer:

   Parameters: $W2 \in \mathcal{R}^{128x48}$ and $b_1 \in \mathcal{R}^{128x1}$

   hidden layer $= W_2$(Embedding layer) $+ b_1 \in \mathcal{R}^{128xN}$

   hidden layer $= \sigma$(hidden layer) where σ is sigmoid function defined as $\sigma(x) = \frac{1}{1+\exp(-x)}$

   Output Layer:

   Parameters: $W_3 \in \mathcal{R}^{250x128}$ and $b_2 \in \mathcal{R}^{250x1}$

   output $= \gamma(W_3$(hidden layer) $+b_2)$ where γ is softmax function defined as

   $\gamma(x) = \frac{exp(x)}{\sum_{n=1}^{250} \exp(x)}$

We use cross-entropy loss function for this project

$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{250} y_{ij}\log(\hat{y}_{ij})$ since we have 250 classes, summation is from 1 to 250. Also, N is number of data points.

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

Gradients for backward propagation

We will need to keep the derivative of loss function with respect to the input, x_i, of softmax function

$\gamma(x) = \frac{exp(x)}{\sum_{n=1}^{250}\exp(x)}$

Since we will call the output of the softmax function as a prediction, we will denote it as $\hat{y}$

if $i = j$: $\frac{\partial \hat{y}_i}{\partial x_i} = \frac{\partial \frac{e^{x_i}}{\sum_{n=1}^{250}\exp(x)}}{\partial x_i} = \frac{e^{x_i}\sum_{n=1}^{250}\exp(x) - e^{x_i}e^{x_i}}{\sum_{n=1}^{250}\exp(x)^2} = \frac{e^{x_i}}{\sum_{n=1}^{250}\exp(x)}\frac{\sum_{n=1}^{250}\exp(x) - e^{x_i}}{\sum_{n=1}^{250}\exp(x)} = \frac{e^{x_i}}{\sum_{n=1}^{250}\exp(x)}(1 - \frac{e^{x_i}}{\sum_{n=1}^{250}\exp(x)}) = \hat{y}_i(1-\hat{y}_i)$

if $i \neq j$: $\frac{\partial \hat{y}_i}{\partial x_j} = \frac{\partial \frac{e^{x_i}}{\sum_{n=1}^{250}\exp(x)}}{\partial x_j} = \frac{0 - e^{x_i}e^{x_j}}{\sum_{n=1}^{250}\exp(x)} = -\frac{e^{x_i}}{\sum_{n=1}^{250}\exp(x)}\frac{e^{x_j}}{\sum_{n=1}^{250}\exp(x)} = -\hat{y}_i\hat{y}_j$

$\frac{\partial L}{\partial x_i} = -\sum_{j=1}^{250}\frac{\partial y_j \log(\hat{y}_j)}{\partial x_i} = -\sum_{j=1}^{250} y_j \frac{\partial \log(\hat{y}_j)}{\partial x_i} = -\sum_{j=1}^{250} y_j \frac{1}{\hat{y}_j}\frac{\partial \hat{y}_j}{\partial x_i}$
$= -\frac{y_i}{\hat{y}_i}\frac{\partial \hat{y}_i}{\partial x_i} - \sum_{j\neq i}^{250}\frac{y_j}{\hat{y}_j}\frac{\partial \hat{y}_j}{\partial x_i} = -\frac{y_i}{\hat{y}_i}\hat{y}_i(1-\hat{y}_i) - \sum_{j\neq i}^{250}\frac{y_j}{\hat{y}_j}(-\hat{y}_j\hat{y}_i)$
$= -y_i + y_i\hat{y}_i + \sum_{j\neq i}^{250} y_j\hat{y}_i = -y_i + \sum_{j=1}^{250} y_j\hat{y}_i = -y_i + \hat{y}_i\sum_{j=1}^{250} y_j = \hat{y}_i - y_i$

Thus, the combination of $j = i$ and $j \neq i$ is $\hat{y}_i - y_i$

Then, derivative of loss function with respect to bias, $b_2 \in \mathcal{R}^{250x1}$

$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial \gamma}\frac{\partial \gamma}{\partial b_2} = \sum_{i=1}^{N}(\hat{y}_i - y_i)^T \in \mathcal{R}^{250x1}$

Then, derivative of loss function with respect to $W_3 \in \mathcal{R}^{250x128}$

$\frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial \gamma}\frac{\partial \gamma}{\partial W_3} = ((\text{hidden layer})(\hat{y}_i - y_i))^T \in \mathcal{R}^{250x128}$

Then, derivative of loss function with respect to hidden layer $\in \mathcal{R}^{128x16}$

$\frac{\partial L}{\partial hidden} = \frac{\partial L}{\partial \gamma}\frac{\partial \gamma}{\partial hidden} = ((\hat{y}_i - y_i)W_3)^T \in \mathcal{R}^{128x16}$

We will need to keep the derivative of sigmoid function for $b_1, W_2, W_1, embedding$

$\frac{d\sigma}{dx} = \frac{d}{dx}(1+e^{-x})^{-1} = -(1+e^{-x})^{-2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{(1+e^{-x})}\frac{e^{-x}}{(1+e^{-x})} = \frac{1}{(1+e^{-x})}\frac{(1+e^{-x})-1}{(1+e^{-x})} = \frac{1}{1+e^{-x}}(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}) = \frac{1}{1+e^{-x}}(1 - \frac{1}{1+e^{-x}}) = \sigma(x)(1-\sigma(x))$

So, now we can calculate the derivative of loss function with respect to sigmoid function

$\frac{\partial L}{\partial \sigma} = \frac{\partial L}{\partial \gamma}\frac{\partial \gamma}{\partial hidden}\frac{\partial hidden}{\partial \sigma} = ((\hat{y}_i - y_i)W_3)^T \circ \sigma(x)(1-\sigma(x))$

Continue with the, derivative of loss function with respect to bias, $b_1 \in \mathcal{R}^{128x1}$

$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial \gamma}\frac{\partial \gamma}{\partial hidden}\frac{\partial hidden}{\partial \sigma}\frac{\partial \sigma}{\partial b_1} = \sum_{i=1}^{N}((\hat{y}_i - y_i)W_3)^T \circ \sigma(x)(1-\sigma(x)) \in \mathcal{R}^{128x1}$

Then, derivative of loss function with respect to $W_2 \in \mathcal{R}^{128x1}$

$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \gamma}\frac{\partial \gamma}{\partial hidden}\frac{\partial hidden}{\partial \sigma}\frac{\partial \sigma}{\partial W_2} = ((\hat{y}_i - y_i)W_3)^T \circ \sigma(x)(1-\sigma(x))(embedding^T)$ where embedding is concatenation of $W1X_1, W1X_2, W1X_3$

We will need derivative of loss function with respect to embedding layer $\in \mathcal{R}^{48x250}$ for W_1

$\frac{\partial L}{\partial embedding} = \frac{\partial L}{\partial \gamma}\frac{\partial \gamma}{\partial hidden}\frac{\partial hidden}{\partial \sigma}\frac{\partial \sigma}{\partial embedding} = W_2^T((\hat{y}_i - y_i)W_3)^T \circ \sigma(x)(1-\sigma(x))$

Finally, derivative of loss function with respect to W_1

$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \gamma}\frac{\partial \gamma}{\partial hidden}\frac{\partial hidden}{\partial \sigma}\frac{\partial \sigma}{\partial embedding}\frac{\partial embedding}{\partial W_1} = W_2^T((\hat{y}_i - y_i)W_3)^T \circ \sigma(x)(1-\sigma(x))[\text{first 16 columns}]X_1$
$+ W_2^T((\hat{y}_i - y_i)W_3)^T \circ \sigma(x)(1-\sigma(x))[\text{second 16 columns}]X_2$
$+ W_2^T((\hat{y}_i - y_i)W_3)^T \circ \sigma(x)(1-\sigma(x))[\text{last 16 columns}]X_3$

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

   The object of Network class has properties that initialize the weights by random normal distribution with 0 mean and, 0.01 standard deviation, and biases with zeros. The class has softmax and sigmoid activation functions, forward propagation, back propagation, and cross entropy loss functions. To create an object of the class codebook should be given as a constructor.

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.

   250 mini batches are given as a default value which makes batch size equals to 1490. As we know from the theory, we should increase the batch size as much as possible. 16 GB RAM capacity allowed me try wide range of possibilities. According

to the experiments, I realized when I increase the batch size, gradient exploding happened, and when I decrease the batch size, as expected it did not converge to anywhere. Also, initialization step is one of the critical experiments, for instance, when I initialize weights as random normal distribution with 0 mean and 1 standard deviation, again gradient exploding is observed, finally I fixed the initialization to with 0 mean and 0.01 standard deviation, it helped to see the updates. As a learning rate I decided on 0.001, and used the learning rate decay method in each epoch. Number of epoch does not important for my case. After 2 epochs later, it converges to somewhere, and weight update does not improve accuracy, because updates are on the 6-7'th decimal digits and in each epoch, changes become negligible, and in some runs loss value has started to increase after 3 epochs. As a summary, according to the experiments and results, I can say that I over-fitted the model, that I uploaded, to the wrong decision rule. I tried many ways to fix this over-fitting problem starting from the initialization and learning rate decisions, I added L1 and L2 regularization terms and all the combinations of these options. Also, I tried the numerically stable softmax function version, but it also did not help me to solve the problem. I searched about it, there many suggestions on dropping the stop words, punctuations and dropout methods. I did not try the dropout method because we do not have that much layer in our network. I also couldn't try the drop the stop words from the dataset and codebook, because in that way, all input and codebook would be different than the requested. When I debug the weights, realised that only update on W3 brings me the same accuracy, so I tried to the give bigger initialization for W1 and W2, then I started to see NA values for them, because they explode after 1 epoch. And after sigmoid function, all values stuck into the 0.2499999 and 0.25 OR 0.499999 and 0.5, even input of sigmoid function has variety between values. Therefore, I tried to change sigmoid function with ReLu, at that time accuracy increased to 0.20, but loss values has started to increase and validation accuracy was the same Table 1. Finally, if I do not have any mistake in my implementation, I decided that this model might not be suitable for this problem. According to [Ganai and Khursheed(2019)], and also their reference papers prefer to use RNN and LSTM architectures for this purpose.

The following Table 1 is the final accuracy table that could do.

Epoch	Training	Validation
1	0.15682	0.17380
2	0.17401	0.17380
3	0.17401	0.17380
4	0.17401	0.17380

Table 1: Training and Validation Accuracy

The Figure 2 shows the loss values. In each iteration, loss is calculated and kept, at the end of the iteration, mean of the iteration recorded as an epoch loss value. It looks like decreasing, but when we look at the values, the differences is only 0.25. It has tendency to decrease, I tried to increase the epoch number but I couldn't reproduce the same results. Also, I did not observe the fluctuations for this model, but I have observed some other runs which gave around 0.003 accuracy, so I believed

that I will encounter that fluctuations all the time, so did not save it. After accuracy has increased, these fluctuations were missed.

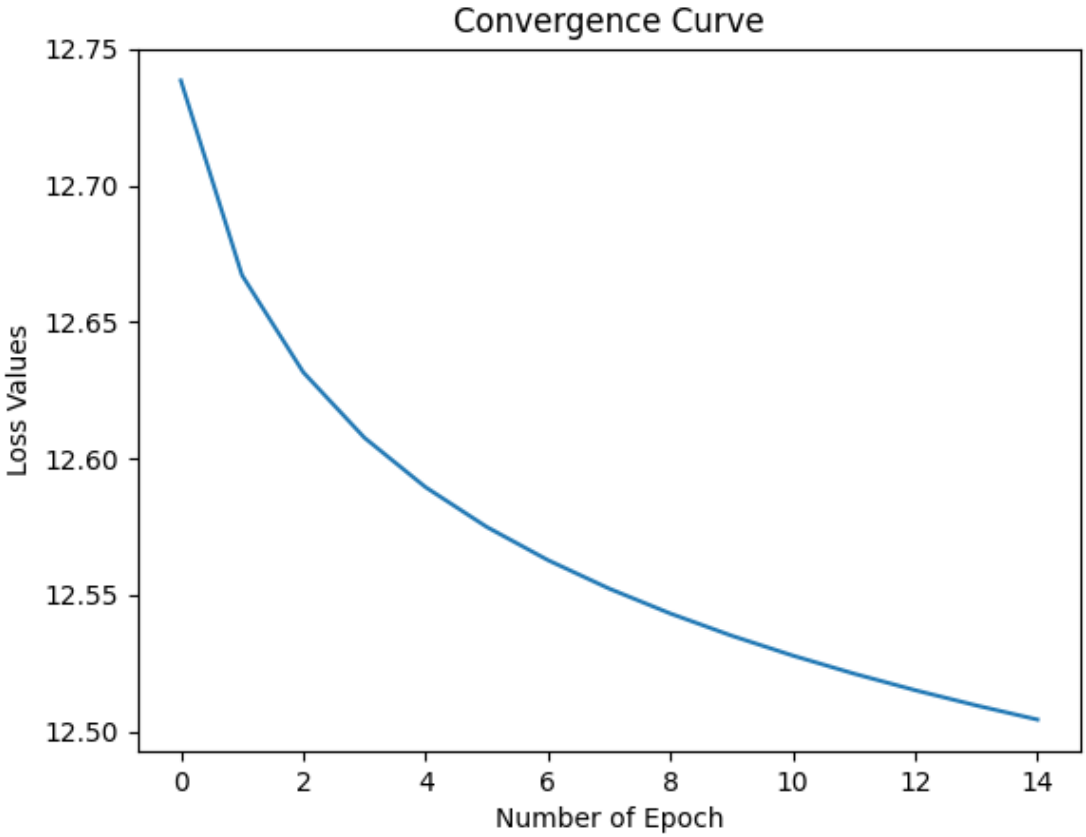

Figure 1

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.

   Test accuracy: 0.17292, this result is also sign of over-fitting to the wrong model.

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.

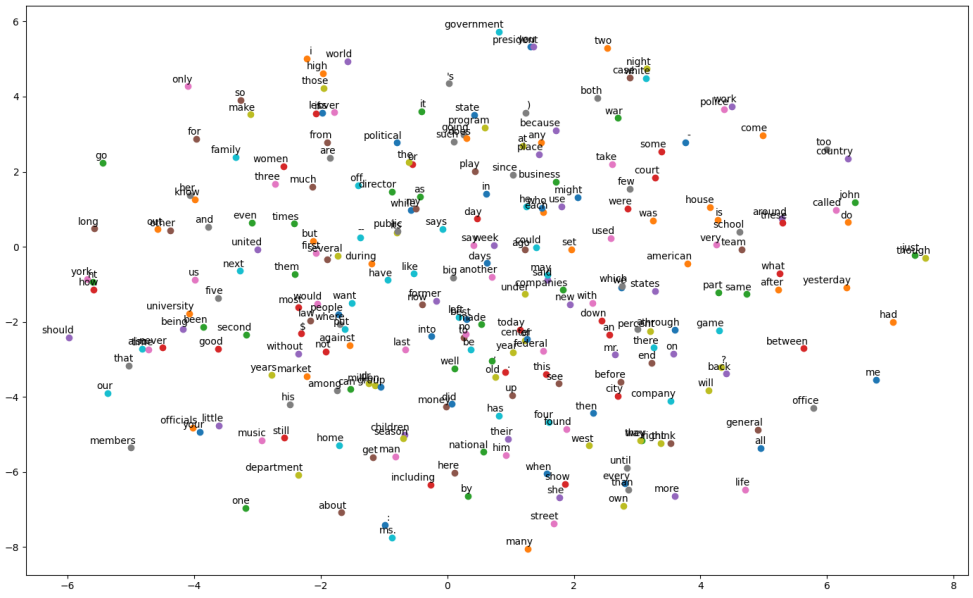

Figure 2

(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?

I can not say embeddings are relatively fine, it might be totally coincidence, too. However, government and president; house and school; was and were; being and been are close to each other, but I was expected to see; was, were, be, been, being are on the same region.

(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?

city of new: "."
life in the: "."
he is the: "."

Unfortunately, model tends to predict to "." for all possible inputs.

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

README.txt file is attached.

References

[Ganai and Khursheed(2019)] A. F. Ganai and F. Khursheed. Predicting next word using rnn and lstm cells: Stastical language modeling. In *2019 Fifth International Conference on Image Information Processing (ICIIP)*, pages 469–474, 2019. doi: 10.1109/ICIIP47207.2019.8985885.