Assignment 1 non-prog

Question 2)

a)

shape of the input vector x is: 750×1

shape of the output vector y is: 250×1

dimension of $W^{(1)}$ is : $K \times 750$

dimention of $W^{(2)}$ is : $250 \times K$

b)

$$\begin{tikzcd}
W^{(1)} \arrow[dr] & & W^{(2)} \arrow[dr] & & \\
x \arrow[r] & m \arrow[r] & h \arrow[r] & z \arrow[r] & y \\
b^{(1)} \arrow[ur] & & b^{(2)} \arrow[ur] & &
\end{tikzcd}$$

c) $y = \text{softmax}(W^{(2)}h + b^{(2)}) = \text{Softmax}(z)$

$$\mathcal{L}(y,t) = -t^T \log(y)$$

$$\frac{\partial \mathcal{L}}{\partial z_i} = -\sum_{j=1}^{k} \frac{\partial t_j \log(y_j)}{\partial z_i} = -\sum_{j=1}^{k} t_j \frac{\partial \log(y_j)}{z_i}$$

$$= -\frac{t_i}{y_i}\frac{\partial y_i}{\partial z_i} - \sum_{j \neq i}^{k} \frac{t_j}{y_j}\frac{\partial y_j}{\partial z_i}$$

$$= -\frac{t_i}{y_i} y_i(1-y_i) - \sum_{j \neq i}^{k} \frac{t_j}{y_j}(-y_j y_i)$$

let $\Sigma_k = \sum_{a=1}^{k} e^{z_a}$ then we have $y_i = \frac{e^{z_i}}{\Sigma_k}$

if $i = j$: $\frac{\partial y_i}{\partial z_i} = \frac{\partial \frac{e^{z_i}}{\Sigma_k}}{\partial z_i} = \frac{e^{z_i}\Sigma_k - (e^{z_i})^2}{\Sigma_k^2} = \frac{e^{z_i}}{\Sigma_k}\,\frac{\Sigma_k - e^{z_i}}{\Sigma_k} = \frac{e^{z_i}}{\Sigma_k}\left(1 - \frac{e^{z_i}}{\Sigma_k}\right) = y_i(1-y_i)$

if $i \neq j$: $\frac{\partial y_i}{\partial z_j} = \frac{\partial \frac{e^{z_i}}{\Sigma_k}}{\partial z_j} = -\frac{e^{z_i} e^{z_j}}{\Sigma_k} = -y_i y_j$

$$= -t_i + t_i y_i + \sum_{j \neq i} t_j y_i = -t_i + y_i \sum_{j=1}^{K} t_j$$

$$= y_i - t_i$$

Gradient descent updates can be derived for each row of W

$$\frac{\partial \mathcal{L}}{\partial W_j^{(2)}} = \frac{\partial \mathcal{L}}{\partial z_j} \cdot \frac{\partial z_j}{\partial W_j^{(2)}} = (y_j - t_j) h$$

$$W_j^{(2)} \leftarrow W_j^{(2)} - \alpha \frac{1}{N} \sum_{i=1}^{N} (y_j^{(i)} - t_j^{(i)}) h^{(i)}$$

$$W^{(2)} \leftarrow W^{(2)} - \alpha \frac{1}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) h$$

d) we have $\mathcal{L} = \frac{1}{2}(y-t)^2$

To find out the update rule, we will use chain rule.

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^{(2)}} = \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial z_{ij}} \frac{\partial z_{ij}}{\partial W_{ij}^{(2)}}$$

$$\frac{\partial \mathcal{L}}{\partial y} = \frac{\partial (\frac{1}{2}(y_i - t_i)^2)}{\partial y} = y_i - t_i$$

if $i = j$: $\frac{\partial y_i}{\partial z_i} = \frac{\partial \frac{e^{z_i}}{\Sigma_k}}{\partial z_i} = \frac{e^{z_i}\Sigma_k - (e^{z_i})^2}{\Sigma_k^2} = \frac{e^{z_i}}{\Sigma_k} \frac{\Sigma_k - e^{z_i}}{\Sigma_k} = \frac{e^{z_i}}{\Sigma_k}\left(1 - \frac{e^{z_i}}{\Sigma_k}\right) = y_i(1-y_i)$

if $i \neq j$: $\frac{\partial y_i}{\partial z_j} = \frac{\partial \frac{e^{z_i}}{\Sigma_k}}{\partial z_j} = -\frac{e^{z_i} e^{z_j}}{\Sigma_k^2} = -y_i y_j$

$$\frac{\partial z_{ij}}{\partial W_{ij}^{(2)}} = \frac{\partial (W_{ij}^{(2)} \cdot h_{ij} + b_i^{(2)})}{\partial W_{ij}^{(2)}} = h_{ij}$$

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^{(2)}} = \begin{cases} (y_i - t_i) \cdot y_i (1 - y_i) h_{ij} & (i=j) \\ (y_i - t_i)(-y_i y_j) \cdot h_{ij} & (i \neq j) \end{cases}$$

Ideally, the gradient should give us strong signals regarding how to update w to do better.

But here $\frac{\partial \mathcal{L}}{\partial w_{ij}}$ is small.

Which means that update $W \leftarrow w - \alpha \frac{\partial \mathcal{L}}{\partial w}$ won't change W much.

So we will not get good gradient signal to update $W_{ij}^{(2)}$ if we use this square loss.

e) $\sigma(z) = \frac{1}{1+e^{-z}}$

$$\sigma'(z) = -\frac{\frac{d}{dz}(e^{-z}+1)}{(e^{-z}+1)^2} = \frac{e^{-z}}{(e^{-z}+1)^2}$$

$$\sigma(z)(1-\sigma(z)) = \frac{1}{1+e^{-z}} \cdot \left(\frac{e^{-z}}{1+e^{-z}}\right)$$

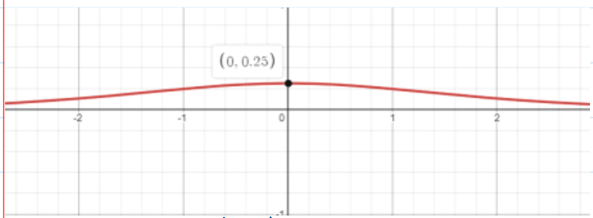
$$= \frac{e^{-z}}{(\ldots)^2}$$

$$\sigma(z)(1-\sigma(z)) = \frac{1}{1+e^{-z}} \cdot \left(\frac{e^{-z}}{1+e^{-z}} \right)$$

$$= \frac{e^{-z}}{(e^{-z}+1)^2}$$

So we showed that $\sigma'(z) = \sigma(z)(1-\sigma(z))$

plot of the function $\sigma'(z)$



Max: $\sigma''(z) = -\frac{(e^z-1)e^z}{(e^z+1)^3}$

$$-\frac{(e^z-1)e^z}{(e^z+1)^3} = 0$$

$$-(e^z-1)e^z = 0$$

$$e^z - 1 = 0$$

$$e^z = 1$$

$$z = 0$$

When $z=0$ $\sigma'(0) = \frac{1}{4} = 0.25$

So the maximum is $(0, 0.25)$, and $\sigma'(z)$ must be positive since both e^{-z} and $(e^{-z}+1)$ are positive

let $z = W_1 x + b_1$

$$\frac{\partial h_1}{\partial x} = \frac{\partial h_1}{\partial z} \frac{\partial z}{\partial x} = \sigma'(z)\, W_1$$

$$= \sigma(z)(1-\sigma(z))\, W_1$$

Since for $\sigma'(z)$ it at most 0.25 which is $\frac{1}{4}$

And $\sigma'(z)$ is positive.

We have $\left|\frac{\partial h_1}{\partial x}\right| \leq \frac{1}{4}|W_1|$

f) The problem for this result is when N is very large, $\left|\frac{\partial h_N}{\partial x}\right|$ will become extremely small.

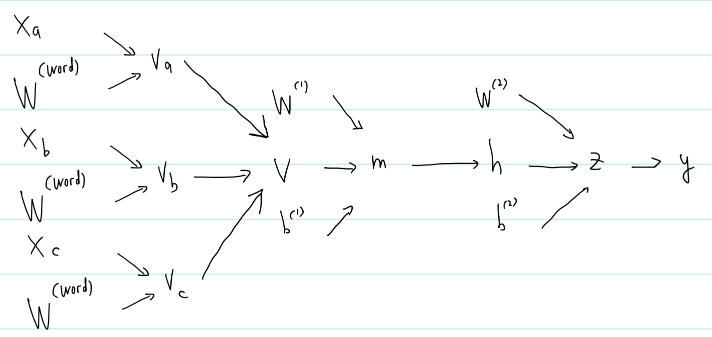
Since it's smaller than $\frac{1}{4^N}|W_1||W_2|\cdots|W_N|$ and $\frac{1}{4^N}$ will surely be very small when N is big.

And this means that x's value will not effect hN a lot. The Nth layer will try to update its weight mostly based on its previous layer's updated weight. More closer to the last layer, the value of x will be less important. In this case, for many of the layers close to the Nth layer, it will only take few information from the input x. It shows that more layers will not help at all and it will even make the learning speed slower and cause problems. On the other hand, if we have many layers and we multiply these gradients together, it's possible that the product of many small values (less than one) will become zero very quickly (because "1/a large number" will be treated as 0 if the number is large enough) since the derivative of the sigmoid function is always smaller than 1. For deep learning, more layers will always improve the learning experience, but if we use sigmoid activation function, more layers means problems. That's why sigmoid activation function would be a problem with this result.

g) No, we will not have the same issue as in part f if we replaced the sigmoid activation with ReLu activation. ReLu activation function reduced likelihood of the gradient to vanish. For the gradient of sigmoid activation function, it will get smaller and smaller as the absolute value of the input x increases. And as we saw in the previous questions, the derivative of the sigmoid function is always smaller than 1/4. In Question e) we just proved that the maximum is (0,1/4). It means that the sigmoid activation function learns slow. But the constant gradient of ReLus will have a faster learning. The gradient of the ReLu activation function is either 0 (if <0) or 1 (if >0). This means the number of the layers will not cause any problems. The gradients will never vanish.

Q3)

a)



b)

$$\frac{\partial y}{\partial W^{(word)}} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial W^{(word)}} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial h}\frac{\partial h}{\partial W^{(word)}} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial h}\frac{\partial h}{\partial m}\frac{\partial m}{\partial W^{(word)}}$$

$$= \frac{\partial y}{\partial z}\frac{\partial z}{\partial h}\frac{\partial h}{\partial m}\frac{\partial m}{\partial V}\frac{\partial V}{\partial W^{(word)}}$$

$$= \frac{\partial y}{\partial z}\frac{\partial z}{\partial h}\frac{\partial h}{\partial m}\frac{\partial m}{\partial V}\left(\frac{\partial V}{\partial V_a}\frac{\partial V_a}{\partial W^{(word)}} + \frac{\partial V}{\partial V_b}\frac{\partial V_b}{\partial W^{(word)}} + \frac{\partial V}{\partial V_c}\frac{\partial V_c}{\partial W^{(word)}}\right)$$

4c)

If we do not call numpy_model.initializeParams(), then the **__init__** method in Numpy WordEmbModel class will initialize all weights and bias matrix with zeros. So no matter what the input is, every hidden unit will get zero. Since 0x+ 0 = 0 (x is the input vector). It will cause all of them to have the same gradient. It means that if all the weights and bias are zero, the learning rate will only affects the scale of the weight v ector, not the direction.

4d)

After applying softmax, each component will be in (0,1) range and they will have a total sum 1.

Then each column of softmax (z) will be $\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ for $j = 1 \dots K$, K = the number of column.

Then if z_i is the maximum column in z. Since every column is divide by the same number $\sum_{k=1}^{K} e^{z_k}$, and e^{z_i} is positive, $\frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$ will still be the maximum column in softmax (z)

The larger input components will correspond to larger probabilities. So the max column of z will still be the max column in y since it will have the largest probability.

5d)
Yes, these predictions do make sense since these are all part of some sentences.

For the 3-grams words, all of the 3-grams words are not appearing next to each other in the training set.

6a)
The shape of the word_emb_weights is (100,250). 100 here represents the emb size and 250 represents the vocab size. This embedding layer is to compute the representation of each word and we will get the result by multiply the word_emb_weights to the one-hot vector for each word (which has a shpe 250*1). For each word multiply by woed_emb_weights, we basically do a matrix multiplication of (100x250) x (250x1) and we will finally get a 100x1 vector and this vector is the representation vector of that word. If we multiply a (250x250)matrix which represent all of the one hot vectors for each word. We will finally get a (100x250) matrix which each column is a word representation vector. But since here we take the transpose, word_emb has a shape (250,100) which ith row is a 1x100 vector and this vector is the representation of the ith word in the vocab.In the example vocab_stoi["any"] will get the index of the word "any" and word_emb[vocab_stoi["any"],:] will take the corresponding row in word_emb matrix. Which is the vector representation of the word "any".

6c)
Cluster 1:
many, few,three,two,several

These word in cluster 1 are all able to describe numbers

Cluster 2: can,could,will,should,might

These words are all verb and they all can be followed with a personal pronoun

7)
This assignment is finished individually by Hongyu Chen