Solution to Assignment 1, Problem 1(a)

Akash Rana

Softmax Prove that $softmax$ is invariant to constant offsets in the input, that is, for any input vector x and any constant c,

$$\text{softmax}(\mathbf{x}) = \text{softmax}(\mathbf{x} + c), \tag{1}$$

where $(\mathbf{x} + c)$ means adding the constant c to every dimension of $\mathbf{x}$.

$$\text{softmax}(\mathbf{x})_j = \frac{e^{x_j}}{\sum_{k=1} e^{x_k}} \tag{2}$$

Solution:

$$\text{softmax}(\mathbf{x} + c)_j = \frac{e^{(x_j+c)}}{\sum_{k=1} e^{(x_k+c)}} \tag{3}$$

$$= \frac{e^c}{e^c} \frac{e^{(x_j)}}{sum_{k=1} e^{(x_j)}} \tag{4}$$

$$= \text{softmax}(\mathbf{x})_j \tag{5}$$

Solution to Assignment 1, Problem 1(b)

Akash Rana

Given an input matrix of `N`-rows and `d`-columns, compute the softmax prediction for each row. Write your implementation in `q1_softmax.py`. You may test by executing python `q1_softmax.py`.

Note: The provided tests are not exhaustive. Later parts of the assignment will reference this code so it is important to have a correct implementation. Your implementation should also be efficient and vectorized whenever possible. A non-vectorized implementation will not receive full credit!

```
import numpy as np

def softmax(x):
#x.shape by default gives column value
    if len(x.shape) > 1:
        # Matrix
        ### YOUR CODE HERE
        c = np.max(x, axis=1).reshape(-1,1) #-1 here means, internally
numpy is just calculating, to get the missing dimension.
        x = np.exp(x-c) / np.sum(np.exp(x-c), axis=1).reshape(-1,1)
        print (np.shape(x))
        ### END YOUR CODE
    else:
        # Vector
        ### YOUR CODE HERE
        c = np.max(x)
        x = np.exp(x-c) / np.sum(np.exp(x-c))
        ### END YOUR CODE

    assert x.shape == orig_shape
    return x
```

Solution to Assignment 1, Problem 2(a)

Akash Rana

Derive the gradients of the sigmoid function and show that it can be rewritten as a function of the function value (i.e. in some expression where only $\sigma(x)$, but not x, is present). Assume that the input x is a scalar for this question.

Denote the sigmoid function as $\sigma(z)$,

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \tag{6}$$

Using chain rule,

$$\begin{aligned}
\sigma'(z) &= \frac{-1}{(1 + e^{-z})^2} \times (-e^{-z}) \\
&= \frac{1}{1 + e^{-z}} \left(\frac{e^{-z}}{1 + e^{-z}} \right) \\
&= \left(\frac{1}{1 + e^{-z}} \right) \left(\frac{1 + e^{-z}}{1 + e^{-z}} - \frac{1}{1 + e^{-z}} \right) \\
&= \sigma(z)(1 - \sigma(z))
\end{aligned}$$

Solution to Assignment 1, Problem 2(b)

Akash Rana

Derive the gradient with regard to the inputs of a softmax function when cross entropy loss is used for evaluation, i.e. find the gradients with respect to the softmax input vector θ, when the prediction is made by $\hat{y} = \text{softmax}(\theta)$. Remember the cross entropy function is

$$\text{CE}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i y_i \log \hat{y}_i \tag{7}$$

where y is the one-hot label vector, and $\hat{y}$ is the predicted probability vector for all classes.

Hint: you might want to consider the fact many elements of y are zeros, and assume that only the k-th dimension of y is one.

Cross entropy error function for multi-class output,

$$\text{CE}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i y_i \log \hat{y}_i \tag{8}$$

Computing the gradient yields,

$$\frac{\partial(\text{CE})}{\partial \hat{y}_i} = -\frac{y_j}{\hat{y}_i} \tag{9}$$

$$\frac{\partial(\text{CE})}{\partial \theta_k} = \frac{\partial(\text{CE})}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial \theta_k} \tag{10}$$

$$= -\frac{y_j}{\hat{y}_i}\frac{\partial \hat{y}_i}{\partial \theta_k} \tag{11}$$

Calculating the partial derivative of $\hat{y}_i$ (derivation using the quotient rule):

$$\text{if } i = k : \frac{\partial y_i}{\partial \theta_i} = \frac{\partial \frac{e^{\theta_i}}{\Sigma_\theta}}{\partial \theta_i} \tag{12}$$

$$= \frac{e^{\theta_i}\Sigma_\theta - e^{\theta_i}e^{\theta_i}}{\Sigma_\theta^2} \tag{13}$$

$$= \frac{e^{\theta_i}}{\Sigma_\theta}\frac{\Sigma_\theta - e^{\theta_i}}{\Sigma_\theta} \tag{14}$$

$$= \frac{e^{\theta_i}}{\Sigma_\theta}(1 - \frac{e^{\theta_i}}{\Sigma_\theta}) = \hat{y}_i(1 - \hat{y}_i) \tag{15}$$

$$\text{if } i \neq k : \frac{\partial y_i}{\partial \theta_j} = \frac{\partial \frac{e^{\theta_i}}{\Sigma_\theta}}{\partial \theta_j} \tag{16}$$

$$= \frac{0 - e^{\theta_i} e^{\theta_j}}{\Sigma_\theta^2} \tag{17}$$

$$= -\frac{e^{\theta_i}}{\Sigma_\theta} \frac{e^{\theta_j}}{\Sigma_\theta} \tag{18}$$

$$= -\hat{y}_i y_k \tag{19}$$

Combining Equations 9, 15, 19, yields

$$\frac{\partial(\text{CE})}{\partial \theta_k} = \begin{cases} -y_j(1 - \hat{y}_k) & \text{for } i = k \\ y_j \hat{y}_k & \text{for } i \neq k \end{cases} \tag{20}$$

Requiring y_j to be non-zero, imposes that the auxiliary condition, $k = j$ and $y_j = 1$, hence it follows immediately,

$$\frac{\partial(\text{CE})}{\partial \theta_j} = \begin{cases} (\hat{y}_j - 1) & \text{for } i = j \\ \hat{y}_j & \text{for } i \neq j \end{cases} \tag{21}$$

Which is equivalent to

$$\frac{\partial(\text{CE})}{\partial \boldsymbol{\theta}} = \hat{\mathbf{y}} - \mathbf{y} \tag{22}$$

Solution to Assignment 1, Problem 2(c)

Akash Rana

Derive the gradients with respect to the inputs x to an one-hidden-layer neural network (that is, find $\partial J/\partial \mathbf{x}$ where J is the cost function for the neural network). The neural network employs sigmoid activation function for the hidden layer, and softmax for the output layer. Assume the one-hot label vector is y, and cross entropy cost is used. (feel free to use $\sigma'(x)$ as the shorthand for sigmoid gradient, and feel free to define any variables whenever you see fit).

In order to simplify the notation used to solve the problem, define the following terms:

$$\mathbf{x}^{(2)} \equiv \quad \mathbf{h} \tag{23}$$

$$\mathbf{z}_i \equiv \quad \mathbf{x^{(i)}}\mathrm{W_i} + \mathbf{b_i} \tag{24}$$

Now, to calculate $\partial J/\partial \mathbf{x^1}$, one can use the back propagation algorithm. Starting with the chain rule and then results from Question 2(b):

$$\frac{\partial J}{\partial \mathbf{x}^{(2)}} = \left(\left(\frac{\partial J}{\partial \mathbf{z}_2} \frac{\partial \mathbf{z}_2}{\partial \mathbf{x}^{(2)}} \right) . \frac{\partial \mathbf{x}^{(2)}}{\partial \mathbf{z}_1} \right) * \frac{\partial \mathbf{z}_1}{\partial \mathbf{x}^{(1)}} \tag{25}$$

$$\frac{\partial J}{\partial \mathbf{z}_2} = \hat{\mathbf{y}} - \mathbf{y} \tag{26}$$

and

$$\frac{\partial \mathbf{z}_i}{\partial \mathbf{x}^{(i)}} = \mathrm{W_i^\top} \tag{27}$$

Sigmoid (σ) derivative can be found in Question 2(a), but we define:

$$\frac{\partial \mathbf{x}^{(2)}}{\partial \mathbf{z}_1} \equiv \sigma'(z_1) \tag{28}$$

Combining these, and using $\cdot$ to denote element-wise product:

$$\frac{\partial J}{\partial z_i} = (\hat{\mathbf{y}} - \mathbf{y})\mathrm{W_2^\top} \cdot \sigma'(\mathbf{z_1}) \tag{29}$$

Finally, using the results from Equation 27 (but for the first layer):

$$\frac{\partial J}{\partial \mathbf{x}^{(1)}} = (\hat{\mathbf{y}} - \mathbf{y})\mathrm{W_2^\top} \cdot \sigma'(\mathbf{z_1}) \cdot \mathrm{W_1^\top} \tag{30}$$

Solution to Assignment 1, Problem 2(d)

Akash Rana

How many parameters are there in this neural network, assuming the input is D_x-dimensional, the output is D_y-dimensional, and there are H hidden units?

W_1 must have dimensions: $D_x \times H$. The bias ($\mathbf{b_1}$) for the first layer must have dimensions H. Adding these two together, yields $(D_x + 1) \times H$. Proceeding to the second layer, there must be $H \times D_y$ parameters associated with the weight matrix W_2. The bias ($\mathbf{b_2}$) for the second layer must have dimensions D_y elements. This yields,

$$(D_x + 1) \times H + D_y \times (H + 1) \tag{31}$$

weights, for each input vector of dimensions D_x.

Solution to Assignment 1, Problem 3(a)

Akash Rana

Assume you are given a predicted word vector $\boldsymbol{v}_c$ corresponding to the center word c for `skipgram`, and word prediction is made with the `softmax` function found in `word2vec` models

$$\hat{\boldsymbol{y}}_{\boldsymbol{o}} = p(\ \boldsymbol{o} \mid \boldsymbol{c}\ ) = \frac{\exp\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)}{\sum_{j=1}^{|W|} \exp\left(\boldsymbol{u}_j^\top \boldsymbol{v}_c\right)} \tag{32}$$

where w denotes the w-th word and $\boldsymbol{u}_w$ ($w = 1, ..., |\mathrm{W}|$) are the 'output' word vectors for all words in the vocabulary (v'_w in the lecture notes). Assume the cross entropy cost is applied to this prediction and word o is the expected word (the o-th element of the one-hot label vector is one), derive the gradients with respect to $\boldsymbol{v}_c$.

Hint: It will be helpful to use notation from question 2. For instance, letting $\hat{\boldsymbol{y}}$ be the vector of the softmax prediction for every word, $\boldsymbol{y}$ as the expected word vector, and the loss function

$$J_{\text{softmax-CE}}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U}) = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) \tag{33}$$

where $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_{|\mathrm{W}|}]$ is the matrix of all the output vectors. *Make sure you state the orientation of your vectors and matrices.*

Applying cross-entropy cost to the above softmax probability defined above:

$$J = -\log p = -\boldsymbol{u}_o^\top \boldsymbol{v}_c + \log \sum_{j=1}^{|V|} \exp\left(\boldsymbol{u}_j^\top \boldsymbol{v}_c\right) \tag{34}$$

Let $z_j = \boldsymbol{u}_j^\top \boldsymbol{v}_c$, and δ_j^i be the indicator function, then

$$\frac{\partial J}{\partial z_k} = -\delta_k^i + \frac{\exp\left(\boldsymbol{u}_i^\top \boldsymbol{v}_c\right)}{\sum_{j=1}^{|V|} \exp\left(\boldsymbol{u}_j^\top \boldsymbol{v}_c\right)} \tag{35}$$

Now, using the chain rule, we can calculate,

$$\frac{\partial J}{\partial \boldsymbol{v}_c} = \frac{\partial J}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{v}_c} \tag{36}$$

$$= \sum_{j=1}^{|V|} \boldsymbol{u}_j^\top \left(\frac{e^{z_j}}{\sum_{k=1}^{|V|} e^{z_k}} - 1 \right) \tag{37}$$

$$= \sum_{k=1}^{|V|} \boldsymbol{P}(\boldsymbol{u}_j | \boldsymbol{v}_c) \boldsymbol{u}_j - \boldsymbol{u}_j \tag{38}$$

Solution to Assignment 1, Problem 3(b)

Akash Rana

As in the previous problem, derive gradients for the 'output' word vectors $\boldsymbol{u}_w$'s (including $\boldsymbol{u}_o$)

This follows immediately from the chain rule:

$$\frac{\partial J}{\partial \boldsymbol{u}_j} = \frac{\partial J}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{u}_j} \tag{39}$$

$$= \boldsymbol{v}_c \left(\frac{\exp\left(\boldsymbol{u}_0^\top \boldsymbol{v}_c\right)}{\sum_{j=1}^{|V|} \exp\left(\boldsymbol{u}_j^\top \boldsymbol{v}_c\right)} - \delta_j^0 \right) \tag{40}$$

Solution to Assignment 1, Problem 3(c)

Akash Rana

Repeat part (a) and (b) assuming we are using the negative sampling loss for the predicted vector $\boldsymbol{v}_c$, and the expected output word is $\boldsymbol{o}(\boldsymbol{u}_o)$. Assume that K negative samples (words) are drawn, and they are $\boldsymbol{u}_1, ..., \boldsymbol{u}_k$, respectively for simplicity of notation ($k \in \{1, ..., K\}$ and $o \notin \{1, ..., K\}$). Again for a given word, $\boldsymbol{o}$, denote its output vector as $\boldsymbol{u}_o$. The negative sampling loss function in this case is,

$$J(\boldsymbol{u}_o, \boldsymbol{v}_c, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)) \tag{41}$$

where $\sigma(\cdot)$ is the sigmoid function.

After you've done this, describe with one sentence why this cost function is much more efficient to compute than the softmax-CE loss (you could provide a speed-up ratio, i.e. the runtime of the softmax-CE loss divided by the runtime of the negative sampling loss).

Note: the cost function here is the negative of what Mikolov *et. al.* had in their original paper, because we are doing a minimization instead of maximization in our code.

$$\frac{\partial J}{\partial \boldsymbol{v}_c} = \frac{\partial J}{\partial \boldsymbol{p}_i}\frac{\partial \boldsymbol{p}_i}{\partial \boldsymbol{x}_j}\frac{\partial \boldsymbol{x}_i}{\partial \boldsymbol{v}_c} + \sum_{k=1}^{K} \frac{\partial J}{\partial \boldsymbol{p}_i}\frac{\partial \boldsymbol{p}_i}{\partial \boldsymbol{z}_k}\frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{v}_c} \tag{42}$$

$$= -(\mathbf{1} - \mathbf{p_i})\mathrm{U_i} + \sum_{\mathbf{k=1}}^{\mathbf{K}} (\mathbf{1} - \mathbf{q_k})\mathrm{U_k} \tag{43}$$

$$\frac{\partial J}{\partial \boldsymbol{u}_j} = \frac{\partial J}{\partial \boldsymbol{p}_i}\frac{\partial \boldsymbol{p}_i}{\partial \boldsymbol{x}_j}\frac{\partial \boldsymbol{x}_i}{\partial \boldsymbol{u}_j} + \sum_{k=1}^{K} \frac{\partial J}{\partial \boldsymbol{p}_i}\frac{\partial \boldsymbol{p}_i}{\partial \boldsymbol{z}_k}\frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{u}_j} \tag{44}$$

$$= -(\mathbf{1} - \mathbf{p_i})\delta_{\mathbf{ij}}\mathrm{V_c} + \sum_{\mathbf{k=1}}^{\mathbf{K}} (\mathbf{1} - \mathbf{q_k})\delta_{\mathbf{ij}}\mathrm{V_c} \tag{45}$$

where, δ_{xy} = kronecker delta
when $x = y = 1$
0 , otherwise.

Solution to Assignment 1, Problem 3(d)

Akash Rana

Derive the gradients for all of the word vectors for `skip-gram` and `CBOW` (optional) given a set of context words $[\texttt{word}_{c-\mathbf{m}}, ..., \texttt{word}_{c-\mathbf{1}}, \texttt{word}_c, \texttt{word}_{c+\mathbf{1}}, ..., \texttt{word}_{c+\mathbf{m}}]$, where $\mathbf{m}$ is the context size. You can denote the 'input' and 'output' word vectors for $\texttt{word}_k$ as $\boldsymbol{v}_k$ and $\boldsymbol{u}_k$ respectively for convenience.

Hint: feel free to use $F(\boldsymbol{o}, \boldsymbol{v}_c)$ (where $\boldsymbol{o}$ is the expected word) as a placeholder for $J_{\texttt{softmax-CE}}(\boldsymbol{o}, \boldsymbol{v}_c, ...)$ or $J_{\texttt{neg-sampling}}(\boldsymbol{o}, \boldsymbol{v}_c, ...)$ cost functions in this part — you'll see that this is a useful abstraction for the coding part. That is, your solution may contain terms of the form $\frac{\partial F_i(\boldsymbol{o}, \boldsymbol{v}_c)}{\partial \lambda}$.

Recall that for `skip-gram`, the cost for a context centered around c is

$$J_{\texttt{skip-gram}} = \sum_{-m \leq j \leq m, j \neq 0} F(\boldsymbol{w}_{c+j}, \boldsymbol{v}_c) \tag{46}$$

where $\boldsymbol{w}_{c+j}$ refers to the word at the j-th index from the center.

`CBOW` is slighltly different. Instead of using $\boldsymbol{v}_c$ as the prediction vector, we use $\hat{\boldsymbol{v}}$ defined below. For (a simpler variant of) `CBOW`, we sum up the input word vectors in the context:

$$\hat{\boldsymbol{v}} = \sum_{-m \leq j \leq m, j \neq 0} \boldsymbol{v}_{c+j} \tag{47}$$

then the `CBOW` cost is

$$J_{\texttt{CBOW}}(\texttt{word}_{c-\mathbf{m}}, ..., \texttt{word}_{c+\mathbf{m}}) = F(\boldsymbol{w}_c, \hat{\boldsymbol{v}}) \tag{48}$$

Note: To be consistent with the $\hat{\boldsymbol{v}}$ notation such as for the code portion, for `skip-gram` $\hat{\boldsymbol{v}} = \boldsymbol{v}_c$

For skip-gram:

$$\frac{\partial J_{\texttt{skip-gram}}}{\partial U} = \sum_{-m \leq j \leq m, j \neq 0} \frac{\partial F(\boldsymbol{w}_{c+j}, \boldsymbol{v}_c)}{\partial U} \tag{49}$$

$$\frac{\partial J_{\texttt{skip-gram}}}{\partial v_c} = \sum_{-m \leq j \leq m, j \neq 0} \frac{\partial F(\boldsymbol{w}_{c+j}, \boldsymbol{v}_c)}{\partial v_c} \tag{50}$$

$$\frac{\partial J_{\texttt{skip-gram}}}{\partial v_i} = 0, Fori \neq c \tag{51}$$

For CBOW:

$$\frac{\partial J_{\texttt{CBOW}}}{\partial U} = \frac{\partial F(\boldsymbol{w}_c, \hat{\boldsymbol{v}})}{\partial U} \tag{52}$$

$$\frac{\partial J_{\texttt{CBOW}}}{\partial v_j} = \frac{\partial F(\boldsymbol{w}_c, \hat{\boldsymbol{v}})}{\partial v} \tag{53}$$

$$\frac{\partial J_{\texttt{CBOW}}}{\partial v_j} = 0, For i \neq c \tag{54}$$