1 Softmax

(a) A constant value c can be added to all values of the input vector $\mathbf{x}$ without changing the output of the softmax function:

$$softmax(\mathbf{x})_i = \frac{e_i^x}{\sum_j e_j^x} \tag{1}$$

proof:

$$\frac{e^(x_i + c}{\sum_j e^(x_j + c)} = \frac{e_i^x e^c}{\sum_j e_j^x e^c} = \frac{e_i^x e^c}{e^c \sum_j e_j^x} = \frac{e_i^x}{\sum_j e_j^x} \tag{2}$$

(b) Implemented in *q1_softmax.py*. Basic pseudocode:

```
max_x = np.max(x, axis=1)
max_x = np.reshape(max_x, (len(max_x), 1))
x = x - max_x
x = np.exp(x)
sum_x = np.sum(x, axis=1)
sum_x = np.reshape(sum_x, (len(sum_x), 1))
x = x / sum_x
```

Listing 1: softmax in python

2 Neural Network Basics

(a) Derivation of the sigmoid function.

$$\sigma(x) = \frac{1}{1+e^{-x}}, \quad \dot{\sigma}(x) = \sigma(x)(1-\sigma(x)) \tag{3}$$

The derivation is as follows:

$$\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\frac{1}{1+e^{-x}} = \frac{d}{dx}(1+e^{-x})^{-1} \\
&= -(1+e^{-x})^{-2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}}\frac{e^{-x}}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}}\frac{(1+e^{-x})-1}{1+e^{-x}} = \frac{1}{1+e^{-x}}(1-\frac{1}{1+e^{-x}}) \\
&= \sigma(x)(1-\sigma(x))
\end{aligned} \tag{4}$$

(b) Derivation of the cross-entropy loss with a softmax as its prediction w.r.t the input of the softmax.

$$CE(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i y_i \log \hat{y}_i = -\sum_i y_i \log \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{5}$$

Considering that $\mathbf{y}$ is a one-hot vector where only element $i = k$ is 1, equation 5 can be simplified to

$$CE(\mathbf{y}, \hat{\mathbf{y}}) = -\log \hat{y}_i = -\log \frac{e^{x_i}}{\sum_j e^{x_j}} = -x_i + \log \sum_j e^{x_j} \tag{6}$$

for $k = i$. Taking the derivative with respect to the softmax input $\mathbf{x}$ results in the following.

$$\frac{\partial}{\partial \mathbf{x}} \log \sum_j e^{x_j} - x_i = \frac{\frac{\partial}{\partial \mathbf{x}} \sum_j e^{x_j}}{\sum_j e^{x_j}} - \frac{\partial}{\partial \mathbf{x}} x_i \tag{7}$$

Now there are two different cases, $i = k$ and $i \neq k$:

$$\begin{aligned}
\frac{\frac{\partial}{\partial x_i} \sum_j e^{x_j}}{\sum_j e^{x_j}} - \frac{\partial}{\partial x_i} x_i = \frac{e^{x_i}}{\sum_j e^{x_j}} - 1 = \hat{y}_i - 1 \quad &, \quad i = k \\
\frac{\frac{\partial}{\partial x_i} \sum_j e^{x_j}}{\sum_j e^{x_j}} - \frac{\partial}{\partial x_i} x_i = \frac{e^{x_i}}{\sum_j e^{x_j}} = \hat{y}_i \quad &, \quad i \neq k
\end{aligned} \tag{8}$$

Again, considering that $\mathbf{y}$ is a one-hot vector with the only 1 at $i = k$, equation 8 can be rewritten in vector form as:

$$\frac{\partial}{\partial \mathbf{x}} CE(\mathbf{y}, \hat{\mathbf{y}}) = \hat{\mathbf{y}} - \mathbf{y} \tag{9}$$

(c) Derivation of the gradient of the cost of a two-layer neural network w.r.t. its input. First denoting the neural-network in standard notation:

$$\begin{aligned}
\mathbf{a}_1 &= \mathbf{x} \\
\mathbf{z}_2 &= \mathbf{a}_1\mathbf{W}_1 + \mathbf{b}_1 \\
\mathbf{a}_2 &= \mathbf{h} = \sigma(\mathbf{z}_2) \\
\mathbf{z}_3 &= \mathbf{a}_2\mathbf{W}_2 + \mathbf{b}_2 \\
\mathbf{a}_3 &= \hat{\mathbf{y}} = softmax(\mathbf{z}_3) \\
J &= CE(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i y_i \log \hat{y}_i
\end{aligned} \tag{10}$$

Using the chain rule $\frac{\partial J}{\partial \mathbf{x}}$ can be expressed in terms of the following chain:

$$\frac{\partial J}{\partial \mathbf{x}} = \frac{\partial J}{\partial \mathbf{z}_3}\frac{\partial \mathbf{z}_3}{\partial \mathbf{a}_2}\frac{\partial \mathbf{a}_2}{\partial \mathbf{z}_2}\frac{\partial \mathbf{z}_2}{\partial \mathbf{x}} \tag{11}$$

Computing these terms:

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{z}_3} &= \hat{\mathbf{y}} - \mathbf{y} = \delta_3 \\
\frac{\partial J}{\partial \mathbf{z}_3}\frac{\partial z_3}{\partial \mathbf{a}_2} &= \delta_3 \mathbf{W}_2^T \\
\frac{\partial J}{\partial \mathbf{z}_3}\frac{\partial z_3}{\partial \mathbf{a}_2}\frac{\partial \mathbf{a}_2}{\partial \mathbf{z}_2} &= (\delta_3 \mathbf{W}_2^T) \circ \dot{\sigma}(\mathbf{z_2}) = \delta_2 \\
\frac{\partial J}{\partial \mathbf{z}_3}\frac{\partial \mathbf{z}_3}{\partial \mathbf{a}_2}\frac{\partial \mathbf{a}_2}{\partial \mathbf{z}_2}\frac{\partial \mathbf{z}_2}{\partial \mathbf{x}} &= \delta_2 \mathbf{W}_1^T
\end{aligned} \tag{12}$$

(d) There are $D_x H$ parameters in $\mathbf{W}_1$, H parameters in $\mathbf{b_1}$, HD_y parameters in $\mathbf{W}_2$ and D_y parameters in b_2. This makes for a total of $H(D_x+1)+D_y(H+1)$ parameters in the network.

3 word2vec

(a) Considering the skip-gram model with softmax activation

$$\hat{y}_i = \frac{e^{\mathbf{u}_i^T \mathbf{v}_c}}{\sum_w^W e^{\mathbf{u}_w^T \mathbf{v}_c}} \tag{13}$$

with a given input/context word-vector/embedding $\mathbf{v}_c$ ($n \times 1$ dimensional column vector), the output word-vector $\mathbf{u}_i$ ($n \times 1$ dimensional column vector), a vocabulary of W words $w_i, i \in [1, .., W]$ and n denoting the number of features in each word-vector.

Each $\hat{y}_i$ denotes the probability (i.e. it is a scalar value) of w_i being the corresponding target word for input w_c and its word vector $\mathbf{v}_c$. By computing $\hat{y}_i$ for each i a $W \times 1$ dimensional output probability vector $\hat{\mathbf{y}}$ is calculated which is the probability distribution over all words in the vocabulary of being the target word for input w_c.

Now taking the cross-entropy loss of this prediction with a one-hot ground truth vector $\mathbf{y}$ ($W \times 1$ dimensional column vector) denoting the actual target word for input w_c results in the following.

$$J = CE(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i y_i \log \hat{y}_i = -\log \hat{y}_i \tag{14}$$

Substituting $\hat{y}_i$ for equation 13 yields

$$J = CE(\mathbf{y}, \hat{\mathbf{y}}) = -\log \frac{e^{\mathbf{u}_i^T \mathbf{v}_c}}{\sum_w^W e^{\mathbf{u}_w^T \mathbf{v}_c}} = -\mathbf{u}_i^T \mathbf{v}_c + \log \sum_w^W e^{\mathbf{u}_w^T \mathbf{v}_c} \tag{15}$$

This calculates the loss w.r.t. each scalar input $\mathbf{u}_i^T \mathbf{v}_c$. By substituting $\mathbf{z}_i = \mathbf{u}_i^T \mathbf{v}_c$, the same expression as 16 is found.

$$CE(\mathbf{y}, \hat{\mathbf{y}}) = -\mathbf{z}_i + \log \sum_w^W e^{\mathbf{z}_j} \tag{16}$$

Now $\mathbf{z}_i$ is only one element of the input to the softmax, corresponding to one particular output vector $\mathbf{u}_i$. This operation can be vectorized, considering all output vectors at the same time, by replacing $\mathbf{u}_i$ with $\mathbf{U} = [u_1, u_2, ..., u_W]$ ($n \times W$ dimensional matrix). Thus $\mathbf{Z} = \mathbf{U}^T \mathbf{v}_c$. The derivative $\frac{\partial}{\partial \mathbf{Z}} CE(\mathbf{y}, \hat{\mathbf{y}})$ is then the same as in equation 9

$$\frac{\partial}{\partial \mathbf{Z}} CE(\mathbf{y}, \hat{\mathbf{y}}) = \hat{\mathbf{y}} - \mathbf{y} \tag{17}$$

And the desired gradient $\frac{\partial J}{\partial \mathbf{v}_c}$ can then be calculated as

$$\frac{\partial J}{\partial \mathbf{v}_c} = \frac{\partial J}{\partial \mathbf{Z}} \frac{\partial \mathbf{Z}}{\partial \mathbf{v}_c} = \mathbf{U}(\hat{\mathbf{y}} - \mathbf{y}) \tag{18}$$

(b) Following the same reasoning as in (a), the gradient $\frac{\partial J}{\partial \mathbf{U}}$ can be found as

$$\frac{\partial J}{\partial \mathbf{U}} = \frac{\partial J}{\partial \mathbf{Z}} \frac{\partial \mathbf{Z}}{\partial \mathbf{U}} = (\hat{\mathbf{y}} - \mathbf{y})\mathbf{v}_c^T \tag{19}$$

(c) Instead of the cross-entropy loss over the entire softmax distribution, a negative sampling loss can be used

$$J_{neg} = -\log \sigma(\mathbf{u}_o^T \mathbf{v}_c) - \sum_{k=1}^{K} \log \sigma(-\mathbf{u}_k^T \mathbf{v}_c) \tag{20}$$

with subscript o denoting the index of the true target word w_o and K being the number of negative samples that are sampled from the vocabulary in a way that $o \notin \{1, .., K\}$.

In order to be able to train the word-vectors with this cost function, again the gradients $\frac{\partial J_{neg}}{\partial \mathbf{v}_c}$, $\frac{\partial J_{neg}}{\partial \mathbf{u}_o}$ and $\frac{\partial J_{neg}}{\partial \mathbf{u}_k}$ are needed.

$$\begin{aligned}
\frac{\partial J_{neg}}{\partial \mathbf{v}_c} &= -\frac{1}{\sigma(\mathbf{u}_o^T \mathbf{v}_c)} \dot{\sigma}(\mathbf{u}_o^T \mathbf{v}_c)\mathbf{u}_o - \sum_{k=1}^{K} \frac{1}{\sigma(-\mathbf{u}_k^T \mathbf{v}_c)} \dot{\sigma}(-\mathbf{u}_k^T \mathbf{v}_c)\mathbf{u}_k \\
&= -\frac{\sigma(\mathbf{u}_o^T \mathbf{v}_c)(1 - \sigma(\mathbf{u}_o^T \mathbf{v}_c))}{\sigma(\mathbf{u}_o^T \mathbf{v}_c)} \mathbf{u}_o - \sum_{k=1}^{K} \frac{\sigma(-\mathbf{u}_k^T \mathbf{v}_c)(1 - \sigma(-\mathbf{u}_k^T \mathbf{v}_c))}{\sigma(-\mathbf{u}_k^T \mathbf{v}_c)} \mathbf{u}_k \\
&= (\sigma(\mathbf{u}_o^T \mathbf{v}_c) - 1)\mathbf{u}_o - \sum_{k=1}^{K} (\sigma(-\mathbf{u}_k^T \mathbf{v}_c) - 1)\mathbf{u}_k
\end{aligned} \tag{21}$$

$$\frac{\partial J_{neg}}{\partial \mathbf{u}_o} = (\sigma(\mathbf{u}_o^T \mathbf{v}_c) - 1)\mathbf{v}_c^T \tag{22}$$

$$\frac{\partial J_{neg}}{\partial \mathbf{u}_k} = -(\sigma(-\mathbf{u}_k^T \mathbf{v}_c) - 1)\mathbf{v}_c^T \tag{23}$$

This negative sampling objective function is much more efficient to compute because we only need to sum over K elements to calculate a cost for the negative samples instead of summing over the entire vocabulary with W items for each training sample. Also computing the dot product of $\mathbf{u}_i^T \mathbf{v}_c$ for only $K + 1$ instead of for the entire output vector matrix $\mathbf{U}^T \mathbf{v}_c$ is more efficient too.

(d) The gradients for all of the word-vectors in skip-gram given a set of context words (i.e. target words in skip-gram) $[w_{c-m}, .., w_{c+m}]$ where m is the context size and denoting $F(\mathbf{o}, \mathbf{v}_c)$ as a general function for both objective functions presented previously can now be computed as follows.

$$\frac{\partial J(w_{c-m,...,c+m})}{\partial \mathbf{U}} = \sum_{-m \leq j \leq m, j \neq 0} \frac{\partial F(w_{c+j}, \mathbf{v}_c)}{\partial \mathbf{U}} \tag{24}$$

$$\frac{\partial J(w_{c-m,...,c+m})}{\partial \mathbf{v}_c} = \sum_{-m \leq j \leq m, j \neq 0} \frac{\partial F(w_{c+j}, \mathbf{v}_c)}{\partial \mathbf{v}_c} \tag{25}$$

$$\frac{\partial J(w_{c-m,...,c+m})}{\partial \mathbf{v}_j} = 0, \text{for all } j \neq c \tag{26}$$

This means that the overall gradients of an entire target context (i.e. several words) w.r.t. the output vectors U are simply the sum of the gradients of each target word w.r.t. U.

Likewise, the gradients of the entire context w.r.t. the predicted word vector $\mathbf{v}_c$ (i.e. the middle word within the context in the skip-gram model) are also simply the sum of the gradients of each target word w.r.t. $\mathbf{v}_c$. Furthermore, the gradients w.r.t. every other $\mathbf{v}_j, j \neq c$ is simply 0. Thus, the input word vectors are only getting updated for the specific current input word w_c.