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:

$$\begin{aligned}
\text{softmax}(\mathbf{x} + c)_j &= \frac{e^{(x_j+c)}}{\sum_{k=1} e^{(x_k+c)}} && (3)\\
&= \frac{e^c}{e^c} \frac{e^{(x_j)}}{sum_{k=1} e^{(x_j)}} && (4)\\
&= \text{softmax}(\mathbf{x})_j && (5)
\end{aligned}$$

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,

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

$$\begin{aligned}\frac{\partial(\text{CE})}{\partial \theta_k} &= \frac{\partial(\text{CE})}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial \theta_k} && (10)\\ &= -\frac{y_j}{\hat{y}_i}\frac{\partial \hat{y}_i}{\partial \theta_k} && (11)\end{aligned}$$

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

$$\begin{aligned}\text{if } i = k : \frac{\partial y_i}{\partial \theta_i} &= \frac{\partial \frac{e^{\theta_i}}{\Sigma_\theta}}{\partial \theta_i} && (12)\\ &= \frac{e^{\theta_i}\Sigma_\theta - e^{\theta_i}e^{\theta_i}}{\Sigma_\theta^2} && (13)\\ &= \frac{e^{\theta_i}}{\Sigma_\theta}\frac{\Sigma_\theta - e^{\theta_i}}{\Sigma_\theta} && (14)\\ &= \frac{e^{\theta_i}}{\Sigma_\theta}(1 - \frac{e^{\theta_i}}{\Sigma_\theta}) = \hat{y}_i(1 - \hat{y}_i) && (15)\end{aligned}$$

$$\begin{aligned}
\text{if } i \neq k : \frac{\partial y_i}{\partial \theta_j} &= \frac{\partial \frac{e^{\theta_i}}{\Sigma_\theta}}{\partial \theta_j} && (16)\\
&= \frac{0 - e^{\theta_i} e^{\theta_j}}{\Sigma_\theta^2} && (17)\\
&= -\frac{e^{\theta_i}}{\Sigma_\theta} \frac{e^{\theta_j}}{\Sigma_\theta} && (18)\\
&= -\hat{y}_i y_k && (19)
\end{aligned}$$

Combining Equations 9, 15, 19, yields

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

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

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

Which is equivalent to

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