Derivative of softmax:

Suppose the tuple $(x^{(i)}, y^{(i)})$ consists of the i^{th} training example and we define the loss for the i^{th} training example as follows:

$$L_i(\theta) = -\log\left(\frac{e^{a_{y^{(i)}}(x^{(i)})}}{\sum_j e^{a_j(x^{(i)})}}\right)$$

where, $a_j(x^{(i)}) = w_j^T x^{(i)}$; $w_j \in \mathbb{R}^d$, $x^{(i)} \in \mathbb{R}^d$

and $a_{y^{(i)}}(x^{(i)})$ is the score corresponding to the correct class of sample $x^{(i)}$.

Let, $$\sigma_{y^{(i)}}(x^{(i)}) = \frac{e^{a_{y^{(i)}}(x^{(i)})}}{\sum_j e^{a_j(x^{(i)})}}$$

then from Discussion 3, we know

$$\frac{\partial \sigma_{y^{(i)}}(x^{(i)})}{\partial a_j(x^{(i)})} = \begin{cases} \sigma_{y^{(i)}}(x^{(i)})[1-\sigma_{y^{(i)}}(x^{(i)})], & y^{(i)} = j \\ -\sigma_{y^{(i)}}(x^{(i)})\,\sigma_j(x^{(i)}), & y^{(i)} \neq j \end{cases}$$

Then using chain rule,

$$\frac{\partial L_i(\theta)}{\partial a_j(x^{(i)})} = \begin{cases} \sigma_{y^{(i)}}(x^{(i)}) - 1, & y^{(i)} = j \\ \sigma_j(x^{(i)}), & y^{(i)} \neq j \end{cases}$$

$$\frac{\partial L_i(\theta)}{\partial w_j} = \begin{cases} [\sigma_{y^{(i)}}(x^{(i)}) - 1]\,x^{(i)}, & y^{(i)} = j \\ \sigma_j(x^{(i)})\,x^{(i)}, & y^{(i)} \neq j \end{cases}$$

Vectorization:

Suppose we define the matrix m as follows:

$$A = \begin{bmatrix} - a_1 - \\ - a_2 - \\ \vdots \\ - a_n - \end{bmatrix}$$

where $a_i \in \mathbb{R}^{1 \times D}$. Hence $A \in \mathbb{R}^{n \times D}$

we also define the matrix B as follows:

$$B = \begin{bmatrix} - b_1 - \\ - b_2 - \\ \vdots \\ - b_m - \end{bmatrix}$$

where $b_i \in \mathbb{R}^{m \times D}$. Hence $B \in \mathbb{R}^{m \times D}$.

Now suppose we want to compute the matrix $P \in \mathbb{R}^{n \times m}$ from A and B, where the entries of P are defined as follows:

$$P(i,j) = \| A(i,:) - B(j,:) \|_2^2$$

Therefore, the $(i,j)^{th}$ entry of P is the squared L-2 distance between the i^{th} row of A and the j^{th} row of B.

Let's expand the expression for $P(i,j)$:

$$P(i,j) = [A(i,:) - B(j,:)][A(i,:) - B(j,:)]^T$$

$$P(i,j) = A(i,:)A(i,:)^T + B(j,:)B(j,:)^T - 2A(i,:)B(j,:)^T$$

We can use the above expression and for loop to fill out the matrix P. However, we can use vectorization to construct P without using for loops. To use vectorization we make the following observations:

(i) $A(i,:)A(i,:)^T$ is the squared 2-Norm of $i^{\underline{th}}$ row of A. we can store the squared 2-Norms of all the rows of A in a vector A_norm:

A_norm = np.sum(A^2, axis=1)

A_norm $\in \mathbb{R}^n$

(ii) Similarly,

$B(j,:)\,B(j,:)^T$

is the squared 2-Norm of jth row of B. We can store the squared 2-Norms of all the rows of B in a vector B_norm:

B_norm = np.sum(B², axis=1)

$B\_norm \in \mathbb{R}^m$

(iii) $A(i,:)B(j,:)^T$

is the dot product between the i^{th} row of A and the j^{th} row of B. We can store all the dot-products in a matrix MN_dot:

$$AB\_dot = AB^T$$

$$AB\_dot \in \mathbb{R}^{n \times m}$$

(iv) Finally, we can use broadcasting to construct P:

$$P = A\_norm + B\_norm - 2\,AB\_dot$$