# Rohan Mahendra Chaudhari

**USCID : 6675-5653-85**

## 1 Problem 1 - Multiclass Perceptron

### 1.1 To optimize this loss function, we need to first derive its gradient. Specifically, for each n $\in$ [N] and c $\in$ [C], write down the partial derivative $\frac{\partial F_n}{\partial \boldsymbol{w}_c}$ (and the reasoning). For simplicity, you can assume that for any n, $\boldsymbol{w}_1^T\boldsymbol{x}_n, ..., \boldsymbol{w}_C^T\boldsymbol{x}_n$ are always C distinct values (so that there is no tie when taking max over them, and consequently no non-differentiable points needed to be considered).

**Solution**

To optimize the said loss function we must consider multiple cases before writing down its partial derivative.

Cases:

$$F_n(\boldsymbol{w}_1, ...., \boldsymbol{w}_n) = \left\{ \begin{array}{ll} 0, & max_{y \neq y_n}(\boldsymbol{w}_y^T\boldsymbol{x}_n) < \boldsymbol{w}_{y_n}^T\boldsymbol{x}_n \\ \boldsymbol{w}_c^T\boldsymbol{x}_n - \boldsymbol{w}_{y_n}^T\boldsymbol{x}_n, & max_{y \neq y_n}(\boldsymbol{w}_y^T\boldsymbol{x}_n) > \boldsymbol{w}_{y_n}^T\boldsymbol{x}_n \end{array} \right\}$$

Here in the first case when $max_{y \neq y_n}(\boldsymbol{w}_y^T\boldsymbol{x}_n) < \boldsymbol{w}_{y_n}^T\boldsymbol{x}_n$ we can conclude that the difference of the given equation would be negative and thus 0 would be picked as the maximum value and thus the output would be 0 as stated.

As in the second case, when $max_{y \neq y_n}(\boldsymbol{w}_y^T\boldsymbol{x}_n) > \boldsymbol{w}_{y_n}^T\boldsymbol{x}_n$ we can conclude that there exists a class 'c' (here, y=c) which returns a value greater than 0 for $max_{y \neq y_n}(\boldsymbol{w}_y^T\boldsymbol{x}_n) - \boldsymbol{w}_{y_n}^T\boldsymbol{x}_n$.

Thus,

$$\frac{\partial F_n}{\partial \boldsymbol{w}_c} = \left\{ \begin{array}{ll} 0, & max_{y \neq y_n}(\boldsymbol{w}_y^T\boldsymbol{x}_n) < \boldsymbol{w}_{y_n}^T\boldsymbol{x}_n \\ \boldsymbol{x}_n, & max_{y \neq y_n}(\boldsymbol{w}_y^T\boldsymbol{x}_n) > \boldsymbol{w}_{y_n}^T\boldsymbol{x}_n \end{array} \right\}$$

Here, the first partial derivative would return 0 whereas we know that $\frac{\partial \boldsymbol{w}_c^T\boldsymbol{x}_n}{\partial \boldsymbol{w}_c} = \boldsymbol{x}_n$, thus the second partial derivative would return $\boldsymbol{x}_n$.

**1.2 Similarly to the binary case, multiclass perceptron is simply applying SGD with learning rate 1 to minimize the multiclass perceptron loss. Based on this information, fill in the missing details in the repeat-loop of the algorithm below (your solution cannot contain implicit quantities such as $\nabla F_n(\mathrm{w})$; instead, write down the exact formula based on your solution from the last question).**

**Solution**

---

**Algorithm 1** Multiclass Perceptron

---

**Input** : A Training set $(\boldsymbol{x}_1, y_1), ...., (\boldsymbol{x}_N, y_N)$
**Initialization** : $\boldsymbol{w}_1 = .... = \boldsymbol{w}_C = 0$
**Repeat** :
  **for** each example in Input $(\boldsymbol{x}_i, y_i)$ do
    Compute the predicted label $\hat{y}_i = argmax_c(\boldsymbol{w}_c^T \boldsymbol{x}_i)$
    **if** $\hat{y}_i \neq y_i$ ***then***
      **if** $max_{y_i \neq y_n}(\boldsymbol{w}_{y_i}^T \boldsymbol{x}_i) < \boldsymbol{w}_{y_n}^T \boldsymbol{x}_i$ **then**
        Increase the score for correct class $\boldsymbol{w}_{y_i} = \boldsymbol{w}_{y_i} + 0$
        Decrease the score for predicted class $\boldsymbol{w}_{\hat{y}_i} = \boldsymbol{w}_{\hat{y}_i} - 0$
      **elif** $max_{y_i \neq y_n}(\boldsymbol{w}_{y_i}^T \boldsymbol{x}_i) > \boldsymbol{w}_{y_n}^T \boldsymbol{x}_i$ **then**
        Increase the score for correct class $\boldsymbol{w}_{y_i} = \boldsymbol{w}_{y_i} + \boldsymbol{x}_i$
        Decrease the score for predicted class $\boldsymbol{w}_{\hat{y}_i} = \boldsymbol{w}_{\hat{y}_i} - \boldsymbol{x}_i$
    **else**
      Don't change the score for correct class $\boldsymbol{w}_{y_i} = \boldsymbol{w}_{y_i}$
      Don't change the score for predicted class $\boldsymbol{w}_{\hat{y}_i} = \boldsymbol{w}_{\hat{y}_i}$
  **end for**

---

**1.3 At this point, you should find that the parameters $w_1, ..., \boldsymbol{w}_C$ computed by Multiclass Perceptron are always linear combinations of the training points $\boldsymbol{x}_1, ..., \boldsymbol{x}_N$, that is, $\boldsymbol{w}_c = \sum_{n=1}^N \alpha_{c,n} \boldsymbol{x}_n$ for some coefficient $\alpha_{c,n}$. Just like kernelized linear regression, this means that one can kernelize multiclass Perceptron as well for any given kernel function k(·,·). Based on this information, fill in the missing details in the repeat-loop of the algorithm below that maintains and updates the coefficient $\alpha_{c,n}$ for all c and n.**

**Solution**

**Algorithm 2** Multiclass Perceptron with Kernel Function k(·,·)

**Input** : A Training set $(\boldsymbol{x}_1, y_1), ...., (\boldsymbol{x}_N, y_N)$
**Initialize** : $\alpha_{c,n} = 0$ for all c $\in$ [C] and n $\in$ [N]
**Repeat** :
  **for** each example in Input $(\boldsymbol{x}_i, y_i)$ do
    // Here K$(\boldsymbol{x}_n, \boldsymbol{x}_i) = (\boldsymbol{x}_n.\boldsymbol{x}_i)$ and $\hat{y}_i = argmax_c(\boldsymbol{w}_c^T \boldsymbol{x}_i)$
    Compute the predicted label $\hat{y}_i = argmax_c(\sum_{n=1}^N \alpha_{c,n} K(\boldsymbol{x}_n, \boldsymbol{x}_i))$
    **if** $\hat{y}_i \neq y_i$ ***then***
      $\alpha_{y_i,i} = \alpha_{y_i,i} + 1$
    **end if**
  **end for**

# 2 Problem 2 - Backpropagation for CNN

## 2.1 Write down $\frac{\partial l}{\partial v_1}$ and $\frac{\partial l}{\partial v_2}$ (show the intermediate steps that use chain rule). You can use the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ to simplify your notation.

**Solution**
**For** $\frac{\partial l}{\partial v_1}$,
By applying chain rule,
We write the above equation as mentioned below since, l is a function of $\hat{y}$ and $\hat{y}$ is a function of $v_1$
$\frac{\partial l}{\partial v_1} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial v_1}$
Here, $\frac{\partial l}{\partial \hat{y}} = \frac{-ye^{-y\hat{y}}}{1+e^{-y\hat{y}}}$
And, $\frac{\partial \hat{y}}{\partial v_1} = o_1$
On multiple both of the above equations we get,
$\frac{\partial l}{\partial v_1} = \frac{-yo_1 e^{-y\hat{y}}}{1+e^{-y\hat{y}}}$
On simplifying using the sigmoid function,
$\sigma(y\hat{y}) = \frac{1}{1+e^{-y\hat{y}}}$
Thus,
$\frac{\partial l}{\partial v_1} = -yo_1 e^{-y\hat{y}} \sigma(y\hat{y})$

**For** $\frac{\partial l}{\partial v_2}$,
By applying chain rule,
We write the above equation as mentioned below since, l is a function of $\hat{y}$ and $\hat{y}$ is a function of $v_2$
$\frac{\partial l}{\partial v_2} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial v_2}$
Here, $\frac{\partial l}{\partial \hat{y}} = \frac{-ye^{-y\hat{y}}}{1+e^{-y\hat{y}}}$
And, $\frac{\partial \hat{y}}{\partial v_2} = o_2$
On multiple both of the above equations we get,
$\frac{\partial l}{\partial v_2} = \frac{-yo_2 e^{-y\hat{y}}}{1+e^{-y\hat{y}}}$

On simplifying using the sigmoid function,
$\sigma(y\hat{y}) = \frac{1}{1+e^{-y\hat{y}}}$
Thus,
$\frac{\partial l}{\partial v_2} = -yo_2 e^{-y\hat{y}} \sigma(y\hat{y})$

## 2.2 Write down $\frac{\partial l}{\partial w_1}$ and $\frac{\partial l}{\partial w_2}$ (show the intermediate steps that use chain rule). The derivative of the ReLU function is H(a) = I$[a > 0]$, which you can use directly in your answer.

**Solution**
**For** $\frac{\partial l}{\partial w_1}$
By applying chain rule,
We write the above equation as mentioned below since, l is a function of $\hat{y}$, $\hat{y}$ is a function of $o_1$ and $o_2$, $o_1$ is a function of $a_1$ & $o_2$ is a function of $a_2$, $a_1$ is a function of $w1$ & $a_2$ is a function of $w_1$.
$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial \hat{y}}[\frac{\partial \hat{y}}{\partial o_1}\frac{\partial o_1}{\partial a_1}\frac{\partial a_1}{\partial w_1} + \frac{\partial \hat{y}}{\partial o_2}\frac{\partial o_2}{\partial a_2}\frac{\partial a_2}{\partial w_1}]$
$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial \hat{y}}[v_1 H(a_1)x_1 + v_2 H(a_2)x_2]$
$\frac{\partial l}{\partial w_1} = \frac{-ye^{-y\hat{y}}}{1+e^{-y\hat{y}}}[v_1 H(a_1)x_1 + v_2 H(a_2)x_2]$
$\frac{\partial l}{\partial w_1} = \frac{-y}{e^{y\hat{y}}+1}[v_1 H(a_1)x_1 + v_2 H(a_2)x_2]$

**For** $\frac{\partial l}{\partial w_2}$
By applying chain rule,
We write the above equation as mentioned below since, l is a function of $\hat{y}$, $\hat{y}$ is a function of $o_1$ and $o_2$, $o_1$ is a function of $a_1$ & $o_2$ is a function of $a_2$, $a_1$ is a function of $w2$ & $a_2$ is a function of $w_2$.
$\frac{\partial l}{\partial w_2} = \frac{\partial l}{\partial \hat{y}}[\frac{\partial \hat{y}}{\partial o_1}\frac{\partial o_1}{\partial a_1}\frac{\partial a_1}{\partial w_2} + \frac{\partial \hat{y}}{\partial o_2}\frac{\partial o_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}]$
$\frac{\partial l}{\partial w_2} = \frac{\partial l}{\partial \hat{y}}[v_1 H(a_1)x_2 + v_2 H(a_2)x_3]$
$\frac{\partial l}{\partial w_1} = \frac{-ye^{-y\hat{y}}}{1+e^{-y\hat{y}}}[v_1 H(a_1)x_2 + v_2 H(a_2)x_3]$
$\frac{\partial l}{\partial w_1} = \frac{-y}{e^{y\hat{y}}+1}[v_1 H(a_1)x_2 + v_2 H(a_2)x_3]$

## 2.3 Using the derivations above, fill in the missing details of the repeat-loop of the Backpropagation algorithm below that is used to train this mini CNN.

**Solution**

**Algorithm 3** Backpropogation for the above mini CNN

**Input** : A Training set $(\boldsymbol{x}_1, y_1), ...., (\boldsymbol{x}_N, y_N)$, learning rate $\eta$

**Initialization** : set $w_1, w_2, v_1, v_2$ randomly

**Repeat** :

randomly pick an example $(\boldsymbol{x}_n, y_n)$

Forward Propagation:

Compute $a_1 = x_1 w_1 + x_2 w_2$

Compute $a_2 = x_2 w_1 + x_3 w_2$

Compute $o_1 = max(0, a_1)$

Compute $o_2 = max(0, a_2)$

Compute $\hat{y} = o_1 v_1 + o_2 v_2$

Backward Propagation:

Compute $\frac{\partial l}{\partial v_1} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial v_1} = -y o_1 e^{-y\hat{y}} \sigma(y\hat{y})$

Compute $\frac{\partial l}{\partial v_2} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial v_2} = -y o_2 e^{-y\hat{y}} \sigma(y\hat{y})$

Update Weight $v_1 = v_1 - \eta(-y o_1 e^{-y\hat{y}} \sigma(y\hat{y}))$

Update Weight $v_2 = v_2 - \eta(-y o_2 e^{-y\hat{y}} \sigma(y\hat{y}))$

Compute $\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial \hat{y}} [\frac{\partial \hat{y}}{\partial o_1} \frac{\partial o_1}{\partial a_1} \frac{\partial a_1}{\partial w_1} + \frac{\partial \hat{y}}{\partial o_2} \frac{\partial o_2}{\partial a_2} \frac{\partial a_2}{\partial w_1}] = \frac{-y}{e^{y\hat{y}}+1} [v_1 H(a_1) x_1 + v_2 H(a_2) x_2]$

Compute $\frac{\partial l}{\partial w_2} = \frac{\partial l}{\partial \hat{y}} [\frac{\partial \hat{y}}{\partial o_1} \frac{\partial o_1}{\partial a_1} \frac{\partial a_1}{\partial w_2} + \frac{\partial \hat{y}}{\partial o_2} \frac{\partial o_2}{\partial a_2} \frac{\partial a_2}{\partial w_2}] = \frac{-y}{e^{y\hat{y}}+1} [v_1 H(a_1) x_2 + v_2 H(a_2) x_3]$

Update Weight $w_1 = w_1 - \eta(\frac{-y}{e^{y\hat{y}}+1} [v_1 H(a_1) x_1 + v_2 H(a_2) x_2])$

Update Weight $w_2 = w_2 - \eta(\frac{-y}{e^{y\hat{y}}+1} [v_1 H(a_1) x_2 + v_2 H(a_2) x_3])$

# 3 Problem 3 - Kernel Composition

## 3.1 Prove that if $k_1, k_2 : R^D \times R^D \to R$ are both kernel functions, then $k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}')k_2(\boldsymbol{x}, \boldsymbol{x}')$ is a kernel function too. Specifically, suppose that $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$ are the corresponding mappings for $k_1$ and $k_2$ respectively. Construct the mapping $\boldsymbol{\phi}$ that certifies k being a kernel function.

**Solution**

To prove this specific theorem we need to make use of Mercer's theorem. Since $k_1$ and $k_2$ are valid kernels, with the help of Mercer we can state that the inner product would also be a kernel function. Let $\boldsymbol{\phi}_1$ be the feature map for $k_1$ and $\boldsymbol{\phi}_2$ be the feature map for $k_2$. Let $f_i(x)$ be the $i^{th}$ feature value under feature map $\boldsymbol{\phi}_1$ and $g_i(x)$ be the $i^{th}$ feature value under feature map $\boldsymbol{\phi}_2$.

$$\begin{gathered}
k_1(\boldsymbol{x}, \boldsymbol{x}')k_2(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{\phi}_1(\boldsymbol{x}) * \boldsymbol{\phi}_1(\boldsymbol{x}'))(\boldsymbol{\phi}_2(\boldsymbol{x}) * \boldsymbol{\phi}_2(\boldsymbol{x}')) \\
= (\textstyle\sum_{i=1}^{\infty} f_i(x)f_i(x'))(\sum_{j=1}^{\infty} g_j(x)g_j(x')) \\
= \textstyle\sum_{i,j} f_i(x)f_i(x')g_j(x)g_j(x') \\
= \textstyle\sum_{i,j} (f_i(x)f_i(x'))(g_j(x)g_j(x'))
\end{gathered}$$

We can now define a feature map $\boldsymbol{\phi}$ with a feature $h_{i,j}(x)$ or each pair $\langle i, j \rangle$ is defined as follows.

$$h_{i,j}(x) = f_i(x)g_j(x)$$

We then have $k_1(\boldsymbol{x}, \boldsymbol{x}')k_2(\boldsymbol{x}, \boldsymbol{x}')$ is $\boldsymbol{\phi}(x)\boldsymbol{\phi}(x')$ where the inner product sums over all pairs $\langle i, j \rangle$.