CSCI 567, Fall 2021
Haipeng Luo

Written Assignment #2

Due: Sep 28, 2021, 11:59 pm, PT

Instructions

Submission: Assignment submission will be via courses.uscden.net. By the submission date, there will be a folder set up in which you can submit your files. Please be sure to follow all directions outlined here.

You can submit multiple times, but only *the last submission* counts. That means if you finish some problems and want to submit something first and update later when you finish, that's fine. In fact you are encouraged to do this: that way, if you forget to finish the homework on time or something happens, you still get credit for whatever you have turned in.

Problem sets must be typewritten or neatly handwritten when submitted. In both cases, your submission must be a single PDF. Please also follow the rules below:

- The file should be named as `firstname_lastname_USCID.pdf` (e.g., Joe_Doe_1234567890.pdf).
- Do not have any spaces in your file name when uploading it.
- Please include your name and USCID in the header of the report as well.

Total points: 40 points

Notes on notation:

- Unless stated otherwise, scalars are denoted by small letter in normal font, vectors are denoted by small letters in bold font and matrices are denoted by capital letters in bold font.
- $\|.\|$ means L2-norm unless specified otherwise i.e. $\|.\| = \|.\|_2$

Problem 1 Multiclass Perceptron (16 points)

Recall that a linear model for a multiclass classification problem with C classes is parameterized by C weight vectors $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_C \in \mathbb{R}^D$. In the class we derive the multiclass logistic regression by minimizing the multiclass logistic loss. In this problem you need to derive the multiclass perceptron algorithm in a similar way. Specifically, the multiclass perceptron loss on a training set $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N) \in \mathbb{R}^D \times [C]$ is defined as

$$F(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_C) = \frac{1}{N} \sum_{n=1}^{N} F_n(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_C), \quad \text{where } F_n(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_C) = \max\left\{0, \max_{y \neq y_n} \boldsymbol{w}_y^{\mathrm{T}} \boldsymbol{x}_n - \boldsymbol{w}_{y_n}^{\mathrm{T}} \boldsymbol{x}_n \right\}.$$

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, $\boldsymbol{w}_1^{\mathrm{T}} \boldsymbol{x}_n, \ldots, \boldsymbol{w}_C^{\mathrm{T}} \boldsymbol{x}_n$ 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). **(8 points)**

   For each n, let $\hat{y}_n = \arg\max_{y \in [C]} \boldsymbol{w}_y^{\mathrm{T}} \boldsymbol{x}_n$. Then by definition, F_n can be written as

   $$\begin{cases} 0, & \text{if } \hat{y}_n = y_n, \\ \boldsymbol{w}_{\hat{y}_n}^{\mathrm{T}} \boldsymbol{x}_n - \boldsymbol{w}_{y_n}^{\mathrm{T}} \boldsymbol{x}_n, & \text{else.} \end{cases}$$

   Its partial derivative with respect to $\boldsymbol{w}_c$ is then

   $$\begin{cases} \boldsymbol{0}, & \text{if } \hat{y}_n = y_n, \\ \boldsymbol{x}_n, & \text{else if } c = \hat{y}_n, \\ -\boldsymbol{x}_n, & \text{else if } c = y_n, \\ \boldsymbol{0}, & \text{else.} \end{cases}$$

   Rubrics: 2 points for each of the 4 cases. There are many other ways to write this, such as using indicator functions. It is of course also possible to combine some of these cases (such as the first and the fourth ones).

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 $\nabla F_n(\boldsymbol{w})$; instead, write down the exact formula based on your solution from the last question). **(4 points)**

Algorithm 1: Multiclass Perceptron

```
Input: A training set $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$
Initialization: $\boldsymbol{w}_1 = \cdots = \boldsymbol{w}_C = \boldsymbol{0}$
Repeat:
    randomly pick an example $(\boldsymbol{x}_n, y_n)$ and compute $\hat{y}_n = \arg\max_{y \in [C]} \boldsymbol{w}_y^{\mathrm{T}} \boldsymbol{x}_n$
    if $\hat{y}_n \neq y_n$ then
        $\boldsymbol{w}_{\hat{y}_n} \leftarrow \boldsymbol{w}_{\hat{y}_n} - \boldsymbol{x}_n$
        $\boldsymbol{w}_{y_n} \leftarrow \boldsymbol{w}_{y_n} + \boldsymbol{x}_n$
```

Rubrics: 1 point for randomly picking an example, 3 points for correctly implementing the rest of SGD (again, there are many equivalent ways of doing so). Do not deduct points if the gradient is wrong solely due to mistakes from the last question.

3. At this point, you should find that the parameters $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_C$ computed by Multiclass Perceptron are always linear combinations of the training points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, 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 $k(\cdot,\cdot)$. 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. **(4 points)**

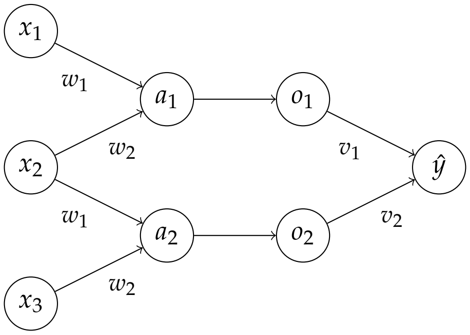
Algorithm 2: Multiclass Perceptron with kernel function $k(\cdot,\cdot)$

```
Input: A training set $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$
Initialize: $\alpha_{c,n} = 0$ for all $c \in [C]$ and $n \in [N]$
Repeat:
    randomly pick an example $(\boldsymbol{x}_n, y_n)$ and compute $\hat{y}_n = \arg\max_{y \in [C]} \left(\sum_{m=1}^N \alpha_{y,m} k(\boldsymbol{x}_m, \boldsymbol{x}_n)\right)$
    if $\hat{y}_n \neq y_n$ then
        $\alpha_{\hat{y}_n,n} \leftarrow \alpha_{\hat{y}_n,n} - 1$
        $\alpha_{y_n,n} \leftarrow \alpha_{y_n,n} + 1$
```

Rubrics: 1 point for randomly picking an example, 3 points for correctly implementing the rest (again, there are many equivalent ways of doing so). Storing the kernel matrix to avoid repeated calculations is of course acceptable. Do not deduct points if the mistake is solely inherited from the first question. However, deduct 2 points if there are any operations involving the feature vectors other than feeding them to the kernel function (since that is one of the most important aspects of kernel methods).

Problem 2 Backpropagation for CNN (18 points)

Consider the following mini convolutional neural net, where (x_1, x_2, x_3) is the input, followed by a convolution layer with a filter (w_1, w_2), a ReLU layer, and a fully connected layer with weight (v_1, v_2).



More concretely, the computation is specified by

$$\begin{aligned} a_1 &= x_1 w_1 + x_2 w_2 \\ a_2 &= x_2 w_1 + x_3 w_2 \\ o_1 &= \max\{0, a_1\} \\ o_2 &= \max\{0, a_2\} \\ \hat{y} &= o_1 v_1 + o_2 v_2 \end{aligned}$$

For an example $(\boldsymbol{x}, y) \in \mathbb{R}^3 \times \{-1, +1\}$, the logistic loss of the CNN is

$$\ell = \ln(1 + \exp(-y\hat{y})),$$

which is a function of the parameters of the network: w_1, w_2, v_1, v_2.

1. Write down $\frac{\partial \ell}{\partial v_1}$ and $\frac{\partial \ell}{\partial v_2}$ (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. **(4 points)**

$$\begin{aligned} \frac{\partial \ell}{\partial v_1} &= \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial v_1} && \text{(1 point)} \\ &= \frac{-ye^{-y\hat{y}}}{1+e^{-y\hat{y}}} o_1 = -\sigma(-y\hat{y}) y o_1 = (\sigma(y\hat{y}) - 1) y o_1 && \text{(1 point)} \end{aligned}$$

$$\begin{aligned} \frac{\partial \ell}{\partial v_2} &= \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial v_2} && \text{(1 point)} \\ &= \frac{-ye^{-y\hat{y}}}{1+e^{-y\hat{y}}} o_2 = -\sigma(-y\hat{y}) y o_2 = (\sigma(y\hat{y}) - 1) y o_2 && \text{(1 point)} \end{aligned}$$

Either one of the last three expressions is acceptable.

2. Write down $\frac{\partial \ell}{\partial w_1}$ and $\frac{\partial \ell}{\partial w_2}$ (show the intermediate steps that use chain rule). The derivative of the ReLU function is $H(a) = \mathbb{I}[a > 0]$, which you can use directly in your answer. **(6 points)**

$$\begin{aligned} \frac{\partial \ell}{\partial w_1} &= \frac{\partial \ell}{\partial a_1} \frac{\partial a_1}{\partial w_1} + \frac{\partial \ell}{\partial a_2} \frac{\partial a_2}{\partial w_1} && \text{(1 point)} \\ &= \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_1} \frac{\partial o_1}{\partial a_1} \frac{\partial a_1}{\partial w_1} + \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_2} \frac{\partial o_2}{\partial a_2} \frac{\partial a_2}{\partial w_1} && \text{(1 point)} \\ &= (\sigma(y\hat{y}) - 1) y (v_1 H(a_1) x_1 + v_2 H(a_2) x_2) && \text{(1 point)} \end{aligned}$$

Similarly

$$\begin{aligned} \frac{\partial \ell}{\partial w_2} &= \frac{\partial \ell}{\partial a_1} \frac{\partial a_1}{\partial w_2} + \frac{\partial \ell}{\partial a_2} \frac{\partial a_2}{\partial w_2} && \text{(1 point)} \\ &= \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_1} \frac{\partial o_1}{\partial a_1} \frac{\partial a_1}{\partial w_2} + \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_2} \frac{\partial o_2}{\partial a_2} \frac{\partial a_2}{\partial w_2} && \text{(1 point)} \\ &= (\sigma(y\hat{y}) - 1) y (v_1 H(a_1) x_2 + v_2 H(a_2) x_3). && \text{(1 point)} \end{aligned}$$

Again, other equivalent expressions are acceptable.

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. **(8 points)**

```
Algorithm 3: Backpropagation for the above mini CNN
Input: A training set $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$, learning rate $\eta$
Initialize: set $w_1, w_2, v_1, v_2$ randomly
Repeat:
    randomly pick an example $(\boldsymbol{x}_n, y_n)$
    Forward propagation: compute                                        (4 points)
        $a_1 = x_{n1}w_1 + x_{n2}w_2, a_2 = x_{n2}w_1 + x_{n3}w_2$
        $o_1 = \max\{0, a_1\}, o_2 = \max\{0, a_2\}, \hat{y} = o_1 v_1 + o_2 v_2$
    Backward propagation: update                                        (4 points)
        $w_1 \leftarrow w_1 - \eta(\sigma(y_n\hat{y}) - 1)y_n(v_1 H(a_1)x_{n1} + v_2 H(a_2)x_{n2})$
        $w_2 \leftarrow w_2 - \eta(\sigma(y_n\hat{y}) - 1)y_n(v_1 H(a_1)x_{n2} + v_2 H(a_2)x_{n3})$
        $v_1 \leftarrow v_1 - \eta(\sigma(y_n\hat{y}) - 1)y_n o_1$
        $v_2 \leftarrow v_2 - \eta(\sigma(y_n\hat{y}) - 1)y_n o_2$
```

- Deduct 1 point for writing x_1, x_2, x_3 instead of x_{n1}, x_{n2}, x_{n3} in the forward propgagation.
- Deduct 1 point for writing x_1, x_2, x_3, y instead of $x_{n1}, x_{n2}, x_{n3}, y_n$ in the backward propgagation.
- Deduct 2 points if updating w_1/w_2 with the updated value of v_1/v_2.
- Do not deduct points for using the wrong gradients solely due to mistakes from previous two questions.

Problem 3 Kernel Composition (6 points)

Prove that if $k_1, k_2 : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ 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.

Let M_1 and M_2 be the dimension of the output of $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$ respectively. Then we have

$$k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}')k_2(\boldsymbol{x}, \boldsymbol{x}') = \left(\sum_{i=1}^{M_1} \phi_1(\boldsymbol{x})_i \phi_1(\boldsymbol{x}')_i\right)\left(\sum_{j=1}^{M_2} \phi_2(\boldsymbol{x})_j \phi_2(\boldsymbol{x}')_j\right) \qquad \text{(2 points)}$$

$$= \sum_{i=1}^{M_1}\sum_{j=1}^{M_2} \phi_1(\boldsymbol{x})_i \phi_1(\boldsymbol{x}')_i \phi_2(\boldsymbol{x})_j \phi_2(\boldsymbol{x}')_j = \sum_{i=1}^{M_1}\sum_{j=1}^{M_2} \left(\phi_1(\boldsymbol{x})_i \phi_2(\boldsymbol{x})_j\right)\left(\phi_1(\boldsymbol{x}')_i \phi_2(\boldsymbol{x}')_j\right) \qquad \text{(2 points)}$$

Therefore, $\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{\phi}_1(\boldsymbol{x})\boldsymbol{\phi}_2(\boldsymbol{x})^{\mathrm{T}}$. Writing it as an $M_1 M_2$-dimensional vector is acceptable. **(2 points)**