Security and Privacy in Machine Learning

Homework 1

Spring 2023

Javad Hezareh

98101074

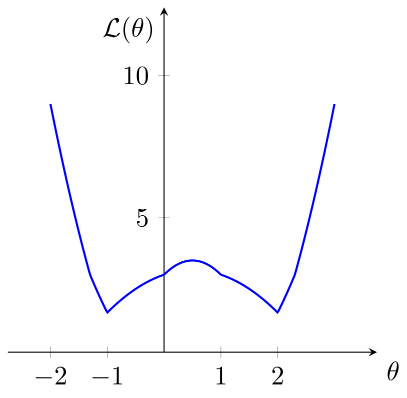
Contents

1 Cost Function

(a) Considering the MAE loss function and $f_\theta(x)$ we have:

$$\begin{aligned}\mathcal{L}(\theta) &= \frac{1}{3}\left(|3-\theta^2+\theta| + |0-2\theta^2+2\theta| + |6-3\theta^2+3\theta|\right)\\ &= \frac{|\theta^2-\theta-3| + 2|\theta^2-\theta| + 3|\theta^2-\theta-2|}{3}\end{aligned}$$

For the graph of $\mathcal{L}(\theta)$ we have:

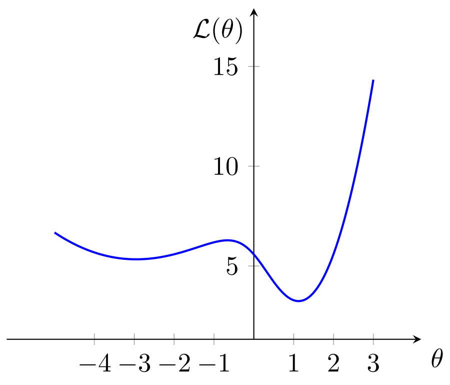


Candidate points for minimum are 2 and -1 where the value of the loss function is $\mathcal{L}(2) = 5/3$ and $\mathcal{L}(-1) = 5/3$. Therefore both of them are the local and global minima of $\mathcal{L}$. If we choose our learning rate enough small, then starting from any point on θ and using gradient descent algorithm we will end up on 2 or -1 (depend on the starting point) which both of them are global minima, therefore using small enough learning rate will always lead us to the global minima of this loss function.

(b) Using MSE loss function and $f_\theta(x) = \ln(1+e^{x\theta})$ we have:

$$\mathcal{L}(\theta) = \frac{1}{3}\left[(0-\ln(1+e^{\theta}))^2 + (3-\ln(1+e^{-\theta}))^2 + (4-\ln(1+e^{3\theta}))^2\right]$$

Therefore the graph of $\mathcal{L}$ will be:



So $\theta = -3$ will be a local minima with value $\mathcal{L}(-3) =?$, and $\theta = 1$ will be the global minima with value $\mathcal{L}(1) =?$. In contrast to the previous part, using gradient descent in this setup will not always converge to the global minima and it depends on the starting point. For example if we start from a point less than -3 it will converge to $\theta = -3$ which is not the global minima.

(c) In deep networks we expect to have a lot of global and local minima where local minima points are not necessarily global as well. One solution to find the global minima is to use stochastic gradient descent which will allow us to skip some local minimas.

2 Training Model

(a) Performing normalization on input data will help the logistic regression model to avoid vanishing gradient and therefore converge faster. It will also help to increase the accuracy of the model a bit. Also, if we don't perform normalization, we can't interpret the coefficients as indicators of feature importance. Because we need to have the same scaling for different features to be able to compare coefficients norm.

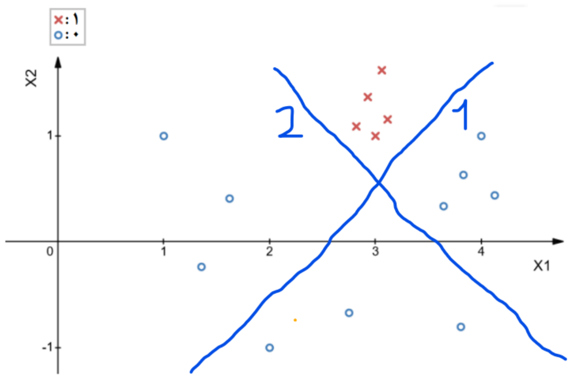
(b) Using regularization in this model will help us to reach a better model in the sense of generalization error and performance.

(c) A network with linear activation has no difference with a linear model. So using a 20 layer network with linear activation function will lead to a simple linear model which is not capable of capturing complex patterns in different data. On the other hand a 3 layer non-linear network has this ability to model the non-linearity and capture complex patterns as well. The later one will also has much less weights and parameters, and therefore is more computation efficient (in case of efficient calculation of non-linearity).

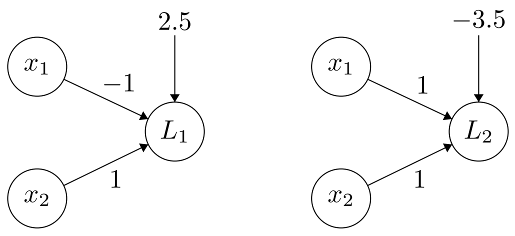
3 Perceptron Network

(a) This is not possible to classify the given points with just one line. At least we need to line to be able to do so.

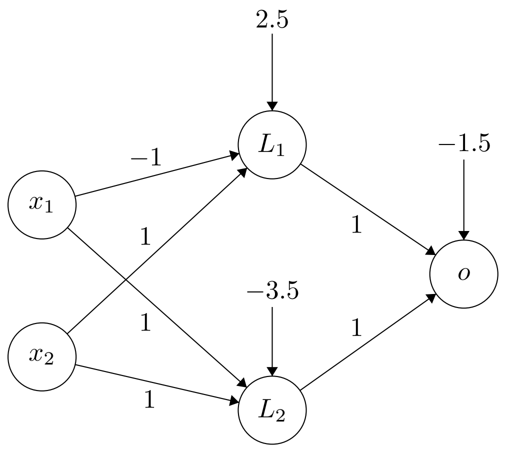
(b) We will use two hyperplane crossing from the center of our data and then if one point be in the right side of both of them, then that point will be classified as class 1 and else as class 0.



For each point on line 1 such as (x_1, x_2) we have $(-1, 1) \cdot (x_1, x_2) + 2.5 = 0$, therefore for each point on the right side of this line we have $w_1 \cdot x + 2.5 > 0$ where $w_1 = (-1, 1)$. For each point on line 2 such as (x_1, x_2) we have $(1, 1) \cdot (x_1, x_2) - 3.5 = 0$, therefore for each point on the right side of this line we have $w_2 \cdot x - 3.5 > 0$ where $w_2 = (1, 1)$. Hence we can use following networks to find out the relative position of each point respect to each line: (all activation functions are threshold)



Now with the following network we can classify data points:



4 Softmax and Cross-Entropy

By substituting the softmax function in our loss we have:

$$\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{i=1}^{M} y_i \log\left(\frac{e^{x_i}}{\sum_{m=1}^{M} e^{x_m}}\right) = -\sum_{i=1}^{M} y_i \left(x_i - \log\left(\sum_{m=1}^{M} e^{x_m}\right)\right)$$

Now for the derivative of loss w.r.t x we have:

$$\begin{aligned}\frac{\partial \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})}{x_j} &= -\sum_{i=1}^{M} y_i \frac{\partial}{\partial x_j}\left(x_i - \log\left(\sum_{m=1}^{M} e^{x_m}\right)\right) \\ &= -\sum_{i=1}^{M} y_i \left(1_{i=j} - \frac{e^{x_j}}{\sum_{m=1}^{M} e^{x_m}}\right)\end{aligned}$$

If we assume that y_is are probability distribution, then they will sum up to 1 and hence we have:

$$\frac{\partial \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})}{x_j} = -y_j + \frac{e^{x_j}}{\sum_{m=1}^{M} e^{x_m}} = -y_j + \hat{y}_j$$

$$\Longrightarrow \boxed{\frac{\partial \mathcal{L}(\hat{y}, y)}{\boldsymbol{x}} = \hat{\boldsymbol{y}} - \boldsymbol{y}}$$

5 Backpropagation

(a) If we assume the middle layer has p neuron, then we have:

$$\begin{cases} \boldsymbol{W_1} \in \mathbb{R}^{p\times n} \\ \boldsymbol{b_1} \in \mathbb{R}^{p\times 1} \\ \boldsymbol{W_2} \in \mathbb{R}^{1\times p} \\ \boldsymbol{b_2} \in \mathbb{R} \end{cases}$$

(b) Let's assume that $\boldsymbol{X} \in \mathbb{R}^{n\times b}$ where $b = 2000$. Then this will not change the weigh matrices and we only need to copy biases in b columns. Therefore:

$$\begin{cases} \boldsymbol{W_1} \in \mathbb{R}^{p\times n} \\ \boldsymbol{b_1} \in \mathbb{R}^{p\times b} \\ \boldsymbol{W_2} \in \mathbb{R}^{1\times m} \\ \boldsymbol{b_2} \in \mathbb{R}^{1\times b} \end{cases}$$

(c) Let's derive this formulas for one data point, in other words $\boldsymbol{x}$ is not a batch of inputs.

For $\partial \boldsymbol{z_1}/\partial \boldsymbol{W_1}$, we need to find the partial derivative of each element of $\boldsymbol{z_1}$ with respect to each element of $\boldsymbol{W_1}$. But we know z_{1k} only depends on the k-th row of $\boldsymbol{W_1}$, so $\partial z_{1k}/\partial W_{1ij} = 0$ if we have $k \neq i$. In case of $k = i$ then z_{1k} is the inner product of k-th row of $\boldsymbol{W_1}$ and $\boldsymbol{x}$, therefore $\partial z_{1k}/\partial W_{1ij} = x_j$ if $k = i$. So we have:

$$\frac{z_{1k}}{W_{1ij}} = \begin{cases} 0 & k \neq i \\ x_j & k = i \end{cases}$$

For $\partial \boldsymbol{a_1}/\partial \boldsymbol{z_1}$, as a_{1i} only depends on z_{1i}, we have:

$$\frac{\partial a_{1i}}{\partial z_{1j}} = \begin{cases} 0 & i \neq j \\ 1 & i = j \text{ and } z_{1j} \geq 0 \\ a & i = j \text{ and } z_{1j} < 0 \end{cases}$$

From the network architecture we know that z_2 is an scalar and hence we have:

$$\frac{\partial z_2}{\partial \boldsymbol{a_1}} = \boldsymbol{W_2}^T$$

Using sigmoid function derivative, for $\partial \hat{y}^{(i)}/\partial z_2$ we have:

$$\frac{\partial \hat{y}^{(i)}}{\partial z_2} = \hat{y}^{(i)}(1 - \hat{y}^{(i)})$$

For the derivative of loss w.r.t $\hat{y}$ we have:

$$\frac{\partial \mathcal{L}}{\partial \hat{y}^{(i)}} = \frac{1}{m} \log\left(\frac{1 - \hat{y}^{(i)}}{\hat{y}^{(i)}}\right)$$

6 Regularization

(a) Using this regularization will make our algorithm to decrease the l_2-norm of the weigh vector to reach less loss value. Therefore it is called ”weight decay”.

(b) By differentiating with respect to $\boldsymbol{w}$ and find its zeros we can find the optimal $\boldsymbol{w}^*$:

$$\frac{\partial}{\partial \boldsymbol{w}}\mathcal{L} = \frac{1}{N}\frac{\partial}{\partial \boldsymbol{w}}\|X\boldsymbol{w} - y\|^2 + \lambda\frac{\partial}{\partial \boldsymbol{w}}\|\boldsymbol{w}\|^2$$

We can differentiate the first term by expanding the norm with inner product:

$$\|X\boldsymbol{w} - y\|^2 = (X\boldsymbol{w} - y)^T(X\boldsymbol{w} - \boldsymbol{y})$$

$$= \|X\boldsymbol{w}\|^2 + \|\boldsymbol{y}\|^2 - 2\boldsymbol{y}^T X\boldsymbol{w}$$

$$\Longrightarrow \frac{\partial}{\partial \boldsymbol{w}}\|X\boldsymbol{w} - \boldsymbol{y}\|^2 = 2(X^T X)\boldsymbol{w} - 2(X^T\boldsymbol{y})$$

Therefore we have:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \frac{(2X^T X - 2\lambda N\boldsymbol{I})\boldsymbol{w} - 2X^T\boldsymbol{y}}{N}$$

$$\Longrightarrow \frac{(2X^T X - 2\lambda N\boldsymbol{I})\boldsymbol{w}^* - 2X^T\boldsymbol{y}}{N} = 0$$

$$\Longrightarrow \boxed{\boldsymbol{w}^* = (X^T X - \lambda N\boldsymbol{I})^{-1}X^T\boldsymbol{y}}$$

(c) As we can see in the above formula for the optimal weight vector, we need to find the inverse of $X^T X - \alpha\boldsymbol{I}$. This might not be possible for all values of α, for example if α be one of the eigenvalues of $X^T X$, then there won’t be any inverse matrix and we can’t use this formula to find the optimal weight vector.