CSE343: Machine Learning

Assignment-3

Divyajeet Singh (2021529)

November 15, 2023

1 Section A (Theoretical)

Solution 1 (3 marks)

For the given regression problem, we assume 3 neurons in the one hidden layer. Since the data and the output is one-dimensional, we use a single neuron in the input and output layer. The architecture of the neural network is given in Figure 1. The bias term for the hidden layer and

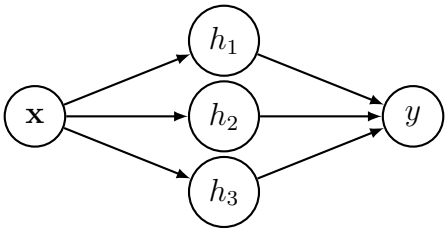

Figure 1: Neural Network Architecture for the given regression problem

for the output layer is h and b respectively. Let us assume the inital weights as follows:

$$W_1 = \begin{bmatrix} w_{11} & w_{12} & w_{13} \end{bmatrix}_{1\times 3} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \end{bmatrix} \text{ and } W_2 = \begin{bmatrix} w_{21} \\ w_{22} \\ w_{23} \end{bmatrix}_{3\times 1} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

The bias terms are initialized to zero. We use ReLU activation with a learning rate of $\eta = 0.01$. The given dataset is

$$\mathbf{x} = \begin{bmatrix} 1.2 \\ 0.8 \\ 2.0 \end{bmatrix}_{3\times 1} \text{ and } \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 3.0 \\ 2.5 \\ 4.0 \end{bmatrix}_{3\times 1}$$

Iteration #1 (Forward Pass)

For the first forward pass, we have (A_1 denotes $ReLU(Z_1)$)

$$Z_1 = \mathbf{x}W_1 + h = \begin{bmatrix} 1.2 \\ 0.8 \\ 2.0 \end{bmatrix} \begin{bmatrix} 1.0 & 1.0 & 1.0 \end{bmatrix} + \mathbf{0} = \begin{bmatrix} 1.2 & 1.2 & 1.2 \\ 0.8 & 0.8 & 0.8 \\ 2.0 & 2.0 & 2.0 \end{bmatrix}_{3\times 3} \tag{1}$$

$$\mathbf{y} = A_1W_2 + b = \begin{bmatrix} 1.2 & 1.2 & 1.2 \\ 0.8 & 0.8 & 0.8 \\ 2.0 & 2.0 & 2.0 \end{bmatrix} \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix} + \mathbf{0} = \begin{bmatrix} 3.6 \\ 2.4 \\ 6.0 \end{bmatrix}_{3\times 1} \tag{2}$$

Clearly, $A_2 = \text{ReLU}(\mathbf{y}) = \mathbf{y}$.

Iteration #1 (Loss)

We find the mean squared error loss for the first iteration is given by

$$\mathcal{L}(\mathbf{W}) = \frac{1}{3}\sum_{i=1}^{3}(\hat{y}_i - y_i)^2 = \frac{(3.6-3.0)^2 + (2.4-2.5)^2 + (6.0-4.0)^2}{3} = \frac{4.37}{3} = 1.4567 \quad (3)$$

Iteration #1 (Backward Pass)

For the backward pass, we first find the gradients of the loss with respect to the output layer weights. This is given by

$$\frac{\partial \mathcal{L}}{\partial W_2} = \left(\frac{\partial \mathbf{y}}{\partial W_2}\right)^T \frac{\partial \mathcal{L}}{\partial \mathbf{y}}$$
$$\frac{\partial \mathbf{y}}{\partial W_2} = \frac{\partial}{\partial W_2}(A_1 W_2 + b) = A_1$$
$$\frac{\partial \mathcal{L}}{\partial \mathbf{y}} = \frac{1}{3}\sum_{i=1}^{3}\frac{\partial}{\partial \mathbf{y}}(\hat{y}_i - y_i)^2 = \frac{1}{3}\sum_{i=1}^{3} 2(\hat{y}_i - y_i)\frac{\partial}{\partial \mathbf{y}}(\hat{y}_i - y_i) = \frac{2}{3}\begin{bmatrix}3.6-3.0\\2.4-2.5\\6.0-4.0\end{bmatrix} = \begin{bmatrix}-0.4\\0.06\\-1.34\end{bmatrix}$$

So, updated weights W_2 are

$$W_2 = W_2 - \eta\frac{\partial \mathcal{L}}{\partial W_2} = W_2 - \eta\begin{bmatrix}1.2 & 0.8 & 2.0\\1.2 & 0.8 & 2.0\\1.2 & 0.8 & 2.0\end{bmatrix}\begin{bmatrix}0.4\\-0.06\\1.34\end{bmatrix} = \begin{bmatrix}1.0\\1.0\\1.0\end{bmatrix} - 0.01\begin{bmatrix}3.112\\3.112\\3.112\end{bmatrix} = \begin{bmatrix}0.968\\0.968\\0.968\end{bmatrix}$$

Next, we find the gradient of the loss with respect to the bias term

$$\frac{\partial \mathcal{L}}{\partial b} = \left(\frac{\partial \mathbf{y}}{\partial b}\right)^T \frac{\partial \mathcal{L}}{\partial \mathbf{y}}$$
$$\frac{\partial \mathbf{y}}{\partial b} = \frac{\partial}{\partial b}(A_1 W_2 + b) = \mathbf{1}$$

So, the updated bias term becomes

$$b = b - \eta\frac{\partial \mathcal{L}}{\partial b} = b - \eta\begin{bmatrix}1 & 1 & 1\end{bmatrix}\begin{bmatrix}-0.4\\0.06\\-1.34\end{bmatrix} = 0 - 0.01 * (-1.68) = 0.0168$$

Next, we find the gradient of the loss with respect to the hidden layer weights. This is given by

$$\frac{\partial \mathcal{L}}{\partial W_1} = \left(\frac{\partial Z_1}{\partial W_1}\right)^T \frac{\partial \mathbf{y}}{\partial Z_1}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{y}}\right)^T$$
$$\frac{\partial \mathbf{y}}{\partial Z_1} = \frac{\partial}{\partial Z_1}(A_1 W_2 + b) = W_2 * 1 = W_2$$
$$\frac{\partial Z_1}{\partial W_1} = \frac{\partial}{\partial W_1}(\mathbf{x}W_1 + h) = \mathbf{x}$$

So, the updated weights W_1 are

$$W_1 = W_1 - \eta\frac{\partial \mathcal{L}}{\partial W_1} = W_1 - \eta\left(\begin{bmatrix}1.2 & 0.8 & 2.0\end{bmatrix}\begin{bmatrix}0.968\\0.968\\0.968\end{bmatrix}\right)\begin{bmatrix}0.4 & -0.06 & 1.34\end{bmatrix}$$
$$= \begin{bmatrix}1.0 & 1.0 & 1.0\end{bmatrix} - 0.01\begin{bmatrix}0.86 & -0.23 & 5.18\end{bmatrix} = \begin{bmatrix}0.99 & 1.02 & 0.94\end{bmatrix}$$

Finally, we find the gradient of the loss with respect to the bias term of the hidden layer. This is given by

$$\frac{\partial \mathcal{L}}{\partial h} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \left(\frac{\partial \mathbf{y}}{\partial Z_1} \right)^T \frac{\partial Z_1}{\partial h}$$
$$\frac{\partial Z_1}{\partial h} = \frac{\partial}{\partial h}(\mathbf{x}W_1 + h) = \mathbf{1}$$

So, the updated bias term h is

$$h = h - \eta \frac{\partial \mathcal{L}}{\partial h} = h - \eta \begin{bmatrix} 0.4 \\ -0.06 \\ 1.34 \end{bmatrix} \left(\begin{bmatrix} 0.968 & 0.968 & 0.968 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right)$$
$$= \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - 0.01 * \begin{bmatrix} 1.16 \\ -0.17 \\ 3.89 \end{bmatrix} = \begin{bmatrix} -0.0116 \\ 0.0017 \\ -0.0389 \end{bmatrix}$$

Solution 2 (2 marks)

To predict the class of point $\mathbf{x}$ through an SVM, we find the sign of the equation

$$\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \tag{4}$$

where $K(\cdot, \cdot)$ is the kernel function. For a gaussian SVM, the kernel function is

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u} - \mathbf{v}\|_2^2}{2\sigma^2} \right) \tag{5}$$

So, to predict the class of a new data point $\mathbf{x}$, we apply the following rule for its classification

$$\text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}\|_2^2}{2\sigma^2} \right) + b \right) \tag{6}$$

Solution 3

Index	Data Points	Label
$\mathbf{x}_1$	$(2, 3)$	$+\mathbf{1}$
$\mathbf{x}_2$	$(6, 7)$	$-\mathbf{1}$
$\mathbf{x}_3$	$(5, 6)$	$-\mathbf{1}$
$\mathbf{x}_4$	$(4, 1)$	$+\mathbf{1}$
$\mathbf{x}_5$	$(1, 1)$	$+\mathbf{1}$
$\mathbf{x}_6$	$(7, 8)$	$-\mathbf{1}$

Table 1: Data points for Problem 3

(a) (1 mark)

Solution

Yes, the points are clearly linearly separable. This is because a line (a 2-dimensional hyperplane) can separate the points into two classes. Empirically, the points with higher x and y values belong to the class $-\mathbf{1}$ and the points with lower x and y values belong to the class $+\mathbf{1}$. The linear separability can be seen in Figure 2.

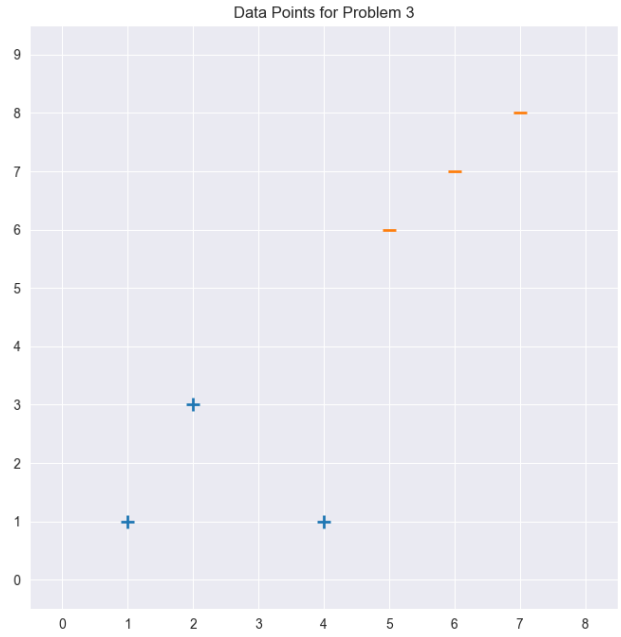

Figure 2: Data points in Problem 3

(b) (1 mark)

Solution

To find the optimal linear decision boundary for the given data points, we need to find the optimal hyperplane that maximizes the margin. Let the equation of the hyperplane be

$$\mathbf{w}^T\mathbf{x} + b = 0 \tag{7}$$

where $\mathbf{w}$ is the normal vector to the hyperplane and b is the bias term. The weights for the normal vector and bias are found by solving the following optimization problem

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2 \quad \text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \forall\ i = 1, 2, \ldots, 6 \tag{8}$$

To solve this, we solve its Lagrangian dual, which is

$$\begin{aligned} &\max_{\alpha} L(\alpha) = \sum_{i=1}^{6} \alpha_i - \frac{1}{2}\sum_{i=1}^{6}\sum_{j=1}^{6} y_i\alpha_i(\mathbf{x}_i^T\mathbf{x}_j)y_j\alpha_j \\ &\text{subject to} \quad \sum_{i=1}^{6} \alpha_i y_i = 0 \quad \text{and} \quad \alpha_i \geq 0 \quad \forall\ i = 1, 2, \ldots, 6 \end{aligned} \tag{9}$$

To find the Lagrangian multipliers, we take the partial derivatives of the Lagrangian dual with respect to each α_i and set them to zero. This gives us

$$\frac{\partial L}{\partial \alpha_i} = 1 - \frac{1}{2}\sum_{j=1}^{6} y_i(\mathbf{x}_i^T\mathbf{x}_j)y_j\alpha_j = 0 \quad \forall\ i = 1, 2, \ldots, 6 \tag{10}$$

Solving the system of these 6 linear equations, we get

$$\mathbf{w} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \text{and} \quad b = -8 \quad \text{i.e.} \quad \mathbf{x}^{(1)} + \mathbf{x}^{(2)} - 8 = 0 \tag{11}$$

In the normalized form, the equation becomes

$$\mathbf{w} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} \quad \text{and} \quad b = -\frac{8}{\sqrt{2}} = -4\sqrt{2} \quad \text{i.e.} \quad \frac{1}{\sqrt{2}}\mathbf{x}^{(1)} + \frac{1}{\sqrt{2}}\mathbf{x}^{(2)} - 4\sqrt{2} = 0 \tag{12}$$

A graph of the complete solution is given in Figure 3.

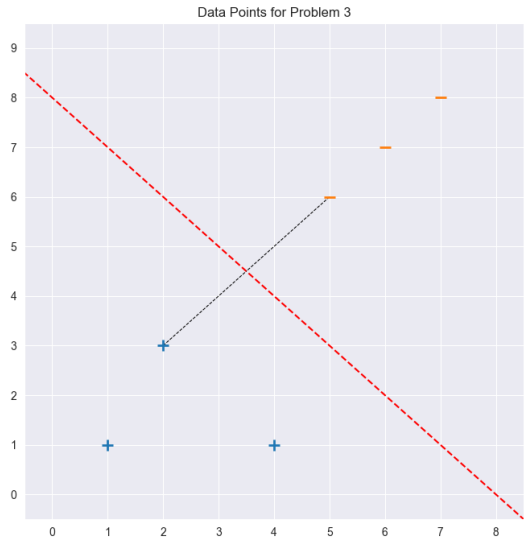

Figure 3: Solution to Problem 3

(c) (1 mark)

The data points $\mathbf{x}^*$ closest to the separating hyperplane, i.e. the support vectors are given in Table 2. Note how the point $\mathbf{x}_4 = (4, 1)$ is not a support vector even though it is the same distance from the hyperplane.

Index	Support Vectors	Label
$\mathbf{x}_1$	$(2, 3)$	$+\mathbf{1}$
$\mathbf{x}_3$	$(5, 6)$	$-\mathbf{1}$

Table 2: Support Vectors for Data in Table 1

(d) (1 mark)

The margin of the decision boundary is given by substituting the support vectors in the equation

$$\begin{aligned}\gamma &= y^* \left(\frac{1}{\|\mathbf{w}\|_2}\left(\mathbf{w}^T\mathbf{x}^* + b\right)\right) \\ &= -1\left[\frac{1}{\sqrt{2}}(1\times 2 + 1\times 3 - 8)\right] = \frac{3}{\sqrt{2}} = \frac{3\sqrt{2}}{2}\end{aligned} \tag{13}$$

Evidently, this is the distance between the hyperplane and the support vectors. The same value for γ can be obtained by substituting the other support vector in the equation. Hence, the single margin is $\frac{3\sqrt{2}}{2}$ and the total margin is $3\sqrt{2}$.

(e) (1 mark)

Clearly, the optimal margin will shift if we remove any support vector. For example, if we remove any support vector. If we remove $\mathbf{x}_1$, then empiraically, the slope of the separating hyperplane will decrease, and if we remove $\mathbf{x}_2$, then its slope increases.

2 Section B (Scratch Implementation)

In this section, we implemented a neural network from scratch using `numpy` package that meets certain requirements. The implementation can be found in `utils.py`. The implemented neural network was then tested on the MNIST dataset. The code for the solution can be found in the `main.ipynb` notebook.

Activation Functions

One can use sigmoid, tanh, ReLU, Leaky ReLU, and Linear as the activation functions for the neural network by passing in the `activation` parameter. The softmax activation function is used in the last layer.

Weight Initialization

The weights of the network can be initialized to three modes - random, normal, and zero, using the `weight_init` parameter.

Trained Neural Network

The neural network is trained to solve the MNSIT classification problem. The hidden layers are set to sizes $[256, 128, 64, 32]$. The plots for accuracies and losses per epoch are given in the notebook. The sigmoid activation function was found to give the best model.

3 Section C (Algorithm Implementation using Packages)

In this section, we use the `MLPClassifier` from the `sklearn` package to train a neural network on the SVHN dataset.

Data Preparation

The dataset was downloaded from the given link, and split into a ratio of 70% training, 20% validation, and 10% testing. Some data visualization was also performed, and can be found in the same notebook.

Model Training and Activation Functions

The neural network was trained with 2 hidden layers. The `GridSearchCV` function was to find the optimal hyperparameters for the netowrk. The ReLU activation function was found to be the best. Other optimal hyperparameters are given in table 3.

Hyperparameter	Value
`activation`	`relu`
`alpha` (learning-rate)	0.05
`batch_size`	250
`hidden_layer_sizes`	$(128, 128)$

Table 3: Optimal hyperparameters for the given neural network classifier

Incorrect Predictions

The best model achieved a score of over 0.803. Some incorrect predictions from each class were visualized and their reasons were noted down. The observations are given in the same notebook.

References

1. An Idiot's Guide to Support Vector Machines
2. `MLPClassifier` in Sci-kit Learn (Official Documentation)