Assignment 5

Course: EE708, Fundamentals of Data Science and Machine Intelligence

Nikhil Jain

Dept: Electrical Engineering

Roll No: 220709

1 .

a. Standard Neural Network (No Input-to-Output Direct Connections)

- **Input Layer (N neurons) → Hidden Layer (H neurons)**
  - Each input neuron is connected to every hidden neuron.
  - Total weights = $N \times H$.
- **Hidden Layer (H neurons) → Output Layer (C neurons)**
  - Each hidden neuron is connected to every output neuron.
  - Total weights = $H \times C$.

$$\text{Total weights} = N \times H + H \times C$$

b. Network with Input-to-Output Direct Connections

- As above, we have:
  - $N \times H$ weights between **Input** and **Hidden** layers.
  - $H \times C$ weights between **Hidden** and **Output** layers.
- Additionally, there are direct connections from the **Input Layer** to the **Output Layer**:
  - $N \times C$ weights between **Input** and **Output** layers.

$$\text{Total weights} = N \times H + H \times C + N \times C$$

2 .

a. **Advantage of Network A over Network B**

Network A has fewer layers (only one layer) compared to Network B. This simplicity reduces computational complexity, making it faster to train and evaluate. Additionally, with fewer layers, it is less prone to the vanishing gradient problem.

b. **Advantage of Network B over Network A**

Network B has more layers, allowing it to model more complex functions and capture more intricate relationships between input and output. Even though the activations are linear, multiple linear layers can still provide more expressive power by combining multiple features.

3 .

The output of a neuron with a logistic activation function is given by:

$$y = \sigma(z) = \frac{1}{1 + e^{-az}}$$

where:

- $a = 2$ (slope parameter)
- $z = w_1x_1 + w_2x_2 + b$
- Inputs $x_1 = -1$, $x_2 = 1$
- Weights $w_1 = 0.1$, $w_2 = 0.5$
- Output $y = 0.73$

Thus, we can write:

$$0.73 = \frac{1}{1 + e^{-2z}}$$

Taking the reciprocal and rearranging:

$$\frac{1}{0.73} = 1 + e^{-2z}$$

$$e^{-2z} = \frac{1}{0.73} - 1 \approx 0.3699$$

Taking the natural logarithm on both sides:

$$-2z = \ln(0.3699)$$

$$z = -\frac{\ln(0.3699)}{2}$$

Calculating the natural logarithm:

$$\ln(0.3699) \approx -0.995$$

$$z = -\frac{-0.995}{2} \approx 0.4975$$

Now, substituting the inputs and weights into the equation for z:

$$z = (0.1)(-1) + (0.5)(1) + b = -0.1 + 0.5 + b$$

$$0.4975 = -0.1 + 0.5 + b$$

$$0.4975 = 0.4 + b$$

$$b = 0.4975 - 0.4 = 0.0975$$

The value of the bias b is approximately 0.0975.

4 .

a. **Perceptron for NOT of its input**

A perceptron can calculate the NOT operation with a single input and one bias term. The NOT operation outputs 1 when the input is 0 and outputs 0 when the input is 1.

Input (x)	Output (NOT x)
0	1
1	0

For a perceptron:

- **Weights** (w): -1 - **Bias** (b): 0.5 - **Activation function**: Step function (Output is 1 if weighted sum ≥ 0, otherwise 0)

$$y = \text{Step}(w \cdot x + b) = \text{Step}(-1 \cdot x + 0.5)$$

b. **Perceptron for NAND of two inputs**

The NAND operation outputs 0 only when both inputs are 1; otherwise, it outputs 1.

x_1	x_2	Output (NAND)
0	0	1
0	1	1
1	0	1
1	1	0

For a perceptron:

- **Weights** (w_1, w_2): $-1, -1$ - **Bias** (b): 1.5 - **Activation function**: Step function

$$y = \text{Step}(w_1x_1 + w_2x_2 + b) = \text{Step}(-1 \cdot x_1 - 1 \cdot x_2 + 1.5)$$

5 .

A **Perceptron** can only solve linearly separable problems, but **parity problems are not linearly separable**. The parity of three inputs is determined by whether the number of 1s in the inputs is even or odd.

For three binary inputs x_1, x_2, x_3, the parity function outputs:

- **0 (Even Parity)** if the number of 1s is even (0 or 2 1s). - **1 (Odd Parity)** if the number of 1s is odd (1 or 3 1s).

x_1	x_2	x_3	Output (Parity)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

A **single-layer perceptron cannot solve this problem** because it is not linearly separable. Instead, we need a **multi-layer perceptron (MLP)** or a **XOR gate combination**.

6 .

Let's derive the weight update equations for a **Multi-Layer Perceptron (MLP)** with: - **One hidden layer** with H units. - **ReLU activation** in the hidden layer. - **Linear output layer** for **regression**. - **Mean Squared Error (MSE)** as the loss function.

Notation

- $\mathbf{X} \in \mathbb{R}^{N \times d}$: Input matrix, where N is the number of samples and d is the input dimension.
- $\mathbf{W}^{(1)} \in \mathbb{R}^{H \times d}$: Weights between the input layer and hidden layer.
- $\mathbf{b}^{(1)} \in \mathbb{R}^{H}$: Biases for the hidden layer.
- $\mathbf{W}^{(2)} \in \mathbb{R}^{1 \times H}$: Weights between the hidden layer and output layer.
- $\mathbf{b}^{(2)} \in \mathbb{R}$: Bias for the output layer.
- $\mathbf{y} \in \mathbb{R}^{N}$: True output (target) for all samples.
- $\hat{\mathbf{y}} \in \mathbb{R}^{N}$: Predicted output for all samples.
- η: Learning rate.

Forward Pass

$$\mathbf{z} = \mathbf{X}\mathbf{W}^{(1)^T} + \mathbf{b}^{(1)}$$

$$\mathbf{h} = \text{ReLU}(\mathbf{z}) = \max(0, \mathbf{z})$$

$$\hat{\mathbf{y}} = \mathbf{h}\mathbf{W}^{(2)^T} + \mathbf{b}^{(2)}$$

Loss Function (MSE)

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

Backward Pass (Gradient Calculation)

$$\frac{\partial \mathcal{L}}{\partial \hat{y}_i} = \frac{2}{N}(\hat{y}_i - y_i)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(2)}} = \frac{2}{N}(\hat{\mathbf{y}} - \mathbf{y})^T \mathbf{h}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(2)}} = \frac{2}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}} = \frac{2}{N}\left(\hat{\mathbf{y}} - \mathbf{y}\right)\mathbf{W}^{(2)}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}} = \left(\frac{2}{N}\left(\hat{\mathbf{y}} - \mathbf{y}\right)\mathbf{W}^{(2)}\right) \odot \mathbf{M}$$

$$\mathbf{M}_j = \begin{cases} 1 & \text{if } \mathbf{z}_j > 0, \\ 0 & \text{if } \mathbf{z}_j \leq 0. \end{cases}$$

Weight Update Equations (Gradient Descent)

$$\mathbf{W}^{(2)} \leftarrow \mathbf{W}^{(2)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(2)}}$$

$$\mathbf{b}^{(2)} \leftarrow \mathbf{b}^{(2)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(2)}}$$

$$\mathbf{W}^{(1)} \leftarrow \mathbf{W}^{(1)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(1)}}$$

$$\mathbf{b}^{(1)} \leftarrow \mathbf{b}^{(1)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(1)}}$$

sectionprogramming questions

7 .

The Perceptron Learning Algorithm for Binary Classification is implemented using the following specifications:

- **Activation function:** Step function
- **Weight initialization:** Random (using numpy package)
- **Learning rate:** 0.01
- **Number of weight update iterations:** 20

The implementation involves the following steps:

1. **Weight Initialization:** Weights are initialized randomly using the numpy package for reproducibility. A bias term is also randomly initialized.
2. **Prediction:** The linear combination of inputs and weights is passed through a step activation function, which returns 1 if the result is non-negative, otherwise 0.
3. **Weight Update Rule:** The weights are updated using the gradient descent rule:

$$w = w + \eta(y - \hat{y})x \tag{1}$$

$$b = b + \eta(y - \hat{y}) \tag{2}$$

where η is the learning rate, y is the true label, and $\hat{y}$ is the predicted label.

4. **Visualization:** The decision boundary is plotted along with the dataset points. Different classes are represented using different colors.

The code is designed to be rerun with different random weight initializations to analyze their effect on the decision boundary. This highlights the importance of initialization in perceptron training.

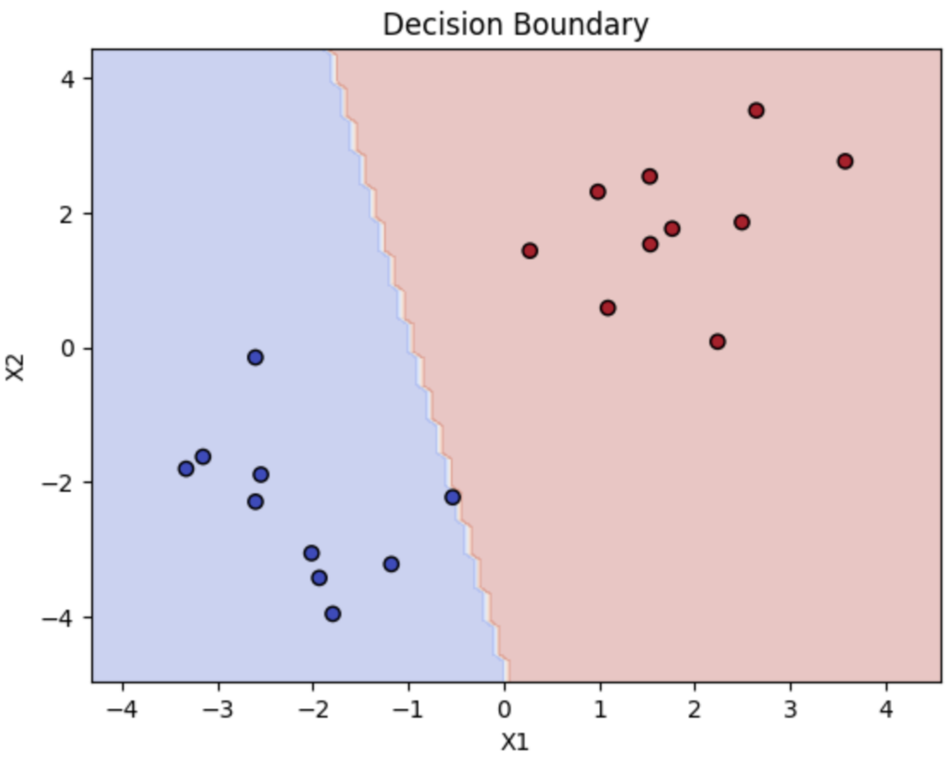

Figure 1: Perceptron

8 .

We will implement a neural network from scratch to classify the Iris dataset. The network architecture consists of:

- One input layer with 4 neurons (corresponding to the 4 features in the Iris dataset).
- One hidden layer with 5 neurons, using the `tanh` activation function.
- One output layer with 3 neurons (corresponding to the 3 classes), using the `softmax` activation function.

Dataset Preparation

- We load the Iris dataset from `sklearn.datasets.load_iris()`.
- The dataset is split into training and testing subsets using an 80%-20% ratio.
- The features are standardized to have zero mean and unit variance using `StandardScaler`.

Network Initialization

- The weights are initialized randomly from a normal distribution.
- Biases are initialized randomly from a normal distribution.

Activation Functions

- Hidden Layer: `tanh` activation function, defined as:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3}$$

- Output Layer: `softmax` activation function, defined as:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{4}$$

Loss Function

- The loss function used is Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5}$$

Training Process

- We use Stochastic Gradient Descent (SGD) for training with a learning rate of 0.01.
- The network is trained for 1000 epochs.
- During each epoch, we perform forward propagation, compute the loss, perform backward propagation, and update the weights and biases.

Evaluation

- The training and testing loss and accuracy are recorded for each epoch.
- Two plots are generated:
  1. Loss vs. Epochs for both training and testing datasets.
  2. Accuracy vs. Epochs for both training and testing datasets.

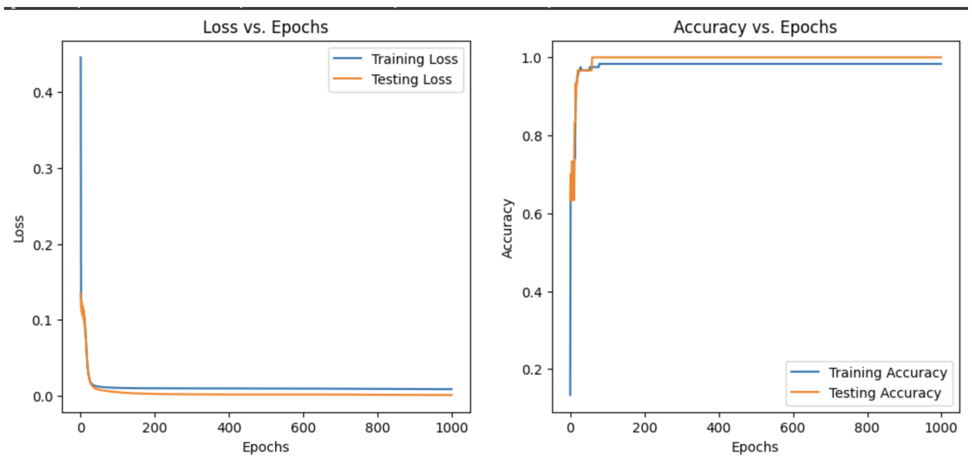

Figure 2: Perceptron