
**Faculty of Computing and Information Technology**

**University of the Punjab,**

**Lahore**

# Artificial Intelligence Lab 14

**Instructor: Qamar U Zaman**

# Backward Propagation for Binary Classification

## Objective

- Understand the concept of forward and backward propagation.
- Implement a neural network with one hidden layer.
- Train the network to classify a custom dataset.
- Visualize the decision boundary.

---

## Key Concepts

1. **Forward Propagation**: Computes the output of the neural network given the input and current weights.
2. **Backward Propagation**: Adjusts the weights to minimize the loss by calculating gradients.
3. **Binary Cross-Entropy Loss**: Evaluates the error for binary classification tasks.

---

## Dataset

| X1 | X2 | Label |
|---|---|---|
| 0.1 | 0.6 | 1 |
| 0.15 | 0.71 | 1 |
| 0.25 | 0.8 | 1 |
| 0.35 | 0.45 | 1 |
| 0.5 | 0.5 | 0 |
| 0.6 | 0.2 | 0 |
| 0.65 | 0.3 | 0 |
| 0.8 | 0.35 | 0 |

---

## Tasks for Students

1. **Dataset Setup**:
    - Define the given dataset in your code.
2. **Parameter Initialization**:
    - Initialize the weights and biases for a simple neural network.
3. **Forward Propagation**:
    - Implement the forward propagation logic to compute outputs.
4. **Loss Calculation**:
    - Write a function to compute the binary cross-entropy loss.
5. **Backward Propagation**:
    - Implement the backward propagation logic to compute gradients for weight updates.

6. **Training**:
    - Train the neural network on the dataset, iterating through epochs and updating weights.
7. **Visualization**:
    - Plot the decision boundary to show how the network classifies the data points.

---

## Code Template

```
# Step 2: Initialize weights and biases
def initialize_parameters(input_size, hidden_size, output_size):
    """
    Initialize the weights and biases for the network.
    """
    pass

# Step 3: Implement forward propagation
def forward_propagation(X, weights):
    """
    Compute the forward pass through the network.
    """
    pass

# Step 4: Compute the loss
def compute_loss(y_true, y_pred):
    """
    Compute binary cross-entropy loss.
    """
    pass

# Step 5: Implement backward propagation
def backward_propagation(X, y, weights, cache):
    """
    Compute gradients for backward propagation.
    """
    pass

# Step 6: Update weights
def update_parameters(weights, gradients, learning_rate):
    """
    Update the weights using gradient descent.
    """
    pass

# Step 7: Training loop
def train_network(X, y, hidden_size, learning_rate, epochs):
    """
    Train the neural network.
    """
    pass

# Step 8: Plot decision boundary
def plot_decision_boundary(X, y, weights):
    """
    Visualize the decision boundary of the trained network.
    """
    pass
```

---

## Instructions

1. **Complete the Code**:
    - Fill in the placeholders in the provided code template to complete each step.
2. **Train the Model**:
    - Train your neural network on the dataset for a specified number of epochs and observe the loss values.
3. **Visualize Results**:
    - Use a plotting library to visualize the decision boundary and data points.