

Assignment 7: Implementing a Single-Layer Neural Network for Binary Classification

Question 1: Design and Initialize Neural Network

1.1 Network Design and Truth Table Creation

- Creating a truth table and corresponding binary output

Step 1.1 Network Design and Truth Table Creation				
The logical function $(x_1 \text{ AND } x_2) \text{ OR } (x_3 \text{ AND } x_4)$				
x_1	x_2	x_3	x_4	
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	0	0	0	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

1.2 Manual Forward Propagation Calculation

- I choose example set of inputs from my truth are
 $x_1=0, x_2=0, x_3=0, x_4=0$ and $x_1=0, x_2=0, x_3=1, x_4=1$ and
 $x_1=1, x_2=1, x_3=0, x_4=0$ and $x_1=1, x_2=0, x_3=1, x_4=1$
- Using sigmoid activation function and weight=0.5, bias = 1

Step 1.2 Manual Forward Propagation Calculation

$$\omega = 0.5, b = 1, z = \text{sigmoid}, f(z) = \frac{1}{1+e^{-z}}$$

$x_1=0, x_2=0, x_3=0, x_4=0$

$$z = (0.5 \times 0) + (0.5 \times 0) + (0.5 \times 0) + (0.5 \times 0) + 1$$

$$z = 0 + 0 + 0 + 0 + 1$$

$$z = 1$$

$$f(1) = \frac{1}{1+e^{-1}} = 0.73, y_{\text{pred}} = 0.73$$

$x_1=0, x_2=0, x_3=1, x_4=0$

$$z = (0.5 \times 0) + (0.5 \times 0) + (0.5 \times 1) + (0.5 \times 0) + 1$$

$$z = 0 + 0 + 0.5 + 0.5 + 1$$

$$z = 2$$

$$f(2) = \frac{1}{1+e^{-2}} = 0.88, y_{\text{pred}} = 0.88$$

$x_1=1, x_2=0, x_3=0, x_4=0$

$$z = (0.5 \times 1) + (0.5 \times 0) + (0.5 \times 0) + (0.5 \times 0) + 1$$

$$z = 0.5 + 0.5 + 0 + 0 + 1$$

$$z = 2$$

$$f(2) = \frac{1}{1+e^{-2}} = 0.88, y_{\text{pred}} = 0.88$$

$x_1=1, x_2=0, x_3=1, x_4=1$

$$z = (0.5 \times 1) + (0.5 \times 0) + (0.5 \times 1) + (0.5 \times 1) + 1$$

$$z = 0.5 + 0 + 0.5 + 0.5 + 1$$

$$z = 2.5$$

$$f(2.5) = \frac{1}{1+e^{-2.5}} = 0.923, y_{\text{pred}} = 0.923$$

1.3 Manual Backpropagation Calculation

Step 1.3 Manual Backpropagation Calculation

Error Calculation

$$\text{Error} = t - y_{\text{pred}}$$
$$x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$$
$$w = 0.5$$
$$b = 1$$
$$t = 1$$
$$y_{\text{pred}} = 0.88$$
$$\text{Error} = 1 - 0.88 = 0.12$$

Gradient Computation

$$E(0) = (t - y_{\text{pred}})^2$$
$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} (t - y_{\text{pred}})^2$$
$$= -2 \cdot (t - y_{\text{pred}}) \cdot \frac{\partial (t - y_{\text{pred}})}{\partial w_i}$$
$$= -2 \cdot (t - y_{\text{pred}}) \cdot \frac{\partial y_{\text{pred}}}{\partial w_i}$$
$$y_{\text{pred}} = \sigma(z)$$

Using the derivation of the sigmoid function

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z) \cdot (1 - \sigma(z))$$

Thus

$$\frac{\partial y_{\text{pred}}}{\partial w_i} = \frac{\partial \sigma(z)}{\partial z} \cdot \frac{\partial z}{\partial w_i}$$
$$\frac{\partial z}{\partial w_i} = x_i$$

Therefore:

$$\frac{\partial y_{\text{pred}}}{\partial w_i} = \sigma(z) \cdot (1 - \sigma(z)) \cdot x_i$$

Error = 0.12

$$\sigma(z) = \sigma(z) \cdot (1 - \sigma(z))$$

$$\sigma(z) = 0.88 \cdot (1 - 0.88)$$
$$= 0.1056$$

$$\frac{\partial y_{pred}}{\partial w_i} = \sigma'(z) \cdot x_i$$

For each weight:

$$x_1 = 1, \frac{\partial y_{pred}}{\partial w_1} = 0.1056 \times 1 = 0.1056$$

$$x_2 = 1, \frac{\partial y_{pred}}{\partial w_2} = 0.1056 \times 1 = 0.1056$$

$$x_3 = 0, \frac{\partial y_{pred}}{\partial w_3} = 0.1056 \times 0 = 0$$

$$x_4 = 0, \frac{\partial y_{pred}}{\partial w_4} = 0.1056 \times 0 = 0$$

$$\frac{\partial E}{\partial w_i} = -2 \cdot \text{error} \cdot \frac{\partial y_{pred}}{\partial w_i}$$

$$\frac{\partial E}{\partial w_1} = -2 \times 0.12 \times 0.1056 = -0.0253$$

$$\frac{\partial E}{\partial w_2} = -2 \times 0.12 \times 0.1056 = -0.0253$$

$$\frac{\partial E}{\partial w_3} = 0$$

$$\frac{\partial E}{\partial w_4} = 0$$

Gradient of the loss function with respect to the bias (b):

$$\frac{\partial E}{\partial b} = -2 \cdot (t - y_{pred}) \cdot \sigma'(z)$$

$$= -2 \times 0.12 \times 0.1056$$

$$= -0.0253$$

Gradient value = -0.0253

Weight Update

$$w_i = w_i - n \cdot \frac{\partial E}{\partial w_i}$$

Learning rate, $n = 0.001$

$$w_1 = 0.5 - 0.001 \times (-0.0253) = 0.5000253$$

$$w_2 = 0.5 - 0.001 \times (-0.0253) = 0.5000253$$

$$w_3 = 0.5 - 0.001 \times 0 = 0.5$$

$$w_4 = 0.5 - 0.001 \times 0 = 0.5$$

bias update:

$$b = b - n \cdot \frac{\partial E}{\partial b}$$

$$b = 1 - 0.01 \times (-0.0253)$$

$$b = 1.0000253$$

weight update value using learning rate 0.001

1.4 Implementation and Training in Python

The image shows two side-by-side sessions of a Python code editor, likely Visual Studio Code, demonstrating the implementation and training of a neural network.

Top Session:

```
Assignment 7.py ...
1 import numpy as np
2
3 # Sigmoid activation function and its derivative
4 def sigmoid(x):
5     return 1 / (1 + np.exp(-x))
6
7 def sigmoid_derivative(x):
8     return sigmoid(x) * (1 - sigmoid(x))
9
10 # Initializing weights and bias
11 np.random.seed(42) # for reproducibility
12 weights = np.array([0.5, 0.5, 0.5, 0.5]) # weights for two inputs
13 bias = np.random.rand(1) # one bias
14
15 # Dataset
16 inputs = np.array([
    [0, 0, 0, 0],
    [0, 0, 0, 1],
    [0, 0, 1, 0],
    [0, 0, 1, 1],
    [0, 1, 0, 0],
    [0, 1, 0, 1],
    [0, 1, 1, 0],
    [0, 1, 1, 1],
    [1, 0, 0, 0]
])
targets = np.array([0, 0, 0, 1, 0, 0, 1, 1, 1, 1])

# Learning rate
learning_rate = 0.001 # Increased for faster convergence

# Training loop
for epoch in range(1000): # Reduced number of epochs
    total_error = 0
    indices = np.arange(len(inputs))
    np.random.shuffle(indices)

    for i in indices:
        input_layer = inputs[i]
        target = targets[i]

        z = np.dot(input_layer, weights) + bias
        output = sigmoid(z)

        error = target - output
        total_error += error ** 2

        # Backpropagation
        slope = sigmoid_derivative(output)
        gradient = input_layer * slope
        weights -= learning_rate * np.dot(gradient, error)
        bias -= learning_rate * error
```

Bottom Session:

```
Assignment 7.py ...
1 import numpy as np
2
3 # Sigmoid activation function and its derivative
4 def sigmoid(x):
5     return 1 / (1 + np.exp(-x))
6
7 def sigmoid_derivative(x):
8     return sigmoid(x) * (1 - sigmoid(x))
9
10 # Initializing weights and bias
11 np.random.seed(42) # for reproducibility
12 weights = np.array([0.5, 0.5, 0.5, 0.5]) # weights for two inputs
13 bias = np.random.rand(1) # one bias
14
15 # Dataset
16 inputs = np.array([
    [0, 0, 0, 0],
    [0, 0, 0, 1],
    [0, 0, 1, 0],
    [0, 0, 1, 1],
    [0, 1, 0, 0],
    [0, 1, 0, 1],
    [0, 1, 1, 0],
    [0, 1, 1, 1],
    [1, 0, 0, 0]
])
targets = np.array([0, 0, 0, 1, 0, 0, 1, 1, 1, 1])

# Learning rate
learning_rate = 0.001 # Increased for faster convergence

# Training loop
for epoch in range(1000): # Reduced number of epochs
    total_error = 0
    indices = np.arange(len(inputs))
    np.random.shuffle(indices)

    for i in indices:
        input_layer = inputs[i]
        target = targets[i]

        z = np.dot(input_layer, weights) + bias
        output = sigmoid(z)

        error = target - output
        total_error += error ** 2

        # Backpropagation
        slope = sigmoid_derivative(output)
        gradient = input_layer * slope
        weights -= learning_rate * np.dot(gradient, error)
        bias -= learning_rate * error
```

The screenshot shows a Python code editor interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, ...
- Search Bar:** week7
- Explorer:** OPEN EDITORS, WEEK7 (containing files: A neural network is a machine learn.txt, Assignment 7.instruction.pdf, Assignment 7.py, Eight lecture Neural Network 1.pdf, neural network.pdf, neural network.py)
- Code Editor:** Assignment 7.py (displayed in the main pane). The code is as follows:

```
        output = sigmoid(z)
        error = 0.5 * (target - output) ** 2
        total_error += error

        dE_dy = output - target
        dy_dz = sigmoid_derivative(z)
        dz_dw = input_layer
        dz_db = 1

        gradient_weights = dE_dy * dy_dz * dz_dw
        gradient_bias = dE_dy * dy_dz * dz_db

        weights -= learning_rate * gradient_weights
        bias -= learning_rate * gradient_bias

if epoch % 100 == 0:
    print(f'Epoch {epoch}, Total Error: {total_error}'
```

- Terminal Tab:** Shows the output of the script execution:

```
Input: [1 0 0 1], Predicted Output: 0.5563, Actual Target: 0
Input: [1 0 0 0], Predicted Output: 0.4705, Actual Target: 0
Input: [1 0 0 0], Predicted Output: 0.4705, Actual Target: 0
Input: [1 0 0 0], Predicted Output: 0.4705, Actual Target: 0
Input: [1 0 0 0], Predicted Output: 0.4705, Actual Target: 0
Input: [1 0 0 0], Predicted Output: 0.4705, Actual Target: 0
Input: [1 0 0 0], Predicted Output: 0.4705, Actual Target: 0
Input: [1 0 0 0], Predicted Output: 0.4705, Actual Target: 0
Input: [1 0 0 0], Predicted Output: 0.4705, Actual Target: 0
Input: [1 0 0 0], Predicted Output: 0.4705, Actual Target: 0
Input: [1 0 0 0], Predicted Output: 0.5563, Actual Target: 0
Input: [1 0 1 0], Predicted Output: 0.5563, Actual Target: 0
Input: [1 0 1 1], Predicted Output: 0.6389, Actual Target: 1
Input: [1 0 0 0], Predicted Output: 0.5563, Actual Target: 1
```

- Bottom Bar:** PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, POSTMAN CONSOLE, COMMENTS, Python 3.11.0 64-bit (3.11.0: pyenv), Go Live, and various system icons.

The screenshot shows a Python code editor interface with the following details:

- File Explorer:** Shows files like "Assignment 7.py", "Assignment 7.instruction.pdf", and "neural network.pdf".
- Code Editor:** The file "Assignment 7.py" is open, containing code for a neural network. It includes imports for `np` and `sigmoid`, defines a `bias` variable, and iterates through inputs to calculate predicted outputs and print them alongside actual targets.
- Output Panel:** Displays the execution results for each input row, showing Predicted Output and Actual Target values.
- Bottom Status Bar:** Shows the current file path as "C:\Users\Admin\OneDrive - Assumption University\Desktop\AI concept\week7>" and the system status bar with "Ln 65, Col 25, Spaces: 4, UTF-8, CRLF, Python 3.11.0 64-bit (3.11.0: pyenv)".

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Search Bar:** week7
- Left Sidebar:** EXPLORER, OPEN EDITORS (Assignment 7.py), WEEK7 (Figure 1).
- Figure 1:** A line plot titled "Loss Over Epochs" showing Average Loss decreasing from approximately 0.16 at epoch 0 to about 0.10 at epoch 1000.
- Code Editor:** Python code for a neural network assignment.
- Output Area:** Terminal output showing loss values for various inputs.
- Bottom Bar:** Includes icons for file operations, search, and help.

```
for i in range(len(inputs)):
    _, weights) + bias

    uts[i]], Predicted Output: {output[0]:.4f}, Actual Target: {targets[i]}')

loss_history)

s")
ochs")
```

```
input: [1 1 0 1], Predicted Output: 0.5563, Actual Target: 0
input: [1 1 0 1], Predicted Output: 0.5563, Actual Target: 0
input: [0 1 1 0], Predicted Output: 0.6389, Actual Target: 1
input: [0 1 1 0], Predicted Output: 0.4705, Actual Target: 0
input: [1 1 1 0], Predicted Output: 0.5563, Actual Target: 0
input: [1 1 1 0], Predicted Output: 0.5563, Actual Target: 0
input: [1 1 1 1], Predicted Output: 0.6389, Actual Target: 1
input: [1 1 1 1], Predicted Output: 0.7141, Actual Target: 1
```