# Central Department of Computer Science & Information Technology

Tribhuvan University, Kirtipur, Kathmandu, Nepal

Tel. +977-01-4333010, email: info@cdcsit.edu.np Website: www.cdcsit.edu.np

## Assignment Report

## Subject: Neural Network

**Submitted To:**

**----------------------------------**
**Arjun Singh Saud**

**Submitted By:**

**Sagar Timalsena**
**Roll No: 34**

**Date of Submission: March 21, 2024**

1. **Write a python program to create a neuron and predict its output using the threshold activation function.**

Program :

```
import numpy as np
def threshold_activation(inputs, weights, bias):
    weighted_sum = sum(x * w for x, w in zip(inputs, weights)) + bias
    activation = 1 if weighted_sum >= 0 else 0
    return weighted_sum, activation

def main():
    num_inputs = int(input("Enter the number of inputs: "))
    weights = []
    for i in range(num_inputs):
        weight = float(input(f"Enter weight for input {i + 1}: "))
        weights.append(weight)

    if len(weights) != num_inputs:
        print("Error: Number of weights should be equal to the number of inputs")
        return

    bias = float(input("Enter the bias: "))

    inputs = []
    for i in range(num_inputs):
        value = float(input(f"Enter input value {i + 1}: "))
        inputs.append(value)

    weighted_sum, output = threshold_activation(inputs, weights, bias)
    print("Inputs:", inputs)
    print("Weights:", weights)
    print("Bias:", bias)
    print(f"The weighted sum is: {weighted_sum}")
    print(f"The output is: {output}")

if __name__ == "__main__":
    main()
```

**Output**

```
Enter the number of inputs: 2
Enter weight for input 1: 0.1
Enter weight for input 2: 0.3
Enter the bias: 0.5
Enter input value 1: 0.6
Enter input value 2: 0.9
Inputs: [0.6, 0.9]
Weights: [0.1, 0.3]
Bias: 0.5
The weighted sum is: 0.8300000000000001
The output is: 1
```

## 2. Write a python program to train AND Gate Using Perceptron Learning Algorithm.

Program:

```python
import numpy as np

class Perceptron:
    def __init__(self, input_size, learning_rate=0.1, epochs=100):
        self.input_size = input_size
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = np.random.rand(input_size)
        self.bias = np.random.rand()

    def activation(self, x):
        return 1 if x >= 0 else 0

    def predict(self, inputs):
        summation = np.dot(inputs, self.weights) + self.bias
        return self.activation(summation)

    def train(self, training_inputs, labels):
        for _ in range(self.epochs):
            for inputs, label in zip(training_inputs, labels):
                prediction = self.predict(inputs)
                self.weights += self.learning_rate * (label - prediction) * inputs
                self.bias += self.learning_rate * (label - prediction)

def main():

    training_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    labels = np.array([0, 0, 0, 1])

    perceptron = Perceptron(input_size=2)


    perceptron.train(training_inputs, labels)

    test_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    for inputs in test_inputs:
        output = perceptron.predict(inputs)
        print(f"Input: {inputs}, Output: {output}")

if __name__ == "__main__":
    main()
```

Output:

```
D:\Neural-Networks-Lab>C:/Python311/python.exe "d:/Neural-Networks-Lab/Lab 1/2.py"
Input: [0 0], Output: 0
Input: [0 1], Output: 0
Input: [1 0], Output: 0
Input: [1 1], Output: 1
```

3. **Write a python program to implement Min-Max Scalar.**

**Program:**

```
class MinMaxScaler:
    def __init__(self):
        self.min_vals = None
        self.max_vals = None

    def fit(self, data):
        self.min_vals = [min(column) for column in zip(*data)]
        self.max_vals = [max(column) for column in zip(*data)]

    def transform(self, data):
        scaled_data = []
        for row in data:
            scaled_row = [(row[i] - self.min_vals[i]) / (self.max_vals[i] - self.min_vals[i]) for i in range(len(row))]
            scaled_data.append(scaled_row)
        return scaled_data

    def fit_transform(self, data):
        self.fit(data)
        return self.transform(data)
data = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]

scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)
print("Original data:")
for row in data:
    print(row)
print("\nScaled data:")
for row in scaled_data:
    print(row)
```

**Output:**

```
Original data:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]

Scaled data:
[0.0, 0.0, 0.0]
[0.5, 0.5, 0.5]
[1.0, 1.0, 1.0]
```

4. **Write a python program to implement Standard Scalar.**

**Program**

```
from sklearn.preprocessing import StandardScaler

# Sample data
data = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]

# Create StandardScaler object
scaler = StandardScaler()

# Fit to data, then transform it
scaled_data = scaler.fit_transform(data)

print("Original data:")
for row in data:
    print(row)

print("\nScaled data:")
for row in scaled_data:
    print(row)
```

**Output**

```
Original data:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]

Scaled data:
[-1.22474487 -1.22474487 -1.22474487]
[0. 0. 0.]
[1.22474487 1.22474487 1.22474487]
```

**5.** Write a python program to train perceptron using given training set and predict class for the input (6,82) and (5.3,52)

| *Height($x_1$)* | *Weight($x_2$)* | *Class(t)* |
|---|---|---|
| 5.9 | 75 | Male |
| 5.8 | 86 | Male |
| 5.2 | 50 | Female |
| 5.4 | 55 | Female |
| 6.1 | 85 | Male |
| 5.5 | 62 | Female |

Program

```python
class Perceptron:
    def __init__(self, learning_rate=0.01, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.activation_func = self._unit_step_func
        self.weights = None
        self.bias = None
    def fit(self, X, y):
        n_samples, n_features = X.shape
        # Initialize parameters
        self.weights = np.zeros(n_features)
        self.bias = 0
        y_ = np.array([1 if i > 0 else 0 for i in y])
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self.activation_func(linear_output)
                # Perceptron update rule
                update = self.lr * (y_[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update
    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        y_predicted = self.activation_func(linear_output)
        return y_predicted

    def _unit_step_func(self, x):
        return np.where(x >= 0, 1, 0)
# Given training dataset
X = np.array([
    [5.9, 75],
    [5.8, 86],
    [5.2, 50],
    [5.4, 55],
    [6.1, 85],
    [5.5, 62]
])

# Classes where 'Male' is 1 and 'Female' is 0
y = np.array([1, 1, 0, 0, 1, 0])

# Predict classes for new inputs
# test_inputs = np.array([[6, 82], [5.3, 52]])
test_inputs = np.array([[6.0764,77.1136],[5.9400,86.750], [5.9979, 81.2432],[5.4144, 58.0020],[5.4761,54.7691],[5.4444, 50.8754]])
predictions = p.predict(test_inputs)

predictions # Output: array of predicted classes (1 for 'Male', 0 for 'Female')
```

Output:

```
array([1, 1, 1, 0, 0, 0])
```

**Conclusion**

In this lab, we covered essential concepts in neuron creation, training a perceptron for an AND gate, and implementing data preprocessing techniques like Min-Max Scalar and Standard Scalar in Python. We learned how to predict outputs using threshold activation functions and trained a perceptron to classify new data points. These exercises demonstrated fundamental principles in machine learning, providing a solid foundation for further exploration in the field.