LabAssignment2

May 7, 2023

1 Lab Assignment 2

1.1 Problem 1

1. Implement Backpropagation algorithm to train an ANN of configuration 2x2x1 to achieve XOR function.

[1]:
```
import numpy as np
from  sklearn.metrics import␣
↪confusion_matrix,accuracy_score,classification_report
```

1.1.1 Using Sigmoid Activation Function

$$f(x) = \frac{1}{1+e^{-x}}$$

[2]:
```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

[3]:
```
def sigmoid_derivative(x):
    return x*(1-x)
```

[4]:
```
class NueuralNetwork:
    def __init__(self,learning_rate=0.1):
        self.input_layer_size = 2
        self.hidden_layer_size = 2
        self.output_layer_size = 1
        self.learning_rate = learning_rate
        self.W1 = np.random.uniform(low=-1, high=1, size=(self.
↪input_layer_size, self.hidden_layer_size))
        self.W2 = np.random.uniform(low=-1, high=1, size=(self.
↪hidden_layer_size, self.output_layer_size))
        self.b1 = np.zeros((1, self.hidden_layer_size))
        self.b2 = np.zeros((1, self.output_layer_size))
    def forward_pass(self,X):
        self.z2 = np.dot(X, self.W1)+self.b1
        self.a2 = sigmoid(self.z2)
        self.z3 = np.dot(self.a2, self.W2)+self.b2
```

```
        y_hat = sigmoid(self.z3)
        return y_hat
    def backward(self, X, y, y_hat):
        delta3 = (y - y_hat) * sigmoid_derivative(y_hat)
        dW2 = np.dot(self.a2.T, delta3)
        db2 = np.sum(delta3, axis=0, keepdims=True)
        delta2 = np.dot(delta3, self.W2.T) * sigmoid_derivative(self.a2)
        dW1 = np.dot(X.T, delta2)
        db1 = np.sum(delta2, axis=0)
        self.W1 += self.learning_rate * dW1
        self.W2 += self.learning_rate * dW2
        self.b1 += self.learning_rate * db1
        self.b2 += self.learning_rate * db2

    def train(self, X, y, epochs):
        for i in range(epochs):
            y_hat = self.forward_pass(X)
            self.backward(X, y, y_hat)

        return self.forward_pass(X)
```

[5]:
```
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
```

[6]:
```
nn = NueuralNetwork()
```

[7]:
```
epoch=10000
```

[8]:
```
y_pred = nn.train(X,y,epochs=epoch)
```

[9]:
```
y_pred
```

[9]:
```
array([[0.05479661],
       [0.66243245],
       [0.66243464],
       [0.66861741]])
```

[10]:
```
threshold = 0.5
y_pred[y_pred >= threshold] = 1
y_pred[y_pred < threshold] = 0

print("Predicted output:")
print(y_pred)
```

```
Predicted output:
[[0.]
 [1.]
```

```
 [1.]
 [1.]]
```

[11]:
```
print(f"The Accuracy Score is {accuracy_score(y, y_pred)}")
```

```
The Accuracy Score is 0.75
```

[12]:
```
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       0.67      1.00      0.80         2

    accuracy                           0.75         4
   macro avg       0.83      0.75      0.73         4
weighted avg       0.83      0.75      0.73         4
```

1.2 Problem 2

2. Implement Backpropagation algorithm to train an ANN of configuration 3x2x2x1 to achieve majority function with 3-bit data. Output of the network must be 1 when there are two or more 1's in the data.

[13]:
```
import numpy as np
```

[14]:
```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

[15]:
```
def sigmoid_derivative(x):
    return x*(1-x)
```

[16]:
```
# Define the majority function
def majority_function(x):
    if np.sum(x) >= 2:
        return 1
    else:
        return -1
```

[17]:
```
class NueuralNetwork:
    def __init__(self,learning_rate=0.1):
        self.input_layer_size = 3
        self.hidden_layer_1_size = 2
        self.hidden_layer_2_size = 2
        self.output_layer_size = 1
        self.learning_rate = learning_rate
```

```
        self.W1 = np.random.uniform(low=-1, high=1, size=(self.
↪input_layer_size, self.hidden_layer_1_size))
        self.W2 = np.random.uniform(low=-1, high=1, size=(self.
↪hidden_layer_1_size, self.hidden_layer_2_size))
        self.W3 = np.random.uniform(low=-1, high=1, size=(self.
↪hidden_layer_2_size, self.output_layer_size))
        self.b1 = np.zeros((1, self.hidden_layer_1_size))
        self.b2 = np.zeros((1, self.hidden_layer_2_size))
        self.b3 = np.zeros((1, self.output_layer_size))
    def forward_pass(self,X):
        self.z1 = np.dot(X, self.W1)+self.b1
        self.a1 = sigmoid(self.z1)
        self.z2 = np.dot(self.a1, self.W2)+self.b2
        self.a2 = sigmoid(self.z2)
        self.z3 = np.dot(self.a2,self.W3)+self.b3
        y_hat = sigmoid(self.z3)
        return y_hat
    def backward(self, X, y, y_hat):
        delta3 = (y - y_hat) * sigmoid_derivative(y_hat)
        delta2 = np.dot(delta3, self.W3.T) * sigmoid_derivative(self.a2)
        delta1 = np.dot(delta2, self.W2.T) * sigmoid_derivative(self.a1)

        dW3 = np.dot(self.a2.T, delta3)
        db3 = np.sum(delta3, axis=0, keepdims=True)
        dW2 = np.dot(self.a1.T, delta2)
        db2 = np.sum(delta3, axis=0, keepdims=True)
        dW1 = np.dot(X.T, delta1)
        db1 = np.sum(delta2, axis=0)

        self.W3 += self.learning_rate * dW3
        self.W2 += self.learning_rate * dW2
        self.W1 += self.learning_rate * dW1
        self.b3 += self.learning_rate * db3
        self.b2 += self.learning_rate * db2
        self.b1 += self.learning_rate * db1
    def train(self, X, y, epochs):
        for i in range(epochs):
            y_hat = self.forward_pass(X)
            self.backward(X, y, y_hat)

        return self.forward_pass(X)
```

[18]:
```
X = np.array([[1, -1, -1], [1, -1, 1], [1, 1, -1], [1, 1, 1]])
y = np.array([[-1], [1], [1], [1]])
```

[19]:
```
nn = NueuralNetwork()
```

[20]:
```python
epoch=10000
```

[21]:
```python
y_pred = nn.train(X,y,epochs=epoch)
```

[22]:
```python
y_pred
```

[22]:
```
array([[0.00099895],
       [0.98014806],
       [0.97904858],
       [0.98914787]])
```

[23]:
```python
threshold = 0.5
y_pred[y_pred >= threshold] = 1
y_pred[y_pred < threshold] = -1

print("Predicted output:")
print(y_pred)
```

```
Predicted output:
[[-1.]
 [ 1.]
 [ 1.]
 [ 1.]]
```

[24]:
```python
print(f"The Accuracy Score is {accuracy_score(y, y_pred)}")
```

```
The Accuracy Score is 1.0
```

[25]:
```python
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

          -1       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         3

    accuracy                           1.00         4
   macro avg       1.00      1.00      1.00         4
weighted avg       1.00      1.00      1.00         4
```