TRIBHUVAN UNIVERSITY

Sagarmatha College of Science & Technology

Lab Report On: Neural Network

Lab Report No.: 03

Date: 2077-08-22

SUBMITTED BY

Name: Ramesh Neupane

Roll no.: 37

SUBMITTED TO

CSIT Department

Question 01

Write a python program to train 2x2x1 network using backpropagation to achieve XOR function.

Source Code

```
import numpy as np

def sigmoid(x):
    return 1/(1 + np.exp(-x))


def sigmoidDerivative(x):
    return x * (1 - x)


# Input dataset
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
t = np.array([[0], [1], [1], [0]])


epoch = 100000
lr = 0.1
ILNeurons, HLNeurons, OLNeurons = 2, 2, 1
# Random weights and bias initialization
wh = np.random.uniform(size = (ILNeurons, HLNeurons))
bh = np.random.uniform(size = (1, HLNeurons))
wo = np.random.uniform(size = (HLNeurons, OLNeurons))
bo = np.random.uniform(size = (1, OLNeurons))


print(f"Initial hidden weights:")
print(wh)
print(f"Initial hidden biases:")
print(bh)
```

```
print(f"\nInitial output weights:")
print(wo)
print(f"Initial output bias:")
print(bo)

# Training Algorithm
for i in range(epoch):
#     Forward Propagation
    vh = np.dot(x, wh)
    vh += bh
    yh = sigmoid(vh)

    vo = np.dot(yh, wo)
    vo += bo
    yo = sigmoid(vo)

#     Back-Propagation
    error = t - yo
    deltao = error * sigmoidDerivative(yo)

    hidden_error = deltao.dot(wo.T)
    deltah = hidden_error * sigmoidDerivative(yh)

#     weight update
    wo += yh.T.dot(deltao) * lr
    bo += np.sum(deltao, axis=0, keepdims=True) * lr
    wh += x.T.dot(deltah) * lr
    bh = np.sum(deltah, axis=0, keepdims=True) * lr
```

```
print(f"\nFinal hidden weights:\n{wh}")
print(f"Final hidden biases:\n{bh}")
print(f"\nFinal output weights:\n{wo}")
print(f"Final output bias:\n{bo}")
print(f"\nOutput after {epoch} epoch:\n{yo}")
```

Output

```
Initial hidden weights:
[[0.50849646 0.45653358]
 [0.74856586 0.0362469 ]]
Initial hidden biases:
[[0.49645444 0.69028227]]

Initial output weights:
[[0.68368952]
 [0.42595845]]
Initial output bias:
[[0.09997033]]

Final hidden weights:
[[7.83208159 0.93013515]
 [7.8252924  0.93012415]]
Final hidden biases:
[[-0.0007043  -0.00207886]]

Final output weights:
[[ 26.27503554]
 [-33.00677212]]
```

Final output bias:

[[-0.00798849]]

Output after 100000 epoch:

[[0.03352632]
 [0.93094489]
 [0.93094471]
 [0.09223843]]

Question 02

Write a python program to train 3x2x2x1 network using backpropagation to achieve majority function for three inputs.

Source Code

```
import numpy as np

def sigmoid(x):
    return 1/(1 + np.exp(-x))


def sigmoidDerivative(x):
    return x * (1 - x)


# Input dataset
x = np.array([[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]])
t = np.array([[0], [0], [0], [1], [0], [1], [1], [1]])


epoch = 100000
lr = 0.1
ILNeurons, HL1Neurons, HL2Neurons, OLNeurons = 3, 2, 2, 1
```

```
# Random weights and bias initialization
wh1 = np.random.uniform(size = (ILNeurons, HL1Neurons))
bh1 = np.random.uniform(size = (1, HL1Neurons))
wh2 = np.random.uniform(size = (HL1Neurons, HL2Neurons))
bh2 = np.random.uniform(size = (1, HL2Neurons))
wo = np.random.uniform(size = (HL2Neurons, OLNeurons))
bo = np.random.uniform(size = (1, OLNeurons))

print(f"Initial hidden layer 1 weights:\n{wh1}")
print(f"initial hidden layer 1 biases:\n{bh1}")
print(f"\nInitial hidden layer 2 weights:\n{wh2}")
print(f"Initial hidden layer 2 biases:\n{bh2}")
print(f"\nInitial output layer weights:\n{wo}")
print(f"Initial output layer bias:\n{bo}")


# Training Algorithm
for i in range(epoch):
#    Forward Propagation
    vh1 = np.dot(x, wh1)
    vh1 += bh1
    yh1 = sigmoid(vh1)

    vh2 = np.dot(yh1, wh2)
    vh2 += bh2
    yh2 = sigmoid(vh2)

    vo = np.dot(yh2, wo)
```

```
    vo += bo
    yo = sigmoid(vo)

#   Back-Propagation
    error = t - yo
    deltao = error * sigmoidDerivative(yo)

    hidden2_error = deltao.dot(wo.T)
    deltah2 = hidden2_error * sigmoidDerivative(yh2)

    hidden1_error = deltah2.dot(wh2.T)
    deltah1 = hidden1_error * sigmoidDerivative(yh1)

#   weight update
    wo += yh2.T.dot(deltao) * lr
    bo += np.sum(deltao, axis=0, keepdims=True) * lr

    wh2 += yh1.T.dot(deltah2) * lr
    bh = np.sum(deltah2, axis=0, keepdims=True) * lr

    wh1 += x.T.dot(deltah1) * lr
    bh1 = np.sum(deltah1, axis=0, keepdims=True) * lr

print(f"\nFinal hidden layer 1 weights:\n{wh1}")
print(f"Final hidden layer 1 biases:\n{bh1}")
print(f"\nFinal hidden layer 2 weights:\n{wh2}")
print(f"Final hidden layer 2 biases:\n{bh2}")
print(f"\nFinal output weights:\n{wo}")
```

```
print(f"Final output bias:\n{bo}")
print(f"\nOutput after {epoch} epoch:\n{yo}")
```

Output

```
Initial hidden layer 1 weights:
[[0.48415914 0.99807726]
 [0.39245753 0.17792472]
 [0.29932184 0.68691752]]
initial hidden layer 1 biases:
[[0.98990028 0.8421257 ]]

Initial hidden layer 2 weights:
[[0.40068737 0.8025034 ]
 [0.87894221 0.81344195]]
Initial hidden layer 2 biases:
[[0.75512805 0.08328155]]

Initial output layer weights:
[[0.10052292]
 [0.57858834]]
Initial output layer bias:
[[0.46549465]]

Final hidden layer 1 weights:
[[ 2.20578715  6.2322205 ]
 [ 4.64169896 -4.33373   ]
 [ 2.20578708  6.23221718]]
Final hidden layer 1 biases:
[[-0.01427052  0.01160984]]
```

Final hidden layer 2 weights:

```
[[ -4.13955628  15.13749799]
 [  5.85240207 -13.28330684]]
```

Final hidden layer 2 biases:

```
[[0.75512805 0.08328155]]
```

Final output weights:

```
[[11.96647674]
 [27.38240986]]
```

Final output bias:

```
[[-31.92428314]]
```

Output after 100000 epoch:

```
[[0.08873681]
 [0.01923363]
 [0.01625852]
 [0.99260297]
 [0.01923363]
 [0.923441  ]
 [0.99260297]
 [0.95463725]]
```