EXPERIMENT 2 (In continuation to Experiment 1)

Build Your Own MLP – XOR

CODE:

```
def dense(inputs, weights):
    return np.matmul(inputs, weights)


input_size = 2
hidden_size = 3
output_size = 1


def initialize_weights():
 # weights for hidden layer, shape: 2x3
    w1 = np.random.uniform(size=(input_size, hidden_size))
 # weights for output layer, shape: 3x1
    w2 = np.random.uniform(size=(hidden_size, output_size))
    return w1, w2


w1, w2 = initialize_weights()


def forward_pass(X):
 # Step 1: Calculate weighted average of inputs (output shape: 4x3)
    net_hidden = dense(X, w1)

 # Step 2: Calculate the result of the sigmoid activation function (shape:
4x3)
    act_hidden = sigmoid(net_hidden)

 # Step 3: Calculate output of neural network (output shape: 4x1)
    y_hat = dense(act_hidden, w2)

    return act_hidden, y_hat


def mse(y_hat, y):
    residual = y_hat - y
    error = np.mean(0.5 * (residual ** 2))
    return residual, error
```

```
def backward(X, y_hat, act_hidden):
 # Step 1: Calculate error
    residual, error = mse(y_hat, y)

 # Step 2: calculate gradient wrt w2
    N = X.shape[0]
    dL_dy = 1.0 / N * residual  # shape (4, 1)
    dy_dw2 = act_hidden  # shape (4, 3)
    dL_dw2 = np.matmul(dL_dy.T, dy_dw2)  # shape (1, 3)

 # According to the math, `dL_dw2` is a row-vector, however, `w2` is a
column-vector.
 # To prevent erroneous numpy broadcasting during the gradient update, we
must make
 # sure that `dL_dw2` is also a column-vector.
    dL_dw2 = dL_dw2.T

 # Step 3: calculate gradient wrt w1
    da_dh = sigmoid_(act_hidden)
    dL_dw1 = np.zeros_like(w1)
    for i in range(w1.shape[0]):
        for j in range(w1.shape[1]):
         # Note: using `residual[:, 0]` instead of just `residual` is
important here, as otherwise
         # numpy broadcasting will make `s` a 4x4 matrix, which is wrong
            s = residual[:, 0] * w2[j, 0] * da_dh[:, j] * X[:, i]
            dL_dw1[i, j] = np.mean(s)
    return dL_dw2, dL_dw


def d_faster(X, y_hat, act_hidden):
 # Step 1: Calculate error
    residual, error = mse(y_hat, y)

 # Step 2: calculate gradient wrt w2
    N = X.shape[0]
    dL_dy = 1.0 / N * residual  # shape (4, 1)
    dy_dw2 = act_hidden  # shape (4, 3)
    dL_dw2 = np.matmul(dL_dy.T, dy_dw2)  # shape (1, 3)

 # According to the math, `dL_dw2` is a row-vector, however, `w2` is a
column-vector.
 # To prevent erroneous numpy broadcasting during the gradient update, we
must make
 # sure that `dL_dw2` is also a column-vector.
```

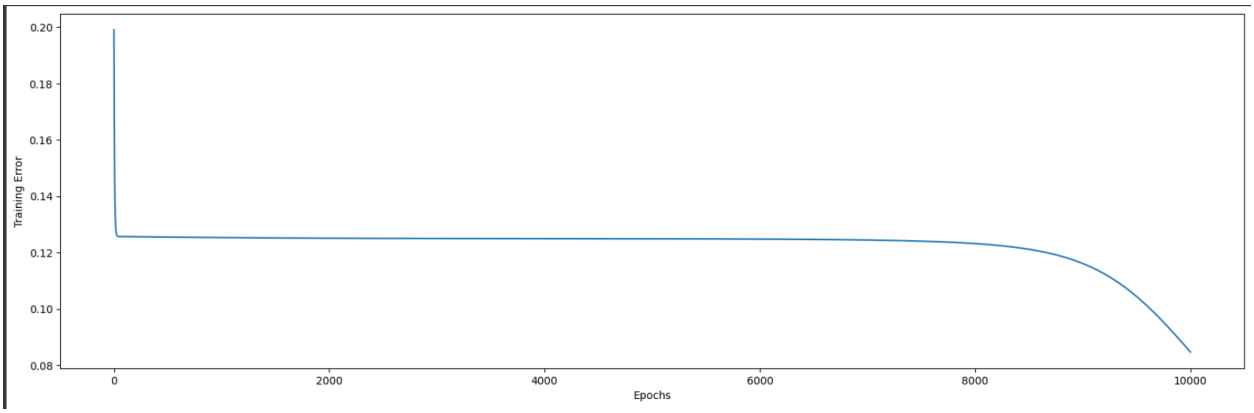
```
    dL_dw2 = dL_dw2.T

 # Step 3: calculate gradient wrt w1
    da_dh = sigmoid_(act_hidden)  # shape (4, 3)asting by numpy
    dL_dw1 = 1.0 / N * np.matmul(X.T, dL_dw1)  # shape (2, 3)

    return dL_dw2, dL_dw1


def backward_pass(X, y_hat, act_hidden):
 # Step 1: Calculate error
    residual, error = mse(y_hat, y)

 # Step 2: calculate gradient wrt w2
    N = X.shape[0]
    dL_dy = 1.0 / N * residual  # shape (4, 1)
    dy_dw2 = act_hidden  # shape (4, 3)
    dL_dw2 = np.matmul(dL_dy.T, dy_dw2)  # shape (1, 3)

 # According to the math, `dL_dw2` is a row-vector, however, `w2` is a
column-vector.
 # To prevent erroneous numpy broadcasting during the gradient update, we
must make
 # sure that `dL_dw2` is also a column-vector.
    dL_dw2 = dL_dw2.T

 # Step 3: calculate gradient wrt w1
    dL_dw1 = 1.0 / N * \
        np.matmul(X.T, np.matmul(residual, w2.T) * sigmoid_(act_hidden))

    return dL_dw2, dL_dw1, error


n_epochs = 10000
learning_rate = 0.1
training_errors = []
# re-initialize the weights to be sure we start fresh
w1, w2 = initialize_weights()
for epoch in range(n_epochs):
 # Step 1: forward pass
    act_hidden, y_hat = forward_pass(X)

 # Step 2: backward pass
    dw2, dw1, error = backward_pass(X, y_hat, act_hidden)
```

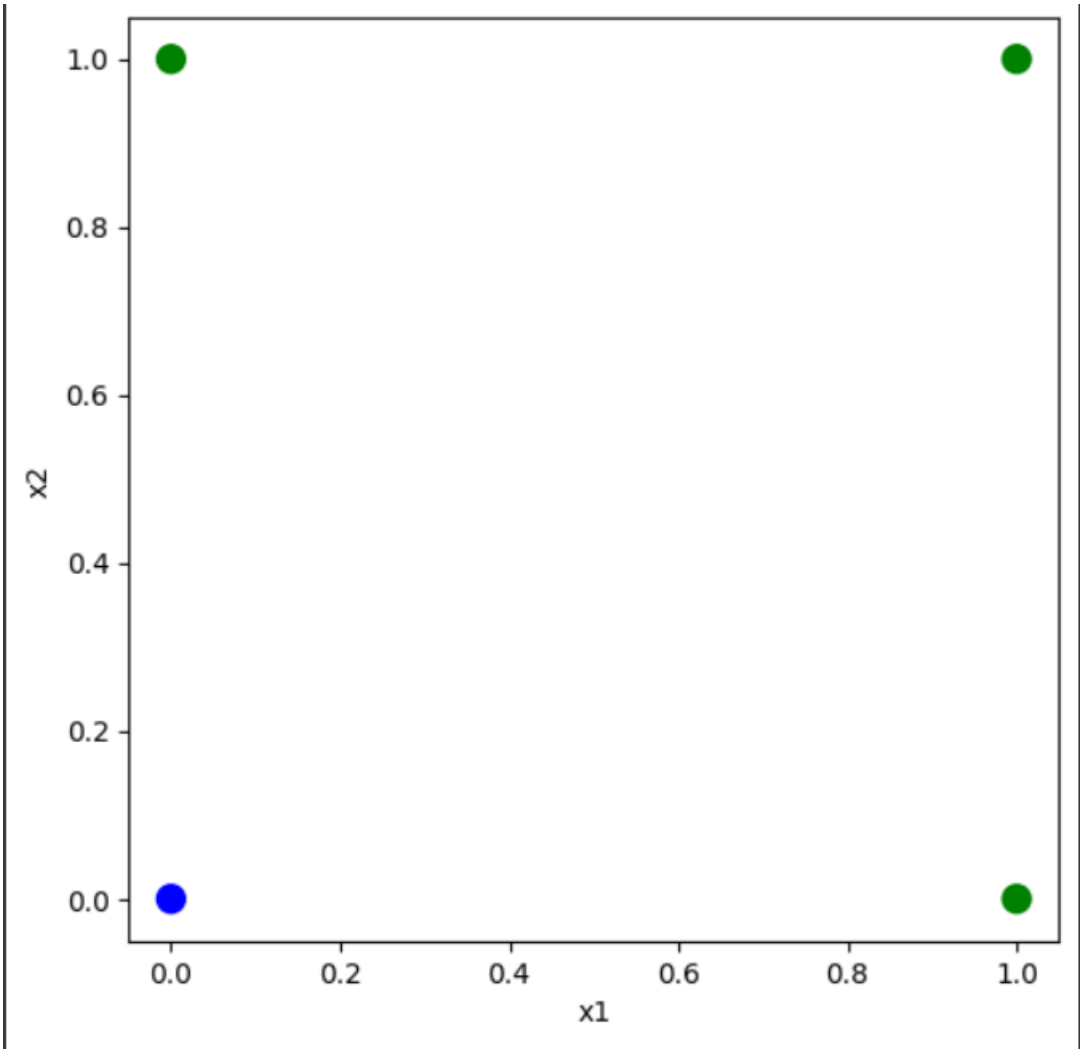
```
 # Step 3: apply gradients scaled by learning rate
    w2 = w2 - learning_rate * dw2
    w1 = w1 - learning_rate * dw1

 # Step 4: some book-keeping and print-out
    if epoch % 200 == 0:
        print('Epoch %d> Training error: %f' % (epoch, error))
    training_errors.append([epoch, error])
# Plot training error progression over time
training_errors = np.asarray(training_errors)
plt.plot(training_errors[:, 0], training_errors[:, 1])
plt.xlabel('Epochs')
plt.ylabel('Training Error')

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_hat = [np.round(forward_pass(x)[1]) for x in X]
# Colors corresponding to class predictions y_hat.
colors = ['green' if y_ == 1 else 'blue' for y_ in y_hat]
fig = plt.figure()
fig.set_figwidth(6)
fig.set_figheight(6)
plt.scatter(X[:, 0], X[:, 1], s=100, c=colors)
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()
resolution = 20
min_x, min_y = 0.0, 0.0
max_x, max_y = 1.0, 1.0
xv, yv = np.meshgrid(np.linspace(min_x, max_x, resolution),
                     np.linspace(min_y, max_y, resolution))
X_extended = np.concatenate(
    [xv[..., np.newaxis], yv[..., np.newaxis]], axis=-1)
X_extended = np.reshape(X_extended, [-1, 2])
y_hat = [np.round(forward_pass(x)[1]) for x in X_extended]
# Colors corresponding to class predictions y_hat.
colors = ['green' if y_ == 1 else 'blue' for y_ in y_hat]
fig = plt.figure()
fig.set_figwidth(6)
fig.set_figheight(6)
plt.scatter(X_extended[:, 0], X_extended[:, 1], s=200, c=colors)
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()
```

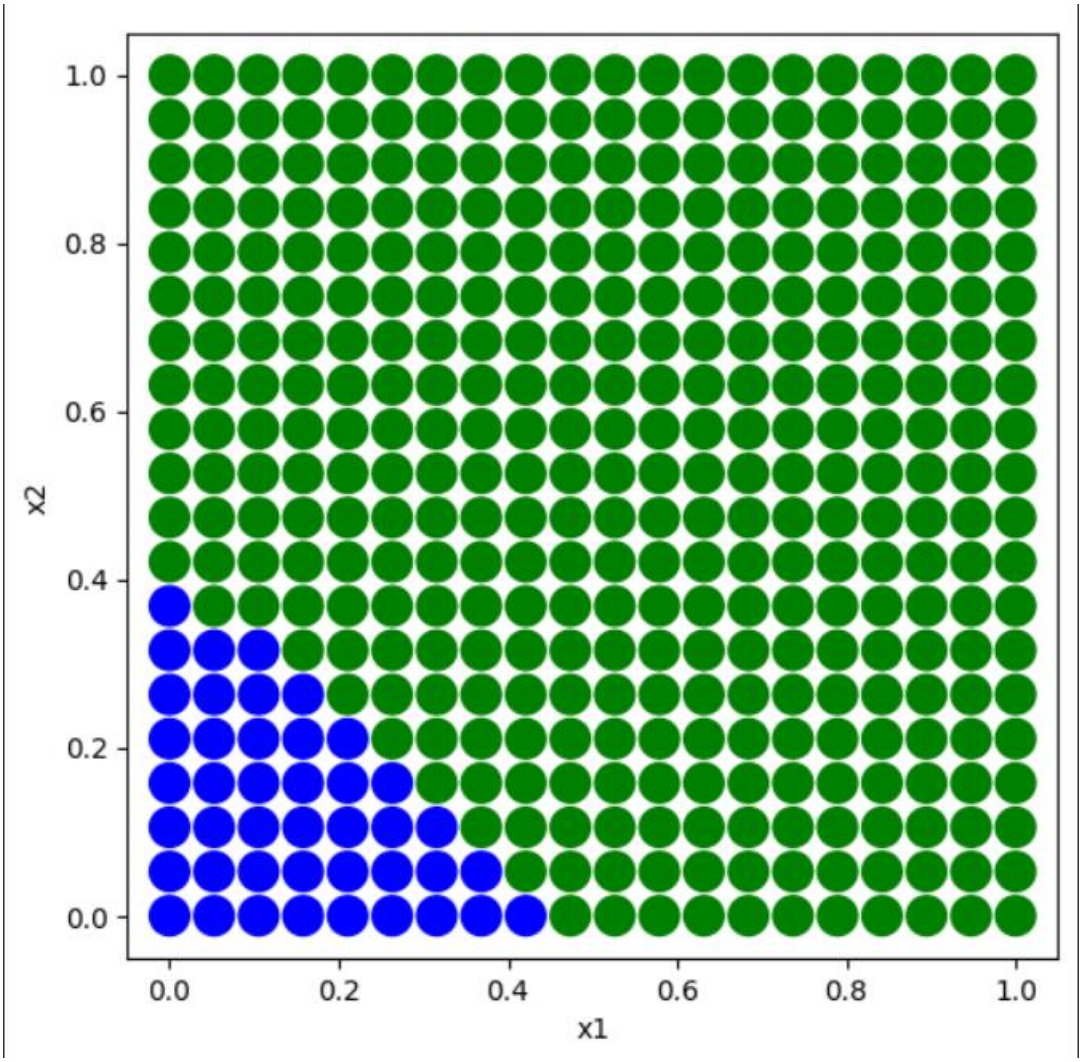
OUTPUT:

```
Epoch 0> Training error: 0.199057
Epoch 200> Training error: 0.125606
Epoch 400> Training error: 0.125517
Epoch 600> Training error: 0.125438
Epoch 800> Training error: 0.125366
Epoch 1000> Training error: 0.125302
Epoch 1200> Training error: 0.125244
Epoch 1400> Training error: 0.125194
Epoch 1600> Training error: 0.125150
Epoch 1800> Training error: 0.125112
Epoch 2000> Training error: 0.125080
Epoch 2200> Training error: 0.125054
Epoch 2400> Training error: 0.125032
Epoch 2600> Training error: 0.125015
Epoch 2800> Training error: 0.125000
Epoch 3000> Training error: 0.124988
Epoch 3200> Training error: 0.124978
Epoch 3400> Training error: 0.124969
Epoch 3600> Training error: 0.124961
Epoch 3800> Training error: 0.124953
Epoch 4000> Training error: 0.124945
Epoch 4200> Training error: 0.124936
Epoch 4400> Training error: 0.124927
Epoch 4600> Training error: 0.124917
Epoch 4800> Training error: 0.124905
Epoch 5000> Training error: 0.124892
Epoch 5200> Training error: 0.124876
Epoch 5400> Training error: 0.124857
Epoch 5600> Training error: 0.124835
Epoch 5800> Training error: 0.124808
Epoch 6000> Training error: 0.124776
Epoch 6200> Training error: 0.124736
Epoch 6400> Training error: 0.124687
Epoch 6600> Training error: 0.124626
Epoch 6800> Training error: 0.124548
Epoch 7000> Training error: 0.124448
```

```
Epoch 7200> Training error: 0.124318
Epoch 7400> Training error: 0.124148
Epoch 7600> Training error: 0.123920
Epoch 7800> Training error: 0.123610
Epoch 8000> Training error: 0.123181
Epoch 8200> Training error: 0.122579
Epoch 8400> Training error: 0.121721
Epoch 8600> Training error: 0.120483
Epoch 8800> Training error: 0.118690
Epoch 9000> Training error: 0.116103
Epoch 9200> Training error: 0.112440
Epoch 9400> Training error: 0.107440
Epoch 9600> Training error: 0.100987
Epoch 9800> Training error: 0.093249
```


Training Error
Epochs
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0
2000
4000
6000
8000
10000


x2
x1
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
x2
0.0
0.2
0.4
0.6
0.8
1.0
x1