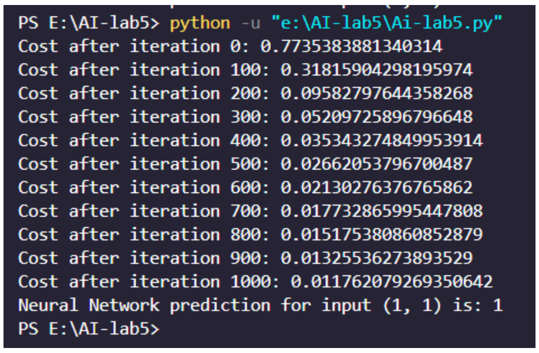
```
import numpy as np

#activation fn
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def initialize_parameters(n_x, n_h, n_y):
    W1 = np.random.randn(n_h, n_x)  # weights for input-hidden layer
    b1 = np.zeros((n_h, 1))         # Bias for hidden layer
    W2 = np.random.randn(n_y, n_h)  # weights for hidden-output
    b2 = np.zeros((n_y, 1))         # Bias for output layer
    parameters = {
        "W1": W1,
        "b1": b1,
        "W2": W2,
        "b2": b2
    }
    return parameters


def forward_prop(X, parameters):
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]


    # A1 and A2... predicted output
    Z1 = np.dot(W1, X) + b1
    A1 = np.tanh(Z1)

    Z2 = np.dot(W2, A1) + b2
    A2 = sigmoid(Z2)

    cache = {"A1": A1, "A2": A2}
    return A2, cache


def calculate_cost(A2, Y):

    m = Y.shape[1]  # no. of training examples
    cost = -np.sum(Y * np.log(A2) + (1 - Y) * np.log(1 - A2)) / m
    return np.squeeze(cost)


def backward_prop(X, Y, cache, parameters):
    m = X.shape[1]
    A1 = cache["A1"]
    A2 = cache["A2"]
    W2 = parameters["W2"]

    dZ2 = A2 - Y                     # Output error

    #--------------gradient for w2 and b2------------------
    dW2 = np.dot(dZ2, A1.T) / m
    db2 = np.sum(dZ2, axis=1, keepdims=True) / m
    #-------------------------------------------------------

    dZ1 = np.dot(W2.T, dZ2) * (1 - np.power(A1, 2))  # Hidden layer error

    #---------------gradient for w1 and b1------------------
    dW1 = np.dot(dZ1, X.T) / m
    db1 = np.sum(dZ1, axis=1, keepdims=True) / m
    #-------------------------------------------------------

    grads = {"dW1": dW1, "db1": db1, "dW2": dW2, "db2": db2}
    return grads


def update_parameters(parameters, grads, learning_rate):
    W1 = parameters["W1"] - learning_rate * grads["dW1"]
    b1 = parameters["b1"] - learning_rate * grads["db1"]
    W2 = parameters["W2"] - learning_rate * grads["dW2"]
    b2 = parameters["b2"] - learning_rate * grads["db2"]

    new_parameters = {"W1": W1, "b1": b1, "W2": W2, "b2": b2}
    return new_parameters
```

```python
# training
def model(X, Y, n_x, n_h, n_y, num_of_iters, learning_rate):
    parameters = initialize_parameters(n_x, n_h, n_y)

    for i in range(num_of_iters + 1):
        A2, cache = forward_prop(X, parameters)
        cost = calculate_cost(A2, Y)
        grads = backward_prop(X, Y, cache, parameters)
        parameters = update_parameters(parameters, grads, learning_rate)

        if i % 100 == 0:
            print(f'Cost after iteration {i}: {cost}')

    return parameters


def predict(X, parameters):
    A2, cache = forward_prop(X, parameters)
    yhat = np.squeeze(A2)

    if yhat >= 0.5:
        return 1
    else:
        return 0



np.random.seed(2)

# XNOR truth table
X = np.array([[0, 0, 1, 1], [0, 1, 0, 1]])  # i/ps
Y = np.array([[1, 0, 0, 1]])  # o/ps


n_x = 2  #neurons in input layer
n_h = 2  #neurons in hidden layer
n_y = 1  #neurons output layer
num_of_iters = 1000
learning_rate = 0.3

# train
trained_parameters = model(X, Y, n_x, n_h, n_y, num_of_iters, learning_rate)

# Test the model with a new input
X_test = np.array([[1], [1]])
y_predict = predict(X_test, trained_parameters)

print(f'Neural Network prediction for input (1, 1) is: {y_predict}')
```



```
PS E:\AI-lab5> python -u "e:\AI-lab5\Ai-lab5.py"
Cost after iteration 0: 0.7735383881340314
Cost after iteration 100: 0.31815904298195974
Cost after iteration 200: 0.09582797644358268
Cost after iteration 300: 0.05209725896796648
Cost after iteration 400: 0.035343274849953914
Cost after iteration 500: 0.02662053796700487
Cost after iteration 600: 0.02130276376765862
Cost after iteration 700: 0.017732865995447808
Cost after iteration 800: 0.015175380860852879
Cost after iteration 900: 0.01325536273893529
Cost after iteration 1000: 0.011762079269350642
Neural Network prediction for input (1, 1) is: 1
PS E:\AI-lab5>
```