```
import pandas as pd
import numpy as np
```

```
df  = pd.DataFrame([[8, 8, 1], [7, 9, 1], [6, 10, 0], [5, 12, 0]], columns = ['CGPA', 'ATS Score', 'Placed'])
df
```

	CGPA	ATS Score	Placed
0	8	8	1
1	7	9	1
2	6	10	0
3	5	12	0

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
def initialize_parameters(layer_dimensions):
    # Set a fixed random seed to make the results reproducible
    np.random.seed(3)

    # Create an empty dictionary to store the weight and bias parameters
    parameters = {}

    # Get the number of layers in the network (including input and output layers)
    L = len(layer_dimensions)

    # Loop through layers 1 to L-1 (we skip layer 0 because it's the input layer)
    for l in range(1, L):
        # Initialize weights for layer l with small values (here all 0.1 for simplicity)
        # Shape: (number of neurons in previous layer, number of neurons in current layer)
        parameters['W' + str(l)] = np.ones((layer_dimensions[l - 1], layer_dimensions[l])) * 0.1

        # Initialize biases for layer l with zeros
        # Shape: (number of neurons in current layer, 1)
        parameters['b' + str(l)] = np.zeros((layer_dimensions[l], 1))

    # Return the dictionary containing all the initialized parameters
    return parameters
```

```
# Utility Functions
def sigmoid(Z):

  A = 1/(1+np.exp(-Z))

  return A
```

```
def linear_forward(A_prev, W, b):
    # A_prev: activations from the previous layer (shape: size of previous layer x number of examples)
    # W: weights matrix for the current layer (shape: size of previous layer x size of current layer)
    # b: bias vector for the current layer (shape: size of current layer x 1)

    # Compute the linear transformation Z = W^T * A_prev + b
    # W.T gives shape: (size of current layer x number of examples)
    Z = np.dot(W.T, A_prev) + b

    # Apply sigmoid activation to the linear output Z
    A = sigmoid(Z)

    # Return the activation output A to be passed to the next layer
    return A
```

```
# Main Forward Propagation function for a single row instance (i.e., one data sample)
def L_layer_forward(X, parameters):
    # X: input vector of shape (input size, 1) - i.e., a single column (one data instance)
    # parameters: dictionary containing all the weights and biases for each layer

    A = X  # Initial activation is the input feature vector
    L = len(parameters) // 2  # Number of layers in the network (each has W and b)

    # Loop through all the layers in the network
    for l in range(1, L + 1):
        A_prev = A  # Store previous activation for computing current layer's activation

        # Retrieve current layer's weights and bias
        Wl = parameters['W' + str(l)]
        bl = parameters['b' + str(l)]

        # Perform linear forward computation: Z = W.T @ A_prev + b
        # NOTE: Your 'W' is stored as shape (prev_layer, current_layer), so we need W.T
        A = linear_forward(A_prev, Wl, bl)

        # Optional debugging prints:
        print("A" + str(l-1) + ": ", A_prev)
        print("W" + str(l) + ": ", Wl)
```

```
        print("b" + str(l) + ": ", bl)
        print("A" + str(l) + ": ", A)
        print("--" * 20)

    # Return final output (last activation), and the last activation before it
    return A, A_prev
```

```
X = df[['CGPA', 'ATS Score']].values[0].reshape(2,1) # Shape(no of features, no. of training example)
y = df[['Placed']].values[0][0]

# Parameter initialization
parameters = initialize_parameters([2,2,1])

y_hat,A1 = L_layer_forward(X,parameters)
y_hat = y_hat[0][0]

update_parameters(parameters,y,y_hat,A1,X)

print('Loss for this student - ',-y*np.log(y_hat) - (1-y)*np.log(1-y_hat))

parameters
```

```
A0:  [[8]
 [8]]
W1:  [[0.1 0.1]
 [0.1 0.1]]
b1:  [[0.]
 [0.]]
A1:  [[0.83201839]
 [0.83201839]]
----------------------------------------
A1:  [[0.83201839]
 [0.83201839]]
W2:  [[0.1]
 [0.1]]
b2:  [[0.]]
A2:  [[0.54150519]]
----------------------------------------
Loss for this student -  0.613402628898913
{'W1': array([[0.10000513, 0.10000513],
        [0.10000513, 0.10000513]]),
 'b1': array([[6.41054186e-07],
        [6.41054186e-07]]),
 'W2': array([[0.10003815],
        [0.10003815]]),
 'b2': array([[4.5849481e-05]])}
```

```
# Above:
# A0 = [8, 8] — This is the input vector (features from the first row of the dataset)
# W1 = Weights for Layer 1 — These connect the input layer to the first hidden layer
# b1 = [b11, b12] — Biases for the neurons in hidden layer 1

# These are passed to linear_forward(A_prev, W1, b1) which computes:
# Z1 = W1.T @ A0 + b1
# A1 = Z1 in this case (since there's no activation function yet), so:
# A1 = [O11, O12] — outputs from the first hidden layer (linear combinations)

# Next layer:
# A1 is now passed as input to the next layer with W2, b2 to get:
# A2 = [O21] — the output from the final layer (the prediction)

# The function L_layer_forward returns:
# - A2: the final output (predicted Package)
# - A1: the activation from the previous layer (needed for backprop)

# WHY RETURN A_prev TOO?
# During backpropagation, we compute gradients like:
# ∂L/∂W = A_prev.T @ dZ
# That’s why we need the activations of the previous layer (A_prev or Oij values)
# They're essential for calculating the gradients during the backward pass.
```

```
def update_parameters(parameters, y, y_hat, A1, X):
    # --------------------------
    # Output layer (layer 2) updates
    # --------------------------

    # Gradient of loss w.r.t W2[0][0] = (dL/dy_hat) * (dy_hat/dz2) * (dz2/dW2[0]) = (y - y_hat) * A1[0]
    parameters['W2'][0][0] += 0.0001 * (y - y_hat) * A1[0][0]

    # Gradient of loss w.r.t W2[1][0] = (y - y_hat) * A1[1]
    parameters['W2'][1][0] += 0.0001 * (y - y_hat) * A1[1][0]

    # Bias update for output layer: gradient = (y - y_hat)
    parameters['b2'][0][0] += 0.0001 * (y - y_hat)

    # --------------------------
    # Hidden layer (layer 1) updates
    # --------------------------

    # For neuron 1 in hidden layer:
```

```
    # Using derivative of sigmoid: A1[0][0] * (1 - A1[0][0])
    # Multiply with W2[0][0] as it contributes to loss via output
    grad_hidden1 = (y - y_hat) * parameters['W2'][0][0] * A1[0][0] * (1 - A1[0][0])

    # Update W1 weights for neuron 1 (based on input X[0] and X[1])
    parameters['W1'][0][0] += 0.0001 * grad_hidden1 * X[0][0]
    parameters['W1'][0][1] += 0.0001 * grad_hidden1 * X[1][0]

    # Update bias for neuron 1
    parameters['b1'][0][0] += 0.0001 * grad_hidden1

    # For neuron 2 in hidden layer:
    grad_hidden2 = (y - y_hat) * parameters['W2'][1][0] * A1[1][0] * (1 - A1[1][0])

    # Update W1 weights for neuron 2
    parameters['W1'][1][0] += 0.0001 * grad_hidden2 * X[0][0]
    parameters['W1'][1][1] += 0.0001 * grad_hidden2 * X[1][0]

    # Update bias for neuron 2
    parameters['b1'][1][0] += 0.0001 * grad_hidden2
```

```
X = df[['CGPA', 'ATS Score']].values[1].reshape(2,1) # Shape(no of features, no. of training example)
y = df[['Placed']].values[1][0]

y_hat,A1 = L_layer_forward(X,parameters)
y_hat = y_hat[0][0]

update_parameters(parameters,y,y_hat,A1,X)

print('Loss for this student - ',-y*np.log(y_hat) - (1-y)*np.log(1-y_hat))

parameters
```

```
A0:  [[7]
 [9]]
W1:  [[0.10000513 0.10000513]
 [0.10000513 0.10000513]]
b1:  [[6.41054186e-07]
 [6.41054186e-07]]
A1:  [[0.83202994]
 [0.83202994]]
----------------------------------------
A1:  [[0.83202994]
 [0.83202994]]
W2:  [[0.10003815]
 [0.10003815]]
b2:  [[4.5849481e-05]]
A2:  [[0.54153291]]
----------------------------------------
Loss for this student -  0.6133514436691428
{'W1': array([[0.10000962, 0.1000109 ],
        [0.10000962, 0.1000109 ]]),
 'b1': array([[1.28227883e-06],
        [1.28227883e-06]]),
 'W2': array([[0.10007629],
        [0.10007629]]),
 'b2': array([[9.16961903e-05]])}
```

```
# Extract input features for the third training example (index 2)
X = df[['CGPA', 'ATS Score']].values[2].reshape(2,1) # Shape(no of features, no. of training example)
y = df[['Placed']].values[2][0]

y_hat,A1 = L_layer_forward(X,parameters)
y_hat = y_hat[0][0]

update_parameters(parameters,y,y_hat,A1,X)

print('Loss for this student - ',-y*np.log(y_hat) - (1-y)*np.log(1-y_hat))

parameters
```

```
A0:  [[ 6]
 [10]]
W1:  [[0.10000962 0.1000109 ]
 [0.10000962 0.1000109 ]]
b1:  [[1.28227883e-06]
 [1.28227883e-06]]
A1:  [[0.83204007]
 [0.83204294]]
----------------------------------------
A1:  [[0.83204007]
 [0.83204294]]
W2:  [[0.10007629]
 [0.10007629]]
b2:  [[9.16961903e-05]]
A2:  [[0.54156062]]
----------------------------------------
Loss for this student -  0.7799272184937318
{'W1': array([[0.10000507, 0.10000333],
        [0.10000507, 0.10000333]]),
 'b1': array([[5.25214767e-07],
        [5.25225084e-07]]),
 'W2': array([[0.10003123],
```

```
               [0.10003123]]),
       'b2': array([[3.75401279e-05]])}
```

```
update_parameters(parameters,y,y_hat,A1,X)
parameters
```

```
{'W1': array([[0.10000053, 0.09999576],
               [0.10000053, 0.09999576]]),
       'b1': array([[-2.31508272e-07],
               [-2.31487641e-07]]),
       'W2': array([[0.09998617],
               [0.09998617]]),
       'b2': array([[-1.66159345e-05]])}
```

```
# Extract input features for the 4th training example (index 2)
X = df[['CGPA', 'ATS Score']].values[3].reshape(2,1) # Shape(no of features, no. of training example)
y = df[['Placed']].values[3][0]

y_hat,A1 = L_layer_forward(X,parameters)
y_hat = y_hat[0][0]

update_parameters(parameters,y,y_hat,A1,X)

print('Loss for this student - ',-y*np.log(y_hat) - (1-y)*np.log(1-y_hat))

parameters
```

```
A0:  [[ 5]
 [12]]
W1:  [[0.10000053 0.09999576]
 [0.10000053 0.09999576]]
b1:  [[-2.31508272e-07]
 [-2.31487641e-07]]
A1:  [[0.84553589]
 [0.84552529]]
----------------------------------------
A1:  [[0.84553589]
 [0.84552529]]
W2:  [[0.09998617]
 [0.09998617]]
b2:  [[-1.66159345e-05]]
A2:  [[0.54216614]]
----------------------------------------
Loss for this student -  0.7812489129203491
{'W1': array([[0.099997  , 0.09998727],
        [0.099997  , 0.09998727]]),
 'b1': array([[-9.39181566e-07],
        [-9.39200617e-07]]),
 'W2': array([[0.09994033],
        [0.09994033]]),
 'b2': array([[-7.08325485e-05]])}
```

```
# Till now, the whole dataset got trained for 1 time(1 epoch)
# Let's start with epoch implementation
```

```
# Outer loop: iterate through entire dataset multiple times (epochs)

parameters = initialize_parameters([2,2,1])
epochs = 50

for i in range(epochs):

  Loss = []

  for j in range(df.shape[0]):

    X = df[['CGPA', 'ATS Score']].values[j].reshape(2,1) # Shape(no of features, no. of training example)
    y = df[['Placed']].values[j][0]

    # Parameter initialization


    y_hat,A1 = L_layer_forward(X,parameters)
    y_hat = y_hat[0][0]

    update_parameters(parameters,y,y_hat,A1,X)

    Loss.append(-y*np.log(y_hat) - (1-y)*np.log(1-y_hat))

  print('Epoch - ',i+1,'Loss - ',np.array(Loss).mean())

parameters
```

```
----------------------------------------
A1:  [[0.83219719]
 [0.83146478]]
W2:  [[0.09932622]
 [0.09932647]]
b2:  [[-0.00076653]]
A2:  [[0.54102728]]
----------------------------------------
A0:  [[ 6]
 [10]]
W1:  [[0.1000849  0.09975928]
 [0.10008503 0.09975881]]
b1:  [[-7.57986149e-06]
 [-7.59695223e-06]]
A1:  [[0.83220728]
 [0.83147768]]
----------------------------------------
A1:  [[0.83220728]
 [0.83147768]]
W2:  [[0.09936441]
 [0.09936463]]
b2:  [[-0.00072064]]
A2:  [[0.54105502]]
----------------------------------------
A0:  [[ 5]
 [12]]
W1:  [[0.1000804  0.09975178]
 [0.10008051 0.09975128]]
b1:  [[-8.33023947e-06]
 [-8.34993427e-06]]
A1:  [[0.84571225]
 [0.84498093]]
----------------------------------------
A1:  [[0.84571225]
 [0.84498093]]
W2:  [[0.09931939]
 [0.09931965]]
b2:  [[-0.00077474]]
A2:  [[0.54168901]]
----------------------------------------
Epoch -  50 Loss -  0.6969136669930831
{'W1': array([[0.10007689, 0.09974336],
        [0.10007699, 0.09974283]]),
 'b1': array([[-9.03191730e-06],
        [-9.05433049e-06]]),
 'W2': array([[0.09927357],
        [0.09927387]]),
 'b2': array([[-0.00082891]])}
```

```
# After all epochs, print the final learned parameters
# These include W1, b1, W2, b2 after being refined over 50 epochs
parameters
```

```
{'W1': array([[0.10007689, 0.09974336],
        [0.10007699, 0.09974283]]),
 'b1': array([[-9.03191730e-06],
        [-9.05433049e-06]]),
 'W2': array([[0.09927357],
        [0.09927387]]),
 'b2': array([[-0.00082891]])}
```

Start coding or generate with AI.