Churn\_Modelling-4 (Score: 100.0 / 100.0)

1. [Test cell](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#cell-3e7d4f612d5c1e6d) (Score: 100.0 / 100.0)

You are tasked with creating a simple artificial neural network (ANN) from scratch

1. Setup the Environment Import necessary libraries Load the dataset
2. Data Preprocessing Normalize the data Split the data into training and testing sets
3. Define the ANN from Scratch Initialize parameters Define the activation function (e.g., ReLU, Sigmoid) Implement forward propagation Implement the cost function (e.g., Cross-Entropy Loss) Implement backward propagation Implement the update rule (Gradient Descent) Compile the model
4. Training the Model Implement a training loop Track and print the loss over epochs
5. Evaluation Implement a function to evaluate accuracy on the test set

In [ ]:

Step 1 Import all the necessary libraries[¶](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#Step-1-Import-all-the-necessary-libraries)

In [ ]:

*#Import all the necessary libraries*

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**from** **sklearn.preprocessing** **import** StandardScaler

**from** **sklearn.model\_selection** **import** train\_test\_split

Step 2 Load the dataset ,Select dependent and Independent variable X,y[¶](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#Step-2-Load-the-dataset-,Select-dependent-and-Independent-variable-X,y)

You are provided with a dataset named Churn\_Modelling.csv, which contains various features related to customers of a company. Your goal is to load this dataset, separate the features from the target variable, and prepare them for training ANN model. Task: Load the Dataset: Use the pandas library to load the Churn\_Modelling.csv file into a DataFrame. Prepare Features (X): Drop the Exited column from the DataFrame, as it is the target variable. Convert the remaining data into a NumPy array. This will serve as the feature set X. Prepare Target (y): Extract the Exited column from the DataFrame. Convert it into a NumPy array and reshape it to ensure it has the correct shape for machine learning algorithms (i.e., a column vector with one output per row).

In [ ]:

data = pd.read\_csv('/srv/shareddata/datasets/ps2/Churn\_Modelling/Churn\_Modelling.csv')

X = data.drop(['Exited'], axis=1).values

y = data['Exited'].values.reshape(-1, 1)

Step 3 Use Standard Scaler(X\_scaled) and split the data into training and testing (X\_train, X\_test, y\_train, y\_test)[¶](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#Step-3-Use-Standard-Scaler(X_scaled)-and-split-the-data-into-training-and-testing-(X_train,-X_test,-y_train,-y_test))

In this task, you will continue preparing the data for training a churn prediction model. After loading and preprocessing the dataset, the next steps involve scaling the features and splitting the data into training and testing sets. These steps are essential to ensure that your machine learning model performs well and can generalize to unseen data. Task: Scale the Features: Use the StandardScaler to scale the features (X) that you prepared in the previous task. Store the scaled features in a variable called X\_scaled. Split the Data: Use the train\_test\_split function from the sklearn.model\_selection module to split the scaled data (X\_scaled) and the target variable (y) into training and testing sets. Assign 80% of the data to the training set and 20% to the testing set. Use a random seed (random\_state=42) to ensure reproducibility. The resulting variables should be X\_train, X\_test, y\_train, and y\_test.

In [ ]:

*# Step 3 Use Standard Scaler(X\_scaled)*

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

*# Split the data into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

In [ ]:

input\_size = X\_train.shape[1] *# Number of input features*

hidden\_size = 10 *# Number of neurons in the hidden layer*

output\_size = 1 *# Binary classification (1 output node)*

Step 3 define a function (initialize\_parameters) to initialize weights(w1,w2) using rand and biases (b1,b2) using zeros[¶](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#Step-3-define-a-function-(initialize_parameters)-to-initialize-weights(w1,w2)-using-rand-and-biases-(b1,b2)-using-zeros)

In [ ]:

In this task, you will implement a function to initialize the parameters of a simple neural network. Proper initialization of the network's parameters is crucial for training effective machine learning models. This initialization includes the weights and biases of the network layers. Function Signature: def initialize\_parameters(input\_size, hidden\_size, output\_size): # Your code here return W1, b1, W2, b2 Parameters: input\_size: An integer representing the number of input features for the network. hidden\_size: An integer representing the number of neurons in the hidden layer. output\_size: An integer representing the number of output neurons (i.e., the number of classes or regression outputs). Expected Outputs: W1: A NumPy array of shape (input\_size, hidden\_size) representing the weights of the layer connecting the input to the hidden layer. Initialize these weights with small random values scaled by 0.01. b1: A NumPy array of shape (1, hidden\_size) representing the biases of the hidden layer. Initialize these biases to zero. W2: A NumPy array of shape (hidden\_size, output\_size) representing the weights of the layer connecting the hidden layer to the output layer. Initialize these weights with small random values scaled by 0.01. b2: A NumPy array of shape (1, output\_size) representing the biases of the output layer. Initialize these biases to zero.

In [ ]:

**def** initialize\_parameters(input\_size, hidden\_size, output\_size):

np.random.seed(42)

W1 = np.random.randn(input\_size, hidden\_size) \* 0.01

b1 = np.zeros((1, hidden\_size))

W2 = np.random.randn(hidden\_size, output\_size) \* 0.01

b2 = np.zeros((1, output\_size))

**return** W1, b1, W2, b2

step 4: Implementing Sigmoid Activation Function and Its Derivative[¶](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#step-4:-Implementing-Sigmoid-Activation-Function-and-Its-Derivative)

In this task, you will implement two fundamental functions used in neural networks: the sigmoid activation function and its derivative. These functions are crucial for introducing non-linearity into the model and for performing backpropagation during the training process. Function Signatures: def sigmoid(z): # Your code here return result def sigmoid\_derivative(z): # Your code here return resultExpected Outputs: sigmoid(z): Input: A NumPy array or a scalar 𝑧 z. Output: A NumPy array or scalar with the sigmoid function applied element-wise. sigmoid\_derivative(z): Input: A NumPy array or a scalar 𝑧 z that represents the output of the sigmoid function (i.e., values between 0 and 1). Output: A NumPy array or scalar with the derivative of the sigmoid function applied element-wise.

In [ ]:

*#Sigmoid activation function*

**def** sigmoid(z):

**return** 1 / (1 + np.exp(-z))

*# Derivative of the sigmoid function*

**def** sigmoid\_derivative(z):

**return** z \* (1 - z)

step5 Implement forward\_propagation[¶](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#step5-Implement-forward_propagation)

In this task, you will implement the forward propagation step of a simple neural network. Forward propagation involves passing input data through the network layers to obtain predictions. This step is crucial for both training the model (to compute the loss) and making predictions on new data.For this task, the network consists of: An input layer with features 𝑋. A hidden layer with weights 𝑊1 and biases b1 An output layer with weights W2 and biases b2. Function Signature: def forward\_propagation(X, W1, b1, W2, b2): # Your code here return Z1, A1, Z2, A2 Parameters: X: A NumPy array of shape (n\_samples, input\_size) representing the input features. W1: A NumPy array of shape (input\_size, hidden\_size) representing the weights of the layer connecting the input to the hidden layer. b1: A NumPy array of shape (1, hidden\_size) representing the biases of the hidden layer. W2: A NumPy array of shape (hidden\_size, output\_size) representing the weights of the layer connecting the hidden layer to the output layer. b2: A NumPy array of shape (1, output\_size) representing the biases of the output layer. Expected Outputs: Z1: A NumPy array of shape (n\_samples, hidden\_size) representing the linear combination of inputs and weights plus biases for the hidden layer. A1: A NumPy array of shape (n\_samples, hidden\_size) representing the activation of the hidden layer, obtained by applying the sigmoid function to Z1. Z2: A NumPy array of shape (n\_samples, output\_size) representing the linear combination of activations and weights plus biases for the output layer. A2: A NumPy array of shape (n\_samples, output\_size) representing the activation of the output layer, obtained by applying the sigmoid function to Z2.

In [ ]:

**def** forward\_propagation(X, W1, b1, W2, b2):

Z1 = np.dot(X, W1) + b1

A1 = sigmoid(Z1)

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

A2 = sigmoid(Z2)

**return** Z1, A1, Z2, A2

step7: define a function (compute\_loss) to calculate loss using binary crossentropy[¶](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#step7:-define-a-function-(compute_loss)-to-calculate-loss-using-binary-crossentropy)

In this task, you will implement the cost function for a neural network used for binary classification. The cost function measures the performance of the model by quantifying the difference between the predicted values and the actual labels. The goal is to minimize this cost during the training process to improve the model's accuracy.Function Signature: def compute\_cost(A2, y): # Your code here return costParameters: A2: A NumPy array of shape (n\_samples, 1) representing the predicted probabilities from the output layer. y: A NumPy array of shape (n\_samples, 1) representing the true binary labels (0 or 1). Expected Output: cost: A scalar representing the average binary cross-entropy cost over all samples. The output should be a single float value.

In [ ]:

**def** compute\_cost(A2, y):

m = y.shape[0]

cost = -(1/m) \* np.sum(y \* np.log(A2) + (1 - y) \* np.log(1 - A2))

**return** np.squeeze(cost)

step7: Backward propagation:[¶](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#step7:-Backward-propagation:)

In this task, you will implement the backward propagation step of a simple neural network. Backward propagation involves computing the gradients of the cost function with respect to the network's parameters. These gradients are used to update the weights and biases during training to minimize the cost function.For this task, the network uses the sigmoid activation function for the output layer and ReLU activation for the hidden layer. You need to compute: Gradients of the cost with respect to weights and biases of the output layer. Gradients of the cost with respect to weights and biases of the hidden layer.Function Signature: def backward\_propagation(X, y, Z1, A1, A2, W2): # Your code here return dW1, db1, dW2, db2Parameters: X: A NumPy array of shape (n\_samples, input\_size) representing the input features. y: A NumPy array of shape (n\_samples, 1) representing the true binary labels. Z1: A NumPy array of shape (n\_samples, hidden\_size) representing the linear combination of inputs and weights plus biases for the hidden layer. A1: A NumPy array of shape (n\_samples, hidden\_size) representing the activation of the hidden layer (output of the ReLU function). A2: A NumPy array of shape (n\_samples, 1) representing the activation of the output layer (output of the sigmoid function). W2: A NumPy array of shape (hidden\_size, output\_size) representing the weights of the layer connecting the hidden layer to the output layer. Expected Outputs: dW1: A NumPy array of shape (input\_size, hidden\_size) representing the gradient of the cost function with respect to the weights of the hidden layer. db1: A NumPy array of shape (1, hidden\_size) representing the gradient of the cost function with respect to the biases of the hidden layer. dW2: A NumPy array of shape (hidden\_size, output\_size) representing the gradient of the cost function with respect to the weights of the output layer. db2: A NumPy array of shape (1, output\_size) representing the gradient of the cost function with respect to the biases of the output layer.

In [ ]:

**def** backward\_propagation(X, y, Z1, A1, A2, W2):

m = X.shape[0]

dZ2 = A2 - y

dW2 = (1/m) \* np.dot(A1.T, dZ2)

db2 = (1/m) \* np.sum(dZ2, axis=0, keepdims=**True**)

dA1 = np.dot(dZ2, W2.T)

dZ1 = dA1 \* sigmoid\_derivative(Z1)

dW1 = (1/m) \* np.dot(X.T, dZ1)

db1 = (1/m) \* np.sum(dZ1, axis=0, keepdims=**True**)

**return** dW1, db1, dW2, db2

step8: Update parameters[¶](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#step8:-Update-parameters)

In this task, you are required to implement the update\_parameters function, which is a crucial step in training an Artificial Neural Network (ANN). This function updates the weights and biases of the network using the gradients computed during backpropagation. In a neural network, weights and biases are adjusted iteratively during training to minimize the error between the predicted output and the actual target. This adjustment is done using gradient descent, where the weights and biases are updated in the direction that reduces the error. Function Signature: def update\_parameters(W1, b1, W2, b2, dW1, db1, dW2, db2, learning\_rate) #your code hereParameters: W1: A numpy array of shape (n\_input\_features, n\_hidden\_units) representing the weights of the first layer. b1: A numpy array of shape (1, n\_hidden\_units) representing the biases of the first layer. W2: A numpy array of shape (n\_hidden\_units, n\_output\_units) representing the weights of the second layer. b2: A numpy array of shape (1, n\_output\_units) representing the biases of the second layer. dW1: A numpy array of the same shape as W1, representing the gradient of the loss with respect to W1. db1: A numpy array of the same shape as b1, representing the gradient of the loss with respect to b1. dW2: A numpy array of the same shape as W2, representing the gradient of the loss with respect to W2. db2: A numpy array of the same shape as b2, representing the gradient of the loss with respect to b2. learning\_rate: A scalar representing the learning rate for gradient descent. This controls the size of the steps taken during the update.

In [ ]:

Student's answer[(Top)](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#top)

*# YOUR CODE HERE*

**def** update\_parameters(W1, b1, W2, b2, dW1, db1, dW2, db2, learning\_rate):

W1 -= learning\_rate \* dW1

b1 -= learning\_rate \* db1

W2 -= learning\_rate \* dW2

b2 -= learning\_rate \* db2

**return** W1, b1, W2, b2

Test Case 5: Update Parameters[¶](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#Test-Case-5:-Update-Parameters)

In [ ]:

Grade cell: cell-3e7d4f612d5c1e6dScore: 100.0 / 100.0 [(Top)](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#top)

step 9: Training loop[¶](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#step-9:-Training-loop)

In this task, you will implement a function to train a neural network model using gradient descent. The training process involves forward propagation to compute predictions, calculating the cost, performing backward propagation to compute gradients, and updating the parameters to minimize the cost function.Function Signature: def train\_model(X, y, hidden\_size, learning\_rate, epochs): # Your code here return W1, b1, W2, b2Parameters: X: A NumPy array of shape (n\_samples, input\_size) representing the input features. y: A NumPy array of shape (n\_samples, 1) representing the true binary labels. hidden\_size: An integer specifying the number of neurons in the hidden layer. learning\_rate: A float representing the learning rate for gradient descent. epochs: An integer specifying the number of iterations for training. Expected Output: W1: A NumPy array of shape (input\_size, hidden\_size) representing the trained weights of the hidden layer. b1: A NumPy array of shape (1, hidden\_size) representing the trained biases of the hidden layer. W2: A NumPy array of shape (hidden\_size, output\_size) representing the trained weights of the output layer. b2: A NumPy array of shape (1, output\_size) representing the trained biases of the output layer.

In [ ]:

In [ ]:

Student's answer[(Top)](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#top)

*# YOUR CODE HERE*

**def** train\_model(X, y, hidden\_size, learning\_rate, epochs):

W1, b1, W2, b2 = initialize\_parameters(input\_size, hidden\_size, output\_size)

**for** epoch **in** range(epochs):

Z1, A1, Z2, A2 = forward\_propagation(X, W1, b1, W2, b2)

loss = compute\_cost(A2, y)

dW1, db1, dW2, db2 = backward\_propagation(X, y, Z1, A1, A2, W2)

update\_parameters(W1, b1, W2, b2, dW1, db1, dW2, db2, learning\_rate)

**return** W1, b1, W2, b2

step 10 Evaluate the model: After training, evaluate the model on the test set and print the test accuracy.[¶](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#step-10-Evaluate-the-model:-After-training,-evaluate-the-model-on-the-test-set-and-print-the-test-accuracy.)

In this task, you will implement a function to make predictions using a trained neural network model. The prediction function will use the network's weights and biases to compute the output for given input data and determine whether each prediction is a positive or negative class based on a threshold.Function Signature: def predict(X, W1, b1, W2, b2): # Your code here return predictions Parameters: X: A NumPy array of shape (n\_samples, input\_size) representing the input features for which predictions need to be made. W1: A NumPy array of shape (input\_size, hidden\_size) representing the trained weights of the hidden layer. b1: A NumPy array of shape (1, hidden\_size) representing the trained biases of the hidden layer. W2: A NumPy array of shape (hidden\_size, output\_size) representing the trained weights of the output layer. b2: A NumPy array of shape (1, output\_size) representing the trained biases of the output layer. Expected Output: predictions: A NumPy array of shape (n\_samples, 1) containing boolean values. Each element is True if the corresponding prediction probability is greater than 0.5, and False otherwise.

In [ ]:

Student's answer[(Top)](http://10.11.51.204:8000/user/23bd1a054t/files/ps2/Churn_Modelling-4/feedback/2025-01-22%2006%3A41%3A12.519232%20UTC/Churn_Modelling-4.html?_xsrf=MnwxOjB8MTA6MTczNzk1MTIxN3w1Ol94c3JmfDEzMjpPRFJsWXpobU9Ua3haRGxtTkRaa1pqbG1OamhoWkdJNFlqRXpNamRsT0RFNk56UTJZemhoTVRZM1pEbGpPR1ZpWlRVME9EZGtZamMwT1dZNU5qSmpOR0V3TkRObE56SmhNV1ZtWkRGbE1EQmlZems0TUdSak1qWTVZekUxTjJWbE1RPT18MDE0YjMxNjQ1NDMyNTAwZDgzNDdlODhhN2NiYzVmYWRiMjE5OWUyN2E4NWVjZjQwNWYxN2JkNDhkZDc2MjU1Yg#top)

*# YOUR CODE HERE*

**def** predict(X, W1, b1, W2, b2):

Z1, A1, Z2, A2 = forward\_propagation(X, W1, b1, W2, b2)

predictions = A2 > 0.5

**return** predictions

In [ ]:

**def** accuracy(y\_true, y\_pred):

**return** np.mean(y\_true == y\_pred)