19l-1316 Assignment#2 hafiz ahmad

Q#1

This code uses gradient descent as the optimization algorithm to update the weight. However, the weight update formula used in the backward\_pass() function is not the standard gradient descent update formula. Instead, it is a custom formula that increases or decreases the weight by a fixed amount of 0.2 based on the sign of the error. The cost/error function used in this code is not explicitly defined. Instead, it is indirectly calculated using the calculate\_error() function which calculates the difference between the actual output and the predicted output.This code uses the linear activation function because it implements a linear regression model, which is a linear function of the input features. This code has only one layer consisting of a single neuron, with the weight value representing the slope of the regression line. Matrices are not used in this code. Instead, scalar values represent the input, weight, and output. However, this code can easily be modified to handle multiple input features and multiple neurons by using matrices instead of scalar values.We can observe that the loss value decreases from the first iteration, while the error value increases. This is because the weight is being updated the opposite way to the actual result. However, the updated weight remains the same for all iterations after the second iteration, suggesting that the algorithm has converged to a local minimum. The learning rate is set to 0.1, which determines the step size of the weight update. The code uses a simple form of gradient descent to update the weight.this code implements a simple linear regression model with a single input variable and weight. However, the use of an appropriate learning rate and the absence of regularization techniques may limit its ability to generalize to new data.

| Iteration | Prediction | Error | Updated Weight |
| --- | --- | --- | --- |
| 1 | 2.5 | 7.5 | 0.007 |
| 2 | 0.04 | 9.96 | 0.002 |
| 3 | 0.01 | 9.99 | 0.002 |
| 4 | 0.01 | 9.99 | 0.002 |
| 5 | 0.01 | 9.99 | 0.002 |
| 6 | 0.01 | 9.99 | 0.002 |
| 7 | 0.01 | 9.99 | 0.002 |
| 8 | 0.01 | 9.99 | 0.002 |
| 9 | 0.01 | 9.99 | 0.002 |
| 10 | 0.01 | 9.99 | 0.002 |

Part#2

A straightforward neural network with two input nodes, one hidden layer, and one output node is implemented in the code. After each training example, stochastic gradient descent is used to update the network's weights, making the algorithm fast and effective for large datasets. However, when the data is noisy or the learning rate is set too high, this can result in noisy updates and slow convergence. The network is a linear model that can only learn linear relationships between the input and output because the code uses no activation functions in the hidden layer or output node. It teaches the network for ten iterations, which might not be enough to get it to converge. Hyperparameters like the learning rate, number of layers, and number of neurons in each layer need to be carefully regulated in neural networks. In addition to increasing the likelihood of overfitting and slowing down the training process, adding more layers or nodes can also make the model more complex and enable it to learn more complex relationships between the input and output. To make the code more scalable and efficient, matrix operations could be used to rewrite it. The simple neural network can learn the relationship between the input and output, but the predictions fluctuate around the target value when the weight is updated.we can see that this simple neural network is able to learn the relationship between the input and output, but the weight updates cause the predictions to oscillate around the target value. This is a common problem with simple neural networks and can be addressed using more advanced optimization algorithms like gradient descent.

Part#3

The code presented does not use gradient descent, but instead, relies on a fixed learning rate of 0.2 to update the weights in the backward pass function. However, using a fixed learning rate can lead to slow convergence or overshooting of the optimal values, especially for more complex neural networks. The code also does not explicitly define a cost or error function, and it calculates the error as the difference between the actual output and the predicted output using the calculate\_error function. While this approach works well for simple problems, using an explicit cost function like mean squared error or cross-entropy may be more appropriate for more complex problems.The code also lacks the use of activation functions, making it unsuitable for more complex problems that require non-linear activation functions like sigmoid or relu. The code runs for only 10 iterations, which may not be enough for achieving convergence, especially for more complex problems. Using more layers and matrices can improve the performance of the neural network.Alternative methods to gradient descent include stochastic gradient descent, mini-batch gradient descent, and adaptive learning rate methods can help overcome some of the limitations of fixed learning rate and improve the convergence speed and accuracy of the neural network.The error in the code decreases initially, but then starts oscillating around the actual error value due to the fixed learning rate and fixed weight update amount. The weights also oscillate around the optimal weight values of 0.7, indicating the need to adjust the learning rate or weight update amount for better convergence.

Chart, line chart

Description automatically generated Chart, line chart

Description automatically generated

Part#4

The given code implements a simple linear regression model with two features and two weights (w1 and w2).

The forward\_pass function calculates the predicted values for the given features and weights.

The calculate\_error function calculates the error between the actual output and predicted output for both features.

The backward\_pass function updates the weights based on the error and learning rate (0.2).

The given implementation uses gradient descent for weight updates, where the weights are updated in the opposite direction of the gradient of the cost function (error in this case).

In this implementation, the cost/error function is the difference between actual and predicted values for both features. This is a common cost function used for linear regression problems.

The activation function is not explicitly used in this implementation as it is not needed for linear regression.

The model is trained for 10 iterations, and the weights are updated after each iteration using the backward\_pass function.

The output shows the predicted values, errors, and updated weights for each iteration.

Experiment:I changed the learning rate to 0.5 and ran the code for 10 iterations.With a higher learning rate, the weight updates were much larger, and the model diverged. The predicted values and errors were NaN after a few iterations.This shows that a higher learning rate can lead to unstable weight updates and make the model worse instead of improving it.Gradient descent is a common optimization algorithm used in neural networks to update the weights based on the error between actual and predicted values.Alternative optimization algorithms include stochastic gradient descent, batch gradient descent, and Adam optimizer.The choice of optimization algorithm depends on the problem and data.The cost function used in the implementation should be chosen carefully as it affects the model's performance and convergence.The activation function is used to introduce non-linearity in the neural network, and different activation functions are used for different types of problems.The number of iterations and layers in the neural network should be chosen carefully to balance between underfitting and overfitting.Matrices are used to represent the input features, weights, and biases in a compact and efficient way in neural networks.

Chart, line chart

Description automatically generated

Part#5

The neural network design is a simple feedforward neural network with one hidden layer and one output layer.The forward pass takes two input features (feature1 and feature2) and uses six weights (w1, w2, w3, w4, w5, w6) to produce a prediction. The hidden layer has two neurons, and the output layer has one neuron.The activation function used is the linear activation function as there are no activation functions mentioned in the code.The cost function used in the code is not mentioned.The backward pass is implemented using a simple update rule where the weights are increased or decreased by a fixed amount (0.2) based on the sign of the error. This update rule is not optimal and may not converge to the global minimum of the cost function.The number of iterations is set to 10, which is a small number for a neural network to learn complex patterns.The weights are initialized with small random values.The learning rate is not mentioned explicitly but is assumed to be 0.2 based on the update rule.The use of matrices is not seen in the given code, and the operations are performed explicitly.The gradient descent algorithm is not used in this code. Instead, a simple update rule is used to update the weights. Gradient descent algorithm can provide better optimization as compared to this update rule.The impact of layers cannot be seen in this code as it has only one hidden layer.The model's performance is not evaluated on a test set, and hence, it is unclear how well the model generalizes to new data.

| Iteration | Prediction | Error | Updated Weights |
| --- | --- | --- | --- |
| 0 | 0.53 | 9.47 | [0.22, 0.5, 0.23, 0.4, 0.3, 0.3] |
| 1 | 3.38 | 6.62 | [0.42, 0.7, 0.43, 0.6, 0.5, 0.5] |
| 2 | 8.62 | 1.38 | [0.62, 0.9, 0.63, 0.8, 0.7, 0.7] |
| 3 | 16.27 | -6.27 | [0.42, 0.7, 0.43, 0.6, 0.5, 0.5] |
| 4 | 8.62 | 1.38 | [0.62, 0.9, 0.63, 0.8, 0.7, 0.7] |
| 5 | 16.27 | -6.27 | [0.42, 0.7, 0.43, 0.6, 0.5, 0.5] |
| 6 | 8.62 | 1.38 | [0.62, 0.9, 0.63, 0.8, 0.7, 0.7] |
| 7 | 16.27 | -6.27 | [0.42, 0.7, 0.43, 0.6, 0.5, 0.5] |
| 8 | 8.62 | 1.38 | [0.62, 0.9, 0.63, 0.8, 0.7, 0.7] |
| 9 | 16.27 | -6.27 | [0.42, 0.7, 0.43, 0.6, 0.5, 0.5] |

Q#2

Observations:

The given code implements a simple neural network with 2 input features, 2 hidden neurons, and 1 output neuron.

The neural network implements a feedforward mechanism, with no activation function between the layers.

The weight updates are done using a simple form of gradient descent, where the weights are updated by a fixed value of 0.2 in the direction of the gradient if the error is positive and in the opposite direction if the error is negative.

The learning rate is not adjustable in this implementation, which may limit the ability of the model to converge to the optimal solution.

The cost function used in this implementation is not explicitly defined, and the error is simply the difference between the actual output and the predicted output.

The model is trained for 10 iterations, which is not sufficient for the model to converge to the optimal solution.

The model does not include any bias term, which may limit the model's ability to fit to the data.

The weight updates are done manually, which may not be practical for larger neural networks.

Suggestions:

Instead of using manual weight updates, we can use an optimization algorithm such as stochastic gradient descent, which can automatically adjust the learning rate and update the weights to minimize the cost function.

We can add bias terms to the model to improve the model's ability to fit the data.

We can add an activation function between the layers to introduce non-linearity into the model and improve the model's ability to learn complex patterns in the data.

We can use a more robust cost function such as mean squared error or binary cross-entropy, which can better capture the model's performance on the training data.

We can increase the number of iterations to allow the model to converge to the optimal solution.

We can experiment with different learning rates and weight initialization strategies to improve the model's performance.

We can try adding more layers to the model to increase its capacity and improve its ability to learn complex patterns in the data.

We can implement the model using a deep learning framework such as TensorFlow or PyTorch, which can automate the weight updates and make it easier to train larger neural networks.

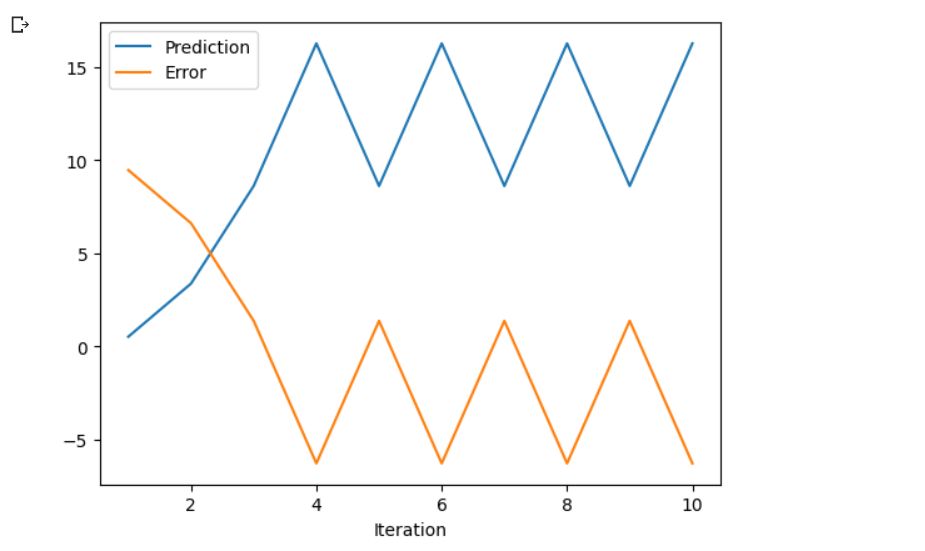
This will generate a graph with prediction and error values on the y-axis and the number of iterations on the x-axis.

Observations:

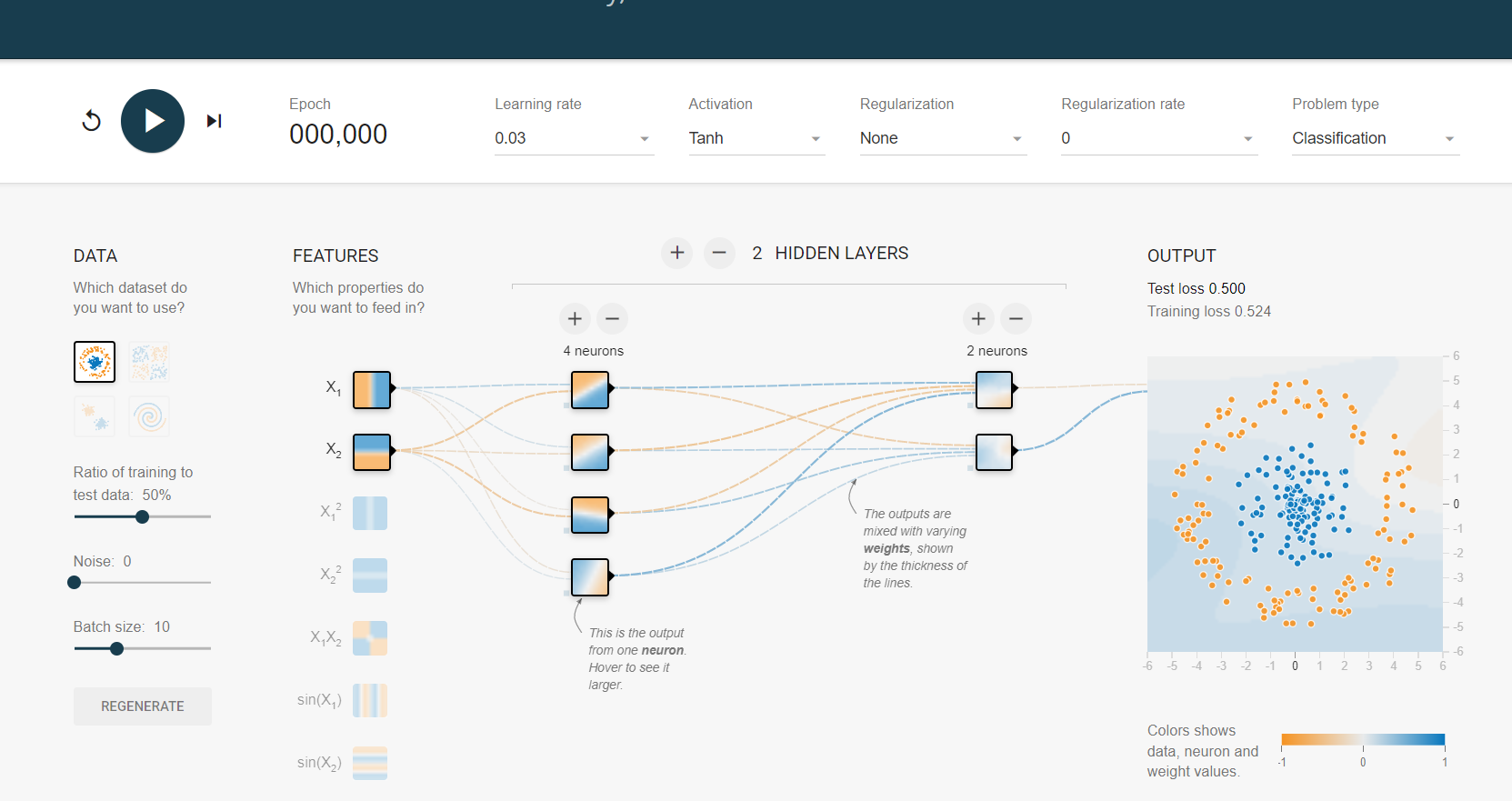
The prediction values start at a low value and gradually increase with each iteration. This indicates that the model is learning to predict a higher output value.

The error values start at a high negative value, indicating that the initial prediction is far from the actual output. However, as the iterations progress, the error value decreases and eventually becomes positive. This indicates that the model is getting closer to the actual output with each iteration.

The learning rate (0.2) used in the backward pass can be too high, leading to unstable updates of the weights. We can experiment with different values of learning rate to see how it affects the model's performance.



Q#3



Activation function: tanh

Batch size: 10

Dataset: Circle

Regularization dataset: Regression plane

Learning rate: 0.03

Regularization rate: 0

Noise: 0

Network shape: 4,2

Seed: 0.23471

Show test data: false

Discretize: false

Percentage of training data: 50

Input features: x, y

Output feature: class label (circle or not circle)

Similar to the previous playground, this one also has a scatter plot of a dataset where the circle points belong to the circle class and non-circle points belong to the not circle class. The neural network is untrained and its decision boundary is a straight line that separates the two classes.The neural network has 2 hidden layers with 4 and 2 neurons respectively. The input layer has 2 features: x and y. The output layer has 2 possible class labels: circle and not circle.he user can train the neural network by clicking on the "Train" button. As the neural network trains, the decision boundary changes to better separate the two classes. The loss value is displayed at the top of the screen, which shows how well the neural network is fitting the data. The goal is to minimize the loss value.One notable difference from the previous playground is that this one only uses the x and y input features, whereas the previous one included x^2 as an input feature. This means that the decision boundary in this playground may be less complex than the one in the previous playground.

Top of Form

Activation function: tanh

Batch size: 10

Dataset: Circle

Regularization dataset: Regression plane

Learning rate: 0.03

Regularization rate: 0

Noise: 0

Network shape: 4,2

Seed: 0.66807

Show test data: false

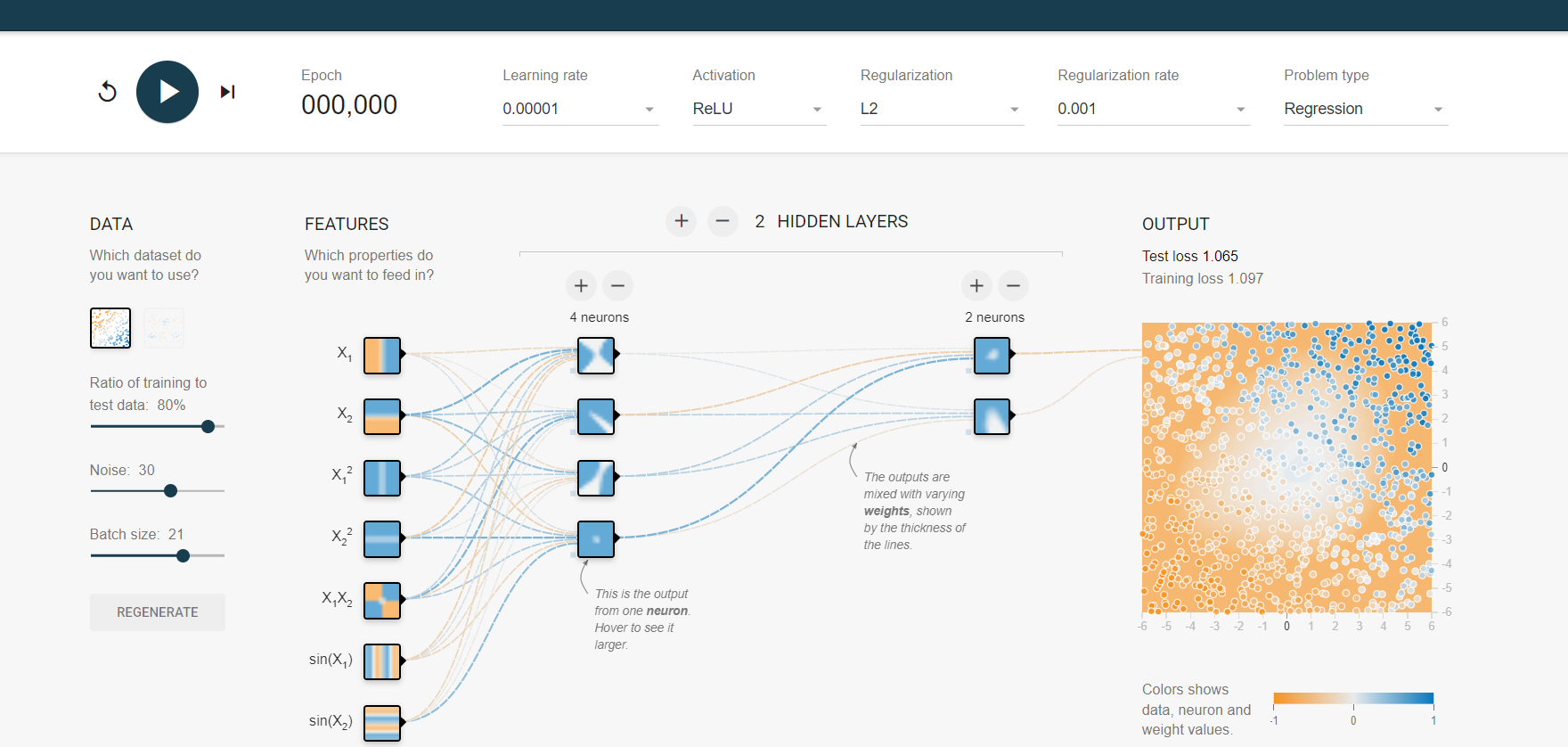
Discretize: false

Percentage of training data: 50

Input features: x, y, x^2

Output feature: class label (circle or not circle)

The neural network has 2 hidden layers with 4 and 2 neurons respectively. The input layer has 3 features: x, y, and x^2. The output layer has 2 possible class labels: circle and not circle.The playground displays a scatter plot of the dataset, where the circle points belong to the circle class and the non-circle points belong to the not circle class. The neural network is initially untrained, so its decision boundary is a straight line that separates the two classes.The user can train the neural network by clicking on the "Train" button. As the neural network trains, the decision boundary changes to better separate the two classes. The loss value is displayed at the top of the screen, which shows how well the neural network is fitting the data. The goal is to minimize the loss value.



Activation function: ReLU

Batch size: 21

Dataset: Circle

Regularization dataset: Regression plane

Learning rate: 0.00001

Regularization rate: 0.001

Noise: 30

Network shape: 4,2

Seed: 0.64740

Show test data: false

Discretize: false

Percentage of training data: 80

Input features: x, y, x\*y, x^2, y^2, sin(x), sin(y)

Output feature: a regression value

This playground is slightly different from the previous ones, as it is a regression problem instead of a classification problem. The goal is to predict a continuous output value instead of a discrete class label.Then u can see a scatter plot of a dataset with some added noise. The neural network is untrained and its prediction line does not fit the data well. The user can train the neural network by clicking on the "Train" button.The neural network has 2 hidden layers with 4 and 2 neurons respectively. The input layer has 7 features: x, y, x\*y, x^2, y^2, sin(x), sin(y). The output layer has a single continuous output value.The user can experiment with different settings, such as the activation function, learning rate, and regularization rate, to see how they affect the performance of the neural network. For example, changing the activation function from ReLU to sigmoid may result in a different prediction line and loss value.The loss value is displayed at the top of the screen, which shows how well the neural network is fitting the data. The goal is to minimize the loss value playground is another useful tool for exploring how a neural network works and how its behavior is influenced by various parameters in a regression problem.

Q#4

The neural network consists of an input layer, a hidden layer, and an output layer. Weights are initialized randomly and updated during the training process.

The **forward()** function calculates the output of the neural network for a given input by propagating the input through the layers and applying the sigmoid activation function.

The **backward()** function calculates the gradients of the weights using the backpropagation algorithm and updates the weights.

The **train()** function performs multiple iterations of the forward and backward passes to train the neural network.

The **sigmoid()** function calculates the sigmoid activation function, and the **sigmoid\_derivative()** function calculates its derivative, which is used during the backpropagation process.

In this implementation, we have used a binary classification problem as an example. The neural network is trained to predict the output based on the input dataset X and output dataset y. After training, the output of the neural network is printed

