Project 1

COSC 525: Deep Learning

Submitted by Yu Jiang and Harshvardhan

```
# import libraries
import numpy as np
```

Question 1

This code defines a class for a single neuron in a neural network. The `Neuron` class has methods for activation, weight calculation, weight update, activation derivative calculation, and partial derivative calculation. The activation function can either be a linear function or a logistic (sigmoid) function. The weights and bias can be initialized randomly or given as input.

The `calculate` method takes an input vector and returns a scalar, which is the output of the neuron. The `update_weight` method updates the weights using the partial derivatives and the learning rate. The `calc_partial_derivative` method calculates the partial derivatives of the weights and bias with respect to the cost function and updates the weights. The output is a vector which can be used as the input to the same method in the next layer.

```
class Neuron:
    # initilize neuron
    def __init__(self, activation, input_num, lr, weights=None):
        self.activation = activation
        self.input_num = input_num
        self.lr = lr

        # if weights is not given, randomly initialize weights
        if weights is None:
            self.weights = np.random.rand(input_num)
            self.bias = np.random.rand()

        else:
            self.weights = weights[:-1]
            self.bias = weights[-1]

        # Other variables
        self.output = None
```

```
        self.input = None
        self.delta = None


    # This method prints the weights of the neuron
    def print_weight(self, layer_idx, neuron_idx):
        print(f"layer {layer_idx}, neuron {neuron_idx}, {self.input = }")
        print(f"layer {layer_idx}, neuron {neuron_idx}, {self.weights =}, {self.bias = }")

    # This method returns the activation function of the neuron
    def activate(self, net):
        if self.activation == 'linear':  # linear activation function
            output = net
        elif self.activation == 'logistic':  # logistic (sigmoid activation function)
            output = 1 / (1 + np.exp(-1 * net))
        else:
           print("Activation function not defined")
        return output

    # Calculate the output of the neuron given the input.
    # The input is a vector of size input_num
    # The output is a scalar
    def calculate(self, input):
        self.input = input
        net = np.dot(input, self.weights) + self.bias
        self.output = self.activate(net)
        return self.output

    # Update the weights using the partial derivatives and the weights
    def update_weight(self):
        self.weights = self.weights - self.lr * self.cal_der_w
        self.bias = self.bias - self.lr * self.cal_der_b

    # This method returns the derivative of the activation function with respect to the ne
    # The input is a scalar
    # The output is a scalar
    def activation_derivative(self):
        if self.activation == 'logistic':  # logistic has x(1-x) derivative
            act_deriv = self.output * (1 - self.output)
        elif self.activation == 'linear':  # linear activation has derivative of 1
            act_deriv = 1
        else:
```

```
            print("Activation function not defined")
        return act_deriv

    # This method calculates the partial derivatives of the weights and bias
    # The input wtimesdelta is a matrix
    # The output is a vector of size input_num
    def calc_partial_derivative(self, wtimesdelta):
        delta = wtimesdelta * self.activation_derivative()

        self.cal_der_w = delta * self.input  # Each node has a partial derivative for each
        self.cal_der_b = delta * 1 # Bias has a no partial derivative

        # update wtimesdelta
        wtimesdelta = delta * self.weights

        # update weights
        self.update_weight()
        return wtimesdelta  # output a vetctor
```

Question 2

This code defines a `FullyConnected` class in Python that represents a fully connected layer in a neural network. The class takes the number of neurons in the layer, the activation function, the number of inputs, the learning rate, and the weights as input to its constructor. If weights are not provided, a list of `Neuron` objects is created with random weights and biases.

The `print_weight` method prints the weights of each neuron in the layer, the `calculate` method calculates the outputs of all the neurons given an input, and the `calcwdeltas` method calculates the partial derivatives of the weights and biases of all the neurons in the layer. These partial derivatives are used to update the weights during backpropagation.

```
class FullyConnected:
    # Initialize the fully connected layer
    def __init__(self, numOfNeurons, activation, input_num, lr, weights=None):
        self.numOfNeurons = numOfNeurons
        self.neurons = []
        self.lr = lr
        # self.neurons = [Neuron (activation, input_num, lr, weights[i]) for i in range (n
        if weights is None:
            for i in range(numOfNeurons):
                self.neurons.append(Neuron(activation, input_num, lr))
```

```
        else:
            for i in range(numOfNeurons):
                self.neurons.append(Neuron(activation, input_num, lr, weights[i]))

    # This method prints the weights of all the neurons in the layer
    def print_weight(self, layer_idx):
        for neuron_index in range(self.numOfNeurons):
            self.neurons[neuron_index].print_weight(layer_idx, neuron_index)

    # This method calculates the output of all the neurons in the layer and return a vecto
    def calculate(self, input):  # input:(2,)
        self.output_vector = []  # list
        for i in range(self.numOfNeurons):
            self.output_vector.append(self.neurons[i].calculate(input))
        return np.array(self.output_vector)  # array

    # This method calculates the partial derivatives of the weights and bias of all the ne
    # The input wtimesdelta is a matrix
    # The output is a vector of size input_num
    def calcwdeltas(self, wtimesdelta):
        lst = []

        for i in range(self.numOfNeurons):
            lst.append(self.neurons[i].calc_partial_derivative(wtimesdelta[i]))
        lst = np.array(lst)

        wtimesdelta = np.sum(lst, axis=0)
        return wtimesdelta
```

Question 3

The NeuralNetwork class is a blueprint for creating a neural network model. It provides a structure for defining the architecture of the model, such as the number of layers and the number of neurons in each layer. It also provides methods for training the model, making predictions, and evaluating its performance.

The `__init__` method initializes the network by taking in the following parameters:

- `numOfLayers`: number of hidden layers in the network
- `numOfNeurons`: a vector that contains the number of neurons in each layer
- `inputSize`: the size of the input data
- `activation`: the activation function to be used in each layer

- `loss`: the loss function to be used to evaluate the performance of the network
- `lr`: the learning rate
- `weights`: the initial weights to be used in each layer (optional)
- `print`: a flag to determine whether to print debug information (optional)

It then creates an array of fully connected layers using the information provided in the parameters. The `FullyConnected` class is implemented previously in the code, and it represents a single fully connected layer in the network.

The `calculate` method calculates the output of the network given the input. It starts by initializing the output to zeros, then it loops over each input sample, calculating the output for each sample by passing it through each layer in the network. If the activation function is `linear` and the loss function is `bce`, it adds a softmax layer to the output to convert the output into probabilities.

The `calculateloss` method calculates the loss of the network given the predicted output and the ground truth output. It implements two loss functions: sum of square errors (SSE) and binary cross entropy (BCE).

The `lossderiv` method calculates the derivative of the loss function with respect to the output of the network. It implements the derivative for the two loss functions implemented in `calculateloss`.

The `train` method trains the network by performing one step of backpropagation. It does this by running a forward pass to get the output, then computing the loss derivative with respect to the output. It then calls `calcwdeltas` for each layer with the appropriate values to update the weights of each layer.

```
class NeuralNetwork:
    # Initialise the neural network with the number of layers, number of neurons in each l
    # activation function (for each layer), loss function and the learning rate
    def __init__(self, numOfLayers, numOfNeurons, inputSize, activation, loss, lr, weights
        self.numOfLayers = numOfLayers
        self.loss = loss
        self.activation = activation
        self.print = print
        self.layers = []
        self.lr = lr

        # Add output layer
        for i in range(self.numOfLayers + 1):
            if i == 0:
                self.layers.append(
                    FullyConnected(numOfNeurons[i], activation, inputSize, lr, None if wei
```

```
            else:
                self.layers.append(FullyConnected(numOfNeurons[i], activation, numOfNeuron
                                                  None if weights is None else weights[i])

    # Calculate output of the neural network given the input
    def calculate(self, input, y):
        self.output = np.zeros(y.shape)

        for j in range(input.shape[0]):
            tmp_output = input[j]
            for i in range(self.numOfLayers + 1):
                tmp_output = self.layers[i].calculate(tmp_output)

            s = tmp_output.shape[0]

            self.output[j] = tmp_output.reshape(s)
            if self.activation == 'linear' and self.loss == 'bce':  # adds a softmax layer
                for k in range(y.shape[1]):
                    self.output[j][k] = np.exp(self.output[j][k]) / np.sum(np.exp(self.out
        return np.array(self.output)

    # Calculate the loss of the neural network given the predicted output and the ground t
    def calculateloss(self, yp, y):
        error = 0
        for i in range(yp.shape[0]):
            if self.loss == 'sse':  # sum of square errors
                error += 1 / 2 * np.sum((yp[i] - y[i]) ** 2)
            if self.loss == 'bce':  # binary cross entropy
                error += np.sum(-y[i] * np.log(yp[i]) - (1 - y[i]) * np.log(1 - yp[i]))

        return error / yp.shape[0]

    # Calculate derivative of loss function given the predicted output and the ground trut
    def lossderiv(self, yp, y):

        if self.loss == 0:
            error_der = 0
            for i in range(yp.shape[0]):
                error_der += (yp[i] - y[i])

        else:
            if self.activation == 'linear':
```

```
                error_der = 0
                for i in range(yp.shape[0]):
                    error_der += ((yp[i] - y[i]) / yp[i] / (1 - yp[i])) * yp[i]
            elif self.activation == 'logistic':
                error_der = 0
                for i in range(yp.shape[0]):
                    error_der += ((yp[i] - y[i]) / yp[i] / (1 - yp[i]))
            else:
                print('Error: loss derivative not implemented for this activation function


        return (1 / yp.shape[0]) * error_der

    # Perform one step of backpropagation by doing the following:
    # - Run a forward pass with the input and get the output
    # - Compute the loss derivative with respect to the output
    # - Call calcwdeltas for each layer with the appropriate values
    def train(self, x, y, num_epochs=10000):
        error = []
        for epoch_index in range(num_epochs):
            output_last = self.calculate(x, y)  # x.shape = (4, 2), output_last.shape = (4
            yp = output_last
            if error == []:
                past_loss = 1e+10
            else:
                past_loss = current_loss

            current_loss = self.calculateloss(yp, y)

            # run till convergence
            if np.abs(current_loss - past_loss) <= 1e-8:
                break

            error.append(current_loss)  # save the error value of each epoch

            error_der = self.lossderiv(yp, y)
            wtimesdelta = error_der  # Output layer to the last hidden layer
            for layer_index in range(self.numOfLayers, -1, -1):  # Layers for L[0] to L[nu
                wtimesdelta = self.layers[layer_index].calcwdeltas(wtimesdelta)

            if self.print == True:
```

```
                for layer_index in range(self.numOfLayers, -1, -1):  # Layers for L[0] to
                    print('layer {} weights: '.format(layer_index))
                    self.layers[layer_index].print_weight(layer_index)

        # if classification problem, return the predicted class
        if len(np.unique(y)) > 2:
            yp = np.zeros(yp.shape)
            for i in range(yp.shape[0]):
                yp[i][np.argmax(output_last[i])] = 1

        return error[0], error[-1], yp
```

Testing Code

Example Problem

```
w = np.array([[[0.15,0.25, 0.35],[0.2,0.3, 0.35]],
              [[0.4,0.5, 0.6],[0.45,0.55, 0.6]]])
x = np.array([[0.05, 0.1]])
y = np.array([[0.01, 0.99]])

numberOfNeurons = [2, y.shape[1]]
test1 = NeuralNetwork(1, numberOfNeurons, np.size(x), activation='linear', loss='sse', lr=
init_error, final_error, yp = test1.train(x, y, 1) # x, y, num_epochs
```

```
layer 1 weights:
layer 1, neuron 0, self.input = array([0.3825, 0.39  ])
layer 1, neuron 0, self.weights =array([-0.28997115, -0.2035    ]), self.bias = -1.203846153
layer 1, neuron 1, self.input = array([0.3825, 0.39  ])
layer 1, neuron 1, self.weights =array([0.45965187, 0.55984112]), self.bias = 0.625233644859
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.11449083, 0.17898167]), self.bias = -0.360183321351
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([0.15559777, 0.21119554]), self.bias = -0.538044572250
```

```
print(f"{init_error=}, {final_error=}")
```

```
init_error=0.4399276953124999, final_error=0.4399276953124999
```

AND gate

```
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [0], [0], [1]])
numberOfNeurons = [y.shape[1]]

test1 = NeuralNetwork(0, numberOfNeurons, x.shape[1], activation='logistic', loss='bce', l

init_error, final_error, yp = test1.train(x, y)
```

```
print(f"{init_error=}, {final_error=}")
print('final prediction = ', yp)
```

```
init_error=1.2111744279678198, final_error=0.7503997456855211
final prediction =  [[0.46955296]
 [0.2927952 ]
 [0.35681819]
 [0.20601609]]
```

XOR gates

```
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
# w = np.array([np.array([[-19., -11., 9.], [21., 22., -29.]]), np.array([[-19, -21., 10.]

numberOfNeurons = [y.shape[1]]
test2 = NeuralNetwork(0, numberOfNeurons, x.shape[1], activation="linear", loss="sse", lr=
init_error, final_error, yp = test2.train(x, y, 1000)
print('XOR using single perceptron:')
print(f"{init_error=}, {final_error=}")
print('final prediction = ', yp)

# ---xor-hidden layer---
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
# w = np.array([np.array([[-20., -20., 10.], [20., 20., -30.]]), np.array([[-20,-20., 10.]


numberOfNeurons = [2, y.shape[1]]
```

```
test2 = NeuralNetwork(1, numberOfNeurons, x.shape[1], activation="linear", loss="sse", lr=
init_error, final_error, yp = test2.train(x, y, 1000)
print('XOR using 1 hidden layer:')
print(f"{init_error=}, {final_error=}")
print('final prediction = ', yp)
```

```
XOR using single perceptron:
init_error=0.35528545215242274, final_error=0.6494158489984959
final prediction =  [[0.79919709]
 [1.10640715]
 [1.7048664 ]
 [2.01207646]]
XOR using 1 hidden layer:
init_error=0.9367545005557036, final_error=0.5896296686667823
final prediction =  [[0.80362383]
 [1.09183607]
 [1.62731961]
 [1.91553185]]
```

Main function for running the code

```
def execute_neural_net(learning_rate, problem_type, w = None):
    ## function to run the codes

    ## example problem
    if problem_type == 'example':

        x = np.array([[0.05, 0.1]])
        y = np.array([[0.01, 0.99]])

        # if weights are not provided, initialize them randomly
        if w is None:
            w = np.random.normal(0, 1, size=(2, 2, 3))

        numberOfNeurons = [y.shape[1]]

        example_nn = NeuralNetwork(0, numberOfNeurons, np.size(x), activation='linear', lo
                                   lr=learning_rate, weights=w, print=True)

        init_error, final_error, yp = example_nn.train(x, y, num_epochs=1)
```

```
    print(f"{init_error=}, {final_error=}")

    return yp, init_error, final_error

## Perceptron with AND gate
if problem_type == 'and':
    x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    y = np.array([[0], [0], [0], [1]])

    numberOfNeurons = [y.shape[1]]

    and_nn = NeuralNetwork(0, numberOfNeurons, x.shape[1], activation='logistic', loss
                           lr=learning_rate, weights = w)
    init_error, final_error, yp = and_nn.train(x, y)

    print(f"{init_error=}, {final_error=}")

    return yp, init_error, final_error


## xor problem with single perceptron
if problem_type == 'xor1':
    x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    y = np.array([[0], [1], [1], [0]])

    # with no hidden layer
    numberOfNeurons = [y.shape[1]]
    xor_nn = NeuralNetwork(0, numberOfNeurons, x.shape[1], activation="linear", loss="
                           lr=learning_rate, weights=w)
    init_error, final_error, yp = xor_nn.train(x, y)
    print('XOR using single perceptron:')
    print(f"{init_error=}, {final_error=}")

    return yp, init_error, final_error

if problem_type == 'xor2':
    x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    y = np.array([[0], [1], [1], [0]])

    # with hidden layer
    numberOfNeurons = [y.shape[1]]
```

```
        xor_nn = NeuralNetwork(0, numberOfNeurons, x.shape[1], activation="linear", loss="
                               lr=learning_rate)
        init_error, final_error, yp = xor_nn.train(x, y)
        print('XOR with hidden layer:')
        print(f"{init_error=}, {final_error=}")

        return yp, init_error, final_error
```

```
execute_neural_net(0.01, 'example')
```

```
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-2.77731793,  0.22122275]), self.bias = -0.490080321
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.62182487, -0.69689984]), self.bias = -0.830288607
init_error=2.0573062848474577, final_error=2.0573062848474577

(array([[-0.61072582, -0.94114267]]), 2.0573062848474577, 2.0573062848474577)
```

```
execute_neural_net(0.0001, 'and')
```

```
init_error=0.9513177182839784, final_error=0.7755452649738896

(array([[0.57613121],
        [0.49329109],
        [0.572581  ],
        [0.48966164]]),
 0.9513177182839784,
 0.7755452649738896)
```

```
execute_neural_net(0.0001, 'xor1')
```

```
XOR using single perceptron:
init_error=0.27220760217067985, final_error=1.543223937101207
```

```
(array([[0.7444063 ],
        [2.08369918],
        [1.81543558],
        [3.15472846]]),
 0.27220760217067985,
 1.543223937101207)
```

```
execute_neural_net(0.0001, 'xor2')
```

```
XOR with hidden layer:
init_error=0.13474508761850362, final_error=0.13042970799406767
```

```
(array([[0.38110822],
        [0.37914246],
        [0.58669709],
        [0.58473133]]),
 0.13474508761850362,
 0.13042970799406767)
```

Function for plotting loss vs learning rate

```
import matplotlib.pyplot as plt

def plot_loss_by_learning_rate(learning_rates, final_errors, title):
    learning_rates = np.log(learning_rates)
    plt.plot(learning_rates, final_errors, label='Final Error')
    plt.xlabel('log(Learning Rate)')
    plt.ylabel('Loss')
    plt.title(title)
    plt.legend()
    plt.savefig(f'{title}.png', dpi=600)
    plt.show()
```

Example problem

```
# Define a range of learning rates to test
learning_rates = np.logspace(-4, 1, 100)
final_errors = []

# Call the execute_neural_net function for each learning rate
for lr in learning_rates:
    yp, init_error, final_error = execute_neural_net(lr, 'example')
    final_errors.append(final_error)

# Plot the loss vs learning rate
plot_loss_by_learning_rate(learning_rates, final_errors, title='Loss vs Learning Rate for
```

```
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.81708908, -0.84065071]), self.bias = 0.0785562090
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 1.27556784, -0.03790521]), self.bias = 0.1978406976
init_error=0.26969936022150726, final_error=0.26969936022150726
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-1.87869155,  1.23195888]), self.bias = 0.7565579312
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.67808394, -0.59532174]), self.bias = 1.2764949929
init_error=0.3198809112625458, final_error=0.3198809112625458
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.02034098, -1.51543115]), self.bias = 0.5088230529
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.65905806, -2.11632251]), self.bias = 0.6382461152
init_error=0.2007190310760012, final_error=0.2007190310760012
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([1.64793647, 0.51669222]), self.bias = 0.3443140707092
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.32966097, -0.58055183]), self.bias = 0.2728686569
init_error=0.4232329374014492, final_error=0.4232329374014492
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.83064713,  0.2021928 ]), self.bias = -0.738004069
```

```
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.77508074, -0.62961989]), self.bias = 0.6519419794
init_error=0.39274861865268634, final_error=0.39274861865268634
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.36534902, 0.33102976]), self.bias = -1.163115175465
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.5865195 , -0.80862419]), self.bias = 1.018440640980
init_error=0.6326181157391496, final_error=0.6326181157391496
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.27395788, -0.4103372 ]), self.bias = 0.047598216780
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([0.70006934, 1.19269492]), self.bias = 0.562302916672
init_error=0.03758089387567312, final_error=0.03758089387567312
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 1.75740367, -0.13864473]), self.bias = -1.118085272
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-1.12925497, -0.04489657]), self.bias = 0.847482577390
init_error=0.5764104226226207, final_error=0.5764104226226207
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 0.82776356, -0.41101295]), self.bias = -0.561032111
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([1.09674819, 1.50974763]), self.bias = 0.709574048707
init_error=0.16572917619262215, final_error=0.16572917619262215
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([1.21077924, 0.09635597]), self.bias = 0.683770462356
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-1.22308997, -0.03013577]), self.bias = 0.155989746
init_error=0.6809975326874438, final_error=0.6809975326874438
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.00479696,  1.25522745]), self.bias = 0.835941604
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.01666621,  0.82618242]), self.bias = 0.914560676
init_error=0.46333254235019683, final_error=0.46333254235019683
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.29698676, -1.23941572]), self.bias = -0.81334051
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
```

```
layer 0, neuron 1, self.weights =array([-0.34791456, -0.22506204]), self.bias = -1.001099384
init_error=2.526239665448188, final_error=2.526239665448188
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 1.27873737, -0.03074508]), self.bias = 0.377920588
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 1.24822934, -0.19355817]), self.bias = -0.12705888
init_error=0.6692380407019599, final_error=0.6692380407019599
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.12696645,  0.00278183]), self.bias = -0.601235759
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.28928216, -0.5347569 ]), self.bias = 0.746364786
init_error=0.23071223467684254, final_error=0.23071223467684254
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.15342945, -0.92302778]), self.bias = 1.355515367
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([0.40582772, 1.52964599]), self.bias = 0.695688083370
init_error=0.7799184204290028, final_error=0.7799184204290028
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.98936561,  0.74308473]), self.bias = 0.459955565
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.78917565, -0.27038664]), self.bias = -0.424252929
init_error=1.2101328961515427, final_error=1.2101328961515427
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.71965907, 1.42523607]), self.bias = 1.397334729328
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 2.65896852, -1.00566936]), self.bias = 2.278789250
init_error=2.095034271732783, final_error=2.095034271732783
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-1.07767939, -2.56726325]), self.bias = -0.000423645
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 1.11570367, -1.29491455]), self.bias = -0.377456165
init_error=1.091110145370828, final_error=1.091110145370828
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 0.67558697, -0.93108723]), self.bias = 0.632690255
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.39773077, -2.52669969]), self.bias = 0.302835911
```

```
init_error=0.620614232185837, final_error=0.620614232185837
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 1.0010411 , -0.32897091]), self.bias = 1.1877125876
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([0.96821097, 0.61005342]), self.bias = -0.9751033188
init_error=2.430784249534372, final_error=2.430784249534372
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.58924245,  0.11763318]), self.bias = 0.033273250
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.99351991, -0.3954581 ]), self.bias = -1.13497097
init_error=2.4536252405596395, final_error=2.4536252405596395
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 0.48368793, -0.38533886]), self.bias = 1.6669146309
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.24923142, -0.3238395 ]), self.bias = 0.869308550
init_error=1.3542278633269744, final_error=1.3542278633269744
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.52110133,  0.26923114]), self.bias = 1.805363158
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.45694007, -1.14752583]), self.bias = 0.011395030
init_error=2.1823680267297125, final_error=2.1823680267297125
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.94209378, 0.79241893]), self.bias = -1.3840428453
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.54467244, -0.30706033]), self.bias = -0.379229487
init_error=1.74873136821124, final_error=1.74873136821124
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 1.78814791, -0.76482183]), self.bias = 0.435932652
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.46876862, -0.54773199]), self.bias = 0.492569709
init_error=0.26349557580252214, final_error=0.26349557580252214
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.03372463, -1.94862468]), self.bias = -0.087484144
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.53234827,  0.88575292]), self.bias = -1.001675434
init_error=1.9088178600054624, final_error=1.9088178600054624
```

```
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.31961702, -0.18785022]), self.bias = -1.587350649
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.56881975,  1.65153221]), self.bias = -1.118616434
init_error=3.2823094445786607, final_error=3.2823094445786607
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.09118718, 1.00607807]), self.bias = -0.413637738989
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.10236921, -0.86078413]), self.bias = -2.232257257
init_error=5.514221094131069, final_error=5.514221094131069
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.00934513, 1.55511816]), self.bias = 1.204896605578
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.3411645 , -0.84180689]), self.bias = -0.292321997
init_error=1.8596508149280924, final_error=1.8596508149280924
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 0.22698027, -0.03965416]), self.bias = -0.178081213
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.16631831, -0.28154902]), self.bias = -0.667501714
init_error=1.428069998161952, final_error=1.428069998161952
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.21223853, 1.12835437]), self.bias = 1.058128699053
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([1.07856701, 1.02364674]), self.bias = 0.463505905601
init_error=0.7279978824663045, final_error=0.7279978824663045
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-1.16261505,  0.18054397]), self.bias = 2.306246933
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.29482409, -1.22610942]), self.bias = 0.311685788
init_error=0.8048932003227669, final_error=0.8048932003227669
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 0.75629379, -0.58523732]), self.bias = -0.499061467
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-1.20668091,  0.12616829]), self.bias = 2.655988711
init_error=1.4437011167461056, final_error=1.4437011167461056
layer 0 weights:
```

```
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.67507967,  0.96641275]), self.bias = 0.5658136288
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.23751583, -0.83591942]), self.bias = -2.058092896
init_error=5.077676828453254, final_error=5.077676828453254
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.95147921,  1.29489769]), self.bias = 1.2323615526
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.80507484, -0.47052871]), self.bias = -0.651470545
init_error=2.32395568357021, final_error=2.32395568357021
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 1.13540363, -0.20929707]), self.bias = -1.841567245
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.12116724, -1.24603723]), self.bias = -0.769645960
init_error=3.4532135446743615, final_error=3.4532135446743615
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.46400795, -0.39046969]), self.bias = -0.377257990
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.41819279, -0.34105792]), self.bias = 0.915212229
init_error=0.11121689831856625, final_error=0.11121689831856625
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 0.59255081, -1.28189478]), self.bias = 2.322839516
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.89661816, -0.56528757]), self.bias = -1.540127891
init_error=5.6708992509215985, final_error=5.6708992509215985
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([1.51773715, 0.75628253]), self.bias = 2.414245455457
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([0.43849749, 1.14022782]), self.bias = -1.046285978387
init_error=5.052431775439134, final_error=5.052431775439134
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 1.24058299, -0.04708934]), self.bias = 1.822091088
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 1.2169929 , -0.84521051]), self.bias = 0.650707348
init_error=1.7788130494910046, final_error=1.7788130494910046
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
```

```
layer 0, neuron 0, self.weights =array([-0.08637752,  1.05564772]), self.bias = 0.624874805
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-1.1966521 , -0.15464195]), self.bias = 2.689574796
init_error=1.582439420445037, final_error=1.582439420445037
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.97348345, -1.14602223]), self.bias = 0.078816177
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([0.35992842, 0.73265748]), self.bias = 0.351815071409
init_error=0.1605938103423652, final_error=0.1605938103423652
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-1.60126818, -1.36341167]), self.bias = 0.562463412
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([0.29747047, 0.40396649]), self.bias = 0.427799566960
init_error=0.19411171385981912, final_error=0.19411171385981912
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.48198684, 0.38943767]), self.bias = -1.816312307
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([0.2915324 , 0.83166229]), self.bias = -1.774333265364
init_error=5.167025838841275, final_error=5.167025838841275
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.4011437 , -2.28416352]), self.bias = -0.86217459
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.66577581, -1.8111331 ]), self.bias = -1.61257910
init_error=4.466841107998082, final_error=4.466841107998082
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.03205086,  0.49080598]), self.bias = -0.32456517
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.2421814 ,  0.88444634]), self.bias = 0.611550939
init_error=0.09382137633844453, final_error=0.09382137633844453
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.07578336, 1.91867567]), self.bias = 1.56619518304
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 1.16089992, -0.57338419]), self.bias = -0.21203858
init_error=2.193678154246444, final_error=2.193678154246444
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 1.32846581, -0.47473178]), self.bias = 0.153983114
```

```
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.20123447,  1.03648691]), self.bias = -1.307065027
init_error=2.4945400325362708, final_error=2.4945400325362708
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.72005535, -0.33034059]), self.bias = -0.024837966
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.69878939, -0.12158352]), self.bias = 0.6806746138
init_error=0.054530175777461554, final_error=0.054530175777461554
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-1.71057528, -0.04574918]), self.bias = 0.730472282
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([0.90081562, 0.27688374]), self.bias = 0.728563818395
init_error=0.27229349774514305, final_error=0.27229349774514305
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([2.37483169, 0.37485472]), self.bias = 1.651757568616
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.76089248, -0.92106074]), self.bias = -0.167766811
init_error=2.2512082327585796, final_error=2.2512082327585796
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.25565655, 0.22673772]), self.bias = -0.0657619206
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([0.33946022, 0.95139658]), self.bias = -0.0169498101
init_error=0.435703025063313, final_error=0.435703025063313
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-1.38515537, -0.65713466]), self.bias = -0.149249207
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([1.21059426, 1.35050307]), self.bias = -0.658690401643
init_error=1.1647755710347194, final_error=1.1647755710347194
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.58522312, -0.3490998 ]), self.bias = 1.944388696
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.15551199,  0.1663599 ]), self.bias = -0.198060493
init_error=2.3019751474647183, final_error=2.3019751474647183
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.25059619, -0.52539534]), self.bias = 0.603665979
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
```

```
layer 0, neuron 1, self.weights =array([-0.2080929 ,  0.09298336]), self.bias = 0.6178204716
init_error=0.4486053382540594, final_error=0.4486053382540594
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([2.38291057, 0.67491053]), self.bias = 0.293084958503
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-1.11453475,  0.40377278]), self.bias = -0.522015985
init_error=1.4081474779130112, final_error=1.4081474779130112
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 0.2415219 , -0.72210289]), self.bias = -1.373741087
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([0.80055656, 0.51445247]), self.bias = -0.4472714714
init_error=2.101284987441354, final_error=2.101284987441354
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 0.57742866, -0.09101593]), self.bias = 1.983467608
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.46323292, -0.30951831]), self.bias = 1.437560980
init_error=1.7321838358375303, final_error=1.7321838358375303
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-2.66055705,  1.60927243]), self.bias = -0.698774917
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-2.84306762,  0.17099937]), self.bias = -1.122071492
init_error=2.9547402530534157, final_error=2.9547402530534157
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.6843551 ,  0.20633044]), self.bias = 0.183076733
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.96407197, -0.76887045]), self.bias = 0.376365164
init_error=0.3643737327898459, final_error=0.3643737327898459
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([1.12901626, 0.06162086]), self.bias = 0.036825730716
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-1.97143653, -1.31547345]), self.bias = 1.767963438
init_error=0.10055916451901417, final_error=0.10055916451901417
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.02534051, -1.35322215]), self.bias = -0.200782801
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 1.41879458, -1.33928562]), self.bias = 1.648173373
```

```
init_error=0.18341092469730866, final_error=0.18341092469730866
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.96500077, 0.3898907 ]), self.bias = 0.219138341033
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.01680078, -0.54536135]), self.bias = 1.944566280
init_error=0.5829707135354532, final_error=0.5829707135354532
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.38980803, 0.12661282]), self.bias = 2.139079787242
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([1.00454541, 0.34285054]), self.bias = 1.688441263744
init_error=0.8836596907285362, final_error=0.8836596907285362
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.82786906, 0.02468455]), self.bias = -0.01057100014
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.35848453, -0.45674446]), self.bias = 1.372106781
init_error=0.01497785895763772, final_error=0.01497785895763772
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.06838158,  0.30396809]), self.bias = -0.43005198
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([1.09425751, 0.96342487]), self.bias = -0.92484329845
init_error=2.0282570125419324, final_error=2.0282570125419324
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 1.08523362, -0.4308284 ]), self.bias = -0.30130557
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.28592051, -1.40590448]), self.bias = -0.05494047
init_error=1.0271155119347595, final_error=1.0271155119347595
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-1.55150029, -0.00514202]), self.bias = 0.19716768995
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.05230838, -0.25723747]), self.bias = 0.870069016
init_error=0.08480385943823289, final_error=0.08480385943823289
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.62533305, 0.43872361]), self.bias = -0.13108690079
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.90641476,  1.08560578]), self.bias = 0.685826202
init_error=0.13443306966147978, final_error=0.13443306966147978
```

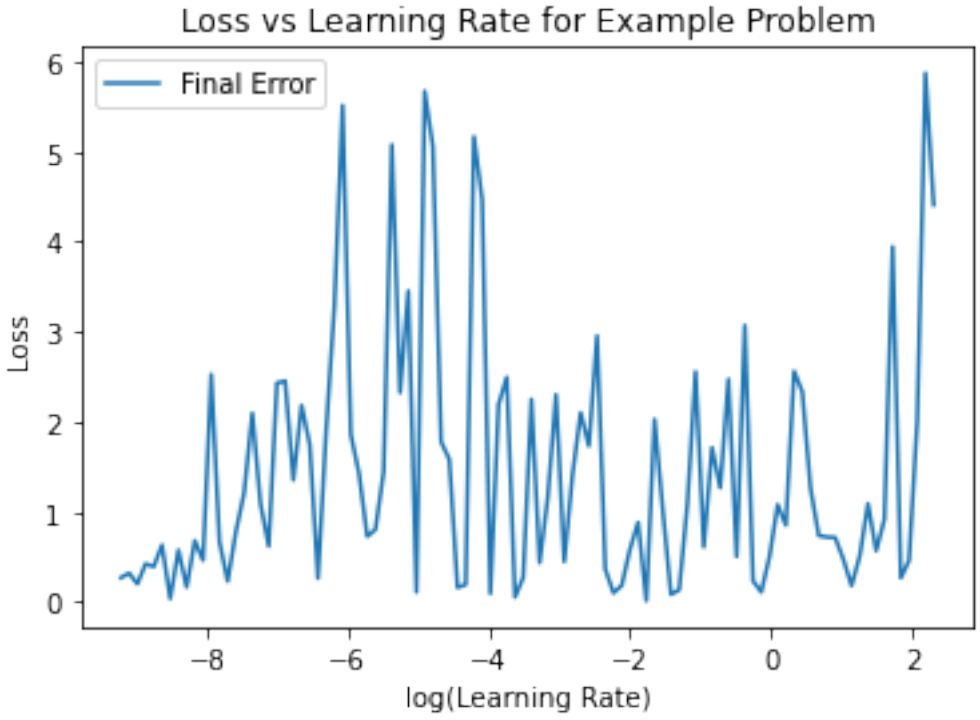
```
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.4093836, 0.9760102]), self.bias = -0.80679319370196
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.38209165, -1.30690959]), self.bias = 0.2361935546
init_error=1.0857810895248021, final_error=1.0857810895248021
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 0.66285366, -0.07087064]), self.bias = 0.025684462
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.07701553,  0.72241657]), self.bias = -0.99312067
init_error=2.5572485917825496, final_error=2.5572485917825496
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.51414895, 1.17367157]), self.bias = 1.618228773692
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.25922017, -0.33214673]), self.bias = 0.3638785100
init_error=0.6105143996065614, final_error=0.6105143996065614
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 1.71157011, -1.03423583]), self.bias = -0.62734364
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([1.07367487, 0.85393822]), self.bias = -0.4040397324
init_error=1.714426754541134, final_error=1.714426754541134
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([1.89612479, 1.5594044 ]), self.bias = -0.25579407128
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([0.58650573, 0.34428961]), self.bias = -0.0923961296
init_error=1.2675659347436226, final_error=1.2675659347436226
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-1.02213011,  0.83052196]), self.bias = -0.990479245
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.9588825 , -2.25642705]), self.bias = -0.10083766
init_error=2.473350215028319, final_error=2.473350215028319
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.22787445, -1.97296917]), self.bias = 0.178987090
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.13491893,  0.21801493]), self.bias = 2.602632566
init_error=0.5044879532222777, final_error=0.5044879532222777
layer 0 weights:
```

```
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.24133882, 0.41364627]), self.bias = -0.020343929762
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.20465508, -1.63411466]), self.bias = -0.631609266
init_error=3.070510195148993, final_error=3.070510195148993
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 0.89131803, -0.51340145]), self.bias = -0.15513571
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([0.70395994, 0.08510196]), self.bias = 1.156578225954
init_error=0.23508104475420244, final_error=0.23508104475420244
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 1.33515029, -1.34193374]), self.bias = 0.079116695
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-1.27481946, -0.30081403]), self.bias = 1.484945535
init_error=0.11050869220810662, final_error=0.11050869220810662
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-1.43505761, -1.36424751]), self.bias = -11.14952637
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-1.31538415,  1.50845335]), self.bias = 1.462837916
init_error=0.5028045681106689, final_error=0.5028045681106689
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.02822473,  0.33682426]), self.bias = -0.51438658
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.62687944, -2.00751329]), self.bias = 1.250219134
init_error=1.08099221681094, final_error=1.08099221681094
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.2175193 , -0.48497435]), self.bias = 6.680822898
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.06671058, -0.24738716]), self.bias = 2.017600592
init_error=0.8519837656474352, final_error=0.8519837656474352
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.61953092,  2.21801896]), self.bias = -1.11336258
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.13617509,  0.20637412]), self.bias = 1.057004939
init_error=2.5612164863287403, final_error=2.5612164863287403
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
```

```
layer 0, neuron 0, self.weights =array([ 0.71951091, -0.57707073]), self.bias = 5.1289469735
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.57669747, -1.17121158]), self.bias = 3.4229052795
init_error=2.3331674635742288, final_error=2.3331674635742288
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.13524509, -0.68304523]), self.bias = 0.1946301073
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.46185919, -0.10477095]), self.bias = 1.2161787435
init_error=1.2630215822236976, final_error=1.2630215822236976
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.64071675, -0.59514985]), self.bias = -3.964184709
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.87203255, -0.42822843]), self.bias = 1.9635329705
init_error=0.7374809160853417, final_error=0.7374809160853417
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.11442794, 0.73276662]), self.bias = 0.153778878624
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.91037626,  3.52039562]), self.bias = 1.924492778
init_error=0.7205043913107383, final_error=0.7205043913107383
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 0.9924705 , -0.32263194]), self.bias = 0.290986364
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.93209777, -1.63898914]), self.bias = 2.780556430
init_error=0.7161407763838414, final_error=0.7161407763838414
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-2.3546627 , -0.88153604]), self.bias = 0.559659764
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.9477178 , -1.15571311]), self.bias = 3.008819590
init_error=0.47412106941635135, final_error=0.47412106941635135
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 0.12629313, -0.56599145]), self.bias = 0.601395764
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.84251565, -1.705193  ]), self.bias = 3.892615916
init_error=0.1819703949914391, final_error=0.1819703949914391
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-1.76853453, -5.59818514]), self.bias = -52.7879690
```

```
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-1.57045056,  1.94499349]), self.bias = 3.940051654
init_error=0.5129398739126645, final_error=0.5129398739126645
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 0.29529205, -0.6211511 ]), self.bias = 1.007445195
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([1.99737559, 1.21239558]), self.bias = 4.125726747124
init_error=1.0908186914299614, final_error=1.0908186914299614
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-0.37750001,  0.6320182 ]), self.bias = 0.336355558
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([1.80710163, 2.49722279]), self.bias = 6.238557667482
init_error=0.5691633587611532, final_error=0.5691633587611532
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 0.16398016, -0.00304354]), self.bias = 1.559791341
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.03016723,  0.41752634]), self.bias = 5.532215068
init_error=0.9172449745735302, final_error=0.9172449745735302
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([ 0.58044684, -0.74005955]), self.bias = 1.955300244
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 0.17544907, -0.29519271]), self.bias = 4.105388481
init_error=3.9457655728857373, final_error=3.9457655728857373
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([-1.58513553, -2.40960006]), self.bias = -15.6280253
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-1.21132225,  1.86321276]), self.bias = 7.833635970
init_error=0.2676651522391442, final_error=0.2676651522391442
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.62428491, 0.59150608]), self.bias = 2.399517870531
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([ 1.59642884, -0.18317082]), self.bias = 7.503352366
init_error=0.4616289646174023, final_error=0.4616289646174023
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([0.13920451, 0.27252156]), self.bias = 2.113815901065
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
```

```
layer 0, neuron 1, self.weights =array([0.39056438, 1.3061419 ]), self.bias = 6.873798800443
init_error=2.0141173758344495, final_error=2.0141173758344495
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([2.96272618, 0.32448863]), self.bias = 19.48297291744
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([-0.57831708,  1.04671347]), self.bias = 7.131112618
init_error=5.872120873110711, final_error=5.872120873110711
layer 0 weights:
layer 0, neuron 0, self.input = array([0.05, 0.1 ])
layer 0, neuron 0, self.weights =array([1.17055019e-05, 2.17058782e-01]), self.bias = 4.3717
layer 0, neuron 1, self.input = array([0.05, 0.1 ])
layer 0, neuron 1, self.weights =array([0.48000833, 2.35269763]), self.bias = 8.11704570289
init_error=4.409442047649335, final_error=4.409442047649335
```



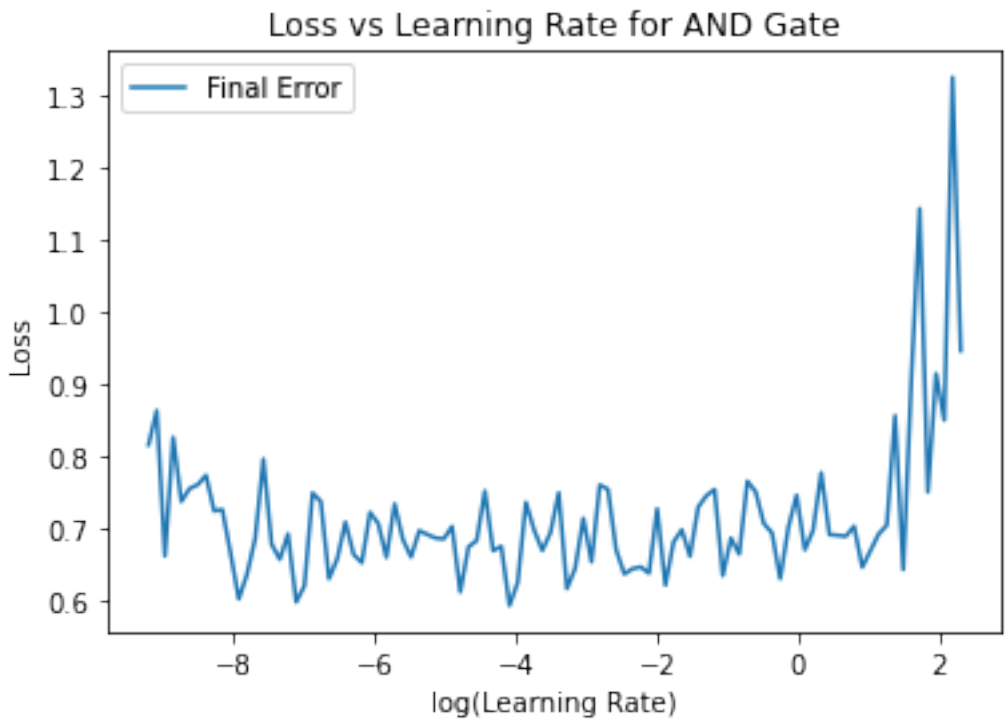
AND Gate

```
# Define a range of learning rates to test
learning_rates = np.logspace(-4, 1, 100)
final_errors = []

# Call the execute_neural_net function for each learning rate
for lr in learning_rates:
    yp, init_error, final_error = execute_neural_net(lr, 'and')
    final_errors.append(final_error)

# Plot the loss vs learning rate
plot_loss_by_learning_rate(learning_rates, final_errors, title='Loss vs Learning Rate for
```

```
init_error=1.0501616439187285, final_error=0.8149839860977218
init_error=1.1732736213098516, final_error=0.862575250354768
init_error=0.7916445890020054, final_error=0.6605697570932648
init_error=1.209268203984964, final_error=0.8253234364417653
init_error=1.0735350660206728, final_error=0.7364374626999373
init_error=1.0311492506060453, final_error=0.7542662712733106
init_error=1.2290097002232783, final_error=0.7595646384227009
init_error=1.3046968053775536, final_error=0.7726534527888185
init_error=1.035311760903309, final_error=0.72320419871272
init_error=0.9248885951354532, final_error=0.7249943005846202
init_error=0.8159430225168153, final_error=0.6644185417965167
init_error=0.8465341990853994, final_error=0.6009740966761753
init_error=0.7964027230010944, final_error=0.6350846771982415
init_error=0.7855782916533749, final_error=0.6850370055840516
init_error=1.0860904413076016, final_error=0.7951996622288582
init_error=0.8479769840373319, final_error=0.6770505318488855
init_error=0.7672429538334287, final_error=0.6561122218581926
init_error=1.267373136750935, final_error=0.6912592043136655
init_error=0.9256643265483346, final_error=0.5967984114183276
init_error=0.7264300222123283, final_error=0.6191795581415583
init_error=1.0016239843271924, final_error=0.7481476030169231
init_error=1.16852169715283, final_error=0.7363056338151337
init_error=0.9117645296523669, final_error=0.629214377611202
init_error=0.8510719746236437, final_error=0.6565557081314148
init_error=0.9533864126111993, final_error=0.707660638137123
init_error=0.8600641227455421, final_error=0.662860834065907
init_error=0.8087444804502792, final_error=0.6514113096518674
init_error=1.1873162843387934, final_error=0.7211833328854604
```

init_error=1.0096871976534034, final_error=0.7057867650640106
init_error=0.8064041068908255, final_error=0.6587770935177204
init_error=1.0851973528859882, final_error=0.7329503701132245
init_error=0.9142444833597136, final_error=0.683810820375568
init_error=0.7181957631499252, final_error=0.6593340665697714
init_error=0.8571902091521999, final_error=0.695766867072539
init_error=0.9645752019269379, final_error=0.6904260109990519
init_error=1.074458407930701, final_error=0.6855273897608366
init_error=1.239805806913213, final_error=0.6844542142667092
init_error=1.0506295856947265, final_error=0.7016474004513351
init_error=0.915168379207877, final_error=0.610793748512316
init_error=0.8603025648356529, final_error=0.6731999823069474
init_error=1.2209712573082074, final_error=0.6820912875603449
init_error=0.9456815489500111, final_error=0.7510001936923925
init_error=0.8190722069220265, final_error=0.6680182446659346
init_error=0.9054940622720724, final_error=0.6740661842924125
init_error=0.8645987107170069, final_error=0.591982700870904
init_error=0.8136662620464351, final_error=0.6238210951519536
init_error=1.1374743559447809, final_error=0.7350901901668971
init_error=1.099183504083551, final_error=0.696777441686782
init_error=0.7660632182632526, final_error=0.6683362889797908
init_error=0.9472566974542069, final_error=0.6932983101097561
init_error=0.965252515808322, final_error=0.7485108740158519
init_error=0.7896317904334759, final_error=0.6154093788298161
init_error=0.8080377974715546, final_error=0.6434279810294692
init_error=1.235303024260475, final_error=0.7130565893020332
init_error=0.8734357303670036, final_error=0.6524828520132301
init_error=1.157701862091333, final_error=0.759220771036716
init_error=1.0037141541688905, final_error=0.7534285307856559
init_error=1.1090348263002583, final_error=0.6691891409078039
init_error=0.8800183052512186, final_error=0.635751499717311
init_error=1.0098461358390205, final_error=0.6429803685391421
init_error=0.7833952995387535, final_error=0.645536762256707
init_error=0.9264484722208652, final_error=0.6366910740124488
init_error=1.1433066295841203, final_error=0.7262329917285801
init_error=0.7997117048200032, final_error=0.6201787048799609
init_error=0.8517006635773022, final_error=0.6801469230768826
init_error=0.7549300271325399, final_error=0.6969193893412493
init_error=0.8794296958812023, final_error=0.6603945368455337
init_error=1.2365161532525317, final_error=0.7285754094453469
init_error=1.1722453043992889, final_error=0.7443076923109807
init_error=1.0310285369907137, final_error=0.7530397020710358
init_error=0.7696018913209395, final_error=0.6334923740598473

init_error=1.0769954264453325, final_error=0.6854390237921704
init_error=1.0013496637838886, final_error=0.663375631758653
init_error=0.961435125905456, final_error=0.7644176520006678
init_error=0.9516176707218351, final_error=0.7500478195198819
init_error=1.3002771536998603, final_error=0.7057930472194618
init_error=1.01397695018065, final_error=0.6926995750401816
init_error=1.0833809866494057, final_error=0.6291714865456501
init_error=1.0754938035129953, final_error=0.7007237869472043
init_error=0.9352755568184176, final_error=0.744923391187446
init_error=1.0584138801854477, final_error=0.669297679146522
init_error=1.0776320676752786, final_error=0.6960980911919002
init_error=1.1209335350714913, final_error=0.7762416987295881
init_error=1.1722267129780495, final_error=0.6902383506350694
init_error=0.8062717295086623, final_error=0.6894174032708672
init_error=0.7811873948789116, final_error=0.6879752517035933
init_error=1.2881427843343465, final_error=0.7019673755303875
init_error=0.9681003845240013, final_error=0.6450681385829149
init_error=0.7905430824656152, final_error=0.6680952904978993
init_error=0.9170010897348946, final_error=0.6911850770007454
init_error=1.2319349538098412, final_error=0.703515966954908
init_error=0.953545395987715, final_error=0.8554105909217706
init_error=1.114330348562795, final_error=0.641696999370935
init_error=1.285351003088863, final_error=0.9098736177881537
init_error=1.098169853916637, final_error=1.1430177594316924
init_error=1.2501725373753632, final_error=0.7493360193189711
init_error=0.9241670590465965, final_error=0.9137741460878669
init_error=1.113473207564847, final_error=0.8491225326060201
init_error=0.8354421492020778, final_error=1.32529590157704
init_error=1.0531249308295563, final_error=0.9456352799311161



XOR Gate with no hidden layer

```
# Define a range of learning rates to test
learning_rates = np.logspace(-4, 1, 100)
final_errors = []

# Call the execute_neural_net function for each learning rate
for lr in learning_rates:
    yp, init_error, final_error = execute_neural_net(lr, 'xor1')
    final_errors.append(final_error)

# Plot the loss vs learning rate
plot_loss_by_learning_rate(learning_rates, final_errors, title='Loss vs Learning Rate for
```

```
XOR using single perceptron:
init_error=0.19488175338587777, final_error=1.0806286747306435
XOR using single perceptron:
init_error=0.23604594155577724, final_error=1.1457296555218994
XOR using single perceptron:
```

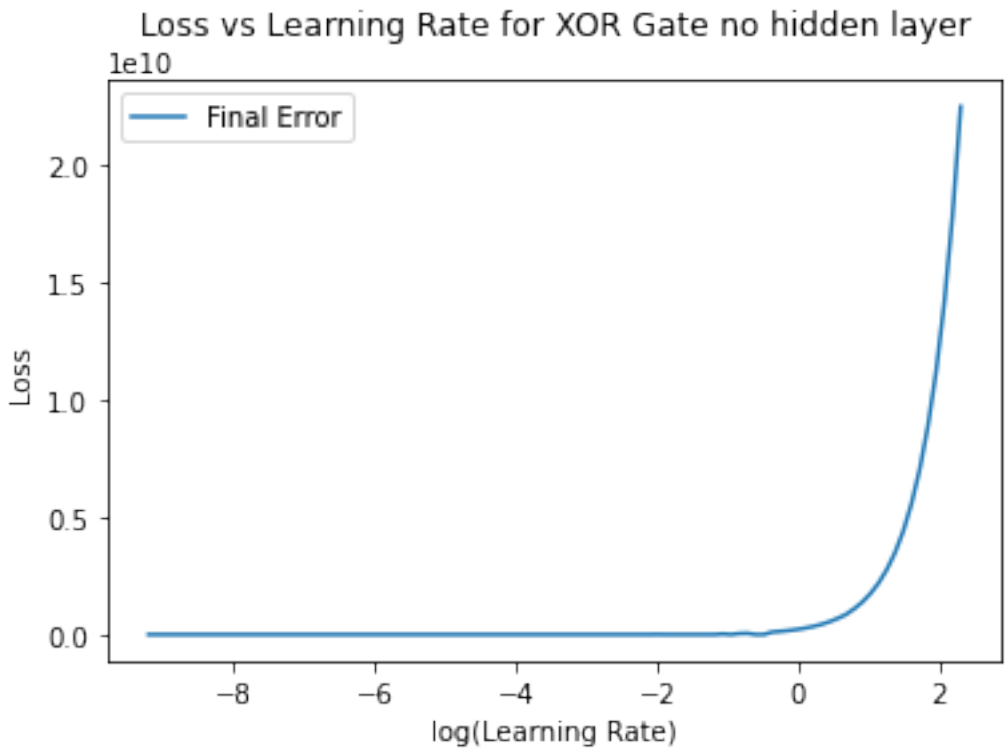
```
init_error=0.4247218490899999, final_error=0.5023073141278305
XOR using single perceptron:
init_error=0.9293780575424018, final_error=1.0812852237037154
XOR using single perceptron:
init_error=0.24593390016518774, final_error=1.3555468798209158
XOR using single perceptron:
init_error=0.5473510417128358, final_error=1.7488219375034508
XOR using single perceptron:
init_error=0.6235943383853629, final_error=1.375411763747778
XOR using single perceptron:
init_error=0.3711927569816466, final_error=0.40855501259467825
XOR using single perceptron:
init_error=0.18292644029570748, final_error=0.9162922741556601
XOR using single perceptron:
init_error=0.8769163448778992, final_error=1.228416551334675
XOR using single perceptron:
init_error=0.5439366166064721, final_error=1.4024571753781165
XOR using single perceptron:
init_error=0.4518906887608783, final_error=1.365727762848838
XOR using single perceptron:
init_error=0.8577143739198516, final_error=0.6745138046526665
XOR using single perceptron:
init_error=0.3004919319786188, final_error=0.3357251153726219
XOR using single perceptron:
init_error=0.8622184524658938, final_error=0.6861481589100575
XOR using single perceptron:
init_error=0.17918633824849944, final_error=1.3756332494608798
XOR using single perceptron:
init_error=0.4266935219410609, final_error=1.479453199466395
XOR using single perceptron:
init_error=0.7868880631289195, final_error=0.9013993912756771
XOR using single perceptron:
init_error=0.17034002950420535, final_error=0.16760520883943616
XOR using single perceptron:
init_error=0.43311807252665047, final_error=1.8691179197552081
XOR using single perceptron:
init_error=0.44137138868076475, final_error=0.41071743338635297
XOR using single perceptron:
init_error=0.8132694184268251, final_error=0.7903309463906989
XOR using single perceptron:
init_error=0.5837428015957282, final_error=0.737124998660199
XOR using single perceptron:
init_error=0.374040780427415, final_error=0.45545457495944525
```

```
XOR using single perceptron:
init_error=0.13334752249217244, final_error=0.13336752421136527
XOR using single perceptron:
init_error=0.3225604414559859, final_error=0.6274975059220799
XOR using single perceptron:
init_error=0.3212701880091121, final_error=1.1781279112935625
XOR using single perceptron:
init_error=0.38276332467720436, final_error=0.3458980908389931
XOR using single perceptron:
init_error=0.17825351624466118, final_error=0.7535631985656551
XOR using single perceptron:
init_error=0.3450386375971074, final_error=0.34440120417097514
XOR using single perceptron:
init_error=0.7170043016380968, final_error=0.9044899330950609
XOR using single perceptron:
init_error=0.4543378433329204, final_error=0.7210583681123792
XOR using single perceptron:
init_error=0.14711617040333655, final_error=0.1315183039299879
XOR using single perceptron:
init_error=0.3504464991006383, final_error=1.1061067531764819
XOR using single perceptron:
init_error=0.23031351171790987, final_error=1.092194334328481
XOR using single perceptron:
init_error=0.3383667919483309, final_error=0.9219378895228022
XOR using single perceptron:
init_error=0.14441095747378635, final_error=0.14449100953944416
XOR using single perceptron:
init_error=0.16797222122360186, final_error=0.1394728812388546
XOR using single perceptron:
init_error=0.31926414751623294, final_error=0.7636897845240536
XOR using single perceptron:
init_error=0.2325495054606688, final_error=1.868634131824796
XOR using single perceptron:
init_error=0.42090231889666746, final_error=0.7230966384524403
XOR using single perceptron:
init_error=0.1368735588794764, final_error=0.13359973403231157
XOR using single perceptron:
init_error=0.15849850656875072, final_error=0.1581201771355423
XOR using single perceptron:
init_error=0.3882617765271154, final_error=0.8671340268536534
XOR using single perceptron:
init_error=0.18333618620597658, final_error=0.15198703105022457
XOR using single perceptron:
```

```
init_error=0.37456718216555035, final_error=0.7658777133040173
XOR using single perceptron:
init_error=0.44841475320782465, final_error=0.337008602754469
XOR using single perceptron:
init_error=0.3362870055477643, final_error=0.2983739359908954
XOR using single perceptron:
init_error=0.24746563314265674, final_error=0.32746208615487893
XOR using single perceptron:
init_error=0.49408739112755407, final_error=1.2856355020586985
XOR using single perceptron:
init_error=0.15901478943997155, final_error=0.1585058253733924
XOR using single perceptron:
init_error=0.6261673374274652, final_error=0.1254876513936977
XOR using single perceptron:
init_error=0.4874728596339456, final_error=0.827623758266746
XOR using single perceptron:
init_error=0.843000466750951, final_error=0.6285804986758595
XOR using single perceptron:
init_error=0.3332771097506218, final_error=1.6231990828650158
XOR using single perceptron:
init_error=0.19551723559274756, final_error=0.14829692423780672
XOR using single perceptron:
init_error=0.3066859948530063, final_error=1.057092027720063
XOR using single perceptron:
init_error=0.35163714465575896, final_error=0.3553113647512882
XOR using single perceptron:
init_error=0.14512084219982507, final_error=0.1449817904453935
XOR using single perceptron:
init_error=0.15969583076305094, final_error=0.1606434543823749
XOR using single perceptron:
init_error=0.389811809712198, final_error=0.3488467191564719
XOR using single perceptron:
init_error=0.21313051258741433, final_error=0.9143367356929422
XOR using single perceptron:
init_error=0.17377130710649402, final_error=4152923.0869908123
XOR using single perceptron:
init_error=0.26950704510012735, final_error=2.0500440880909627
XOR using single perceptron:
init_error=0.5871956078784903, final_error=0.1576869573868295
XOR using single perceptron:
init_error=0.48042366862002367, final_error=0.5470106767379475
XOR using single perceptron:
init_error=0.4308243742296316, final_error=0.1308381297315438
```

```
XOR using single perceptron:
init_error=0.15079710566862142, final_error=0.13809003964164324
XOR using single perceptron:
init_error=0.5181114011127764, final_error=1.7314412047989594
XOR using single perceptron:
init_error=0.45151856895767206, final_error=0.7968818202827067
XOR using single perceptron:
init_error=0.20869501685215422, final_error=26531360.912342925
XOR using single perceptron:
init_error=0.5427095739222174, final_error=1.1555743264857292
XOR using single perceptron:
init_error=0.8153401669405725, final_error=42189945.13812913
XOR using single perceptron:
init_error=0.2116663865339395, final_error=53272449.86582832
XOR using single perceptron:
init_error=0.15373656731369156, final_error=0.1430906704367741
XOR using single perceptron:
init_error=1.3372719744077903, final_error=11.260177198035196
XOR using single perceptron:
init_error=0.4027010312899516, final_error=106867193.7738869
XOR using single perceptron:
init_error=0.660308266648414, final_error=134930734.56280664
XOR using single perceptron:
init_error=0.5917549240331842, final_error=170344943.47703922
XOR using single perceptron:
init_error=0.31796823657272166, final_error=214885721.64470124
XOR using single perceptron:
init_error=0.25231950203816916, final_error=271147037.8706509
XOR using single perceptron:
init_error=0.23663894471162195, final_error=342133859.28900695
XOR using single perceptron:
init_error=0.2739099281617291, final_error=431685998.6194799
XOR using single perceptron:
init_error=0.2661356122259359, final_error=544685229.2596006
XOR using single perceptron:
init_error=0.1865682602246385, final_error=686647504.4805512
XOR using single perceptron:
init_error=0.9169251257974658, final_error=850266247.9990455
XOR using single perceptron:
init_error=0.43075784869256595, final_error=1094169918.9553308
XOR using single perceptron:
init_error=0.6503447369935014, final_error=1380654000.2999249
XOR using single perceptron:
```

```
init_error=0.14865851898140114, final_error=1746835100.338322
XOR using single perceptron:
init_error=0.1496975128714231, final_error=2197773855.6322994
XOR using single perceptron:
init_error=0.29876817080190826, final_error=2773694965.0711355
XOR using single perceptron:
init_error=0.760895972817376, final_error=3492914783.5634346
XOR using single perceptron:
init_error=0.8433601867253325, final_error=4404826777.706676
XOR using single perceptron:
init_error=0.36665229574756425, final_error=5573183268.869962
XOR using single perceptron:
init_error=0.17250568903718366, final_error=7029034867.184996
XOR using single perceptron:
init_error=0.5289332103526929, final_error=8871160999.916462
XOR using single perceptron:
init_error=0.32619248366675024, final_error=11197406141.126766
XOR using single perceptron:
init_error=0.19974810360091638, final_error=14133675849.305824
XOR using single perceptron:
init_error=0.1472757381748502, final_error=17823997192.349815
XOR using single perceptron:
init_error=0.16379126950392822, final_error=22490373771.23959
```



XOR Gate with single hidden layer

```
# Define a range of learning rates to test
learning_rates = np.logspace(-4, 1, 100)
final_errors = []

# Call the execute_neural_net function for each learning rate
for lr in learning_rates:
    yp, init_error, final_error = execute_neural_net(lr, 'xor1')
    final_errors.append(final_error)

# Plot the loss vs learning rate
plot_loss_by_learning_rate(learning_rates, final_errors, title='Loss vs Learning Rate for
```

```
XOR using single perceptron:
init_error=0.4822595686664164, final_error=0.4224916505969806
XOR using single perceptron:
init_error=0.44491117041039957, final_error=0.41899978899308316
```

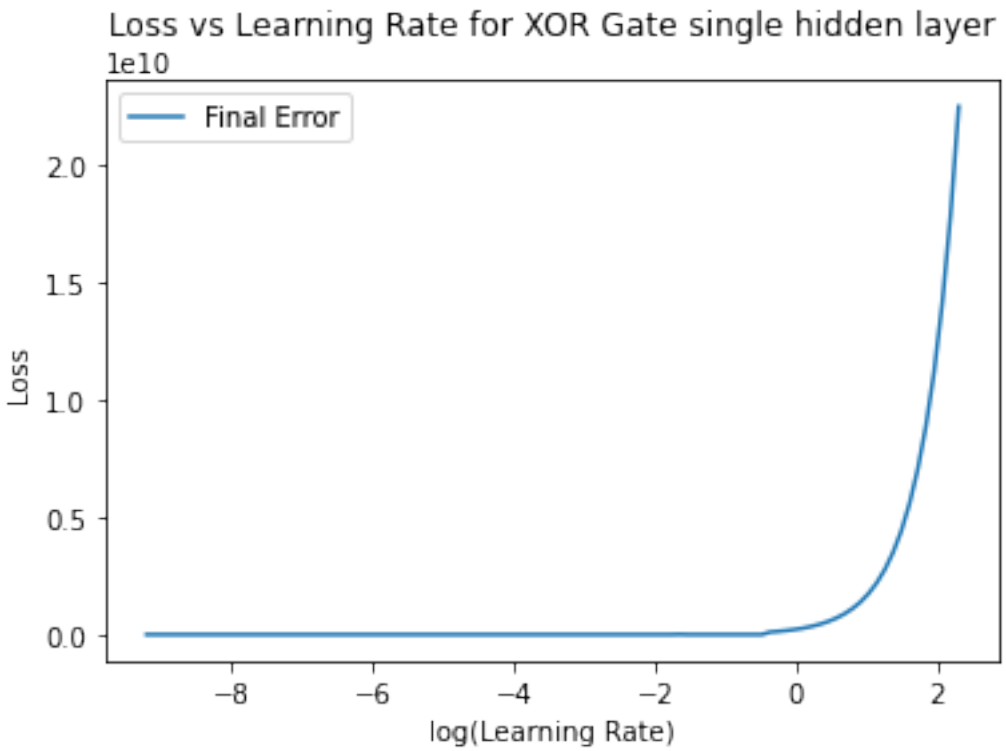
```
XOR using single perceptron:
init_error=0.8197515251359695, final_error=0.4763562573893592
XOR using single perceptron:
init_error=0.3724367231599793, final_error=1.5701313029331796
XOR using single perceptron:
init_error=0.5494147210741289, final_error=0.33687831092357085
XOR using single perceptron:
init_error=0.5825510808049282, final_error=1.3784233239397943
XOR using single perceptron:
init_error=0.7007419584895828, final_error=1.1122780051272039
XOR using single perceptron:
init_error=0.28789420125285226, final_error=0.6991245529188181
XOR using single perceptron:
init_error=0.20444936269652692, final_error=0.1396106622825715
XOR using single perceptron:
init_error=0.21082712901281092, final_error=1.5689597960301624
XOR using single perceptron:
init_error=0.18566168716292641, final_error=0.15835825572739187
XOR using single perceptron:
init_error=0.6694397835150583, final_error=0.5060263933243974
XOR using single perceptron:
init_error=0.28015927703191235, final_error=0.45635196973168035
XOR using single perceptron:
init_error=0.21890754118654293, final_error=1.3360315981953255
XOR using single perceptron:
init_error=0.3427269664493229, final_error=0.5150243604047594
XOR using single perceptron:
init_error=0.2699721144128559, final_error=1.1766380122405025
XOR using single perceptron:
init_error=0.8484248513027834, final_error=0.576524473609977
XOR using single perceptron:
init_error=0.3728182561731191, final_error=0.6114749452563701
XOR using single perceptron:
init_error=0.30462959216899826, final_error=1.0711164155365809
XOR using single perceptron:
init_error=0.18195849172678322, final_error=0.17114125834563992
XOR using single perceptron:
init_error=0.3769787339996888, final_error=2.5038236208963958
XOR using single perceptron:
init_error=0.2640053901905039, final_error=1.120000684770572
XOR using single perceptron:
init_error=0.8865437695648214, final_error=0.7149476925370988
XOR using single perceptron:
```

```
init_error=0.14956996843969889, final_error=0.150620588525256
XOR using single perceptron:
init_error=0.4564550304360752, final_error=1.9296573263827763
XOR using single perceptron:
init_error=0.2398528352819914, final_error=0.3587386846633734
XOR using single perceptron:
init_error=0.46048933709655526, final_error=0.6948684259170386
XOR using single perceptron:
init_error=0.2788405445634599, final_error=1.2268955256802814
XOR using single perceptron:
init_error=0.20296424252689887, final_error=0.20402085691292518
XOR using single perceptron:
init_error=0.6050209642162896, final_error=0.5817047483261687
XOR using single perceptron:
init_error=0.6742933639669918, final_error=0.8695205821249492
XOR using single perceptron:
init_error=0.1703920040420735, final_error=0.16970310543715186
XOR using single perceptron:
init_error=0.33477799854801066, final_error=1.3227683861034336
XOR using single perceptron:
init_error=0.26940840770559155, final_error=1.9159726319431183
XOR using single perceptron:
init_error=0.3593762141242298, final_error=0.7928439322998693
XOR using single perceptron:
init_error=0.16640578530016828, final_error=0.15539191408220704
XOR using single perceptron:
init_error=0.6568392340338325, final_error=0.5096745950234118
XOR using single perceptron:
init_error=0.21961029203287194, final_error=1.7939048411638714
XOR using single perceptron:
init_error=0.23852511464316978, final_error=0.8547617377105937
XOR using single perceptron:
init_error=1.208342265441687, final_error=0.803373062952369
XOR using single perceptron:
init_error=0.7206292497589875, final_error=0.8366818526878432
XOR using single perceptron:
init_error=0.1737076310569117, final_error=0.16989605497565036
XOR using single perceptron:
init_error=0.7123898066921001, final_error=0.41636967390730617
XOR using single perceptron:
init_error=0.7133481301572193, final_error=716.724490781188
XOR using single perceptron:
init_error=0.42318858643228585, final_error=0.9305087914256249
```

```
XOR using single perceptron:
init_error=0.25138145704574283, final_error=0.3214030570802411
XOR using single perceptron:
init_error=0.8863550304722755, final_error=0.5911736800293876
XOR using single perceptron:
init_error=0.6086351151196526, final_error=0.7729363724251817
XOR using single perceptron:
init_error=0.20178195522790496, final_error=0.8553108846410874
XOR using single perceptron:
init_error=0.21636162300552783, final_error=0.8018967868839455
XOR using single perceptron:
init_error=0.4522553557777226, final_error=0.4397979611264159
XOR using single perceptron:
init_error=0.3966581356559855, final_error=1.6416448748252555
XOR using single perceptron:
init_error=0.2132553748481318, final_error=0.7960889295590394
XOR using single perceptron:
init_error=0.6474148244296467, final_error=1.2140893885433246
XOR using single perceptron:
init_error=0.5447097839206666, final_error=0.8008560095362252
XOR using single perceptron:
init_error=0.2587181136165658, final_error=0.791598312665187
XOR using single perceptron:
init_error=0.5715207514819972, final_error=0.1354955284318731
XOR using single perceptron:
init_error=0.3092906951756562, final_error=0.9849184074967073
XOR using single perceptron:
init_error=0.5880818127861223, final_error=0.8190262102348111
XOR using single perceptron:
init_error=0.5717051390858531, final_error=1.9454580971993487
XOR using single perceptron:
init_error=0.2457320302456959, final_error=1.0019471910941091
XOR using single perceptron:
init_error=0.7828544201376311, final_error=1.655672961085684
XOR using single perceptron:
init_error=0.41192344916631557, final_error=0.4864383779946438
XOR using single perceptron:
init_error=0.6521356775779785, final_error=1.2554436310134767
XOR using single perceptron:
init_error=0.7537238797239441, final_error=0.8910128008786118
XOR using single perceptron:
init_error=0.18240720710290492, final_error=8367993.50916205
XOR using single perceptron:
```

```
init_error=0.5470290084003708, final_error=0.7125771269928367
XOR using single perceptron:
init_error=0.1652457124375532, final_error=0.13763480606220493
XOR using single perceptron:
init_error=0.16068331399574393, final_error=0.1376472201187458
XOR using single perceptron:
init_error=0.8568094399297606, final_error=1.188176973016713
XOR using single perceptron:
init_error=0.3173729678243705, final_error=0.9396744390873917
XOR using single perceptron:
init_error=0.23638041899376197, final_error=1.2291673937435075
XOR using single perceptron:
init_error=0.15479122711677917, final_error=0.15482366665827937
XOR using single perceptron:
init_error=0.5851749004986357, final_error=0.16673375818775857
XOR using single perceptron:
init_error=0.43617690655981967, final_error=2.1730488563142343
XOR using single perceptron:
init_error=0.15851977557606084, final_error=0.5444299491551923
XOR using single perceptron:
init_error=0.17686551750581003, final_error=106524219.10929471
XOR using single perceptron:
init_error=0.5855559151543623, final_error=135076768.17008412
XOR using single perceptron:
init_error=0.37784588743009406, final_error=170321665.6796696
XOR using single perceptron:
init_error=0.26658651234666075, final_error=214907807.55113858
XOR using single perceptron:
init_error=1.1906024718003831, final_error=269683453.7525668
XOR using single perceptron:
init_error=1.0549744263494094, final_error=341841157.9135922
XOR using single perceptron:
init_error=0.45165044407854127, final_error=431686969.87821805
XOR using single perceptron:
init_error=0.9083328479168385, final_error=544748288.3100705
XOR using single perceptron:
init_error=0.3541604581527902, final_error=687262369.5399253
XOR using single perceptron:
init_error=0.16060541387680508, final_error=867216848.8672166
XOR using single perceptron:
init_error=0.44222835240765407, final_error=1093842224.82653
XOR using single perceptron:
init_error=0.17400696916219924, final_error=1377071401.996091
```

```
XOR using single perceptron:
init_error=0.5077977737126267, final_error=1741633127.7346249
XOR using single perceptron:
init_error=0.550188525126216, final_error=2197732657.3287196
XOR using single perceptron:
init_error=0.49986579436081313, final_error=2773134629.4894643
XOR using single perceptron:
init_error=0.27063442917888647, final_error=3500361333.037001
XOR using single perceptron:
init_error=0.16383991433692374, final_error=4411006621.63793
XOR using single perceptron:
init_error=0.8708443236294268, final_error=5572610646.73945
XOR using single perceptron:
init_error=0.5523905892615606, final_error=7031916921.490482
XOR using single perceptron:
init_error=0.34607926179869275, final_error=8873670208.71717
XOR using single perceptron:
init_error=0.4324992698648376, final_error=11188693361.750195
XOR using single perceptron:
init_error=0.5849118103564066, final_error=14122406242.272015
XOR using single perceptron:
init_error=0.1939246264270597, final_error=17834155045.790382
XOR using single perceptron:
init_error=0.15178792959799026, final_error=22494329609.67749
```


Loss vs Learning Rate for XOR Gate single hidden layer
1e10
Final Error
2.0
1.5
1.0
0.5
0.0
Loss
−8
−6
−4
−2
0
2
log(Learning Rate)