In [21]:

```
import numpy as np
import matplotlib.pyplot as plt
```

## init the perceptron

In [22]:

```
import numpy as np
import matplotlib.pyplot as plt

# create a new class for a perceptron with a hidden layer


class Perceptron:
    rng = np.random.default_rng()
    errArr = []

    def __init__(self, input_size, hidden_size, output_size):
        self.hiddenWeights = np.random.normal(0, 1/2, (hidden_size, input_size))
        self.hiddenThreshold = np.zeros([hidden_size,1])
        self.outputWeights = np.random.normal(0, 1/hidden_size, (hidden_size))
        self.outputThreshold = np.zeros(output_size)

        if(0):
            #shapes of the initializations
            print("hiddenWeights.shape: ", self.hiddenWeights.shape)
            print("hiddenThreshold.shape: ", self.hiddenThreshold.shape)
            print("outputWeights.shape: ", self.outputWeights.shape)
            print("outputThreshold.shape: ", self.outputThreshold.shape)

            # print the values
            print("hiddenWeights: ", self.hiddenWeights)
            print("hiddenThreshold: ", self.hiddenThreshold)
            print("outputWeights: ", self.outputWeights)
            print("outputThreshold: ", self.outputThreshold)


    def tanh(self, x):
        return np.tanh(x)

    def tanhDerivative(self, x):
        return 1 - np.power(np.tanh(x), 2)

    def forward(self, input):
        self.hidden_b = np.dot(self.hiddenWeights, input.T) - self.hiddenThreshold
        self.hiddenLayer = self.tanh(self.hidden_b)
        self.output_B = np.dot(self.outputWeights.T, self.hiddenLayer) - self.outputThreshold
        self.outputLayer = self.tanh(self.output_B)
        return self.outputLayer

    def backward(self, X, target, eta, output):
        error = target - output                                                #(batch_si

        error_2 = error * self.tanhDerivative(self.output_B)                        #(bat

        error_2_reshaped = error_2[:, np.newaxis]
        output_weights_reshaped = self.outputWeights[np.newaxis, :]
        tmp = error_2_reshaped * output_weights_reshaped
        gprime = self.tanhDerivative(self.hidden_b)
        error_1 = (tmp * gprime.T)

        self.outputWeights += error_2 @ self.hiddenLayer.T *eta
        self.outputThreshold -= eta * np.sum(error_2, axis=0, keepdims=True)

        self.hiddenWeights += error_1.T @ X * eta
        self.hiddenThreshold -= eta * np.sum(error_1, axis=0, keepdims=True).T


    def classificationError(self, input, target):
        prediction = np.sign(self.forward(input))
        return 0.5 * np.mean(np.abs(prediction-target))

    def trainMiniBatches(self, xTrain, yTrain, eta, epochs, mB):

        for _ in range(epochs):
            mu = self.rng.choice(xTrain.shape[0],size=mB, replace=True)
            xTrain_rand = xTrain[mu]
            yTrain_rand = yTrain[mu]

            output = self.forward(xTrain_rand)
            self.backward(xTrain_rand, yTrain_rand, eta=eta)
        return

    def trainMiniBatchesWhile(self, xTrain, yTrain, xValid, yValid, eta, epochs, mB):
        i = 0
        err = 1
        lowestErr = 1

        while err > 0.115:

            if i > 1000:
                eta = 0.001


            oldErr = err
            mu = self.rng.choice(xTrain.shape[0],size=mB, replace=False)
            xTrain_rand = xTrain[mu]
            yTrain_rand = yTrain[mu]

            output = self.forward(xTrain_rand,)
            self.backward(xTrain_rand, yTrain_rand, eta=eta, output=output)

            # calculate the error
            err = self.classificationError(xValid, yValid)
            self.errArr.append(err)

            if err<lowestErr:
                lowestErr = err

            if (i % 100 == 0):
                print("epoch: ", i, "error: ", err, "lowestErr: ", lowestErr)

            i += 1

            # if new lowest error, and error < 0.13 save the weights and thresholds
            if (err < 0.12):
                np.savetxt("csv/w1.csv", self.hiddenWeights, delimiter=",")
                np.savetxt("csv/w2.csv", self.outputWeights, delimiter=",")
                np.savetxt("csv/t1.csv", self.hiddenThreshold, delimiter=",")
                np.savetxt("csv/t2.csv", self.outputThreshold, delimiter=",")


        print("epoch: ", i, "error: ", err, "lowestErr: ", lowestErr)
        """print("hidden Weights: ", self.hiddenWeights)
        print("hidden Threshold: ", self.hiddenThreshold)
        print("output Weights: ", self.outputWeights)
        print("output Threshold: ", self.outputThreshold)"""


        return




perceptron = Perceptron(2, 2, 1)
```

In [23]:

```
def main():
    # read the training and validation data
    training_set = np.loadtxt('training_set.csv', delimiter=',')
    validation_set = np.loadtxt('validation_set.csv', delimiter=',')

    xTraining = training_set[:, :2]
    yTraining = training_set[:, 2]

    xValidation = validation_set[:, :2]
    yValidation = validation_set[:, 2]

    xTraining_mean = xTraining.mean(axis=0)
    xTraining_std = xTraining.std(axis=0)

    xTraining_normalised = (xTraining-xTraining_mean)/xTraining_std
    xValidation_normalised = (xValidation-xTraining_mean)/xTraining_std

    perceptron = Perceptron(input_size=2, hidden_size=50, output_size=1)

    # train the perceptron
    #perceptron.trainMiniBatches(xTraining_normalised, yTraining, eta=0.001, epochs=100000, mB=2**8)

    perceptron.trainMiniBatchesWhile(xTraining_normalised, yTraining, xValidation_normalised, yValid

    # classify the validation data
    valErr = perceptron.classificationError(xValidation_normalised, yValidation)
    print("Validation error: ", valErr)
```

In [24]:

```
if __name__ == "__main__":
    main()
```

```
epoch:  0 error:  0.397 lowestErr:  0.397
epoch:  100 error:  0.1516 lowestErr:  0.1516
epoch:  200 error:  0.1516 lowestErr:  0.1516
epoch:  300 error:  0.1516 lowestErr:  0.1418
epoch:  400 error:  0.1516 lowestErr:  0.1392
epoch:  500 error:  0.1516 lowestErr:  0.138
epoch:  600 error:  0.142 lowestErr:  0.1376
epoch:  700 error:  0.1384 lowestErr:  0.1366
epoch:  800 error:  0.15 lowestErr:  0.1366
epoch:  900 error:  0.1422 lowestErr:  0.1358
epoch:  1000 error:  0.1464 lowestErr:  0.1358
epoch:  1100 error:  0.138 lowestErr:  0.1348
epoch:  1200 error:  0.1378 lowestErr:  0.1346
epoch:  1300 error:  0.1356 lowestErr:  0.1344
epoch:  1400 error:  0.1376 lowestErr:  0.1344
epoch:  1500 error:  0.141 lowestErr:  0.1344
epoch:  1600 error:  0.1368 lowestErr:  0.1342
epoch:  1700 error:  0.1404 lowestErr:  0.1342
epoch:  1800 error:  0.1354 lowestErr:  0.1342
```