```
import numpy as np
import pandas as pd


# Preprocess

def load(name):
    data = pd.read_csv(name).values
    a,b  = data[:, :2], data[:, 2].reshape(-1, 1)
    return a,b


def normalize_data(x_train, x_val):
    mean_t = np.average(x_train, axis=0)
    std_t = np.sqrt(np.var(x_train, axis=0))
    mean_v = np.average(x_val,axis=0)
    std_v = np.sqrt(np.var(x_val,axis = 0))


    xt = (x_train - mean_t) / std_t
    xv = (x_val - mean_v) / std_v

    return xt, xv



# Prepare data

x_train, y_train = load('training_set.csv')
x_val, y_val = load('validation_set.csv')

x_train, x_val = normalize_data(x_train, x_val)
x_train
```

```
array([[-1.24461396, -0.46158835],
       [-0.61676414,  0.69498638],
       [ 0.27876648,  1.16213117],
       ...,
       [-0.30456223, -1.74746903],
       [ 0.33866478, -1.38420147],
       [ 0.69156081,  1.47744741]])
```

```
def Tanh(x):
    return 1 - np.tanh(x)**2


#Perception built: Capital for the former,lower case for the later propagation


class Perceptron:
    def __init__(self, M):
        self.W = (1/np.sqrt(2)) * np.random.randn(2, M)
        self.w = (1/np.sqrt(52)) * np.random.randn(M,1)     #chosen for #neurons =52, var = 1/52, could change

        self.T = np.zeros((1, M))
        self.t = 0

    def forward_propagation(self, x):
        self.E = np.dot(x, self.W) - self.T
        self.H = np.tanh(self.E)
        self.e = np.dot(self.H, self.w) - self.t
        self.O = np.tanh(self.e)
        return self.O

    def backward_propagation(self, x, t):
        delta = (t - self.O) * Tanh(self.b)
        self.dw = np.dot(self.H.T, delta)
        self.dt = np.sum(delta)

        dH = np.dot(delta, self.w.T) * Tanh(self.E)
        self.dW = np.dot(x.T, dH)
        self.dT = np.sum(dH, axis=0)

    def refresh_parameters(self, l=0.008): #learning rate could change
        self.W += self.dW *l
        self.w += self.dw *l
        self.T -= self.dT *l
        self.t -= self.dt *l


# Mini-batches built, size could change
def build_mini_batches(x, y, batch_size):
    n = x.shape[0]
```

```
    n = x.shape[0]
    indice = np.arange(n)
    np.random.shuffle(indice)

    x_shuffled = x[indice]
    y_shuffled = y[indice]

    x_m = [x_shuffled[i:i + batch_size] for i in range(0, n, batch_size)]
    y_m = [y_shuffled[i:i + batch_size] for i in range(0, n, batch_size)]

    return [(x_batch, y_batch) for x_batch, y_batch in zip(x_m, y_m)]



#Training mechanism built
#epoch, mini batches, calculate errors in validation set

class Trainer:
    def __init__(self, model, learningrate, mini_batch_size, nEpoch):
        self.model = model
        self.lr = learningrate
        self.batch_size = mini_batch_size
        self.epoch = nEpoch

    def compute_classification_error(self, a, b):
        O = self.model.forward_propagation(a)
        C = 0.5 / len(b) * np.sum(np.abs(np.sign(O) - b))
        return C

    def train_epoch(self, x_train, y_train):
        mini_batches = build_mini_batches(x_train, y_train, self.batch_size)

        for (x_m, y_m) in mini_batches:
            O = self.model.forward_propagation(x_m)
            self.model.backward_propagation(x_m, y_m)
            self.model.refresh_parameters(self.lr)

    def train(self, x_train, y_train, x_val, y_val):
        for epoch in range(self.epoch):
            self.train_epoch(x_train, y_train)
            error = self.compute_classification_error(x_val, y_val)

            if epoch % 100 == 0:
                print(f"At Epoch: {epoch}, Classification Error is: {error * 100:.2f}%")

            #target
            if error < 0.12:
                print(f"At Epoch: {epoch} as Classification Error {error* 100:.2f}% is below the target!")
                break



# Tune and test
M = 52
Prc = Perceptron(M)
trainer = Trainer(Prc, learningrate=0.008, mini_batch_size=128, nEpoch=2000)
trainer.train(x_train, y_train, x_val, y_val)
```

```
At Epoch: 0, Classification Error is: 15.16%
At Epoch: 100, Classification Error is: 12.32%
At Epoch: 200, Classification Error is: 12.08%
At Epoch: 222 as Classification Error 11.98% is below the target!
```