```
import pandas as pd
import numpy as np

### FUNCTIONS ###

# Normalization
def normalize(data, data_val):
    outdata = data.copy()
    outdata_val = data_val.copy()
    m1 = np.mean(data[:,0])
    m2 = np.mean(data[:,1])
    outdata[:,0] = (outdata[:,0] - m1) / np.std(data[:,0])
    outdata[:,1] = (outdata[:,1] - m2) / np.std(data[:,1])
    outdata_val[:,0] = (outdata_val[:,0] - m1) / np.std(data[:,0])
    outdata_val[:,1] = (outdata_val[:,1] - m2) / np.std(data[:,1])
    return outdata, outdata_val

# Initialization of weights and thresholds
def init_weights_W(M1):
    return np.random.normal(loc = 0, scale =np.sqrt(1/M1), size =
M1)
def init_weights_w(M1):
    return np.random.normal(loc = 0, scale = np.sqrt(0.5), size =
(M1,2))
def init_thresholds(M1):
    return np.zeros(M1), 0

# Local fields
def calc_local_field_b_mat(x_mat, w_mat, theta_vec):
    M1 = w_mat.shape[0]
    p = x_mat.shape[0]
    b_mat = np.zeros((M1, p))
    for mu in range(p):
        for jt in range(M1):
            b_mat[jt, mu] = (-theta_vec[jt] +
w_mat[jt,0]*x_mat[mu,0] + w_mat[jt,1]*x_mat[mu,1])
    return b_mat
def calc_local_field_B(V_mat, W_mat, theta):
    M1, p = V_mat.shape
    B_vec = np.zeros(p)
    for mu in range(p):
        tmp = 0
        for jt in range(M1):
            tmp += W_mat[jt] * V_mat[jt,mu]
        B_vec[mu] = tmp - theta
    return B_vec

# Neuron calucations
def calc_neurons(b_mat):
    return np.tanh(b_mat)
def calc_output(B_vec):
    return np.tanh(B_vec)

# Back propagation step 1
```

```
def deriv_tanh(bi):
    return 1 - np.tanh(bi)**2
def calc_Delta(t_vec,O_vec, B_vec):
    p = t_vec.shape[0]
    Delta_vec = np.zeros(p)
    for mu in range(p):
        Delta_vec[mu] =  (t_vec[mu] - O_vec[mu]) *
deriv_tanh(B_vec[mu])
    return Delta_vec
def calc_change_W_vec(eta, Delta_vec, V_mat):
    return eta * np.matmul(V_mat, Delta_vec)
def calc_change_Theta(Delta_vec, eta):
    return -eta * sum(Delta_vec)

# Back propagation step 2
def calc_delta_mat(Delta_vec, W_mat, b_mat):
    M1, p = b_mat.shape
    delta_mat = np.zeros((M1,p))
    for mu in range(p):
        for jt in range(M1):
            delta_mat[jt,mu] = Delta_vec[mu] * W_mat[jt] *
deriv_tanh(b_mat[jt,mu])
    return delta_mat
def calc_change_w_mat(eta, delta_mat, x_mat):
    return eta * np.matmul(delta_mat, x_mat)
def calc_change_theta_vec(eta, delta_mat):
    return -eta * np.sum(delta_mat, axis = 1)

# Update weights
def update_weights(W_mat, w_mat, eta, change_W, change_w):
    return np.add(W_mat, change_W), np.add(w_mat, change_w)
def update_thresholds(theta_vec, Theta, change_theta_vec,
change_Theta):
    return np.add(theta_vec, change_theta_vec), Theta + change_Theta

# Classification error
def calc_C(t_vec, O_vec):
    tmp = np.sum( np.absolute( np.subtract( np.sign(O_vec),
t_vec ) ) ) /(2*t_vec.shape[0])
    return tmp
def class_error_validation(x_val, t_val, w_mat, W_mat, theta_vec,
Theta):
        b_mat = calc_local_field_b_mat(x_val, w_mat, theta_vec)
        V_mat = calc_neurons(b_mat)
        B_vec = calc_local_field_B(V_mat, W_mat, Theta)
        O_vec = calc_output(B_vec)
        return calc_C(t_val, O_vec), O_vec

# Running the network
def run_training(M1, x_mat, t_vec, x_val, t_val, epochs, eta,
train_frac):
    # Initialize
    W_mat = init_weights_W(M1)
    w_mat = init_weights_w(M1)
```

```
    theta_vec, Theta = init_thresholds(M1)
    val_C = np.zeros(epochs)
    best_val_C = np.inf
    # Training loop
    for ep in range(epochs):
        for it in range(x_mat.shape[0]):
            indices = np.random.choice(range(x_mat.shape[0]), size =
int (x_mat.shape[0]*train_frac))
            x_it = np.copy(x_mat[indices,:])
            t_it = np.copy(t_vec[indices])
            b_mat = calc_local_field_b_mat(x_it, w_mat, theta_vec)
            V_mat = calc_neurons(b_mat)
            B_vec = calc_local_field_B(V_mat, W_mat, Theta)
            O_vec = calc_output(B_vec)
            Delta_vec = calc_Delta(t_it, O_vec, B_vec)
            change_W_vec = calc_change_W_vec(eta, Delta_vec, V_mat)
            change_Theta = calc_change_Theta(Delta_vec, eta)
            delta_mat = calc_delta_mat(Delta_vec, W_mat, b_mat)
            change_w_mat = calc_change_w_mat(eta, delta_mat, x_it)
            change_theta_vec = calc_change_theta_vec(eta,delta_mat)
            W_mat, w_mat = update_weights(W_mat, w_mat, eta,
change_W_vec, change_w_mat)
            theta_vec, Theta = update_thresholds(theta_vec, Theta,
change_theta_vec, change_Theta)

        # Classification error validation set
        val_C[ep], output = class_error_validation(x_val,t_val,
w_mat, W_mat, theta_vec, Theta)
        if (val_C[ep] < best_val_C):
            best_val_C = val_C[ep]
            best_w = w_mat
            best_W = W_mat
            best_t = theta_vec
            best_T = Theta
    return val_C, best_w, best_W, best_t, best_T

### Calculations ###

# Load data
colnames = ['x1', 'x2', 't']
train = pd.read_csv('training_set.csv', names = colnames)
validation = pd.read_csv('validation_set.csv', names = colnames)
train_norm, validation_norm = normalize(train.to_numpy(),
validation.to_numpy())
x_mat = train_norm[:,:2]
t_vec = train_norm[:,2]
x_val = validation_norm[:,:2]
t_val = validation_norm[:,2]

# Run training
M1 = 50
eta = 0.005
epochs = 50
train_frac = 1/500
```

```
val_C, best_w, best_W, best_t, best_T = run_training(M1, x_mat,
t_vec, x_val, t_val, epochs, eta, train_frac)
print(val_C)

# Checking output
best_val, output = class_error_validation(x_val, t_val, best_w,
best_W, best_t, best_T)
print(best_val)

# Plotting result
import matplotlib.pyplot as plt
plt.scatter(x_val[:,0], x_val[:,1],s = 10, c = np.sign(output))
plt.scatter(x_val[:,0], x_val[:,1],s = 10, c = t_val)

# Save as CSV
pd.DataFrame(best_w).to_csv('w1.csv', header = False, index = False)
pd.DataFrame(best_W).to_csv('w2.csv', header = False, index = False)
pd.DataFrame(best_t).to_csv('t1.csv', header = False, index = False)
best_T.tofile('t2.csv',sep=',')
```