```python
import numpy as np
import numpy.random as rd
from progress.bar import FillingSquaresBar

N = 2
Out = 1
M1 = 20
N_EPOCHS = 1000
LR = 0.005


def inertia(t):
    e = 1 - np.exp(- t * 1e-10)
    return e


def load_data(URL_train, URL_test):
    return np.loadtxt(open(URL_train, "rb"), delimiter=",", skiprows=1), np.loadtxt(open(URL_test, "rb"), delimiter=",",
                                                                                    skiprows=1)


def standardize_data(train, validate):
    inputs = train[:, :-1]
    means = np.mean(inputs, axis=0)
    stds = np.std(inputs, axis=0)
    train[:, :-1] = (train[:, :-1] - means) / stds
    validate[:, :-1] = (validate[:, :-1] - means) / stds


def init_network():
    w = rd.normal(loc=0., scale=1 / N, size=(M1, N))
    W = rd.normal(loc=0., scale=1 / M1, size=(Out, M1))
    theta = np.zeros((M1, 1))
    THETA = np.zeros((Out, 1))
    return w, W, theta, THETA


d_tanh = np.vectorize(lambda x: 1. - (np.tanh(x)) ** 2)

def feed_forward(x, w1, w_out, tht1, tht_out):

    h_b = np.matmul(w1, x) - tht1
    h_v = np.tanh(h_b)

    o_b = np.matmul(w_out, h_v) - tht_out
    o_v = np.tanh(o_b)
    return h_b, h_v, o_b, o_v


def energy(data, w1, w_out, tht1, tht_out):
    n_data = data.shape[0]
    out = np.zeros((n_data, 1))
    for p in range(n_data):
        p_mu = data[p]
        x_mu = p_mu[:2].reshape((N, 1))
        _, _, _, out[p] = feed_forward(x_mu, w1, w_out, tht1, tht_out)
    t = data[:, 2].reshape((n_data,1))
    diff = (t - out)
    return 0.5 * np.sum(diff ** 2)


def c_error(data_val, w1, w_out, tht1, tht_out):
    sum = 0
    p_val = data_val.shape[0]
    for p in range(p_val):
        p_mu = data_val[p]
        x_mu = p_mu[:2].reshape((N, 1))
        t_mu = p_mu[2]
        _, _, _, v_out = feed_forward(x_mu, w1, w_out, tht1, tht_out)
        v_out = np.sign(v_out)
        sum += np.abs( v_out - t_mu )
    return (1 / (2 * p_val)) * sum


def train_network(learn, validate):
    n_learn = learn.shape[0]
    w, W, theta, THETA = init_network()
    print("\nC_ERROR = {0}\n\n".format(c_error(validate, w, W, theta, THETA)))
    e = energy(validate, w, W, theta, THETA)

    Dw, DW = 0, 0
    t = 0
    for i_epoch in range(N_EPOCHS):
        bar = FillingSquaresBar("Epoch " + str(i_epoch) + ": ", max=15*n_learn)
        err_s = 0

        for p in range(15*n_learn):
            # Choose random pattern
            p_mu = learn[rd.randint(0, n_learn)]
            x_mu = p_mu[:2].reshape((N, 1))
            t_mu = p_mu[2]

            # FEED FORWARD
            h_b, h_v, o_b, o_v = feed_forward(x_mu, w, W, theta, THETA)

            # BACKPROPAGATION

            ERR = (t_mu - o_v) * d_tanh(o_b)
            DW = LR * ERR * h_v.T + inertia(t) * DW

            err = np.matmul(W.T, ERR) * d_tanh(h_b)
            Dw = LR * np.matmul(err, x_mu.T) + inertia(t) * Dw

            # ADJUST WEIGHTS AND THRESHOLDS

            W += DW
            err_s += DW
            THETA -= LR * ERR

            w += Dw
            theta -= LR * err

            t += 1
            bar.next()
        bar.finish()
        ce = c_error(validate, w, W, theta, THETA)
        e_p = e
        # e = energy(validate, w + Dw, W + DW, theta - (LR * err), THETA - (LR * ERR))
        e = energy(validate, w, W, theta, THETA)
        delta_e = e - e_p
        print("\nC_ERROR = {0} , deltaH = {1}, H = {2}\n\n".format(ce, delta_e, e))
        if ce < 0.118:
            break
    return w, W, theta, THETA


data_train, data_val = load_data("training_set.csv", "validation_set.csv")
standardize_data(data_train, data_val)
w, W, theta, THETA = train_network(data_train[:-3500],
data_val)

np.savetxt("w1.csv", w, delimiter=",")
np.savetxt("w2.csv", W, delimiter=",")
np.savetxt("t1.csv", theta, delimiter=",")
np.savetxt("t2.csv", THETA, delimiter=",")

```