%

% Neural Network MATLAB Code

%

% Training and testing of neural network for the prediction of self-diffusion in binary mixtures

% using the full ANN1 model as detailed in:

%

% Prediction of Self-Diffusion in Binary Fluid Mixtures Using Artificial Neural Networks

% Joshua Allers, Jane Keth, Todd M. Alam (2022) J. Phys. Chem B., XXXXXXX jp-2022-01723f.R1

%

% Code written by Joshua Allers, Sandia National Laboratories

% Last modification: May 2022

%

clear

clc

close all

format long

rng(50)

% Pull in data from excel

[data, strings] = xlsread('Binary\_mean\_v4.xlsx');

substance\_1 = strings(2:end, 1);

substance\_2 = strings(2:end, 2);

inputs = data(:, 4:21);

y = data(:, 22)';

n = size(inputs, 2);

m = length(y);

% Scale Inputs and Outputs

y = log10(y);

[x\_scale, PS] = mapstd(inputs');

y\_scale = (y - min(y)) ./ (max(y) - min(y));

% Separate out validation and test set

indices = 1:m;

shuffled\_ind = indices(randperm(m));

train\_split = ceil(0.7 \* m);

val\_split = train\_split + floor((m - train\_split)/2);

test\_split = val\_split + floor((m - train\_split)/2);

train\_ind = shuffled\_ind(1:train\_split);

val\_ind = shuffled\_ind((train\_split + 1):val\_split);

test\_ind = shuffled\_ind((val\_split + 1):test\_split);

%% ANN Optimization

n = 50;

m = 50;

mu = linspace(0.001, 0.009, 81);

nodes\_1 = 5:5:100;

nodes\_2 = [5 10 15 20 25 30 35 40 45 50];

error\_matrix = zeros(m, n);

for i = 1:n

i

val\_mse = zeros(1, m);

for j = 1:m

rng(i)

net = feedforwardnet(25); % Number of nodes in hidden layer

net.trainFcn = 'trainlm'; % Define optimization method

net.divideFcn = 'divideind';

net.divideParam.trainInd = train\_ind;

net.divideParam.valInd = val\_ind;

net.divideParam.testInd = [];

net.layers{1}.transferFcn = 'tansig';

net.inputs{1}.processFcns = {};

net.outputs{1}.processFcns = {};

net.trainParam.mu = 0.001;

net.trainParam.showWindow = 0; % Suppresses GUI

[net, tr] = train(net, x\_scale, y\_scale);

% Define Train, Validation, and Test sets

x\_val = x\_scale(:, tr.valInd);

y\_val = y\_scale(tr.valInd);

% Test Neural Net on All Sets

NN\_val = net(x\_val);

% Calculate Error for Test Set

len = length(y\_val);

val\_mse(j) = sum((y\_val - NN\_val) .^ 2)/len;

end

error\_matrix(:, i) = val\_mse;

end

min\_err = min(error\_matrix(:))

[row, col] = find(error\_matrix == min\_err);

i\_var = col

j\_var = row

%% Assessment on Test Set

rng(31)

net = feedforwardnet([50 25]); % Number of nodes in hidden layer

net.trainFcn = 'trainlm'; % Define optimization method

net.divideFcn = 'divideind';

net.divideParam.trainInd = [train\_ind val\_ind];

net.divideParam.valInd = [];

net.divideParam.testInd = test\_ind;

net.layers{1}.transferFcn = 'tansig';

net.inputs{1}.processFcns = {};

net.outputs{1}.processFcns = {};

net.trainParam.epochs = 80;

net.trainParam.max\_fail = 6;

net.trainParam.mu = 0.001;

net.trainParam.showWindow = 1; % Suppresses GUI

[net, tr] = train(net, x\_scale, y\_scale);

% Define Train, Validation, and Test sets

x\_train = x\_scale(:, tr.trainInd);

y\_train = y\_scale(tr.trainInd);

x\_test = x\_scale(:, tr.testInd);

y\_test = y\_scale(tr.testInd);

% Test Neural Net on All Sets

NN\_test = net(x\_test);

NN\_train = net(x\_train);

sub\_1\_train = substance\_1(tr.trainInd);

sub\_2\_train = substance\_2(tr.trainInd);

sub\_1\_test = substance\_1(tr.testInd);

sub\_2\_test = substance\_2(tr.testInd);

sub\_1 = {[sub\_1\_train; sub\_1\_test]};

sub\_2 = {[sub\_2\_train; sub\_2\_test]};

sub\_1 = vertcat(sub\_1{:});

sub\_2 = vertcat(sub\_2{:});

% Revert to Raw Scale

y\_train\_raw = y\_train \* (max(y) - min(y)) + min(y);

y\_test\_raw = y\_test \* (max(y) - min(y)) + min(y);

NN\_train\_raw = NN\_train \* (max(y) - min(y)) + min(y);

NN\_test\_raw = NN\_test \* (max(y) - min(y)) + min(y);

figure()

hold on

scatter(y\_train\_raw, NN\_train\_raw, 'b')

scatter(y\_test\_raw, NN\_test\_raw, 'r')

plot(y, y, '-k')

% ylim([0 3.813466984378636e-04])

% xlim([0 3.813466984378636e-04])

ylabel('Predicted Diffusion')

xlabel('Experimental Diffusion')

legend('Training Data', 'Testing data', 'Location', 'northwest')

hold off

y\_train\_raw = 10 .^ y\_train\_raw;

y\_test\_raw = 10 .^ y\_test\_raw;

NN\_train\_raw = 10 .^ NN\_train\_raw;

NN\_test\_raw = 10 .^ NN\_test\_raw;

% Calculate Error for Test Set

test\_len = length(y\_test);

train\_len = length(y\_train);

test\_rmse = sqrt(sum((y\_test\_raw - NN\_test\_raw) .^ 2)/test\_len);

train\_rmse = sqrt(sum((y\_train\_raw - NN\_train\_raw) .^ 2)/train\_len);

rel\_test\_err = (y\_test\_raw - NN\_test\_raw) ./ y\_test\_raw;

rel\_train\_err = (y\_train\_raw - NN\_train\_raw) ./ y\_train\_raw;

train\_aad = (sum(abs(NN\_train\_raw - y\_train\_raw)./y\_train\_raw)/train\_len) \* 100;

test\_aad = (sum(abs(NN\_test\_raw - y\_test\_raw)./y\_test\_raw)/test\_len) \* 100;

% Test Neural Net on all Data and Calculate R

R\_input = [NN\_train\_raw NN\_test\_raw];

R\_target = [y\_train\_raw y\_test\_raw];

R\_NN = (sum((R\_target-mean(R\_target)).\*(R\_input-mean(R\_input))))./(sqrt(sum((R\_target-mean(R\_target)).^2).\*sum((R\_input-mean(R\_input)).^2)));

all\_aad = (sum(abs(R\_input - R\_target)./R\_target)/m) \* 100;

% Find Outliers

diff = abs(R\_input - R\_target)./R\_target;

[m, ind] = maxk(diff, 10);

for u = 1:10

outlier\_1 = sub\_1{ind(u)};

outlier\_2 = sub\_2{ind(u)};

sprintf('Outlier %d comes from %s and %s in index %d with a deviation of %.3f', u, outlier\_1, outlier\_2, ind(u), m(u))

end