CIS 520, Machine Learning, Fall 2018: Assignment 4

Simran Arora

Collaborator:

Shubhankar Patnakar

1 Convolutional Neural Network

1. Number of weights = (108)(162)(3) = 52488 weights for a single neuron in the first hidden layer because the dimensions are $108x162$ and there are 3 color channels.

2. Number of weights = (18)(6)(3) = 324 weights for a single neuron in the convolutional layer if we use a filter of size 18x6x3.

3. Number of neurons = (16)(27)(3) = 1296. The output size is defined by the equation:

$$Output = \frac{W - F + 2P}{S} + 1$$

In the above equation, W is the input volume size, F is the receptive field size, S is the stride, and P is the padding. In our case, for the $x - direction$, the input size is $W = 108$, $S = 6$, $P = 0$, $F = 18$, and thus the output size in the $x - direction$ is $\frac{108-18+2(0)}{6} + 1 = 15 + 1 = 16$. For the $y - direction$, the input size is $W = 162$, $S = 6$, $P = 0$, $F = 6$, and thus the output size in the $y - direction$ is $\frac{162-6+2(0)}{6} + 1 = 26 + 1 = 27$. The depth remains the same at 3.

4. To calculate the element in row 0 and column 0 from output filter 1, we take the following steps. First, element wise multiply the upper left 3x3 matrix of the input image by filter 1. The upper left 3x3 matrix of the input image is:

$$\begin{bmatrix} 4 & 4 & 1 \\ 2 & 2 & 4 \\ 5 & 1 & 2 \end{bmatrix}$$

Then using this upper left 3x3 matrix of the input image and multiplying element-wise by filter 1, and summing the products, we get the sum of:

$$(4)(-1) + (4)(0) + (1)(1) + (2)(-3) + (2)(0) + (4)(2) + (5)(1) + (1)(1) + (2)(2) = 9$$

Given that the bias for filter 1 and filter 2 are 0 we calculate the element in (row 0, column 0) of the output filter 1 to be $9 + 0 = 9$. Next, since the stride is 1, we shift the 3x3 matrix that we consider in the input image by 1 column or 1 row to get the next value in the output filter. Using this process, the following output filters are obtained.

$$\text{output filter 1} = \begin{bmatrix} 9 & 8 & 2 \\ 4 & 16 & 9 \\ 16 & 22 & 3 \end{bmatrix} \quad \text{output filter 2} = \begin{bmatrix} 20 & 8 & -6 \\ 7 & -1 & 19 \\ 24 & 7 & 0 \end{bmatrix}$$

Java Code for Calculating CNN Output Filter Matrices

```
public class CNN {

public static void main(String[] args) {

Integer[][] filter1 = new Integer[3][3];
filter1[0] = new Integer[]{-1, 0, 1};
filter1[1] = new Integer[]{-3, 0, 2};
filter1[2] = new Integer[]{1, 1, 2};

Integer[][] filter2 = new Integer[3][3];
filter2[0] = new Integer[]{2, -2, 1};
filter2[1] = new Integer[]{-1, 0, 2};
filter2[2] = new Integer[]{3, -2, 0};

Integer[][] input = new Integer[5][5];
input[0] = new Integer[]{4, 4, 1, 3, 2};
input[1] = new Integer[]{2, 2, 4, 1, 2};
input[2] = new Integer[]{5, 1, 2, 5, 1};
input[3] = new Integer[]{2, 1, 5, 2, 4};
input[4] = new Integer[]{4, 3, 4, 5, 1};

Integer[][] output1 = new Integer[3][3];
Integer[][] output2 = new Integer[3][3];

//each slot in output array
System.out.println("Output 1:");
for (int i = 0; i < filter1.length; i ++) {
for (int j = 0; j < filter1.length; j ++) {
int sum = 0;
//filter iter
for (int k = 0; k < filter1.length; k++) {
for (int l = 0; l < filter1.length; l++) {
int n = k + i;
int m = l + j;
sum = sum + filter1[k][l]*input[n][m];
}
}
output1[i][j] = sum;
}
}

for (int i = 0; i < output1.length; i ++) {
for (int j = 0; j < output1.length; j ++) {
System.out.print(output1[i][j] + " ");
}
System.out.println("");
}

//each slot in output array
System.out.println("");
System.out.println("Output 2:");
for (int i = 0; i < filter1.length; i ++) {
for (int j = 0; j < filter1.length; j ++) {
```

```
int sum = 0;
//filter iter
for (int k = 0; k < filter2.length; k++) {
for (int l = 0; l < filter2.length; l++) {
int n = k + i;
int m = l + j;
sum = sum + filter2[k][l]*input[n][m];
}
}
output2[i][j] = sum;
}
}

for (int i = 0; i < output2.length; i ++) {
for (int j = 0; j < output2.length; j ++) {
System.out.print(output2[i][j] + " ");
}
System.out.println("");
}
}
}
```

2 Optimization and Lagrangian Duality

1. $\mathcal{L}(x_1, x_2, \nu) = (\frac{1}{2})(x_1^2 + x_2^2) + \nu(3x_1 + 2x_2 - 1)$

2. $$\frac{dL}{dx_1} = x_1 + 3\nu \rightarrow x_1 = -3\nu$$
$$\frac{dL}{dx_2} = x_2 + 2\nu \rightarrow x_2 = -2\nu$$
$\therefore \phi(\nu) = \frac{1}{2}((-3\nu)^2 + (-2\nu)^2) + \nu(3(-3\nu) + 2(-2\nu) - 1) = \frac{13}{2}\nu^2 + (-13\nu^2) - \nu = -\frac{13}{2}\nu^2 - \nu$

3. $$\frac{d\phi(\nu)}{d\nu} = -13\nu - 1 = 0$$
$$\nu^* = \frac{-1}{13}$$
$\therefore x_1^* = \frac{3}{13}, x_2^* = \frac{2}{13}$ if we plug in the obtained $v*$. We can see that the x_1* and x_2* pass the constraint as $(3\frac{3}{13} + 2\frac{2}{13}) = \frac{9}{13} + \frac{4}{14} = 1$.

3 Kernel Functions

We define $k : XxX \to \mathbb{R}$ as positive semidefinite if $\sum_i \sum_j c_i c_j k(x_i, x_j) \geq 0 = \vec{c}' K \vec{c}$.

1. $\phi(x) = \sqrt{c}\phi_2$: Using the above have that $\vec{c}^T c K_1 \vec{c} = c\vec{c}' K \vec{c} \geq 0$ because $c \geq 0$. Thus we know that this is a valid kernel. Further $\phi(x) = \sqrt{c}\phi_2$. To prove this we know that $K_2(x, x') = \phi_2(x)^T \phi_2(x')$ thus $(\sqrt{c}\phi_2(x)^T)(\sqrt{c}\phi_2(x')) = (\sqrt{c})^2 \phi_2(x)^T \phi_2(x') = c\phi_2(x)^T \phi_2(x') = cK_2(x, x')$.

2. For $K(x, x') = K_1(x, x') - cK(x, x')$ we can write this as $\phi(x) = \phi_1(x)^T \phi_1(x') - c\phi_2(x)^T \phi_2(x')$. This is not a valid kernel. We know $\sum_i \sum_j c_i c_j K_1(x_i, x_j) \geq 0$ and $\sum_i \sum_j c_i c_j K_2(x_i, x_j) \geq 0$. Thus we have $\sum_i \sum_j c_i c_j K_1(x_i, x_j) - c \sum_i \sum_j c_i c_j K_2(x_i, x_j)$. Since $c \geq 0$, we know $(-c) \leq 0$, and thus $\vec{c}^T K \vec{c} = \vec{c}^T K_1 \vec{c} + (-c)\vec{c}^T K_2 \vec{c}$ will not be ≥ 0 if $(c)\vec{c}^T K_2 \vec{c} \geq vecc^T K_1 \vec{c}$. Thus $K(x, x')$ is not positive semidefinite. Thus the valid mapping does not exist.

3. $\phi(x)\phi(x') = (\phi_1(x)\phi_2(x))\begin{pmatrix} \phi_1(x') \\ \phi_2(x') \end{pmatrix}$ For $K(x, x') = K_1(x, x') + K(x, x')$ we can think of $\phi(x)$ as the concatenation of $\phi_1(x)$ and $\phi_2(x)$. This gives us $\phi(x)\phi(x') = \phi_1(x)^T \phi_1(x') + \phi_2(x)^T \phi_2(x^T) = (\phi_1(x)\phi_2(x))\begin{pmatrix} \phi_1(x') \\ \phi_2(x') \end{pmatrix}$. This gives us a valid kernel because the property $\sum_i \sum_j c_i c_j k(x_i, x_j) \geq 0$ still holds on K

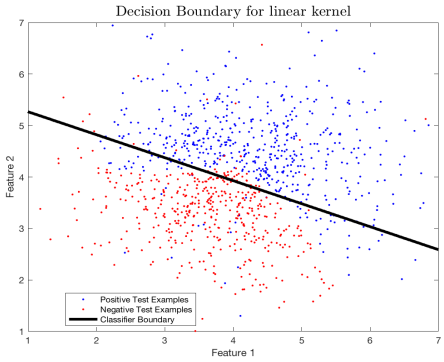
4. We know that $\phi_i(x)$ corresponds to the i^{th} feature of vector x. We have the definition that $K(x, x^T) = \sum_{i=1}^{n} \phi_i(x)\phi_i(x)$. Thus $K_1(x, x^T)K_2(x, x^T) = \sum_{i=1}^{d_1} \phi_{1i}(x)\phi_{1i}(x^T) \sum_{j=1}^{d_2} \phi_{2j}(x)\phi_{2j}(x^T)$ This equals $\sum_{i=1}^{d_1} \sum_{j=1}^{d_2} \phi_{1i}(x)\phi_{2j}(x)\phi_{1i}(x^T)\phi_{2j}(x^T)$. This effectively multiplies out the feature map of K_1 and K_2. We can define $\phi(x)$ as the sum of $d_1 x d_2$ items, such that item (i, j) the value $\phi_{1i}(x)\phi_{1i}(x^T)\phi_{2j}(x)\phi_{2j}(x^T)$. Let $m = (d_1)(d_2)$. Then for $k \in [1..m]$ and a given pair $i \in S_1 = [1..d_1]$, $j \in S_2 = [1..d_2]$, let $\phi_m(x) = \phi_{1i}(x)\phi_{2j}(x)$. We can then define our $\phi(x)$ as the sum of the set of all possible $\phi_m(x)$, or in other words consider the Cartesian Product of $S_1 x S_2$ and for each produced (i, j) pair, get sum the $\phi_m(x)$ to produce ϕ. The product of the two is also positive semidefinite, and thus, the product of kernels produces a valid kernel, with the given ϕ.

5. $\phi(x) = \phi_1(f(x))$ Kernel 1 is valid for all vectors for all x in our space X. Since $f : X \to X$, then we know that the function doesn't leave the vector space. Since ϕ_1 is defined already for these same vectors X as the domain, we can use $\phi_1(f(x))$ as $\phi(x)$.

4 SVM and Neural Nets: Programming Exercise

4.1 SVM on synthetic data

1. (a) report value of C and insert plot of decision boundary for linear kernel
   C = 10000

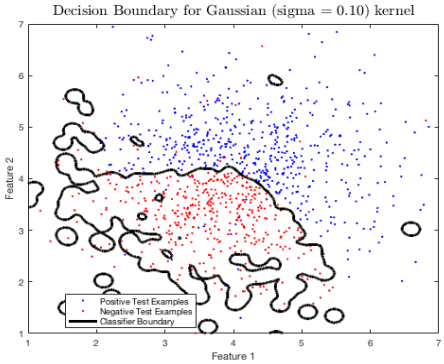


   (b) Cross Validation Error = 0.0840

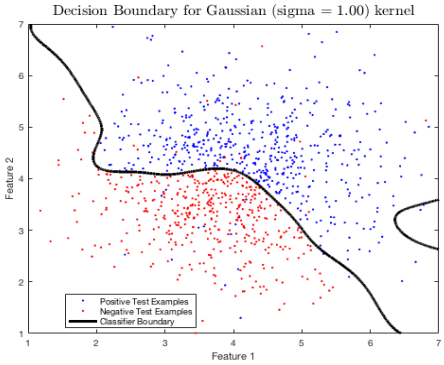
   (c) Training Error = 0.0860

   (d) Test Error = 0.1090

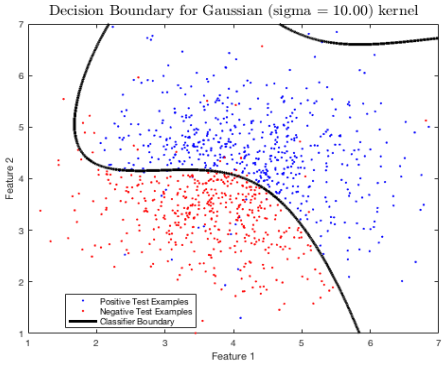
2. (a) report value of C and insert plot of decision boundary for the rbf-0.1
   C = 1



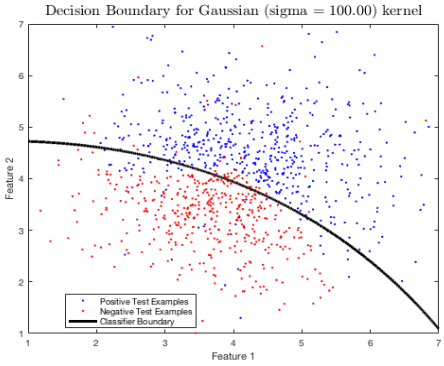
   (b) report value of C and insert plot of decision boundary for the rbf-1
   C = 100



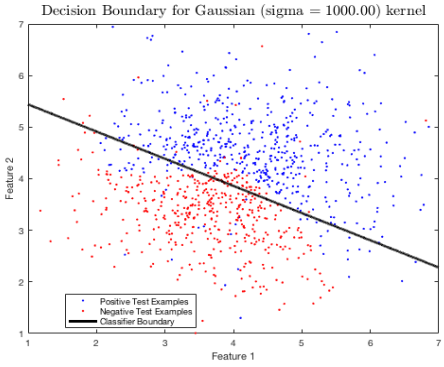
(c) report value of C and insert plot of decision boundary for the rbf-10
C = 100000



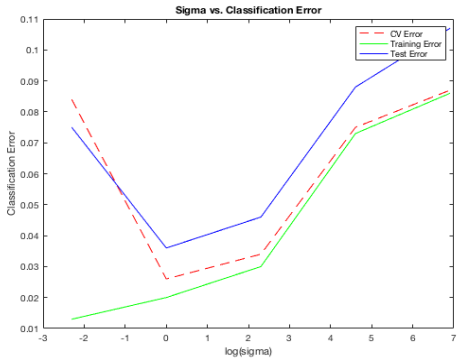
(d) report value of C and insert plot of decision boundary for the rbf-100
C = 100000



(e) report value of C and insert plot of decision boundary for the rbf-1000
C = 100000



(f) insert line plot of errors



(g) σ that achieves lowest test error is $\sigma = 1$.

(h) σ that achieves lowest cross-validation error is $\sigma = 1$

3. (a) Absolute difference between test errors is 0

   (b) The result of 0 difference between test errors makes sense because the cross validation error is a subtest of the overall test error. This indicates that the cross validation test error generalizes well. The cross-validation is used to estimate what the test error would be and we see that this provides a good procedure for selecting the σ parameter based on our result.

4.2 SVM and Neural Nets on Breast Cancer Dataset

1. Summary of SVM results:

   In the result for SVM with the linear kernel, the lowest cross-validation error was achieved with a penalty value of $C = 1000$. The error value was observed as $CVerror = 0.0314$ and $TrainingError = 0.1000$. In the result for SVM with the RBF kernel, both $(\sigma = 100, C = 10)$ and $(\sigma = 1000, C = 1000)$ gave the same cross-validation error result. The CV error in both of these cases was $CVerror = 0.0267$ and the training error in both of these cases was $TrianingError = 0.0267$. The RBF kernel gives better performance as indicated by the lower $CVerror$ achieved, as compared to the linear kernel. Because there are two pairs that provided the low error result for the RBF result, we can decide between them by picking the more complex alternative of $\sigma = 1000$ and $C = 1000$. The downside of higher complexity is that if the complexity is too high, then the test error can start to increase again, as the variance on test data increases, however in our result, this does not happen, and so we can choose the more complex result. Overall with higher complexity, bias tends to decrease. In our autograder results, the SVM test error was 0.024096. This is very close to the ovserved $CVerror = 0.0267$ in the RBF result.

2. Summary of NN results:

   To analyze the created neural network, we tested with both the ReLU and sigmoidal activation functions. Within each case, to determine the best C value, we performed cross-validation tests with different candidate values of C, used the C value yielding the lowest $CVerror$ to determine the best value. For our tests, between different repetitions of the NN, we obtained different results for the best C value. This is because the neural network is initialized randomly. To yield consistent results between tests, we can add a seed to the random generator to produce the same stream of random values between repetitions. Even given the different choices for best C, the $CVerror$ values outputted were very consistent for the chosen C value. In our results, the $CVerror$ for sigmoid and ReLU were very similar and consistently approximately 0.0687. In the autograder results, the NN test error was 0.072289. The $training - error$ was approximately 0.0667 for ReLU and 0.0867 for $sigmoid$. Looking at the training error, sigmoid consistently has higher training error by approximately 0.02.

   Because both activation functions yielded approximately the same $CVerror$, we decided to evaluate the better choice by the amount of time required to train the NN for the two activation functions.

We found that training the neural networks with the sigmoidal function required more time than training the neural network with the ReLU. This makes sense to us because weights in a neural network are learned using back-propagation and gradient descent and it is computationally easier to compute gradients of the ReLU. Therefore, a ReLU activated neural network is a better model compared to a sigmoid activated NN. The choice of C for the ReLU does not seem to affect the CV error in a discernible way. Values of CV error for different C values remain very close to each other.

4.3 Appendix

Code for 4.2.1 lin-kern-cancer.m

```
clc; close all; clear;


%% Pick a C using CV

C = [1, 10, 10^2, 10^3, 10^4, 10^5];
error = zeros(5, size(C, 2));
cd Breast-Cancer/CrossValidation
% make sure to be in the CrossValidation folder for the loop to work
for i = 1:5
    newFolder = sprintf('Fold%d', i);
    cd(newFolder);
    load('cv-train.mat');
    load('cv-test.mat');
    X = cv_train(:,1:9);
    Y = cv_train(:,10);
    X_test = cv_test(:,1:9);
    Y_test = cv_test(:,10);
    for j = 1:size(C, 2)
        SVMModel = fitcsvm(X, Y, 'BoxConstraint', C(j), 'KernelFunction', 'linear');
        labels_test = predict(SVMModel, X_test);
        % error(i,j) = sum((labels_train - Y_test).^2)/(numel(Y_test));
        % error(i,j) = sum(labels_test ~= Y_test)/(numel(Y_test));
        error(i,j) = classification_error(labels_test, Y_test);
    end
    cd .. % return to CrossValidation from Foldi
end
error_av_CV = mean(error, 1); % this is the cross-validation error
% 10^4 looks to be the smallest
[~, idx] = min(error_av_CV);
error_CV = error_av_CV(idx);
C_choice = C(idx);

%% Use chosen C to train and test
clearvars -except error_av_CV C_choice

cd .. % return to Breast-Cancer
load('trainingdata.mat');
load('testdata.mat');
% X = train(:,1:9);
X = train_inputs;
% Y = train(:,10);
Y = train_labels;
% X_test = test(:,1:9);
```

```
X_test = test_inputs;
% Y_test = test(:,10);

SVMModel = fitcsvm(X, Y, 'BoxConstraint', C_choice, 'KernelFunction', 'linear');
labels_train = predict(SVMModel, X); % for training error
% train_error = sum((labels_train - Y).^2)/(numel(Y));
% train_error = sum(labels_train ~= Y)/(numel(Y));
train_error = classification_error(labels_train, Y);

labels_test = predict(SVMModel, X_test); % for testing error
% test_error = sum((labels_test - Y_test).^2)/(numel(Y_test));
% test_error = sum(labels_test ~= Y_test)/(numel(Y_test));

%% Visualize boundary
% cd .. % return to hw4-kit
% decision_boundary_SVM(X_test, Y_test, SVMModel, 1000, 'linear');

%% Helper Function

function err = classification_error(y_pred, y_true)
% This function computes the classification error for the predicted labels
% with respect to the ground truth. The returned error value is a real number
% between 0 and 1 (fraction of misclassications).

% y_true: vector of true labels (each label +1/-1)
% y_pred: vector of predicted labels (each prediction +1/-1)
% err: classification error (fraction of misclassifications)

err = 1 - length(find(y_pred == y_true)) / length(y_true);
end
```

Code for 4.2.2: neural-net.m

```
clc; close all; clear;

C = [0, 10^-4, 10^-3, 10^-2, 10^-1, 1];
error = zeros(5, size(C, 2));
cd Breast-Cancer/CrossValidation
% make sure to be in the CrossValidation folder for the loop to work

%% Activation Function: Sigmoid

for i = 1:5
    newFolder = sprintf('Fold%d', i);
    cd(newFolder);
    load('cv-train.mat');
    load('cv-test.mat');
    X = cv_train(:,1:9); % first 9 columns are features
    Y = cv_train(:,10); % 10-th column is labels
    Y(Y == -1) = 0;
    X_test = cv_test(:,1:9);
    Y_test = cv_test(:,10);
    Y_test(Y_test == -1) = 0;
    for j = 1:size(C, 2)
        net = patternnet(10, 'trainrp', 'crossentropy');
```

```
        net.layers{1}.transferFcn = 'logsig';
        net.performParam.regularization = C(j);
        net = trainrp(net, X', Y');
        labels_pred = net(X_test');
        % labels_pred(labels_pred < 0.5) = 0;
        labels_pred(labels_pred < 0.5) = -1;
        labels_pred(labels_pred > 0.5) = 1;
        labels_pred(labels_pred == 0.5) = 1;
        % error(i,j) = sum(labels_pred' ~= Y_test)/(numel(Y_test));
        error(i,j) = classification_error(labels_pred', Y_test);
    end
    cd .. % return to CrossValidation from Foldi
end
error_av_CV_sig = mean(error, 1); % this is the cross-validation error
% 10^4 looks to be the smallest
[error_min_sig, idx] = min(error_av_CV_sig);
C_choice_sig = C(idx);


%% Activation Function: ReLU

error = zeros(5, size(C, 2)); % reset error matrix
for i = 1:5
    newFolder = sprintf('Fold%d', i);
    cd(newFolder);
    load('cv-train.mat');
    load('cv-test.mat');
    X = cv_train(:,1:9); % first 9 columns are features
    Y = cv_train(:,10); % 10-th column is labels
    Y(Y == -1) = 0;
    X_test = cv_test(:,1:9);
    Y_test = cv_test(:,10);
    Y_test(Y_test == -1) = 0;
    for j = 1:size(C, 2)
        net = patternnet(10, 'trainrp', 'crossentropy');
        net.layers{1}.transferFcn = 'poslin';
        net.performParam.regularization = C(j);
        net = trainrp(net, X', Y');
        labels_pred = net(X_test');
        % labels_pred(labels_pred < 0.5) = 0;
        labels_pred(labels_pred < 0.5) = -1;
        labels_pred(labels_pred > 0.5) = 1;
        labels_pred(labels_pred == 0.5) = 1;
        % error(i,j) = sum(labels_pred' ~= Y_test)/(numel(Y_test));
        error(i,j) = classification_error(labels_pred', Y_test);
    end
    cd .. % return to CrossValidation from Foldi
end
error_av_CV_relu = mean(error, 1); % this is the cross-validation error
% 10^4 looks to be the smallest
[error_min_relu, idx] = min(error_av_CV_relu);
C_choice_relu = C(idx);

%% Predictions on Test Inputs
```

```
clearvars -except error_av_CV_sig error_min_sig C_choice_sig ...
    error_av_CV_relu error_min_relu C_choice_relu

cd .. % return to Synthetic
load('trainingdata.mat');
load('testdata.mat');
cd .. % return to hw4-kit

% X = train(:,1:9);
X = train_inputs;
% Y = train(:,10);
Y = train_labels;
% X_test = test(:,1:9);
X_test = test_inputs;
% Y_test = test(:,10);

net = patternnet(10, 'trainrp', 'crossentropy');
net.layers{1}.transferFcn = 'logsig';
net.performParam.regularization = C_choice_sig; % chosen C for sigmoid act.
net = trainrp(net, X', Y');
labels_pred_sig = net(X_test');
% labels_pred_sig(labels_pred_sig < 0.5) = 0;
labels_pred_sig(labels_pred_sig < 0.5) = -1;
labels_pred_sig(labels_pred_sig > 0.5) = 1;
labels_pred_sig(labels_pred_sig == 0.5) = 1;

% for training error
labels_train_sig = net(X'); % run neural net on training data
% labels_train_sig(labels_train_sig < 0.5) = 0;
labels_train_sig(labels_train_sig < 0.5) = -1;
labels_train_sig(labels_train_sig > 0.5) = 1;
labels_train_sig(labels_train_sig == 0.5) = 1;
% train_error_sig = sum(labels_train_sig' ~= Y)/(numel(Y));
train_error_sig = classification_error(labels_train_sig', Y);

net = patternnet(10, 'trainrp', 'crossentropy');
net.layers{1}.transferFcn = 'poslin';
net.performParam.regularization = C_choice_relu; % chosen C for ReLU act.
net = trainrp(net, X', Y');
labels_pred_relu = net(X_test');
% labels_pred_relu(labels_pred_relu < 0.5) = 0;
labels_pred_relu(labels_pred_relu < 0.5) = -1;
labels_pred_relu(labels_pred_relu > 0.5) = 1;
labels_pred_relu(labels_pred_relu == 0.5) = 1;

% for training error
labels_train_relu = net(X'); % run neural net on training data
% labels_train_relu(labels_train_relu < 0.5) = 0;
labels_train_relu(labels_train_relu < 0.5) = -1;
labels_train_relu(labels_train_relu > 0.5) = 1;
labels_train_relu(labels_train_relu == 0.5) = 1;
% train_error_relu = sum(labels_train_relu' ~= Y)/(numel(Y));
train_error_relu = classification_error(labels_train_relu', Y);
```

```
%% Helper Function

function err = classification_error(y_pred, y_true)
% This function computes the classification error for the predicted labels
% with respect to the ground truth. The returned error value is a real number
% between 0 and 1 (fraction of misclassications).

% y_true: vector of true labels (each label +1/-1)
% y_pred: vector of predicted labels (each prediction +1/-1)
% err: classification error (fraction of misclassifications)

err = 1 - length(find(y_pred == y_true)) / length(y_true);
end
```