49275 - A2 - Assignment 2

Neural Networks and Fuzzy Logic (University of Technology Sydney)

Contents:

29/01/2021

Question 1 – Multi-Layer Network

Q1 Part 1

Using the error back propagation training, the calculated next weights w(2) and wHat(2) are:

```
w(2):
    0.0041   -0.2509   -0.3902    0.1110
    0.6453    0.3733   -0.6868    0.8594
    0.5393   -0.3868    0.7724    0.2069

wHat(2):
  Columns 1 through 14

   -0.2118    0.2227    0.4676   -0.1798    0.5863   -0.7785   -0.2855    0.9715   -0.5857   -0.1988    0.7699    0.9723   -0.1715   -0.0415
    0.1927    0.8431   -0.6470    0.3360    0.1790   -0.0547    0.6258   -0.7217   -0.6031    0.3551   -0.2035   -0.0685    0.6929    0.5942
    0.6574    0.3574   -0.7176   -0.2217    0.9050    0.0575    0.3249   -0.7605    0.8211   -0.2152   -0.1713    0.3887   -0.9711    0.4616

  Columns 15 through 26

    0.3531   -0.4724    0.8713    0.3245    0.1653    0.7699    0.3299    0.7739   -0.9908    0.2333    0.3624   -0.1474
    0.6757    0.2797   -0.1768    0.5525    0.2583    0.3360    0.7623   -0.5177    0.3363    0.7873   -0.1106    0.8212
    0.2260   -0.8637    0.0159    0.9290   -0.5001    0.1159   -0.7646    0.2069    0.3329    0.9466   -0.5078   -0.3550
```

Q1 Part 2

After 50 cycles, the final weight matrices are shown as follows:

```
w_Final = w(301)
   -0.4180   -0.7340   -3.1413    0.7084
   -1.0147    2.6563    0.7973    0.8016
    2.5589   -0.9046    1.3918    1.1120

wHat_Final =wHat(301)
  Columns 1 through 14

   -0.0478    0.2501    0.1967   -0.3290    0.4612   -0.3174   -0.5045    1.1207   -0.8288   -0.3239    1.2310    1.4334    0.0716    0.3015
    0.2624    0.6505   -0.5339    0.4632   -0.0448   -0.4216    0.8900   -0.8489    0.0121    0.1313   -0.5704   -0.4354    0.0777    1.0663
    0.8145    0.1428   -0.9059   -0.3081    0.8127    0.6960    0.3979   -0.6741    0.8999   -0.3075    0.4672    1.0272   -1.0499    1.2711

  Columns 15 through 26

    0.5413   -0.5975    0.7462    0.5676   -0.0778    0.6448    0.2048    0.6488   -0.7477   -0.0098    0.2373   -0.2725
    0.3944    0.0559   -0.4006   -0.0627    0.8735    0.1122    0.5385   -0.7415   -0.2789    1.4025   -0.3344    0.5974
    0.3773   -0.9560   -0.0764    0.8502   -0.4213    0.0236   -0.8569    0.1146    0.2541    1.0254   -0.6001   -0.4473
```

The cycle error curve for this training exercise is shown in Figure 1 below.

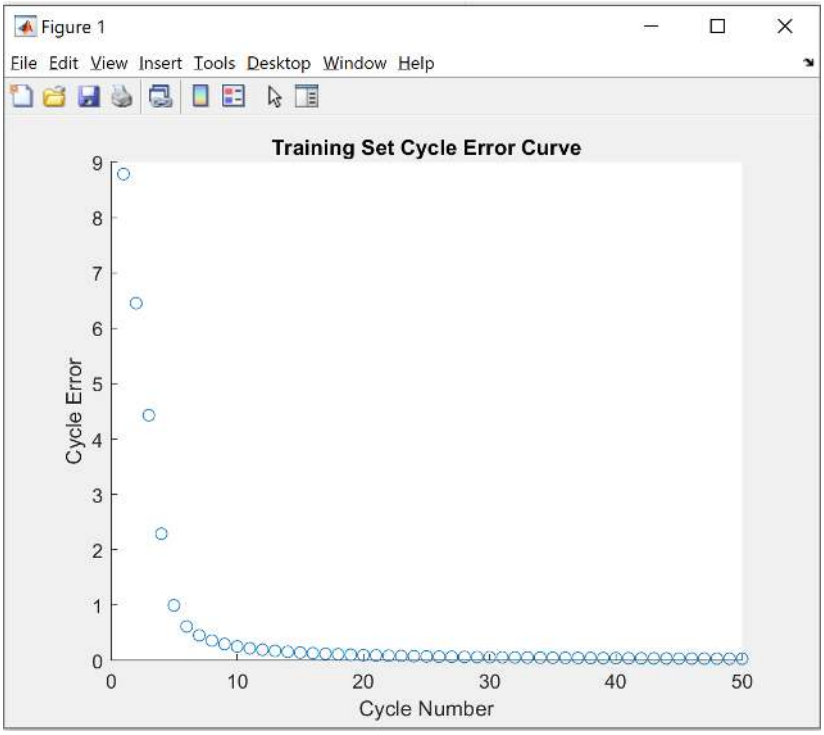


Figure 1: Training Set Cycle Error Curve

Q1 Part 3

i) Calculate the output vector z which is generated from the above input. How would the neural network classify this feature input?

Given the testing number pattern shown in Figure 2 below, the output vector z was:

```
The output vector z for test input
   -0.9169
    0.6380
   -0.8709
```

Therefore, it classified the test input vector belonging to class 2, which is the number '7'.

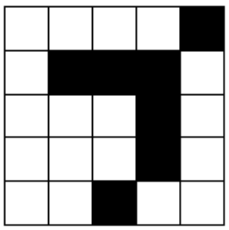


Figure 2: Testing Number Pattern Provided

ii) Suggest one solution if we want to improve the testing classification performance. Please provide a reason and testing result.

One solution to improve the testing classification performance of the model would be to alter the initial learning rate (n) from 0.4 to 0.8. As by changing this can have a significant effect on the calculated error. A larger learning constant (n) will result in a more rapid convergence, however with more risk of overshooting/overfitting the solution. Whilst a smaller value may take too long to train.

Figure 3 below shows the values of the increased learning constant rate (0.8), vs the original learning constant rate (0.4).

```
Output Z (Increased Learning Rate):        Output Z (Original Learning Rate):
   -0.9580                                    -0.9169
    0.7784                                     0.6380
   -0.9115                                    -0.8709
```

Figure 3: Increased LR Vs Original LR

This meant the model still classified the testing output as class 2, but with a higher result, meaning that it was more definite that the testing input belong to that class.

Additionally, the following cycle error curve of the increased learning constant is shown in Figure 4.

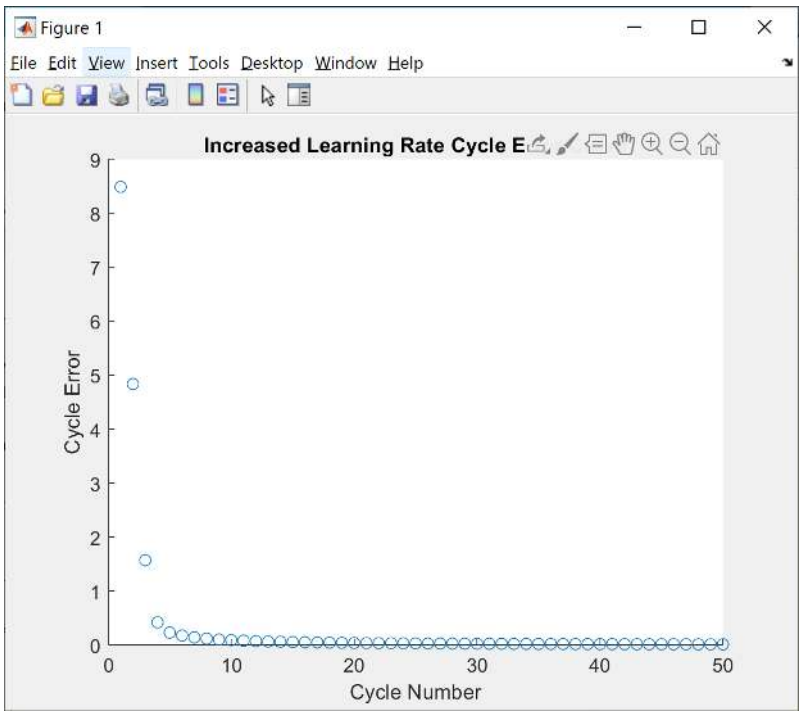


Figure 4: Increased Learning Rate Cycle Error Curve

MATLAB Code for Q1 Part 1:

```
%Inputs with -1 added for Bias
x1 = [ -1 -1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 ...
    -1 -1 1 -1 -1 -1]';
x2 = [ -1 1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 ...
    -1 -1 1 -1 -1 -1]';
x3 = [1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 ...
    -1 -1 1 -1 -1]';
x4 = [-1 -1 -1 -1 -1 -1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 ...
    -1 -1 1 -1 -1]';
x5 = [1 -1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 1 1 -1 -1 1 -1 -1 -1 -1 ...
    1 -1 -1 -1]';
x6 = [-1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 1 1 -1 -1 -1 1 -1 -1 -1 -1 ...
    1 -1 -1 -1]';

x = [x1 x2 x3 x4 x5 x6];

%Desired
d1 = [1 -1 -1]';
d2 = [1 -1 -1]';
d3 = [-1 1 -1]';
d4 = [-1 1 -1]';
d5 = [-1 -1 1]';
d6 = [-1 -1 1]';

d = [d1 d2 d3 d4 d5 d6];

%Hidden To Output Layer Weight
w = [0.2007 -0.0280 -0.1871 0.3366
     0.5522 0.2678 -0.7830 0.7526
     0.4130 -0.5299 0.6420 0.0620];

%Input to Hidden Layer Weight
wHat = [
-0.2206 0.2139 0.4764 -0.1886 0.5775 -0.7873 -0.2943 0.9803 -0.5945 ...
-0.2076 0.7611 0.9635 -0.1627 -0.0503 0.3443 -0.4812 0.8625 0.3333 ...
0.1565 0.7611 0.3211 0.7651 -0.9820 0.2245 0.3536 -0.1562;
0.1932  0.8436  -0.6475 0.3365  0.1795 -0.0542 0.6263 -0.7222 ...
-0.6026 0.3556 -0.2030 -0.0680 0.6924 0.5947 0.6762 0.2802 -0.1763 ...
0.5520 0.2588 0.3365 0.7628 -0.5172 0.3358 0.7878 -0.1101 0.8217; ...
0.6525 0.3525 -0.7127 -0.2266 0.9001 0.0526 0.3200 -0.7556 0.8162 ...
-0.2201 -0.1762 0.3838 -0.9662 0.4567 0.2211 -0.8686 0.0110 0.9339 ...
-0.5050 0.1110 -0.7695 0.2020 0.3378 0.9417 -0.5127 -0.3599];

n = 0.4; %Learning Constant
I = 26;%Num I/p nodes (25 + 1 for Bias)
J = 4;%Hidden Layers (3 + 1 for Bias)
K = 3;%Num O/p nodes

%%Q1.1: Using the error back propagation training,calculate
%the next weight updates w(2), wHat(2)
%Steps:
%1: Find y and z (Feed-Forward phase)
%2: Find error and error signal (Back-Propagation phase)
%3: Find the delta_w and delta_wHat
%4: Update weights [w = w + delta_w]

%Step 1-2:
vHat = wHat*x(:,1);
y = (1-exp(-vHat))./(1+exp(-vHat));%Bipolar LF to calculate y (o/p of hidden layer)
dy = 0.5*(1-y.^2); %Calculate error slope bwteen hidden & i/p layer
v = w*[y; -1];%Adding -1 to y inputs for Bias
z= (1-exp(-v))./(1+exp(-v));%Bipolar LF to calculate z (o/p of 2nd layer)
dz = 0.5*(1-z.^2); %Calculate error slope between o/p & hidden layer
r = d(:,1)- z; %Error between Desired and Actual
delta = r.*dz; %Error signal between o/p & hidden layer
deltaHat = dy.*(w(:,1:J-1)'*delta);%Error signal between hidden & i/p layer
%Step 3-4:
delta_w = n*delta*[y; -1]';
delta_wHat = n*deltaHat*x(:,1)';
w = w + delta_w;
wHat = wHat + delta_wHat;

disp("w(2):")
disp(w)
disp("wHat(2):")
disp(wHat)
```

MATLAB Code for Q1 Part 2:

```
%Inputs with -1 added for Bias
x1 = [ -1 -1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 ...
     -1 -1 1 -1 -1 -1]';
x2 = [ -1 1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 ...
     -1 -1 1 -1 -1 -1]';
x3 = [1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 ...
     -1 -1 1 -1 -1]';
x4 = [-1 -1 -1 -1 -1 -1 -1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 ...
     -1 -1 1 -1 -1]';
x5 = [1 -1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 1 1 -1 -1 1 -1 -1 -1 -1 ...
     1 -1 -1 -1]';
x6 = [-1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 1 -1 -1 -1 1 -1 -1 -1 -1 ...
     1 -1 -1 -1]';

x = [x1 x2 x3 x4 x5 x6];

%Desired
d1 = [1 -1 -1]';
d2 = [1 -1 -1]';
d3 = [-1 1 -1]';
d4 = [-1 1 -1]';
d5 = [-1 -1 1]';
d6 = [-1 -1 1]';

d = [d1 d2 d3 d4 d5 d6];

%Hidden To Output Layer Weight
w = [0.2007 -0.0280 -0.1871 0.3366
      0.5522 0.2678 -0.7830 0.7526
      0.4130 -0.5299 0.6420 0.0620];

%Input to Hidden Layer Weight
wHat = [
-0.2206 0.2139 0.4764 -0.1886 0.5775 -0.7873 -0.2943 0.9803 -0.5945 ...
-0.2076 0.7611 0.9635 -0.1627 -0.0503 0.3443 -0.4812 0.8625 0.3333 ...
0.1565 0.7611 0.3211 0.7651 -0.9820 0.2245 0.3536 -0.1562;
0.1932  0.8436  -0.6475 0.3365  0.1795 -0.0542 0.6263 -0.7222 ...
-0.6026 0.3556 -0.2030 -0.0680 0.6924 0.5947 0.6762 0.2802 -0.1763 ...
0.5520 0.2588 0.3365 0.7628 -0.5172 0.3358 0.7878 -0.1101 0.8217; ...
0.6525 0.3525 -0.7127 -0.2266 0.9001 0.0526 0.3200 -0.7556 0.8162 ...
-0.2201 -0.1762 0.3838 -0.9662 0.4567 0.2211 -0.8686 0.0110 0.9339 ...
-0.5050 0.1110 -0.7695 0.2020 0.3378 0.9417 -0.5127 -0.3599];

n = 0.4; %Learning Constant
I = 26;%Num I/p nodes (25 + 1 for Bias)
J = 4;%Hidden Layers (3 + 1 for Bias)
K = 3;%Num O/p nodes

% Q1.2: Determine the final weight matrices w(301) & wHat(301) after 50 cycles.
%Plot the cycle error curve for this training exercise.

cycles = 50;
numInputNodes = 6;
cycleEr = [];%Initialise cycle error array

for j = 1:cycles %Number of Cycles
    e = 0;%Initialise error count
    for i = 1:numInputNodes %Number of I/p (x1-x6)
         vHat = wHat*x(:,i);
         y = (1-exp(-vHat))./(1+exp(-vHat));%Bipolar LF to calculate y (o/p of hidden layer)
         dy = 0.5*(1-y.^2); %Calculate error slope bwteen hidden & i/p layer
         v = w*[y; -1];%adding -1 to y inputs for Bias
         z = (1-exp(-v))./(1+exp(-v));%Bipolar LF to calculate z (o/p of 2nd layer)
         dz = 0.5*(1-z.^2); %Calculate error slope bwteen o/p & hidden layer
         r = d(:,i)- z; %error between Desired and Actual
         delta = r.*dz; %error signal between o/p & hidden layer
         deltaHat = dy.*(w(:,1:J-1)'*delta);%error signal between hidden & i/p layer
         %Step 3-4
         delta_w = n*delta*[y; -1]';
         delta_wHat = n*deltaHat*x(:,i)';
         w = w + delta_w;
         wHat = wHat + delta_wHat;
         er = r.^2;
         e = e + 0.5*sum(er);%cycle error
    end
    %Obtaining Final Weight Values
    if j == cycles
        w_Final = w;
        wHat_Final = wHat;
    end
    cycleEr = [cycleEr e]; %#ok<AGROW> % All the cycle error
end

disp('w_Final = w(301)')
disp(w_Final)
disp('wHat_Final =wHat(301)')
disp(wHat_Final)

% plotting cycle errors
step = 1:cycles;
scatter(step,cycleEr);
xlabel('Cycle Number');
ylabel('Cycle Error');
title('Training Set Cycle Error Curve')
```

MATLAB Code for Q1 Part 3:

```
%% Q1.3: Classifying Test Number Patterns
%Put testInput into same format as other inputs(x1-x6)
%with -1 Bias at the end
x_input = [-1 -1 -1 -1 1 -1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 ...
             -1 -1 1 -1 -1 -1]'; % -1 is white square

vHat_test = wHat_Final * x_input;
y_test = (1-exp(-vHat_test))./(1+exp(-vHat_test));
v_test = w_Final*[y_test; -1];
z_test = (1-exp(-v_test))./(1+exp(-v_test));
disp('The output vector z for test input')
disp(z_test)
```

Question 2 – Convolutional Neural Network

The first step required before creating a convolutional neural network (CNN), is to make sure that the images being used are of similar size. Thus, this required the rescaling of the cat & dog images to a uniform size.

The below MATLAB code shows the script used for resizing the cat and dog images into a 200x200.

```
%%Resizing Cat & Dog Images

clc;
clear all; %#ok<CLALL>

%File Path of Cats & Dogs Images
catFP = 'C:\Users\Jeremy\Dropbox\Coding\MATLAB_code\Assignment 2\Images\cats';
dogFP = 'C:\Users\Jeremy\Dropbox\Coding\MATLAB_code\Assignment 2\Images\dogs';

%File Path of the resized Cats & Dogs Images
resizedCatFP ='C:\Users\Jeremy\Desktop\CNN Images\A2_CNN_Q2\Resized Images 200x200\cats';
resizedDogFP = 'C:\Users\Jeremy\Desktop\CNN Images\A2_CNN_Q2\Resized Images 200x200\dogs';

catFileString = '\cat.';
dogFileString = '\dog.';

%%Resizing to 200x200 and saving images to resized folder

%Cat Images
for i = 0:299 %images start from 0
    catImageString = [catFP catFileString num2str(i) '.jpg'];
    catImage = imread(catImageString);
    rsCat = imresize(catImage , [200, 200]);

    baseFileName = sprintf('cat.%d.jpg',i);
    filePath = fullfile(resizedCatFP,baseFileName);
    imwrite(rsCat, filePath);
end

%Dog Images
for i = 0:299
    dogImageString = [dogFP dogFileString num2str(i) '.jpg'];
    dogImage = imread(dogImageString);
    rsDog = imresize(dogImage , [200, 200]);

    baseFileName = sprintf('dog.%d.jpg',i);
    filePath = fullfile(resizedDogFP,baseFileName);
    imwrite(rsDog, filePath);
end
```

Q2 Part 1

Now that the images were resized to a suitable size, we could now load and split them accordingly.

Step 1: Load File Paths into Workspace

Step 2: Image Storing & Splitting into Training and Validation sets (7:3 or 70%:30%)

Step 3: Creating the CNN layers with the specified values.

- 4 convolution layers with 24, 28, 32, and 36 filters, respectively,
- Filter size of the convolution layers is 5, stride = 1, padding = 2,
- Batch normalization,
- Active function = ReLU,
- After each convolutional layer, a max pooling layer is followed with filter size = 2, stride = 2.

Step 4: Create CNN training options with the specified values.

- Adaptive learning rate method = Adam,
- Initial learning rate = 0.001,
- Max epochs = 50

After training the CNN with the created layers and training options, the accuracy of the model was 71.67%. The training progress can be seen below.

MATLAB Code for Q2 Part 1:

```
%%Step 1: Load FilePaths Into WorkSpace
clc;
clear all;%#ok<CLALL>

%File Path of where both folders of resized Cats & Dogs Images
resizedFP = 'C:\Users\Jeremy\Dropbox\Coding\MATLAB_code\Assignment 2\Resized Images 200x200';

%%Step 2: Image Storing & Splitting into Training and Validation sets (7:3)
imds = imageDatastore(resizedFP,'IncludeSubfolders',true,...
                      'LabelSource','foldernames');
[imdsTrain, imdsValidation] = splitEachLabel(imds, 0.7, 'randomize');

%%Creating CNN Layers
layers = [
    imageInputLayer([200 200 3]) %input size
    convolution2dLayer(5, 24,'Padding', 2,'Stride', 1 )
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride', 2)

    convolution2dLayer(5, 28,'Padding', 2,'Stride', 1 )
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride', 2)

    convolution2dLayer(5, 32,'Padding', 2,'Stride', 1 )
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride', 2)

    convolution2dLayer(5, 36,'Padding', 2,'Stride', 1 )
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride', 2)

    fullyConnectedLayer(2)%2 classes cat or dog
    softmaxLayer
    classificationLayer
];

%% CNN Training Options
options = trainingOptions('adam', ...
    'InitialLearnRate',0.001, ...
    'MaxEpochs',50, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',10, ...
    'Verbose',true, ...
    'Plots','training-progress');

%% Training CNN
net = trainNetwork(imdsTrain, layers, options);
```

Q2 Part 2

In order to achieve an improved training model of 75% and above, optimising the training options hyperparameters will enable this.

I found that by resizing the images to 28 x 28, my network was able to obtain a better accuracy. I found this strange as usually by giving a network more pixels, it is able to have more weights, however, 200x200x3 = 120,000 weights. Which meant that the training of my network would be much longer to that of 28x28x3 = 2,352 weights. In this situation, resizing the images to a smaller size did not negatively impact my network.

The following hyperparameters were used to train the new model.

```
updatedOptions = trainingOptions('adam', ...
'InitialLearnRate',0.001, ...
'MaxEpochs',50, ...
'MiniBatchSize', 10, ...
'Shuffle','every-epoch', ...
'ValidationData',imdsValidation, ...
'ValidationFrequency',5, ...
```

By having the learning rate at 0.001, it allows the model to learn at an optimal rate without having to trade off accuracy and time.

Epochs is the number of times the model goes through the training algorithm. After numerous testing, having max epochs of 50 seemed to be enough to properly train the network.

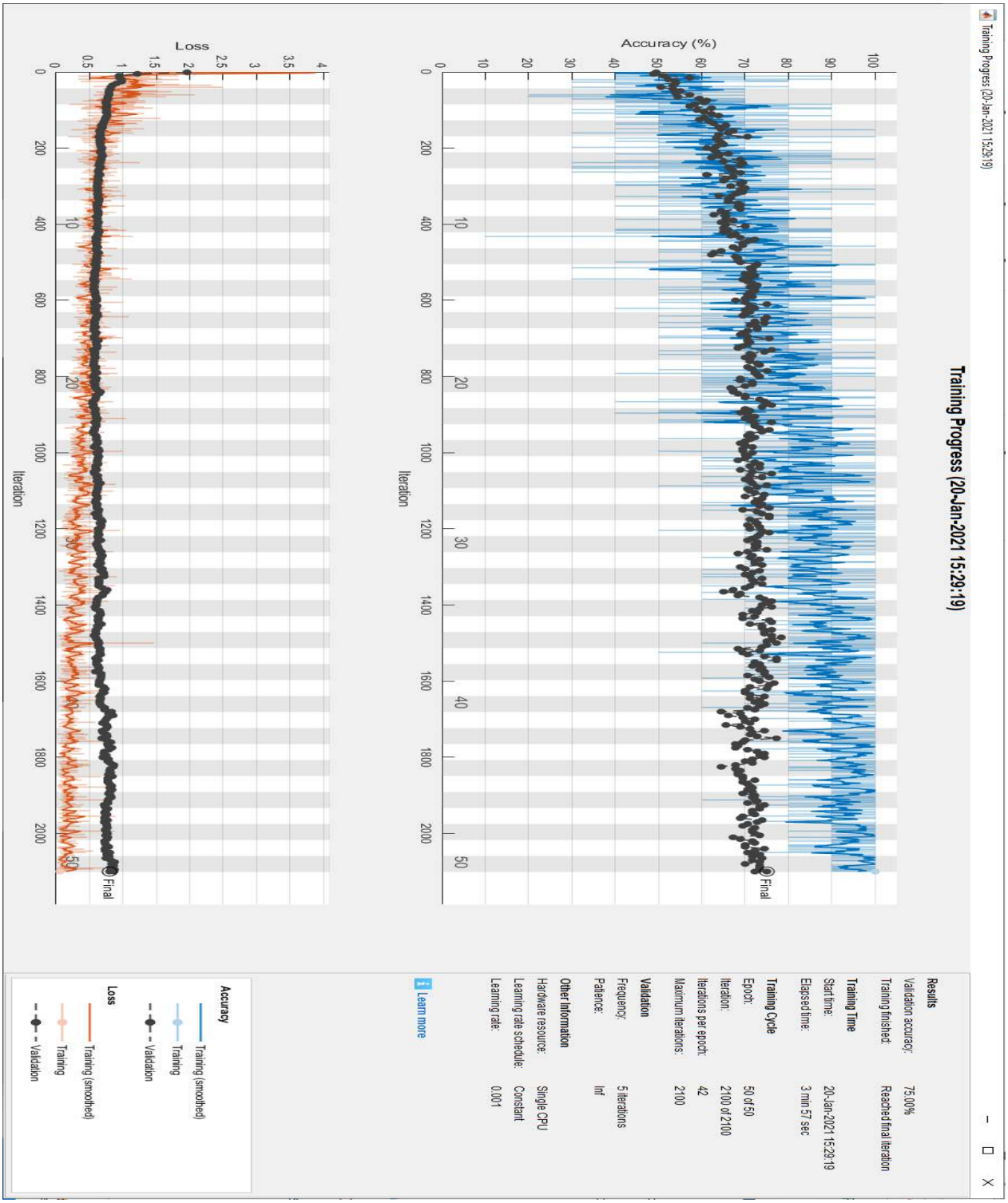
Many training samples are included in each split of the epoch, this is known as mini-batch learning. The amount of data included in each sub-epoch weight change is known as the batch size. Therefore, in this case the data set is 600, setting the size to 10, means that there is 60 mini-batches.

Having the validation frequency set to 5, meant that for each pass through of an epoch, it would require 5 batches to be validated against the other validation images.

Lastly, to reduce the possibility of the model overfitting. A dropout layer of 20% was added after each convolutional layer, as well as a 50% dropout layer before the fully connected layer.

The dropout works by randomly setting the outgoing edges of hidden units (neurons that make up hidden layers) to 0 at each update of the training phase. However, when you increase dropout beyond a certain threshold, it results in the model not being able to fit properly, thus potentially resulting in worse accuracy.

The accuracy of the trained model was 75.00%. The training progress is shown as follows.



MATLAB Code for Q2 Part 2:

```
%%Load FilePaths Into WorkSpace
clc;
clear all;%#ok<CLALL>

%File Path of where both folders of resized Cats & Dogs Images
resizedFP = 'D:\Jeremy\Desktop\Dropbox\Coding\MATLAB_code\Assignment 2\Resized Images 28x28';

% Image Storing 28x28
imds = imageDatastore(resizedFP,'IncludeSubfolders',true,...
                      'LabelSource','foldernames');
[imdsTrain, imdsValidation] = splitEachLabel(imds, 0.7, 'randomize');

% Updated CNN Layers
updatedLayers = [
    imageInputLayer([28 28 3]) %input size
    convolution2dLayer(3, 24,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride', 2)
    dropoutLayer(0.1)

    convolution2dLayer(3, 28,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride', 2)
    dropoutLayer(0.1)

    convolution2dLayer(3, 32,'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride', 2)
    dropoutLayer(0.1)

    convolution2dLayer(3, 38,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride', 2)
    dropoutLayer(0.5)

    fullyConnectedLayer(2)
    softmaxLayer
    classificationLayer
    ];

% Updated CNN Training Options
updatedOptions = trainingOptions('adam', ...
'InitialLearnRate',0.001, ...
'MaxEpochs',50, ...
'MiniBatchSize', 10, ...
'Shuffle','every-epoch', ...
'ValidationData',imdsValidation, ...
'ValidationFrequency',5, ...
'Verbose',false, ...
'Plots','training-progress');

% Training Updated CNN
updatNet = trainNetwork(imdsTrain, updatedLayers, updatedOptions);
```