# Machine Learning HW4: CIS 519

Rutuja Moharil

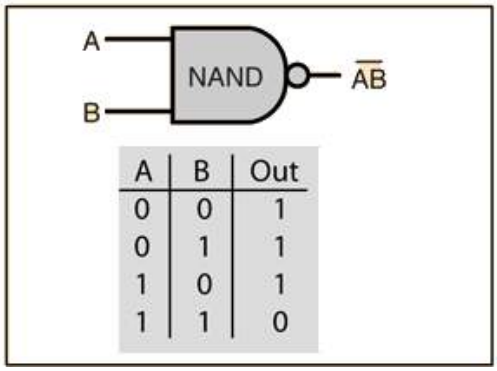
9 April 2019

Taking 2 Late days out of the 3 days

## 1 Logical function with neural nets

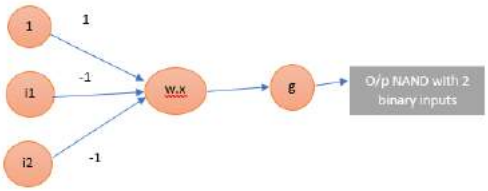
1. Nand Gate

   (a) Truth table




| A | B | Out |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

We can notice from the truth table that the NAND values is 0 only when both inputs are 1 . The network looks like this :




   (b) Formulation

   Let's formulate the NAND gate using the perceptron rule or algorithm i.e Prediction(y)=1 if $Wx + b >= 0$ and 0 if $Wx + b < 0$

   Let's check the neural network using the truth table.Let us take the equation of a neural network as :

   $w_1x_1 + w_2x_2 + b$

   where $w_1 and w_2$ are initialized as -1 and b is intialized as 1 ; thus the equation we get is :

$x_1(-1) + x_2(-1) + 1$
From the first row of the truth table we get $x_1 = 0 and x_2 = 0$ thus we can substitute in the perceptron equation and we get $0 + 0 + 1 = +1$
From the perceptron rule if $wx + b > 0$ then Prediction(y)=1 which is satisfied in this case.
In row 2 we have $x_1 = 0 and x_2 = 1$ thus we can substitute in the perceptron equation and we get $0 + -1 + 1 = 0$
From the perceptron rule if $wx + b > 0$ then Prediction(y)=1 which is satisfied in this case.
In row 3 we have $x_1 = 1 and x_2 = 0$ thus we can substitute in the perceptron equation and we get $1 + 0 + 1 = 2$
From the perceptron rule if $wx + b > 0$ then Prediction(y)=1 which is satisfied in this case.
In row 4 we have $x_1 = 1 and x_2 = 1$ thus we can substitute in the perceptron equation and we get $-1 + -1 + 1 = -1$
From the perceptron rule if $wx + b < 0$ then Prediction(y)=0 which is satisfied in this case.

2. 3-Bit parity
The failure of a single layer of perceptron to be able to learn XOR logic is is because one layer of perceptron provides only linear separation. Thus a 3 bit parity function cannot be modified with a single perceptron layer.The addition of another layer can form a network that is a powerful general nonlinear model that can learn to map multiple inputs to multiple outputs with very small errors.

   (a) Truth Table

| $X_1$ | $X_2$ | $X_3$ | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

   (b) Formulation
   So basically the 3 bit parity can be though of as xor of $X_3$ with output of xor of $X_1$ and$X_2$. $Y = X_1 \bigoplus X_2 \bigoplus X_3$

   (c) So lets find xor of $X_1 and X_2$ with the weights being $w_0 1 = \begin{vmatrix} -1 & 3 \\ 2 & -2 \\ 2 & -3 \end{vmatrix}$ and for the

hidden layer we can take $w_12 = \begin{vmatrix} -3 \\ 2 \\ 2 \end{vmatrix}$

(d) Now we can calculate the output of the XOR of $X_1 and X_2$ at the third layer . Now at the third and fourth layer also we need weights . For easier computation purpose I have considered the same weights i.e $w_23 = \begin{vmatrix} -1 & 3 \\ 2 & -2 \\ 2 & -3 \end{vmatrix}$ $w_34 = \begin{vmatrix} -3 \\ 2 \\ 2 \end{vmatrix}$ The output of the XOR which is obtained at the the third hidden layer is :

$$h = g\left(-3x_0 + 2g\left(-1x_0 + 2x_1 + 2x_2\right) + 2g\left(3x_0 - 2x_1 - 2x_2\right)\right. \tag{1}$$

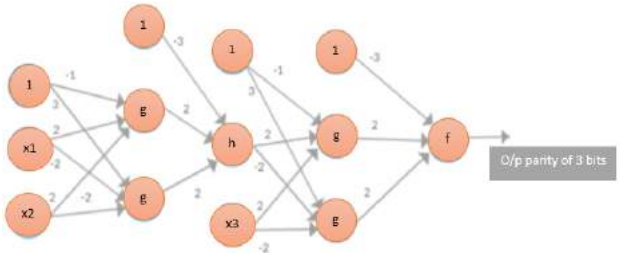
The output of the final layer is :

$$f = g\left(-3x_0 + 2g\left(-1x_0 + 2h + 2x_3\right) + 2g\left(3x_0 - 2h - 2x_3\right)\right. \tag{2}$$

Here g is the activation function:

$$\begin{cases} g(z) = 1 & z \geq 0.5 \\ g(z) = 0 & z < 0.5 \end{cases} \tag{3}$$

n/w n/w for Parity of 3 (4)




$$\text{For} x_1 = 0, x_2 = 0, x_3 = 0 \tag{5}$$

$$\begin{aligned} &h = g(-3(1) + 2g(-1(1) + 2(0) + 2(0)) + 2g(3(1) - 2(0) - 2(0)) \\ &= g(-3 + 2g(-1) + 2g(3)) = g(-3 + 2(0) + 2(1)) = \\ &g(-1) = 0 \end{aligned} \tag{6}$$

$$\begin{aligned} &f = g(-3(1) + 2g(-1(1) + 2(0) + 2(0)) + 2g(3(1) - 2(0) - 2(0)) \\ &= g(-3 + 2g(-1) + 2g(3)) = g(-3 + 0 + 2) = \\ &g(-1) = 0 \end{aligned} \tag{7}$$

For $x_1 = 0, x_2 = 0, x_3 = 1$

$$\begin{aligned} &h = g(-3(1) + 2g(-1(1) + 2(0) + 2(0)) + 2g(3(1) - 2(0) - 2(0)) \\ &= g(-3 + 2g(-1) + 2g(3)) = 0 \\ &f = g(-3(1) + 2g(-1(1) + 2(0) + 2(1)) + 2g(3(1) - 2(0) - 2(1)) \\ &= g(-3 + 2g(1) + 2g(1)) = 1 \end{aligned} \tag{8}$$

For $x_1 = 0, x_2 = 1, x_3 = 0$

$$
\begin{aligned}
&h = g(-3(1) + 2g(-1(1) + 2(0) + 2(1)) + 2g(3(1) - 2(0) - 2(1)) \\
&= g(-3 + 2g(1) + 2g(1)) = 1 \\
&f = g(-3(1) + 2g(-1(1) + 2(1) + 2(0)) + 2g(3(1) - 2(1) - 2(0)) \\
&= g(-3 + 2g(1) + 2g(1)) = 1
\end{aligned} \tag{9}
$$

$$
\begin{aligned}
&\text{For } x_1 = 1, x_2 = 0, x_3 = 0 \\
&h = g(-3(1) + 2g(-1(1) + 2(1) + 2(0)) + 2g(3(1) - 2(1) - 2(0)) \\
&= g(-3 + 2g(1) + 2g(1)) = 1 \\
&f = g(-3(1) + 2g(-1(1) + 2(1) + 2(0)) + 2g(3(1) - 2(1) - 2(0)) \\
&= g(-3 + 2g(1) + 2g(1)) = 1
\end{aligned} \tag{10}
$$

$$
\begin{aligned}
&\text{For } x_1 = 0, x_2 = 1, x_3 = 1 \\
&h = g(-3(1) + 2g(-1(1) + 2(0) + 2(1)) + 2g(3(1) - 2(0) - 2(1)) \\
&= g(-3 + 2g(1) + 2g(1)) = 1 \\
&f = g(-3(1) + 2g(-1(1) + 2(1) + 2(1)) + 2g(3(1) - 2(1) - 2(1)) \\
&= g(-3 + 2g(3) + 2g(-1)) = 0
\end{aligned} \tag{11}
$$

$$
\begin{aligned}
&\text{For } x_1 = 1, x_2 = 1, x_3 = 0 \\
&h = g(-3(1) + 2g(-1(1) + 2(1) + 2(1)) + 2g(3(1) - 2(1) - 2(1)) \\
&= g(-3 + 2g(3) + 2g(-1)) = 0 \\
&f = g(-3(1) + 2g(-1(1) + 2(0) + 2(0)) + 2g(3(1) - 2(0) - 2(0)) \\
&= g(-3 + 2g(-1) + 2g(3)) = 0
\end{aligned} \tag{12}
$$

$$
\begin{aligned}
&\text{For } x_1 = 1, x_2 = 0, x_3 = 1 \\
&h = g(-3(1) + 2g(-1(1) + 2(1) + 2(0)) + 2g(3(1) - 2(1) - 2(0)) \\
&= g(-3 + 2g(1) + 2g(1)) = 1
\end{aligned} \tag{13}
$$

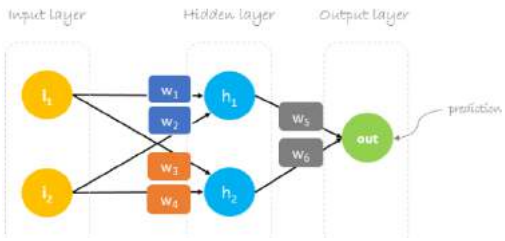
$$
\begin{aligned}
&f = g(-3(1) + 2g(-1(1) + 2(1) + 2(1)) + 2g(3(1) - 2(1) - 2(1) \\
&) = g(-3 + 2g(3) + 2g(-1)) = 0 \\
&\text{For } x_1 = 1, x_2 = 1, x_3 = 1 \\
&h = g(-3(1) + 2g(-1(1) + 2(1) + 2(1)) + 2g(3(1) - 2(1) - 2(1)) \\
&= g(-3 + 2g(4) + 2g(-1)) = 0
\end{aligned} \tag{14}
$$

$$
\begin{aligned}
&\text{For } x_1 = 1, x_2 = 1, x_3 = 1 \\
&h = g(-3(1) + 2g(-1(1) + 2(1) + 2(1)) + 2g(3(1) - 2(1) - 2(1)) \\
&= g(-3 + 2g(4) + 2g(-1)) = 0
\end{aligned} \tag{15}
$$

$$
f = g(-3(1)+2g(-1(1)+2(0)+2(1))+2g(3(1)-2(0)-2(1)) = g(-3+2g(1)+2g(1)) = 1 \tag{16}
$$

# 2 Backprop

In this particular question we are supposed to find or calculate the backpropagation .Basically this image below is a representation of the layer network .




The image above gives a very easy representation for the calculations to be performed .
Here we have been given $W_01 = \begin{vmatrix} w_1 & w_2 \\ w_3 & w_4 \end{vmatrix} = \begin{vmatrix} 0.1 & 0.2 \\ -0.4 & 0.3 \end{vmatrix}$
The weight between the hidden and output layer is given as $w_12 = \begin{vmatrix} w_5 & w_5 \end{vmatrix}^T = \begin{vmatrix} 0.1 & 0.2 \end{vmatrix}^T$
So the way we will calculate at the first layer is as follows :

$$\begin{aligned} z_01 &= i_1 * w_1 + i_2 * w_2 \\ z_01 &= 5 * 0.1 + 4 * 0.2 = 1.3 \\ z_02 &= i_1 * w_3 + i_2 * w_3 \\ z_02 &= 5 * (-0.4) + 4 * (0.3) = -0.8 \end{aligned}$$

Then $z_01 and z_02$ pass through an activation function which is the sign function to give $h_1, h_2$.

$$\begin{aligned} h_1 &= sign(z_01) = sign(1.3) = 1 \\ h_2 &= sign(z_02) = sign(-0.8) = -1 \end{aligned}$$

Now from the hidden layer to the output layer we have the following calculations.

$$\begin{aligned} z_12 &= h_1 * w_5 + h_2 * w_6 \\ z_12 &= 1 * (0.1) + (-1) * 0.2 = -0.1 \\ \sigma(z_12) &\text{ goes into the output layer} \\ \sigma(z_12) &= \frac{1}{1 + \exp 0.1} = 0.475 \end{aligned}$$

Differentiation of sigmoid is as follows

$$\sigma(x) = \sigma(x) * (1 - \sigma(x))$$

The output gradient w.r.t weight $w_6$

$$\frac{d(\sigma(z_1 2)}{dw_6} = \frac{d(\sigma(z_1 2)}{dz_1 2}\frac{d(z_1 2)}{dw_6}$$

$$= \sigma(z_1 2) * (1 - \sigma(z_1 2)) * (h_2) = 0.475 * (1 - 0.475)(-1) = -0.249375$$

The output gradient w.r.t weight $w_5$ is

$$\frac{d(\sigma(z_1 2)}{dw_5} = \frac{d(\sigma(z_1 2)}{dz_1 2}\frac{d(z_1 2)}{dw_5}$$

$$= \sigma(z_1 2) * (1 - \sigma(z_1 2)) * (h_1) = 0.475 * (1 - 0.475)(1) = 0.249375$$

For the hidden layer the output gradient w.r.t weight $w_1$ is

$$\frac{d(\sigma(z_1 2)}{dw_1} = \frac{d(\sigma(z_1 2)}{dz_1 2}\frac{d(z_1 2)}{dh_1}\frac{d(h_1)}{z_0 1}\frac{d(z_0 1)}{w_1}$$

$$= \sigma(z_1 2) * (1 - \sigma(z_1 2)) * w_5 * \nabla sign(1.3)(i_1) = 0.475 * (1 - 0.475)(0.1)(0)(5) = 0$$

For the hidden layer the output gradient w.r.t weight $w_2$ is

$$\frac{d(\sigma(z_1 2)}{dw_2} = \frac{d(\sigma(z_1 2)}{dz_1 2}\frac{d(z_1 2)}{dh_1}\frac{d(h_1)}{z_0 1}\frac{d(z_0 1)}{w_2}$$

$$= \sigma(z_1 2) * (1 - \sigma(z_1 2)) * w_6 * \nabla sign(1.3)(i_2) = 0.475 * (1 - 0.475)(0.1)(0)(4) = 0$$

For the output gradient w.r.t weight $w_3$ is

$$\frac{d(\sigma(z_1 2)}{dw_3} = \frac{d(\sigma(z_1 2)}{dz_1 2}\frac{d(z_1 2)}{dh_2}\frac{d(h_2)}{z_0 2}\frac{d(z_0 2)}{w_3}$$

$$= \sigma(z_1 2) * (1 - \sigma(z_1 2)) * w_6 * \nabla sign(-0.8)(i_1) = 0.475 * (1 - 0.475)(0.2)(1)(5) = 0.24937$$

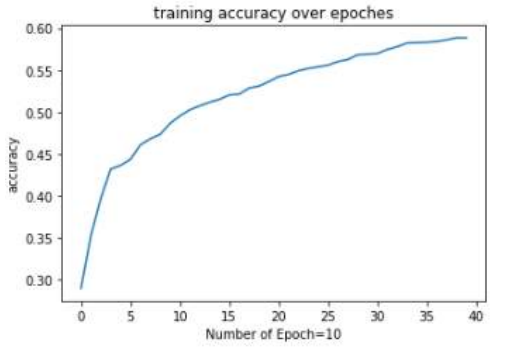
For the output gradient w.r.t weight $w_4$ is

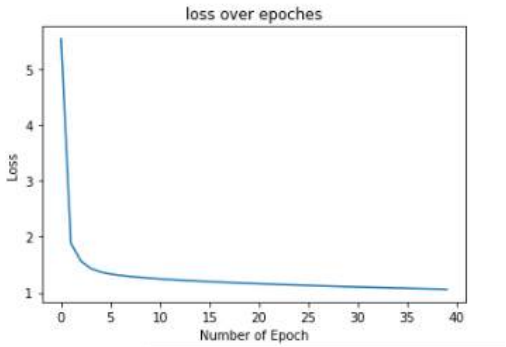
$$\frac{d(\sigma(z_1 2)}{dw_4} = \frac{d(\sigma(z_1 2)}{dz_1 2}\frac{d(z_1 2)}{dh_2}\frac{d(h_2)}{z_0 2}\frac{d(z_0 2)}{w_4}$$

$$= \sigma(z_1 2) * (1 - \sigma(z_1 2)) * w_6 * \nabla sign(-0.8)(i_1) = 0.475 * (1 - 0.475)(0.2)(1)(4) = 0.1995$$

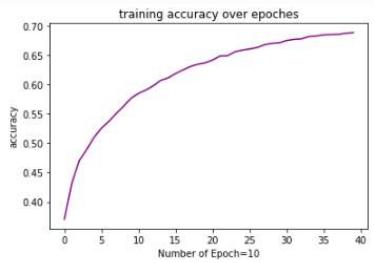
# 3 Neural Nets for classification

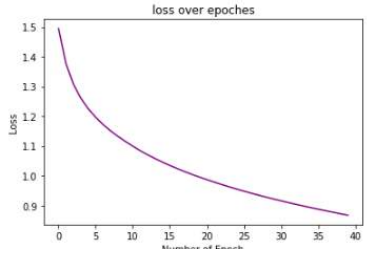
1. Experiment 1 : FeedForward NN
   Here we have been asked to plot the training accuracy ,validation accuracy and loss over 40 epochs over raw images for baseline Feedforward NN.So I observed that with every epoch the loss decreases and training and validation accuracy are increasing.So without the hyper-parameter tuning the train accuracy was 58.1 and test accuracy was 43.25 percent and Loss was 1.30988 . Here are the training accuracy , loss curves for baseline :




2. Experiment 2
   Here we have been asked to plot the training accuracy ,validation accuracy and loss over 40 epochs for Baseline Convolutional Neural Network over raw images . This has better training accuracy compared to FeedForward network since we are implementing backpropagation and updating the weights.
   The training accuracy is 68.3 and validation accuracy is 64.56 ,whereas the loss is 0.584.
   Here are the training accuracy , loss curves



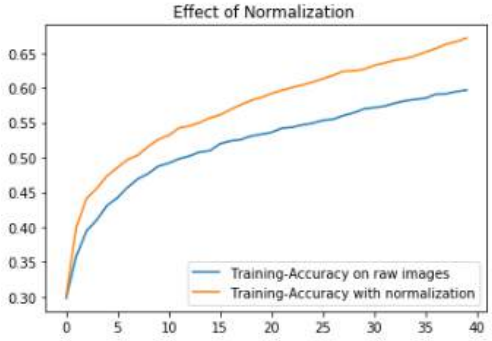

3. Experiment 3
   Here we have to give the CNN and feedforward network the normalised images . The

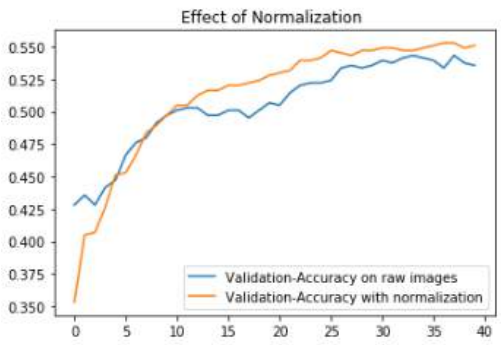
advantages of normalization is

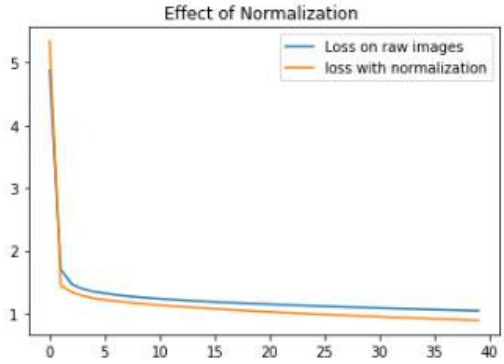
(a) First, if data is not normalized, features with larger numerical values dominate features with smaller numerical values and consequently we will not get contributions from features with smaller values.

(b) Second, many learning algorithms behave well with normalized data. This manifests in higher test accuracy for normalized data than with non-normalized data.

Feedforward normalization:
For Feedfoward raw images Train accuracy is 59.66 ,loss is 1.2709 and validation accuracy 55.08
For Feedforward normalized images Train accuracy is 0.67163462 , Loss is 1.15807 and validation accuracy is 0.53550864
Graphs for feedforward normalized and raw images :

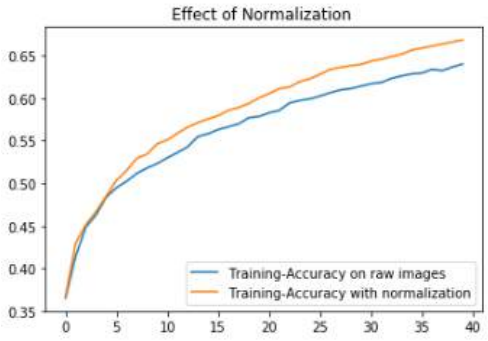


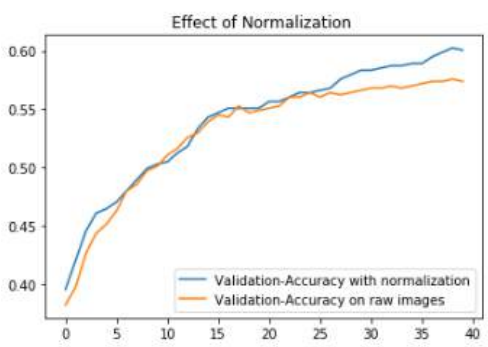



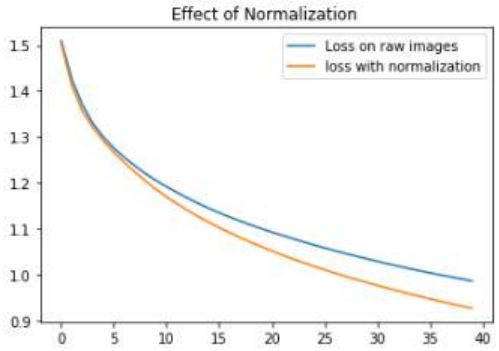




CNN:
For CNN raw images : Train accuracy is 63.56 ,loss is 1.0457 and validation accuracy 55.08
For CNN normalized images : Train accuracy is 66.36 ,loss is 0.923 and validation accuracy is 59.865 percent

Graphs for comparison of normalized and non -normalized image for CNN :




4. Experiment 4
   Here we have been given the task to choose any one of the above experiments and optimize the network using hyper-parameter tuning . So after several experimentations with the hyperparameters I found the best performance or the most optimized performance with the following architecture:

| Layers | Hyperparameters |
|---|---|
| Conv 1 | Out channels=16, kernel_size = 3,stride=1, padding=0 : ReLu |
| Max Pool 1 | kernel_size = 2 |
| Conv2 | Out Channels= 32,kernel_size = 3,stride=1, padding=0 : ReLu |
| Max Pool 2 | kernel_size = 2 |
| Conv3 | Out Channels= 64, kernel_size = 3,stride=1, padding=0 |
| Max Pool 3 | kernel_size = 2 |
| Linear | 200 |
| Dropout | p=0.5 |
| Linear | 5 |
| Dropout | p=0.2 |

   The way I arrived at these parameters is as follows :

   (a) Learning rate : Learning rate controls how much to update the weight in the optimization algorithm.Here I experimented with Adam and Adagrad. Both have adaptive learning rates. I noticed that Adam gave better performance than Adagrad.The learning rate used was 0.005.

   (b) Number of Epochs: Number of epochs is the the number of times the entire training set pass through the neural network. I noticed better performance for 40 epochs Vs 20 epochs.

   (c) Batch size : Mini-batch is usually preferable in the learning process of convnet. Smaller batch sizes ensure avoiding overfitting , thus batch size of 32 has been used.

(d) Number of hidden layers and units ; A good practice for hyparparameter tuning is to add more layers until the test error no longer improves. However this becomes computationally expensive. So I used 3 layers . Also the units in the linear layers should not be small else it may lead to underfitting . Units used are 200 and 5.

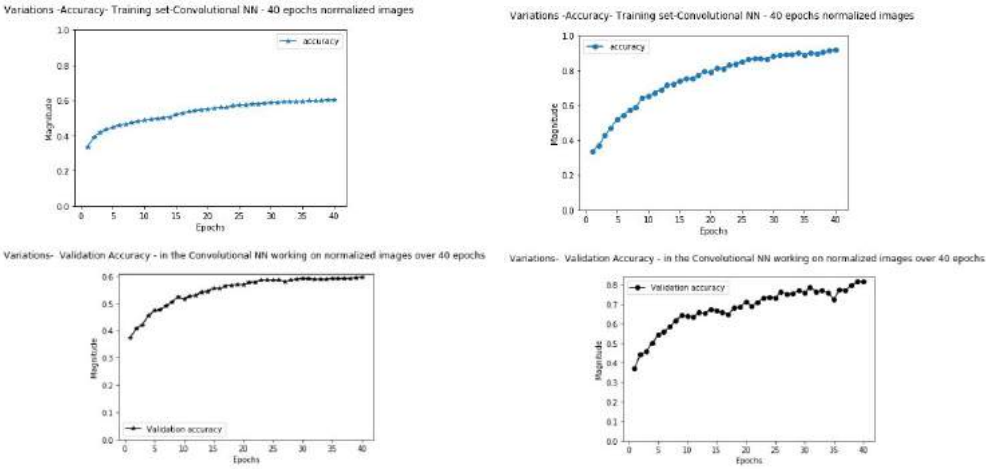
(e) Dropout : Dropout is a preferable technique to avoid overfitting neural networks. The method simply drops out units in neural network according to the desired probability. I used a choice of 0.3 as dropout.

(f) Padding : Padding should be used so that edges of the images are accurately taken into account .Padding did increase the performance and this can be seen with the padded and non padded plots shown in the report (after the Adam Vs Adagrad plots).

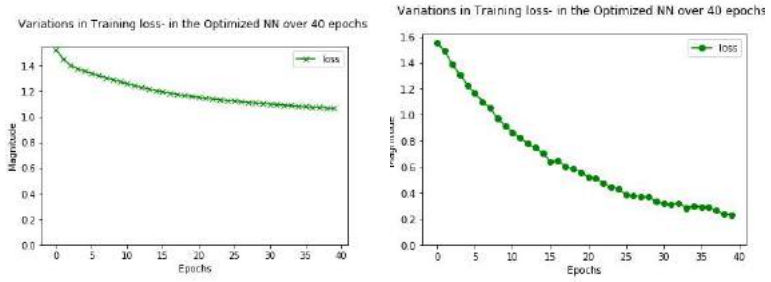
(g) Batch normalization Batch normalization doesn't really help much in our case and gave a slightly better performance after using it on the second layer

Following are the plots which I observed while experimenting with hyperparameter values :

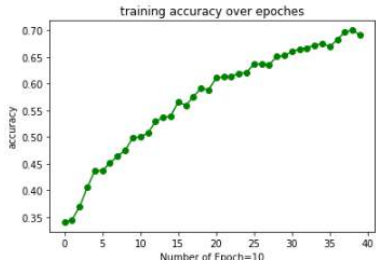
(a) This is a comparison of Adam and Adagrad optimizer used over 40 epochs for the learning rate 0.005. The left plot is for Adagrad and the right plot is for Adam.

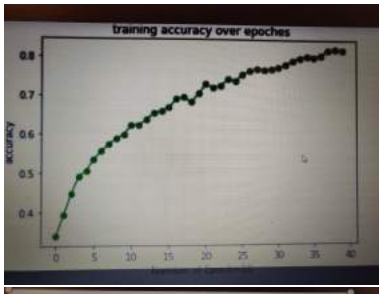


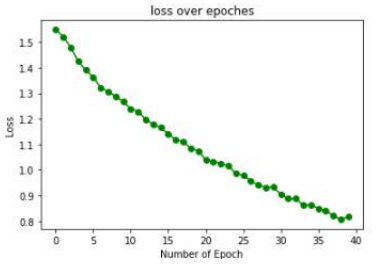

Here are the plots for the loss and comparison between Adagrad and Adam. Left plot is Adagrad and right one is Adam



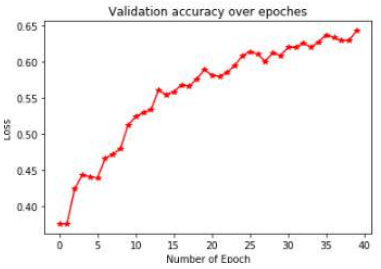
(b) Here I kept Adam as the optimizer and then experimented with the padding . The left side are without padding and right side images are with padding .Without padding validation accuracy 66.39 percent is and whereas with padding (3,1,0) the accuracy 74.49 percent

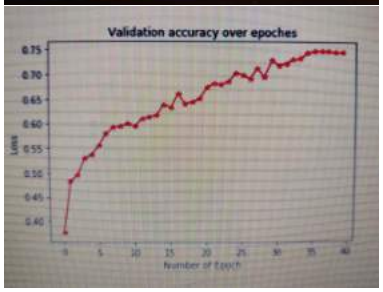




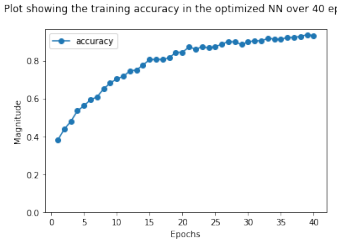


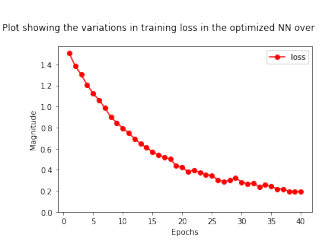



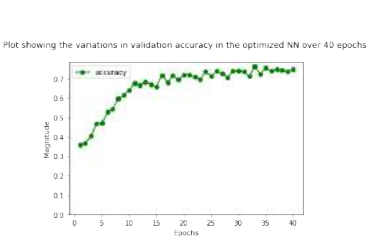

5. My final choice of hyperparameter for input as normalized images are :

| | |
|---|---|
| Conv2D | 3 layers (16,32,64) |
| Kernel | Size =(5,3,3) |
| Dropout | p=0.5 ,p=0.5 |
| Maxpool | Size=2 |
| BatchNorm | After Conv2 layer |
| Padding | Size = (3,1,0) |
| Learning Rate | 0.0001 |
| Optimizer | Adam |

We get the following optimised curves .



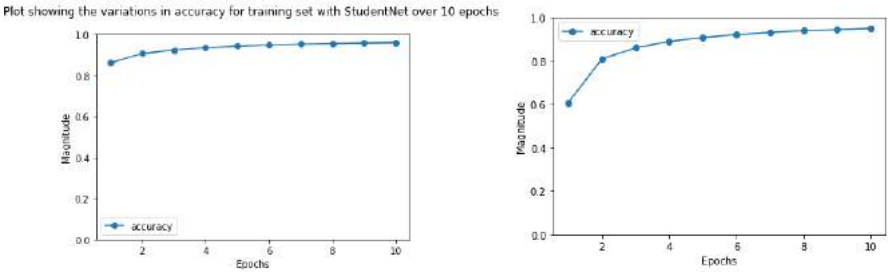
# 4 Debugging Neural networks

6. Section 4.1
   In this section we have been asked to find which aspect of the neural network is faulty.

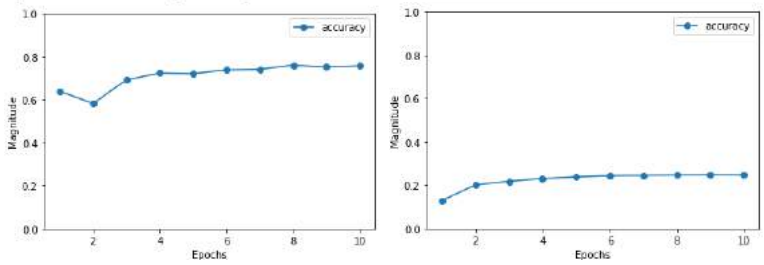
   (a) Feed forward implementation
   So here we implemented the class StudentNet, a feed forward network wth no convolutions to compare it with the faulty networks.
   Feed forward network StudentNet architecure explanation:
   Here we have used 3 feedforward layers and a dropout of 0.3.

| Layer | Parameters |
|---|---|
| Linear layer 1 | 512 units |
| Linear layer 2 | 128 units |
| Dropout | 0.3 |
| Linear | 26 units |

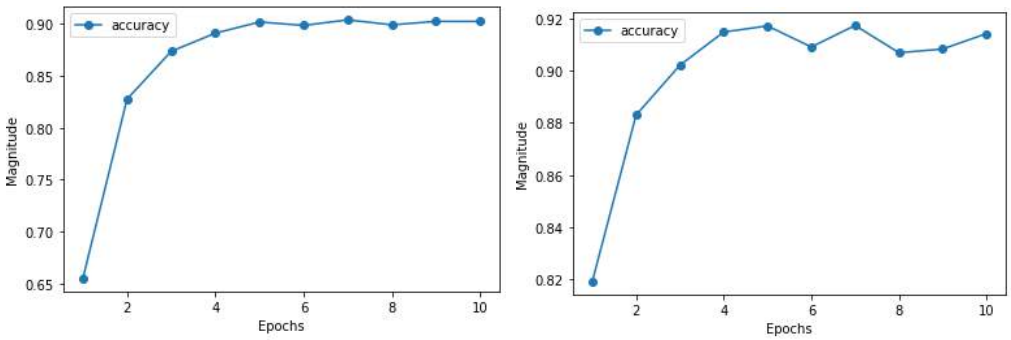
   Optimizer = Adam and learning rate 0.001. The activation function used was ReLU after every layer (except last).
   Here we have to plot the training accuracy and validation accuracy as well as loss curves
   The left plot is the test accuracy curves and the right plot is the training accuracy curve. The training accuracy value is 0.951 and the test accuracy value is 0.9615



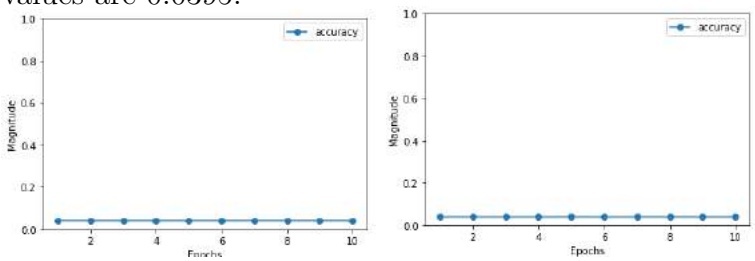

   (b) Faulty DNN 1
   The left figure is the testing accuracy which is at 0.753075 and the right figure is the training accuracy which is at 0.2463.



Here we need to find the issue with the faulty network . The faulty part of the network is the train loader.We can see from the training accuracy values which are very low , because the labels for the images are faulty in the training set .
Validation accuracy is around 75 percent and is still somewhat better since there are only a few labels which are faulty , most of the other samples have correct labels in the vaidation set.
On replacing the trainloader with our correct trainloader (from previous step) we observe the training accuracy to be 90.225 and testing accuracy as 91.425
The graphs after the correct implementation are as follows




(c) Faulty DNN 2
The left plot is testing accuracy and right is the training accuracy . Both these values are 0.0395.




In this situation both the training and testing accuracies were very low .So I tried displaying optimizer parameters and this is what I found .

```
net= net_dict['net']
net=net.to(device)
train_load= net_dict['train_load']
test_load= net_dict['test_load']
optimizer= net_dict['optimizer']
```

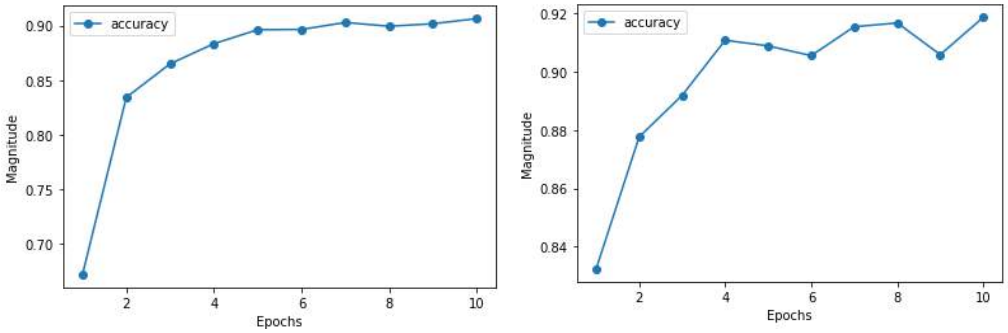
```
[44] #run the network here
     #optimizer = optim.Adam(net.parameters(), lr=0.0005)
     run_net(net,train_load,test_load,criterion,optimizer,10)
     def get_lr(optimizer):
         for param_group in optimizer.param_groups:
             return param_group['lr']
```

```
Train acc : 0.08991 Valid Acc : 0.160475
Train acc : 0.08774 Valid Acc : 0.160475
Train acc : 0.09102 Valid Acc : 0.160475
Train acc : 0.09062 Valid Acc : 0.160475
Train acc : 0.08881 Valid Acc : 0.160475
Train acc : 0.09026 Valid Acc : 0.160475
Train acc : 0.0903 Valid Acc : 0.160475
Train acc : 0.09015 Valid Acc : 0.160475
Train acc : 0.09098 Valid Acc : 0.160475
Train acc : 0.08822 Valid Acc : 0.160475
```

```
get_lr(optimizer)
```

```
300
```

So in this case of the given options we observe that optimizer is the faulty component as it has a learning rate of 300 !!.
This conclusion holds since both training and testing accuracies are very low and similar as well,which means that the model isn't able to learn anything . The Adam optimizer has a very high value of learning rate and hence the model does not learn anything. We can rectify this by using a new optimizer and see the results from the below graphs .




Here I could achieve a training accuracy of 90.64 percent and testing accuracy of 91.8 percent by using a new Adam optimizer with a learning rate of 0.001 (from 4.1).

Contributors : Gsai ,Tien Pham ,Gauri Pradhan