CSC420 A2

Q1) Assume we use some linear activation function like $y = wx + b$

Also assume we are given an input layer x and two hidden layers h as well weights and a final output layer y.

We know that:

$$y = h_2 w_3 + b_3$$

However, we can expand h_2:

$$y = (h_1 w_2 + b_1) w_3 + b_3$$

Again, we can expand h_1:

$$y = ((x w_1 + b_1) w_2 + b_1) w_3 + b_3$$

So, we get (by properties of linear transformations):

$$y = x w_1 w_2 w_3 + b_1 w_2 w_3 + b_2 w_3 + b_3$$

We now notice that this can be expressed as:

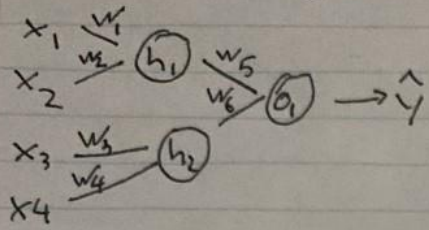
$$y = x w_{new} + b_{new}$$

Where $w_{new} = w_1 w_2 w_3$ and $b_{new} = b_1 w_2 w_3 + b_2 w_3 + b_3$

Effectively, the number of layers in a fully connected network has essentially no impact on the output layer as these layers may be collapsed down into a single layer as described by the process above. Also, note that this was an example for two hidden layers, but the same logic follows for any n number of hidden layers.

Q2)

② I will merge Σ and σ nodes



Define net as the Σ and out as the σ parts.

After a forward pass we get:

$net\ h_1 = (0.75 * 0.9) + (-0.63 * -1.1)$

$out\ h_1 = 0.7971$

$net\ h_2 = (0.24 * -0.3) + (-1.7 * 0.8)$

$out\ h_2 = 0.1928$

$net\ o_1 = (0.7971 * 0.8) + (0.1928 * -0.2)$

$out\ o_1 = 0.6455$

$$\frac{dL}{dw_6} = \frac{dL}{dout_{o_1}} \frac{dout_{o_1}}{dnet_{o_1}} \frac{dnet_{o_1}}{dw_6}$$

$$\frac{dL}{dout_{o_1}} = 2\|0.5 - 0.6455\| = 0.2910$$

$$\frac{dout_{o_1}}{dnet_{o_1}} = out_{o_1}(1 - out_{o_1}) = 0.2288$$

$$\frac{dnet_{o_1}}{dw_6} = (out_{h_1} * w_5 + out_{h_2} * w_6)' = out_{h_2} = 0.1928$$

$$\frac{dL}{dw_3} = \frac{dL}{dout_{h_2}} \frac{dout_{h_2}}{dnet_{h_2}} \frac{dnet_{h_2}}{dw_3}$$

$$\frac{dL}{dout_{h_2}} = \frac{dL_{o_1}}{dnet_{o_1}} \frac{dnet_{o_1}}{dout_{h_2}} = \frac{dL_{o_1}}{dout_{o_1}} \frac{dout_{o_1}}{dnet_{o_1}} \frac{dnet_{o_1}}{dout_{h_2}}$$

$$= (0.2910)(0.2288)(-0.2)$$
$$= -0.0133$$

$$\frac{dout_{h_2}}{dnet_{h_2}} = out_{h_2}(1 - out_{h_2}) = 0.1556$$

$$\frac{dnet\, h_2}{dw_3} = x_3 = -0.3$$

$$= -0.0133 * 0.1556 * -0.3 = \underline{0.0006}$$

Q3) The stride of 2 implies that we will have the filter in 6x6 different positions in C. At each of these positions there are 50 channels and a filter size of 4x4 (x20 as we have 20 filters). So at C we perform (6x6x20x50x4x4) flops from multiplications if we ignore bias. If we account for bias then it is (6x6x20x50x4x4x2). We will also perform 50x20x6x6x(4x4-1) flops from additions without the bias and 50x20x6x6x(4x4) with bias. At P we know that the output size of C is 6x6 and given max pooling with 3x3 receptive fields and a stride of 1 we apply max pooling in 4x4 different positions. At each position there are 20 channels and given that we have 3x3 fields and the knowledge that this takes n-1 (8 operations in this case) flops for each channel we are performing 8x4x4x20 operations at P.

Hence if we ignore bias we have: 6x6x20x50x4x4 + 50x20x6x6x(4x4-1) + 8x4x4x20=1118560 flops.

If we account for bias, we have: 6x6x20x50x4x4x2 + 50x20x6x6x(4x4) + 8x4x4x20=1730560 flops.

Q4)

- The number of learnable parameters in an input layer is 0.
- The number of learnable parameters in a conv layer is given by $((width\ of\ the\ filter\ *\ height\ of\ the\ filter\ *\ number\ of\ filters\ in\ the\ previous\ layer + 1) * number\ of\ filters)$ here the +1 is added for the bias term.
- Subsampling layers require no learning so the parameters here are 0.
- A FC layer has a number of learnable parameters quantified by $((current\ layer\ neurons\ *\ previous\ layer\ neurons) + 1 * current\ layer\ neurons)$

Layer	**Parameters**
C1	$\big((5 * 5 * 1 + 1) * 6\big) = 156$
S2	0
C3	$\big((5 * 5 * 6 + 1) * 6\big) = 906$
S4	0
C5	$\big((5 * 5 * 16 + 1) * 16\big) = 6416$
F6	$\big((84 * 120) + 1 * 84\big) = 10164$
Output	$\big((10 * 84) + 1 * 10\big) = 850$

Q5) Let us consider a single node with output (out), input (net) weight (w) and loss (L). In backpropagation we wish to find:

$$\frac{dL}{dw} = \frac{dL}{dout}\frac{dout}{dnet}\frac{dnet}{dw}$$

Assuming the same error function as in question 2:

$$\frac{dL}{dout} = -(target - out)$$

As you can see this is independent of the input. That said, any error function usually only requires the output (out) and the target output (target).

Next as we are using a logistic function:

$$\frac{dout}{dnet} = out(1 - out)$$

Again, this is entirely independent of any input and only requires us to know the output.

Finally, as net is the sum of all weights multiplied with the output of the previous neurons:

$$\frac{dnet}{dw} = (some\ previous\ output)w$$

Hence, when calculating $\frac{dL}{dw}$ in backpropagation in a NN with logistic activation function we only need the output of neurons.

Q6) A) tanh(x) is a shifted and scaled version of sigmoid. Whereas the sigmoid output range is between 0 and 1, the output range for tanh(x) is between -1 and 1.

B) We first note that we can write the tanh(x) function in terms of the sigmoid logistic activation function.

$$\tanh(x) = 2\sigma(2x) - 1$$

We then take the derivative of the right-hand side:

$$= 4\sigma'(2x) = 4\sigma(2x)\big(1 - \sigma(2x)\big)$$

Hence, the gradient of tanh(x) may be expressed as a function of the logistic function.

C) Both activation functions are used for binary classification in feed forward neural nets. Generally, tanh is preferred to sigmoid for faster convergence given that the gradient on tanh is higher than for sigmoid, but this might change based on the data we are using.

Part 2 A)

I wrote a script that rearranges the data into a directory structure such as root/train/label or root/validation/label. I used this script to first shuffle all the images inside their label directories. Then, the script would copy 1500 images from each label to the corresponding label in the train directory as well as the following 100 to the label inside the validation directory. The last 272 images of each label were copied to the testing directory. This process helps ensure that we have balanced

division when we later use datasets.ImageFolder from pytorch when loading the data into training, validation and testing subsets.

I also used the following transform

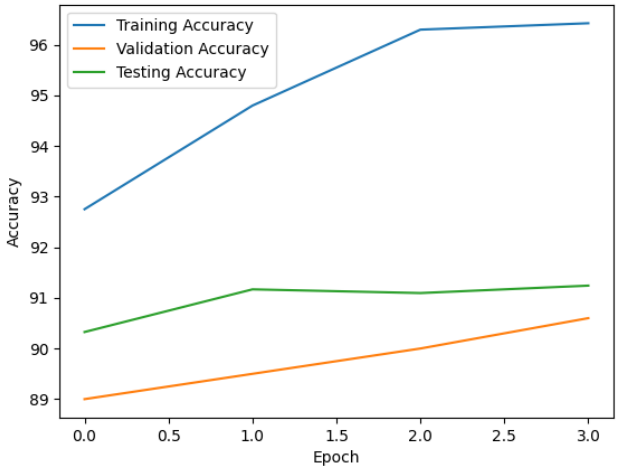
```
transform = transforms.Compose([
        #gray scale
        transforms.Grayscale(),
        #resize
        transforms.Resize((28,28)),
        #converting to tensor
        transforms.ToTensor(),
        #normalize
        transforms.Normalize( (0.1307,), (0.3081,)),
    ])
```

This is because I wanted to ensure that all images are grayscale and of size 28x28. I also needed to load them into tensors for pytorch to process. The normalizing step is done to reduce skewedness of data and help the neural network learn faster.

Part 2 B)

The hyperparameters for tuning in my neural network are the epoch number, batch size, number of hidden units, learning rate and weight decay. For simplicity I have fixed the number of epochs to 4, hidden units to 1000 and the batch size to 100. I tested learning rate values of 0.5,0.1,0.01,0.001 and 0.0001. The best validation accuracy was achieved for a learning rate of 0.001 (having weight decay at 0). After testing 0.0,0.1.0.001,0.0001 and 0.00001 for weight decay, decay did not seem to have a noticeably favorable impact on the validation accuracy, so I fixed this at 0.

For these final values a plot of accuracy as training continues:



The final training accuracy: 96.4%

The final validation accuracy: 90.6%

The final test accuracy: 91.2%

Part 2 C)
Validation Accuracy:

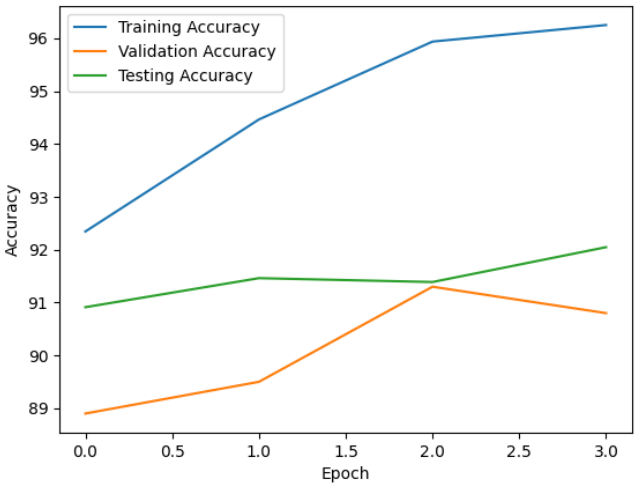
100 Hidden Units: 89.0%

500 Hidden Units: 90.7%

1000 Hidden Units: 91.2%

Using 1000 hidden units the final test accuracy was 91.3%. Increasing the number of hidden units did benefit our neural network by giving us more features to learn.

Part 2 D)
Using a two-layer network (each with 500 hidden units) the final validation accuracy was 90.8%. This is slightly worse than the one-layer case which achieved 91.2% validation accuracy. Strangely, the two layer case managed to score 92.0% on the test set as opposed to just 91.3% for the single layer case.



Part 2 E)

Dropout helped reduce overfitting on the training set leading to a slightly higher validation accuracy of 91.4% as compared to the 91.2% achieved without it.