CSC-578

Hithesh Shanmugam

Assignment-4

Student ID:2068266

Question 1:

In the feedforward there is no cycles are repetition in it that is the connections are fed forward. If an input is given at input layer it goes through the layers in this order first input and to hidden and to output. So in the Network class the feedforward () function in evaluate() it doesn't adjust the weights as activations are calculated and the output activation is returned.

However, in case of back propagation two things are done first feedforward the values and calculate the error and then backpropagate it to the previous layers. Now in the backprop() function the activations which are calculated during the feedforward it is saved in the network in order to adjust the weights in each layer.

Question 2:

```
delta = self.cost_derivative(activations[-1], y) * \
            sigmoid_prime(zs[-1])
```

For computing the partial derivatives of the cost function w.r.t the weights and biases we need to compute an intermediate quantity which is known as delta. The error of the neuron at the output layer of a network is the delta. In the above code we calculate delta which is a cost derivative function w.r.t the activation function multiplied by the derivative of the activation function w.r.t the weighted sum. *The * is used to compute the Hadamard product of the two vectors.* In the above code the index -1 is used to use the activations from the last layer of the network. The two functions within the network class are cost_derivative and sigmoid_prime.

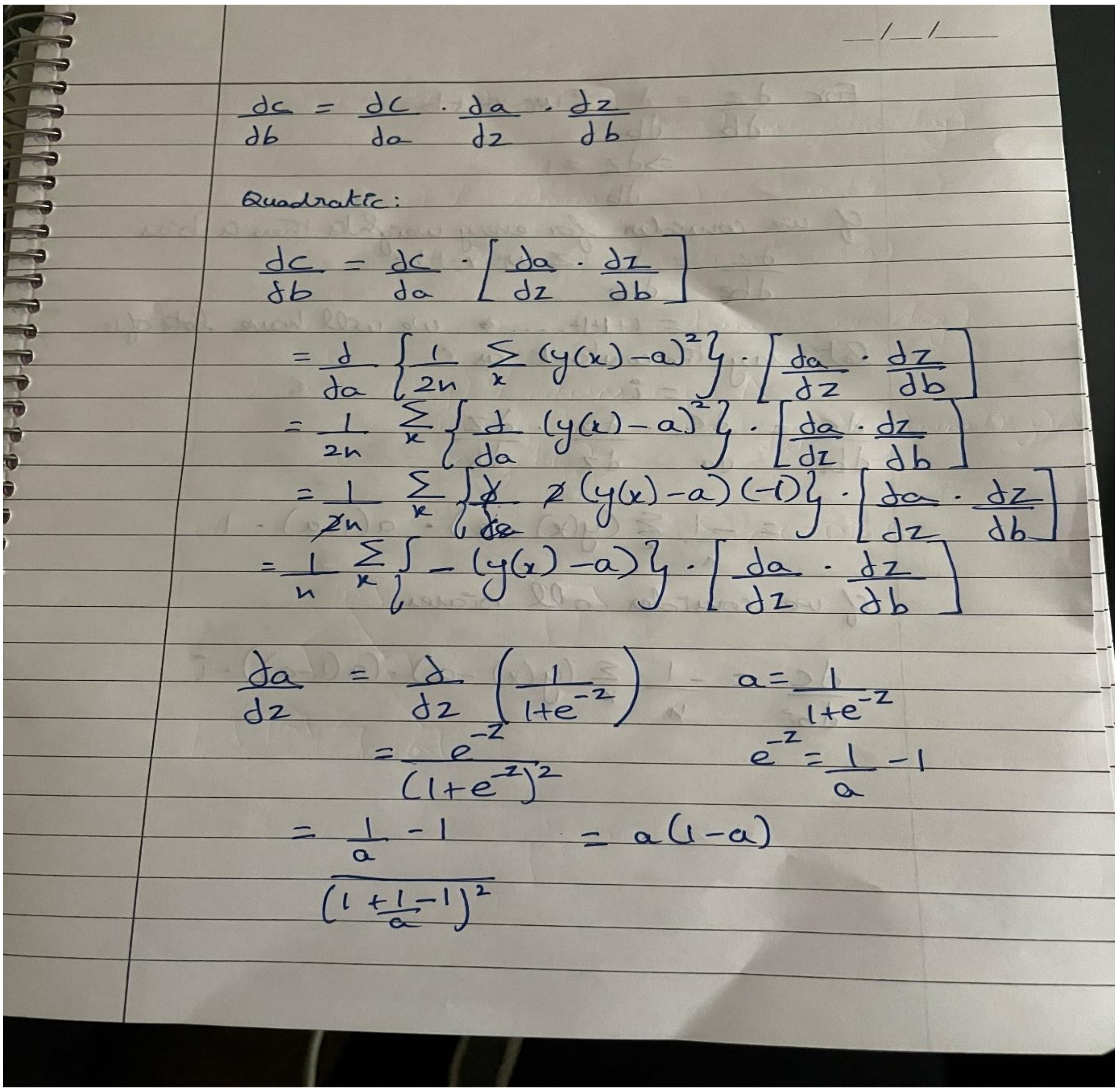
```
nabla_b[-1] = delta
```

In this case the computed delta is same as the bias of the last layer. "nabla_b" is the rate of change of cost w.r.t the bias in the network.

```
nabla_w[-1] = np.dot(delta, activations[-2].transpose())
```

The “nabla_w” that is the nabla weights are computed based on the delta computed earlier. The change in those weights is computed by taking the dot product between the delta and activations of the previous layer.
*The **transpose** operation is used as the shape of a vector or a matrix which needs to be corrected to perform dot operation.*
The” nabla_w” is the rate of change of cost w.r.t the weights in the network.
In order to update the weights and biases of the network first we need to find the weight and bias shifts and use them to update the weights and biases of the network. This updating is done repeatedly to minimize the cost function an eventually reduce to zero.

Question 3:



$$\frac{dc}{db} = \frac{dc}{da} \cdot \frac{da}{dz} \cdot \frac{dz}{db}$$

Quadratic:

$$\frac{dc}{db} = \frac{dc}{da} \cdot \left[\frac{da}{dz} \cdot \frac{dz}{db}\right]$$

$$= \frac{d}{da}\left\{\frac{1}{2n} \sum_x (y(x)-a)^2\right\} \cdot \left[\frac{da}{dz} \cdot \frac{dz}{db}\right]$$

$$= \frac{1}{2n} \sum_x \left\{\frac{d}{da} (y(x)-a)^2\right\} \cdot \left[\frac{da}{dz} \cdot \frac{dz}{db}\right]$$

$$= \frac{1}{\cancel{2}n} \sum_x \left\{\frac{\cancel{d}}{\cancel{da}} \cancel{2}(y(x)-a)(-1)\right\} \cdot \left[\frac{da}{dz} \cdot \frac{dz}{db}\right]$$

$$= \frac{1}{n} \sum_x \left\{-(y(x)-a)\right\} \cdot \left[\frac{da}{dz} \cdot \frac{dz}{db}\right]$$

$$\frac{da}{dz} = \frac{d}{dz}\left(\frac{1}{1+e^{-z}}\right) \qquad a = \frac{1}{1+e^{-z}}$$

$$= \frac{e^{-z}}{(1+e^{-z})^2} \qquad e^{-z} = \frac{1}{a} - 1$$

$$= \frac{\frac{1}{a} - 1}{\left(1 + \frac{1}{a} - 1\right)^2} \qquad = a(1-a)$$

__/__/__

For $\frac{dz}{db} = \frac{d}{db} \Sigma w_i \cdot x_i + b$

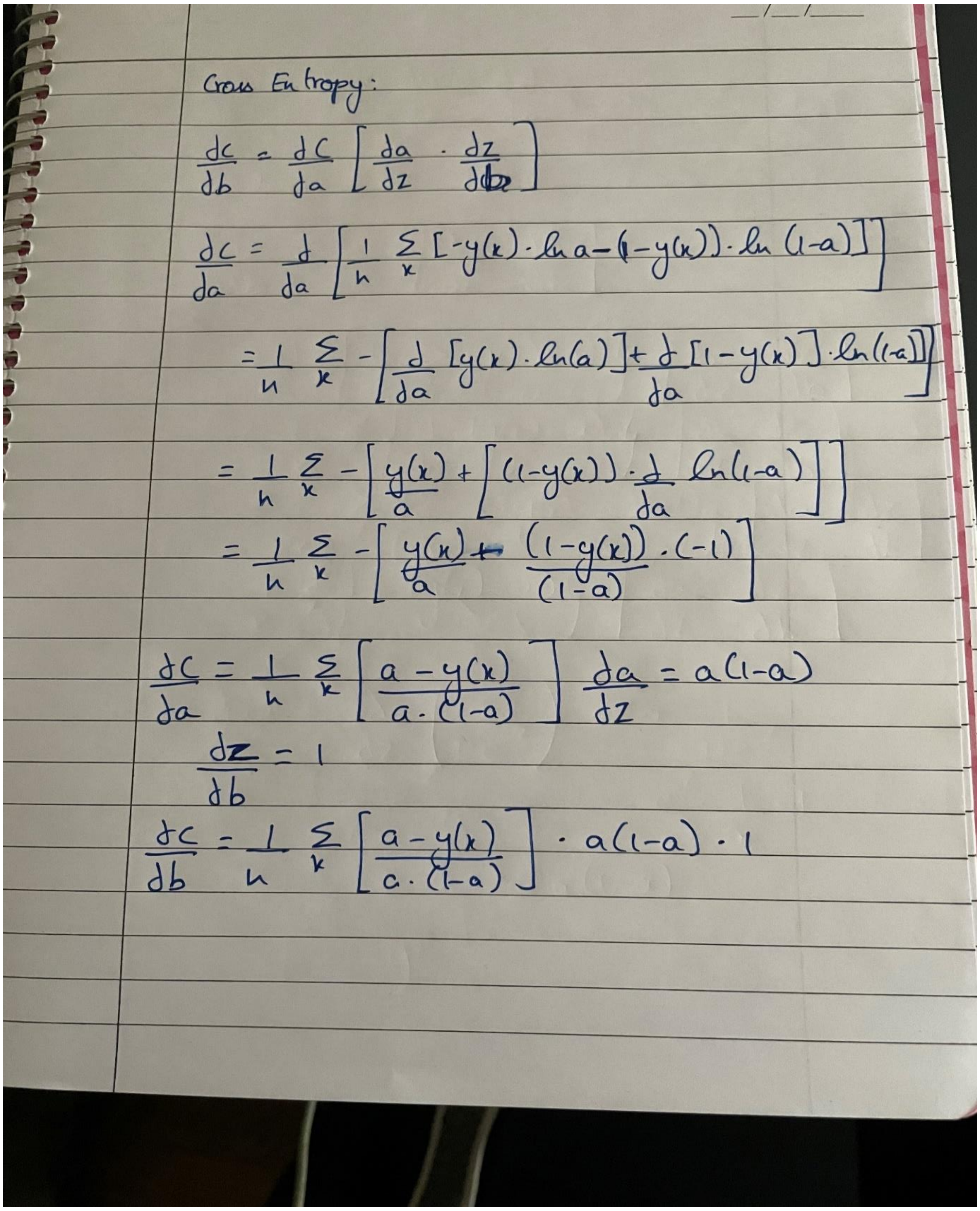
$$= \frac{d}{db}(w_i \cdot x_i) + \frac{d}{db}(b)$$

$$\frac{dz}{db} = 1$$

$$\frac{dC}{db} = \frac{dC}{da} \cdot \frac{da}{dz} \cdot \frac{dz}{db}$$

$$= \frac{-1}{n} \sum_x (y(x) - a)) \cdot a(1-a) \cdot (1)$$

$$= \frac{-1}{n} \sum_x (y(x) - a)) \cdot a(1-a)$$



Cross Entropy:

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial a}\left[\frac{\partial a}{\partial z}\cdot\frac{\partial z}{\partial b}\right]$$

$$\frac{\partial C}{\partial a} = \frac{\partial}{\partial a}\left[\frac{1}{n}\sum_x\left[-y(x)\cdot\ln a-(1-y(x))\cdot\ln(1-a)\right]\right]$$

$$= \frac{1}{n}\sum_x -\left[\frac{\partial}{\partial a}\left[y(x)\cdot\ln(a)\right]+\frac{\partial}{\partial a}\left[1-y(x)\right]\cdot\ln(1-a)\right]$$

$$= \frac{1}{n}\sum_x -\left[\frac{y(x)}{a}+\left[(1-y(x))\cdot\frac{\partial}{\partial a}\ln(1-a)\right]\right]$$

$$= \frac{1}{n}\sum_x -\left[\frac{y(x)}{a}+\frac{(1-y(x))}{(1-a)}\cdot(-1)\right]$$

$$\frac{\partial C}{\partial a} = \frac{1}{n}\sum_x\left[\frac{a-y(x)}{a\cdot(1-a)}\right] \qquad \frac{\partial a}{\partial z} = a(1-a)$$

$$\frac{\partial z}{\partial b} = 1$$

$$\frac{\partial C}{\partial b} = \frac{1}{n}\sum_x\left[\frac{a-y(x)}{a\cdot(1-a)}\right]\cdot a(1-a)\cdot 1$$

From the NNDL book in chapter 3 after applying chain rule without differentiating the sigmoid function the output is given as:

$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z)$$
$$\frac{\partial C}{\partial b} = (a-y)\sigma'(z) = a\sigma'(z),$$

Question 4:

Original model Vs Another hidden layer

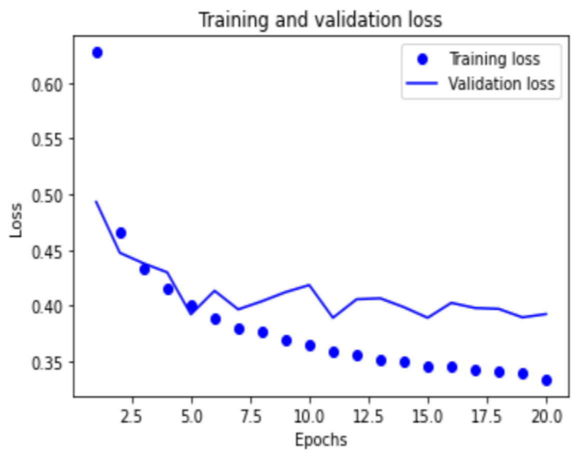
Parameters:

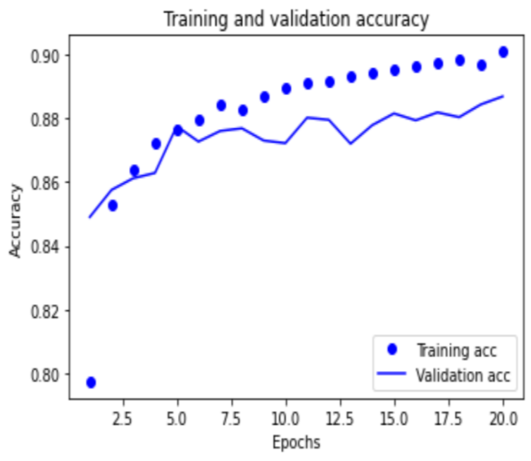
Epochs-20

Number of hidden layer-1

Activation function for hidden layers- relu

Learning rate-0.0001





Test accuracy: 87.83%; Training accuracy: 90.85%

Test loss:42.17%; Training loss: 31.38%

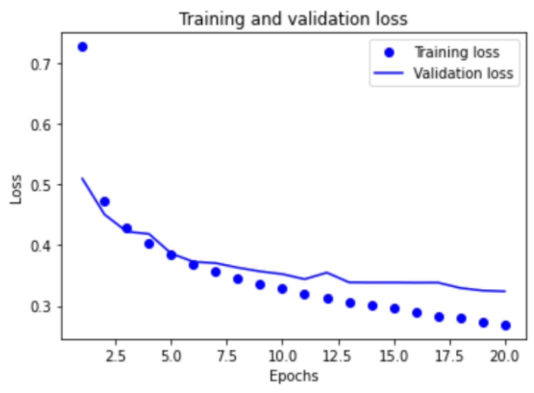
Parameters:

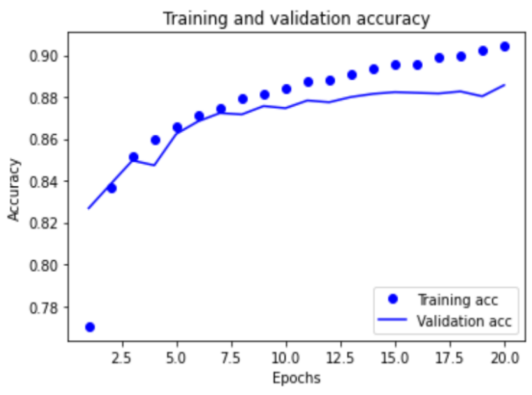
Epochs-20

Number of hidden layer-2

Activation function for hidden layers- relu

Learning rate-0.0001





Test accuracy: 87.66%; Training accuracy: 90.59%

Test loss:34.69%; Training loss: 26.46%

layers	**Learning rate**	**Regularizer class**	**Dropout rate**	**Test accuracy**	**Training accuracy**	**Test loss**	**Training loss**
1	0.0001	*Nil*	*Nil*	87.83%	90.85%	42.17%	31.38%
2	0.0001	*Nil*	*Nil*	87.66%	90.59%	34.69%	26.46%

With the increase in another hidden layer with the size of 32, I was expecting the network to learn classification better than with lesser neurons that is with 1 layer. In my experience with the neural networks, this has always been the case. The number of neurons in first layer was 128 and the other was 32. My expectation with the increased 32 neurons was that the model would perform better on training set and test set.

Results- The result with my expectation is not the model performed with the additional layer but the difference in that is not much a big difference. It's not only in the calculated rate but in the visualization also there is not a vast difference in it. But if I want to choose a model in between these two I'll choose the one with 2 layers because it's just 32 additional neurons. If the neurons count was even higher and it's still the same, I won't go for it. But here I will choose the one with additional layer, because loss and accuracy was smoother in the graph so I am sticking with this model.

Learning rate experimentation:

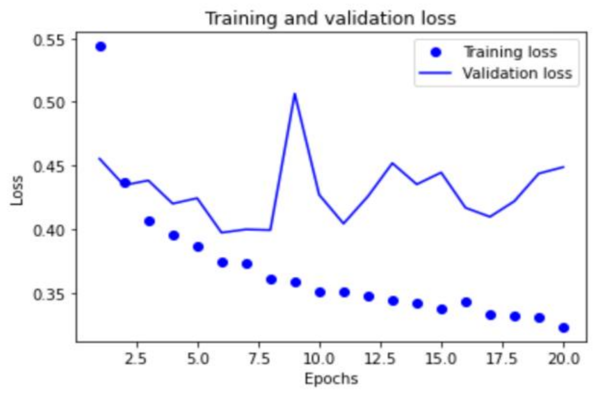
Parameters:

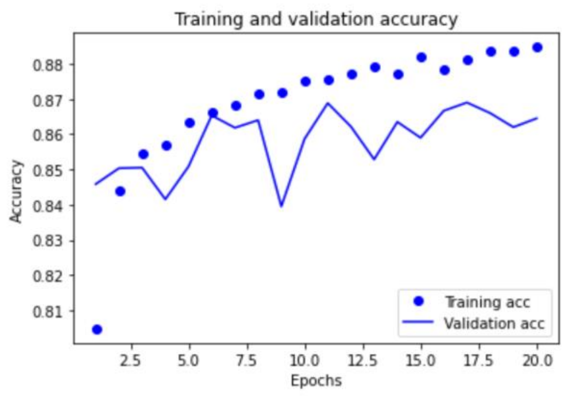
Epochs-20

Number of hidden layer-2

Activation function for hidden layers- relu

Learning rate-0.01





Test accuracy: 85.32%; Training accuracy: 88.13%

Test loss:48.01%; Training loss: 34.01%

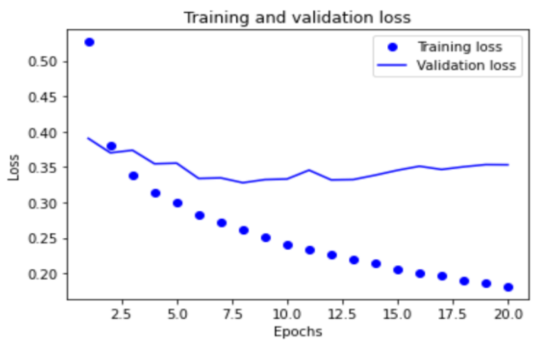
Parameters:

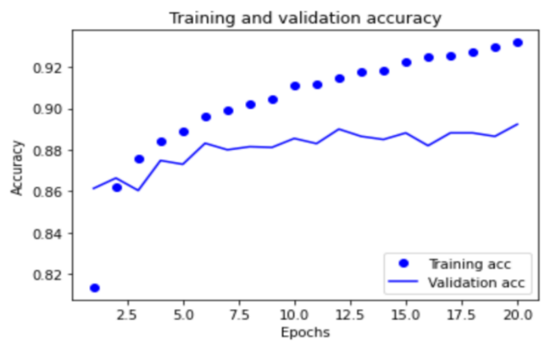
Epochs-20

Number of hidden layer-2

Activation function for hidden layers- relu

Learning rate-0.001





Test accuracy: 88.35%; Training accuracy: 93.17%

Test loss:36.49%; Training loss: 18.88%

layers	Learning rate	Regularizer class	Dropout rate	Test accuracy	Training accuracy	Test loss	Training loss
2	0.0001	*Nil*	*Nil*	87.63%	90.59%	34.69%	26.46%
2	0.01	*Nil*	*Nil*	85.32%	88.13%	48.01%	34.01%
2	0.001	*Nil*	*Nil*	88.35%	93.17%	36.49%	18.88%

Expectation- From previous experimentations, I have found that the learning was slow and hence I expected an increase in the learning rate would in turn increase the performance of the model quickly and efficiently.

Results-The performance of the model increased for a gradual change from 0.0001 to 0.001 but if I increase the learning rate to 0.01 then comparing with the others it was not good learning rate. I am sticking with 0.001 not only because of the last results but also in the curve because I think the model with 0.0001 is overfitting because the learning rate is too small for a change.

Regularizer class experimentation

Parameters:

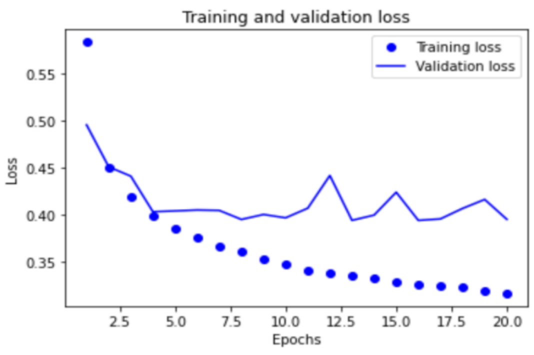
Epochs-20

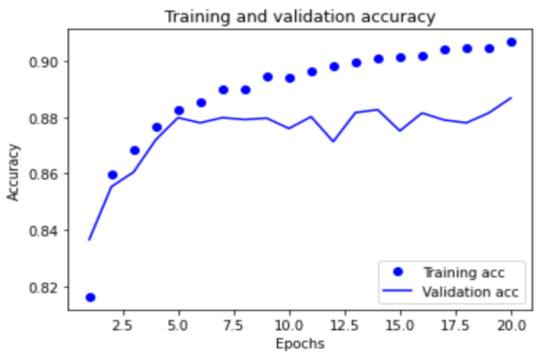
Number of hidden layer-2

Activation function for hidden layers- relu

Learning rate-0.001

Regularizer class=L1L2(L1=1e-5; L2=1e-4)





Test accuracy: 87.87%; Training accuracy: 90.98%

Test loss:41.71%; Training loss: 31.05%

Parameters:

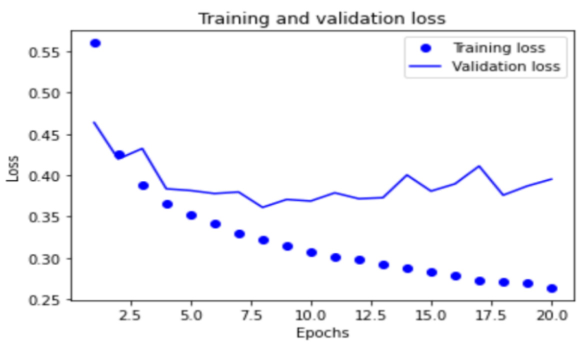
Epochs-20

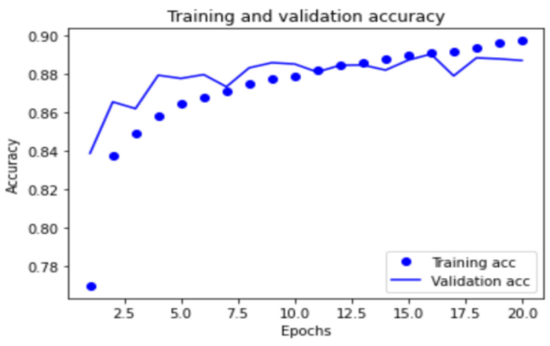
Number of hidden layer-2

Activation function for hidden layers- relu

Learning rate-0.001

Regularizer class=L1(L1=1e-5)





Test accuracy: 88.27%; Training accuracy: 92.15%

Test loss:40.68%; Training loss: 27.06%

layers	Learning rate	Regularizer class	Dropout rate	Test accuracy	Training accuracy	Test loss	Training loss
2	0.001	L1L2	*Nil*	87.01%	89.99%	44.59%	32.91%
2	0.001	L1	*Nil*	87.99%	91.98%	39.41%	26.79%

Expectation- After the experimentation with the regularizer I was expecting that adding regularizer class in L1L2 rather than just a single layer classes the results would be better.

Result- The results I was expecting was not the result I got. The better results were in the regularizer with just single regularizer class(L1). As in case of the graph the graph was smoother in just the single regularizer class. But it was not quite good as the one without any regularizer.

Dropout experimentation

Parameters:

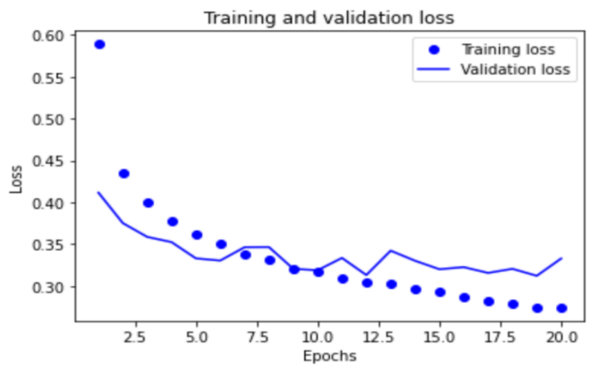
Epochs-20

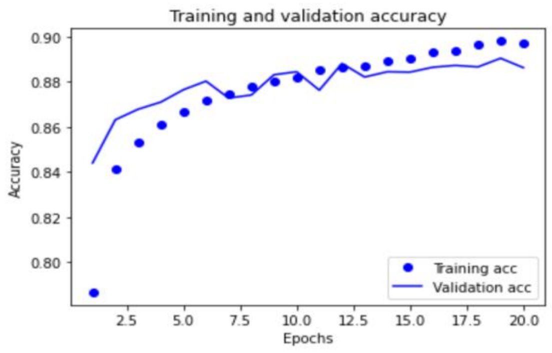
Number of hidden layer-2

Activation function for hidden layers- relu

Learning rate-0.001

Dropout rate=0.3





Test accuracy: 87.95%; Training accuracy: 90.77%

Test loss:35.33%; Training loss: 24.53%

Parameters:

Epochs-20

Number of hidden layer-3

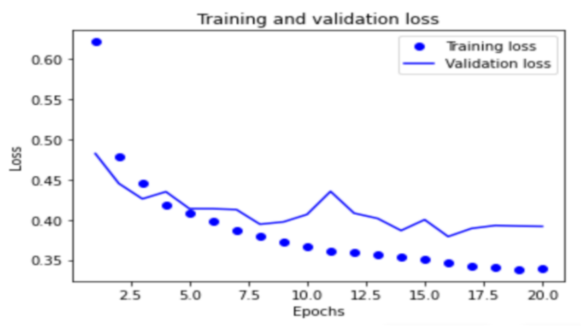
Activation function for hidden layers- relu

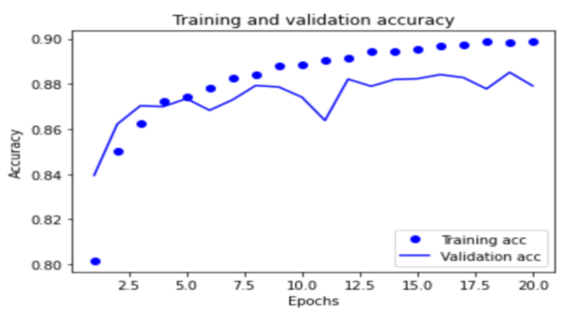
Learning rate-0.001

Regularizer class=L1L2(L1=1e-5; L2=1e-4)

Regularizer class=L1(L1=1e-5)

Dropout rate=0.3





Test accuracy: 87.99%; Training accuracy: 90.86%

Test loss:40.79%; Training loss: 31.86%

layers	Learning rate	Regularizer class	Dropout rate	Test accuracy	Training accuracy	Test loss	Training loss
2	0.001	*Nil*	0.3	88.27%	91.60%	33.10%	22.63%
3	0.001	L1L2 & L1	0.3	87.83%	90.85%	42.17%	31.83%

Expectation-I expected a drop in the performance of the model on the train set whereas I expected a slight increase in the test accuracy. This is because we are adding a regularization in terms of dropout to the network.

Results- I have done with just a dropout rate and also with 3 layers and regularizer classes and also drop out rate. Every result was better in without the regularizer. The test accuracy was also as close as the best one.

Comparison

layers	Learning rate	Regularizer class	Dropout rate	Test accuracy	Training accuracy	Test loss	Training loss
1	0.0001	*Nil*	*Nil*	87.83%	90.85%	42.17%	31.38%
2	0.0001	*Nil*	*Nil*	87.66%	90.59%	34.69%	26.46%
2	0.01	*Nil*	*Nil*	85.32%	88.13%	48.01%	34.01%
2	0.001	*Nil*	*Nil*	**88.35%**	**93.17%**	36.49%	**18.88%**
2	0.001	L1L2	*Nil*	87.86%	90.98%	41.71%	31.05%
2	0.001	L1	*Nil*	88.27%	92.15%	40.68%	27.06%
2	0.001	*Nil*	0.3	87.95%	90.77%	35.33%	24.53%
3	0.001	L1L2 & L1	0.3	87.99%	90.86%	40.79%	31.86%

The best cases are in green colour and the worst are in red colour. By comparing with all the experimentations, the best case in overall would be the 2 marked in green and the worst is the 2 marked in red.