Muhammad Muneeb Ur Rahman **BSCS19057**

Backpropagation

$$\begin{tikzcd}
b \arrow[dr] & & & & & \\
x \arrow[r] \arrow[rrr, bend right] & z \arrow[r] & h \arrow[r] \arrow[ddrr, bend right] & y \arrow[r] & S \arrow[dr] & \\
& & & & & E \\
& & & & R \arrow[ur] &
\end{tikzcd}$$

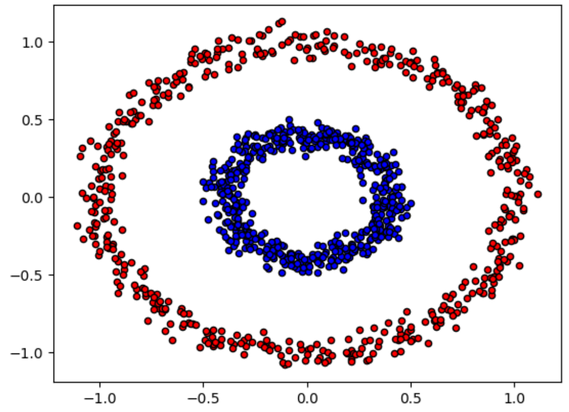
$\frac{\partial \varepsilon}{\partial S} = 1$	$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial S}\frac{\partial S}{\partial y}\frac{\partial y}{\partial x} + \frac{\partial E}{\partial R}\frac{\partial R}{\partial h}\frac{\partial h}{\partial z}\frac{\partial z}{\partial x} + \frac{\partial E}{\partial S}\frac{\partial S}{\partial y}\frac{\partial y}{\partial h}\frac{\partial h}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial \varepsilon}{\partial R} = 1$	$\frac{\partial E}{\partial x} = (1)(y - s)(1) + (1)(r^{T})\sigma'(z)W^{(1)} + (1)(y - s)(W^{(2)})\sigma'(z)W^{(1)}$
$\frac{\partial S}{\partial y} = y - s$	$\frac{\partial E}{\partial x} = (y - s) + (r^{T})\sigma'(z)W^{(1)} + (y - s)(W^{(2)})\sigma'(z)W^{(1)} \quad Ans$
$\frac{\partial R}{\partial h} = r^{T}$	
$\frac{\partial y}{\partial h} = W^{(2)}$	
$\frac{\partial y}{\partial x} = 1$	
$\frac{\partial h}{\partial z} = \sigma'(z)$	
$\frac{\partial z}{\partial x} = W^{(1)}$	

Task 2

First Data

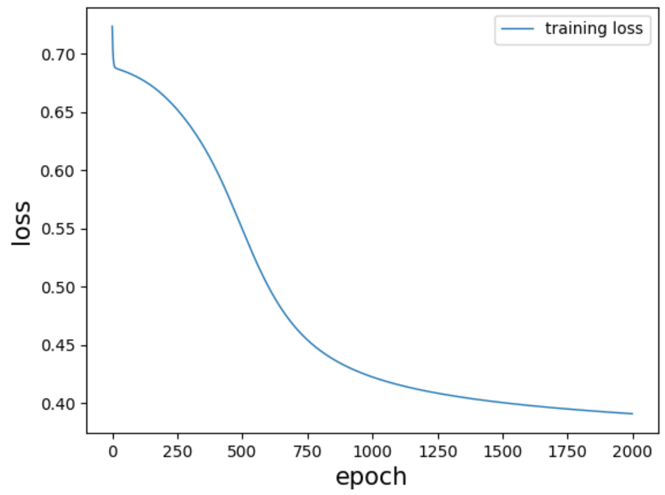
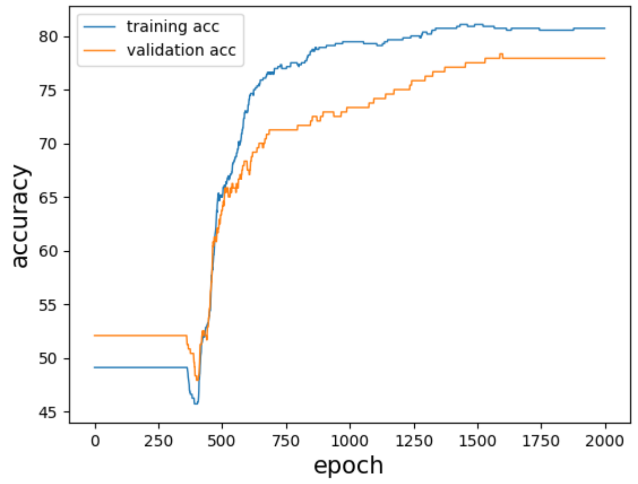
This is the first data distribution that is used to train the neural network.

```
data, label = ds.make_circles(n_samples=1000, factor=.4, noise=0.05)
```



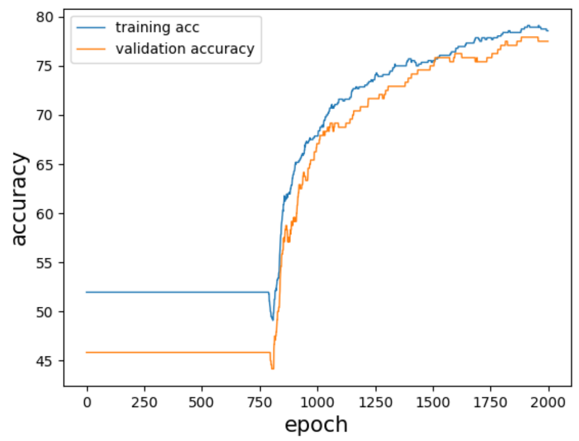
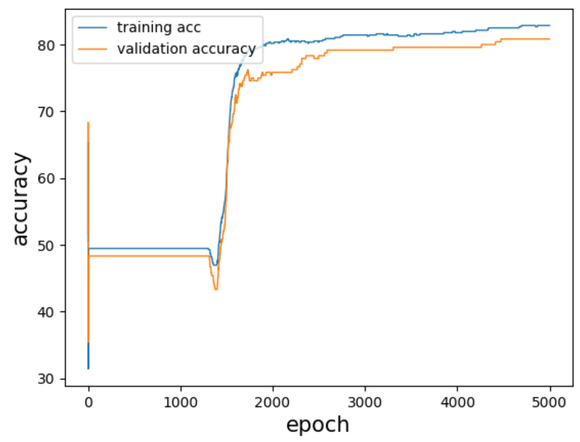
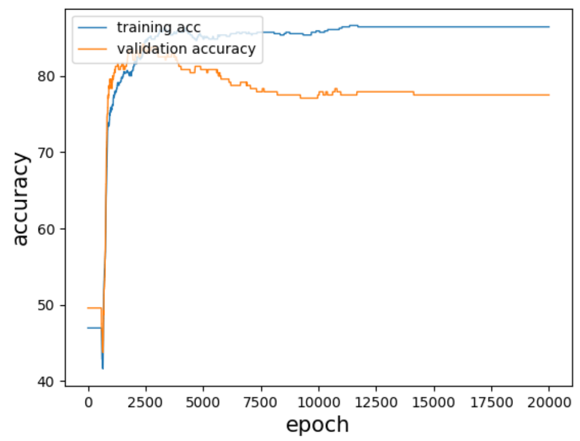
- This dataset is simple with almost no noise in it. the model can easily differentiate between the two classes by learning parameters that are kind of making a circular boundary to classify the X.

Initial Learning:

Initial Parameters	training loss	training and validation acc
• Model = **[2, 2, 1]** • epochs = **2000** • lr= **0.001** • activation function=**sigmoid** • loss = **cross-entropy**		
• This is the example of overfitting because we have very less data (1000). We are intentionally overfitting in order to check that our network is training well and weights are updated accurately in backpropagation.		

Epochs

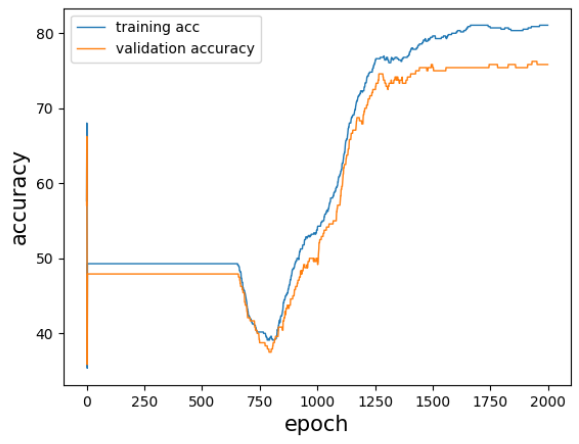
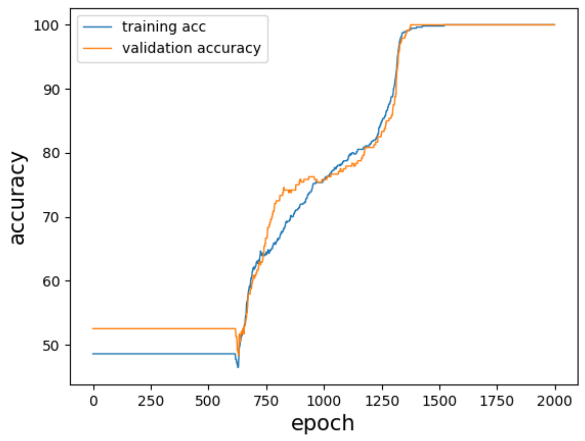
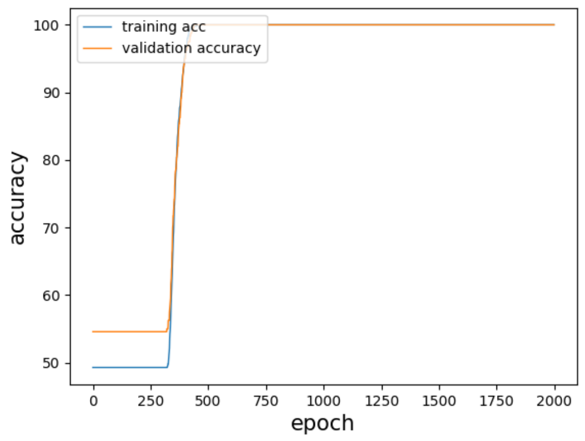
The initially model is retrained by only increasing the epochs.

 **MAX accuracy for training: 79**	 **MAX accuracy for training: 80**	 **MAX accuracy for training: 80**

- After increasing the number of epochs from 2000 to 5000 there is little gain in accuracy by 1%.
- But from 5000 to 20000, the accuracy didn't improve or the loss didn't decrease much because the hidden neurons are saturated and have less scope to learn. This means the model [2, 2, 1] does not have the **capacity to overfit the data**. Hence, we have to change some other hyperparameters in order to improve the accuracy. For this, I changed the number of neurons in the hidden layer.

Model

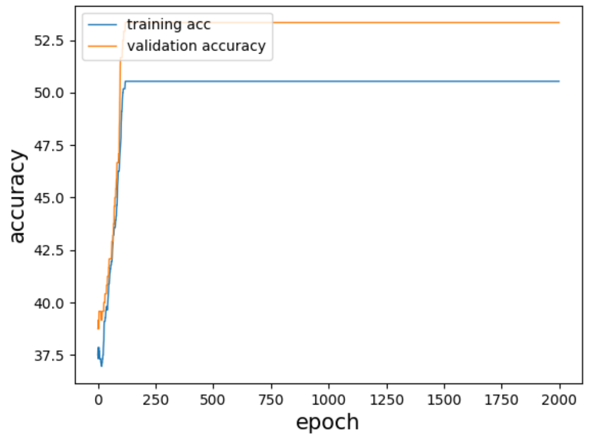
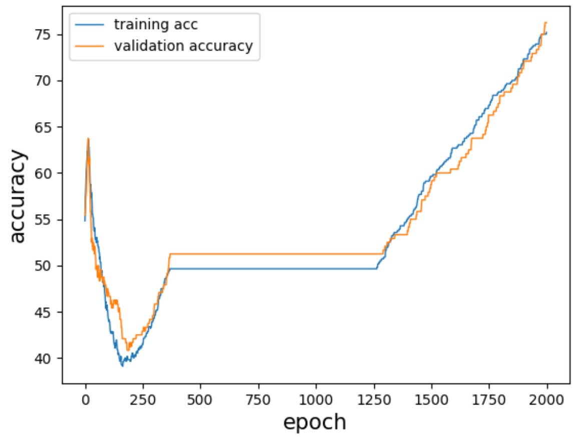
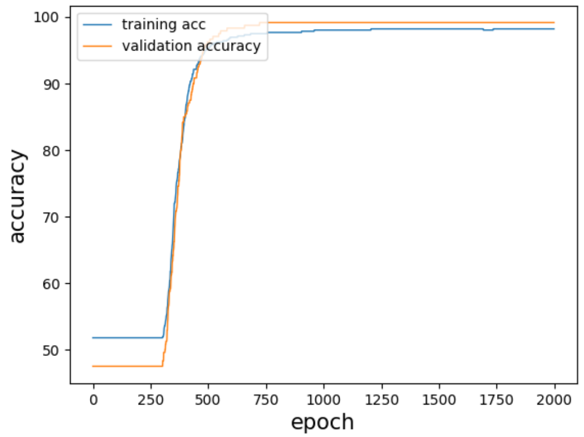
The initial model is retrained by changing the neurons in the middle layers.

[2, 2, 1]	[2, 3, 1]	[2, 5, 1]
 **testing accuracy: 75**	 **testing accuracy: 100**	 **testing accuracy: 100**

- The testing and training accuracy immediately improved when the hidden layer neurons were increased from 2 to 3 because more number of parameters contribute to classifying accurately.
- With 5 hidden neurons, the accuracy improved suddenly because within a few epochs the weights were updated accurately reaching 100% accuracy.

Learning Rate:

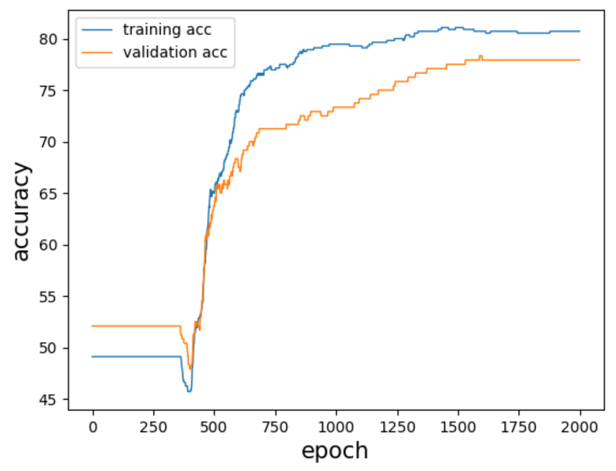
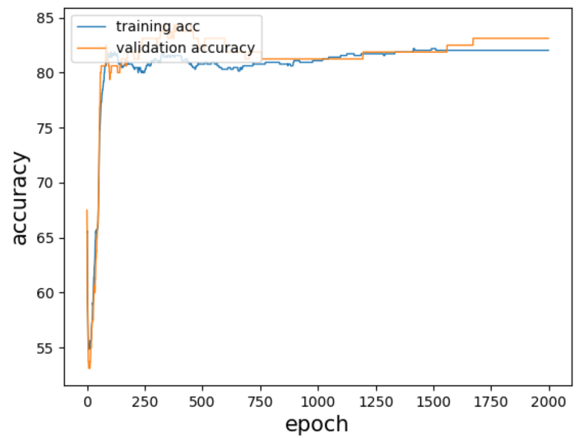
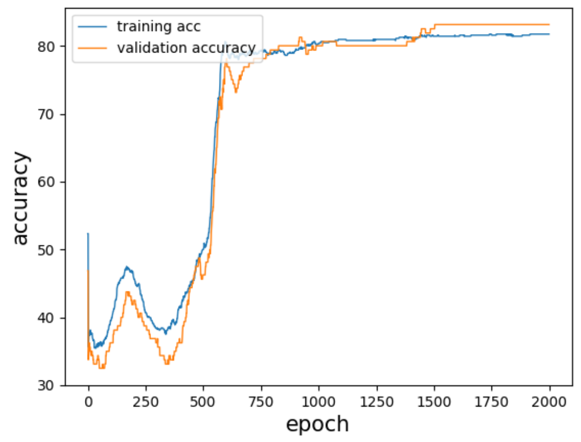
Initial model is retrained with different learning rates.

0.00001	**0.0001**	**0.001**
		

- All other parameters are similar to the initial ones.
- After increasing the learning rate by 10 times, the accuracy improved substantially, with the same number of epochs the network learned faster this time.
- Secondly, with the learning rate (0.00001), the model is **stuck on the local minimum** so that's why it is unable to improve the accuracy.

Activation Functions

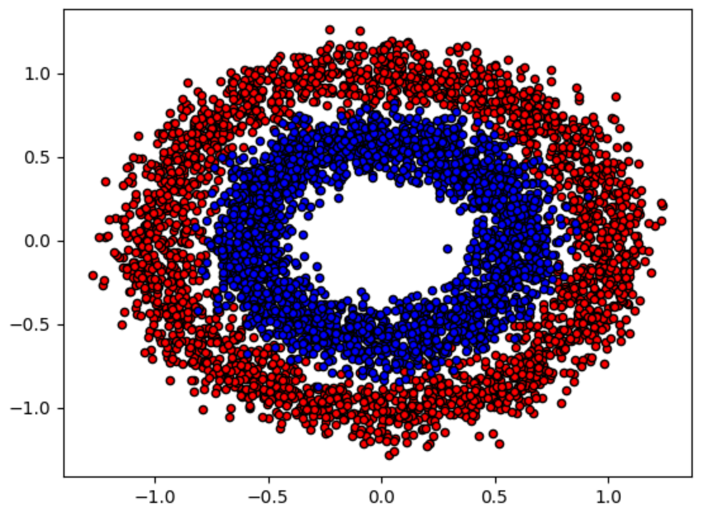
Output of initial model with different activation fuctions.s

sigmoid	tanh	relu
		
• The behaviour with each activation function is different and notable. ○ With sigmoid, the network learns gradually. ○ With the tanh, there is a sharp decrease in accuracy and then a sharp increase leading to 80% accuracy. This shows that the tanh adds rigorous changes to the model learning. ○ With the relu, there is high noise in training this neuron doesn't learn when the input is lesser the 0, because the output is 0 hence deltaW is 0		

Second Data

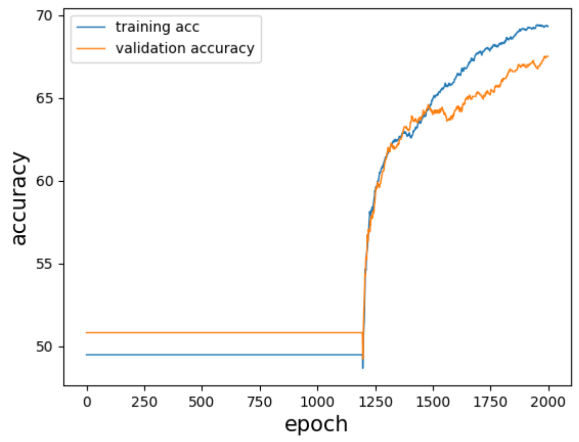
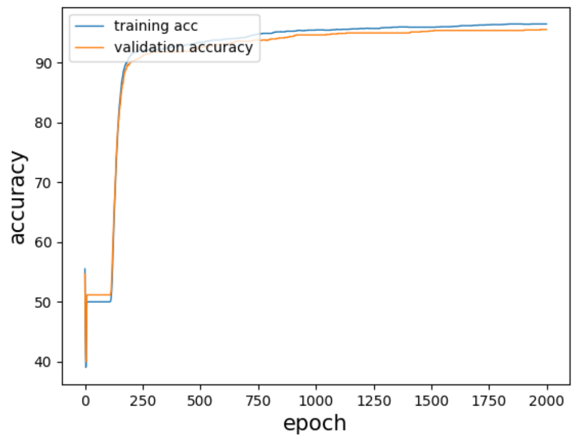
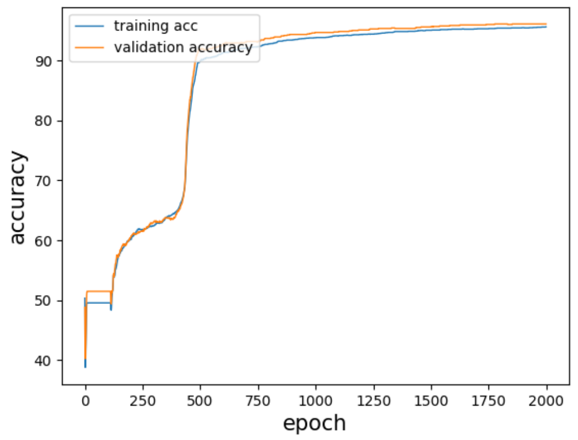
This is the first data distribution that is used to train the neural network.

```
data, label = ds.make_circles(n_samples=5000, factor=.6, noise=0.1)
```



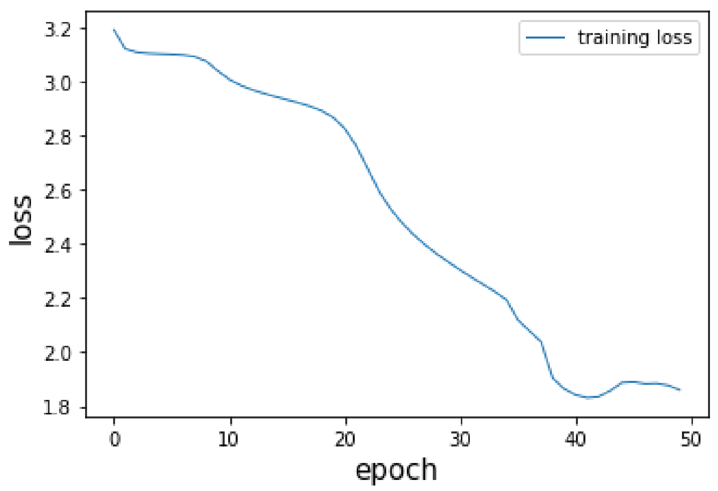
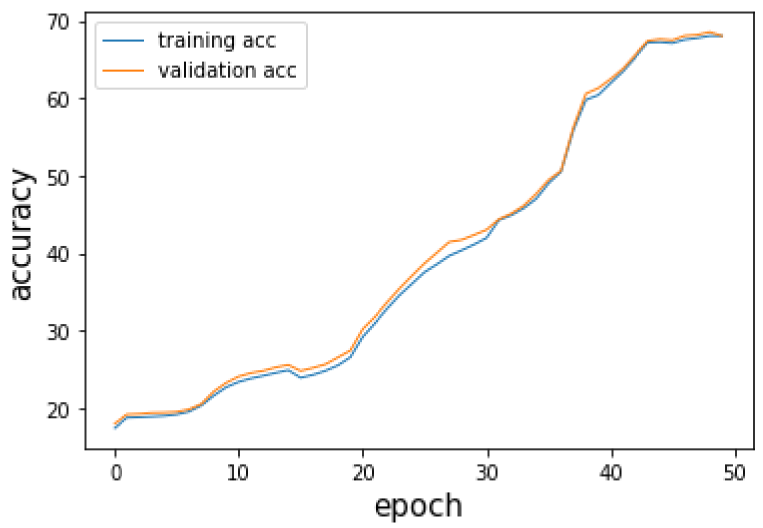
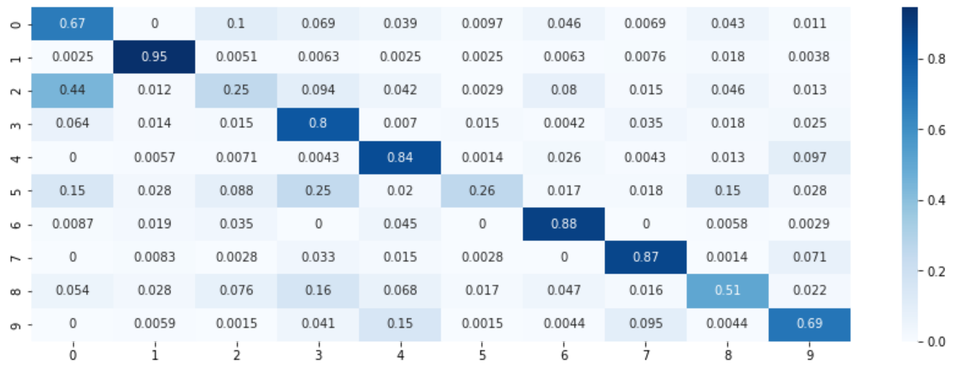
- This dataset is complex as compared to previous one and with more noise in it. So, here we will fit the same models and see how well they perform on this dataset.

Model

[2, 2, 1]	[2, 3, 1]	[2, 5, 1]
 **testing accuracy: 67**	 **testing accuracy: 96**	 **testing accuracy: 96**
● The point here is that, when the noise increases and the dataset become complex, the performance of the same model decreased from 100% to 96%.		

Task 3

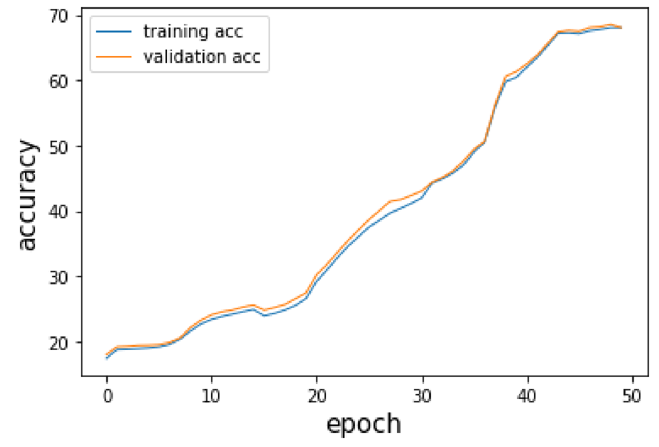
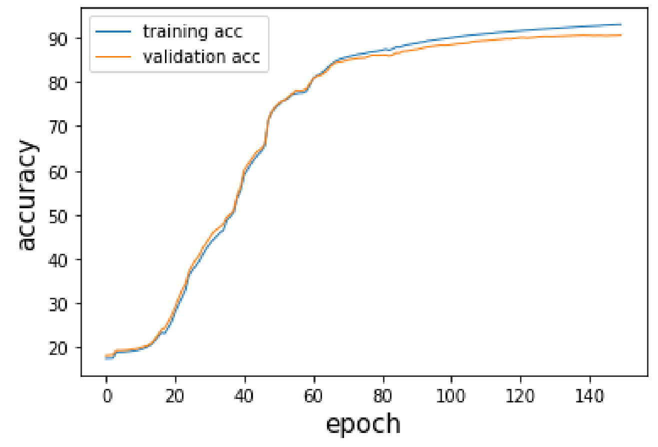
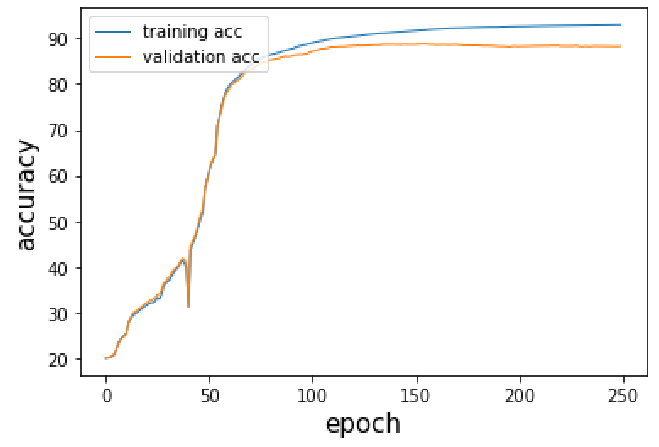
Initial Learning:

Initial Parameters	training loss	training and validation acc
• Model = **[784, 128, 64, 10]** • epochs = 50 • lr= 0.001 • activation function=sigmoid • LL activation = Softmax • loss = cross-entropy • train data = 0.9X • test data = 0.1X • vaidation datal = 0.09X • where X = whole data		
• Here the model is underfit, because of the lack of training. It is just to check whether everything is working fine and to compare the hyperparameters with each other.		
This is the confusion matrix in which maximum TP is for 1 (0.95) and minimum TP is for 4 (0.04)		

	0	1	2	3	4	5	6	7	8	9
0	0.67	0	0.1	0.069	0.039	0.0097	0.046	0.0069	0.043	0.011
1	0.0025	0.95	0.0051	0.0063	0.0025	0.0025	0.0063	0.0076	0.018	0.0038
2	0.44	0.012	0.25	0.094	0.042	0.0029	0.08	0.015	0.046	0.013
3	0.064	0.014	0.015	0.8	0.007	0.015	0.0042	0.035	0.018	0.025
4	0	0.0057	0.0071	0.0043	0.84	0.0014	0.026	0.0043	0.013	0.097
5	0.15	0.028	0.088	0.25	0.02	0.26	0.017	0.018	0.15	0.028
6	0.0087	0.019	0.035	0	0.045	0	0.88	0	0.0058	0.0029
7	0	0.0083	0.0028	0.033	0.015	0.0028	0	0.87	0.0014	0.071
8	0.054	0.028	0.076	0.16	0.068	0.017	0.047	0.016	0.51	0.022
9	0	0.0059	0.0015	0.041	0.15	0.0015	0.0044	0.095	0.0044	0.69

Epochs

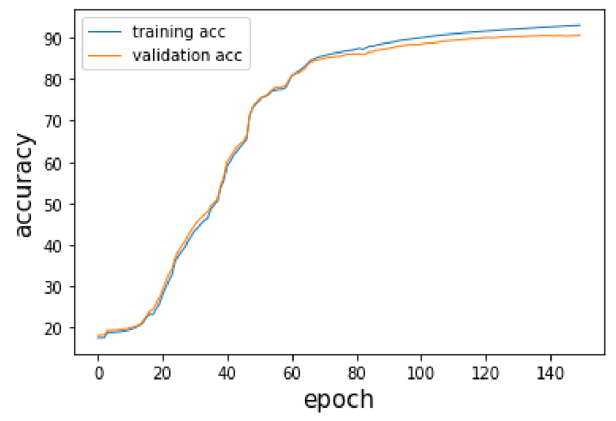
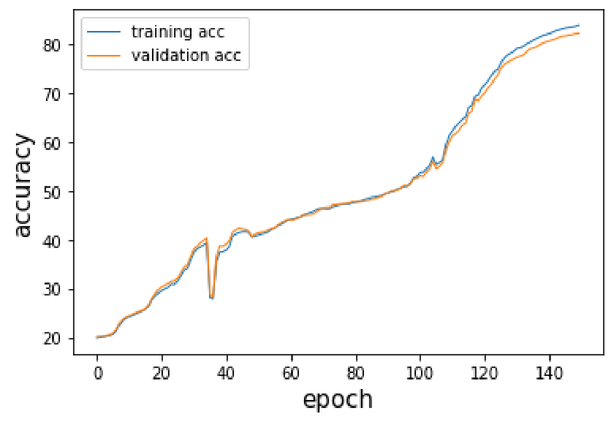
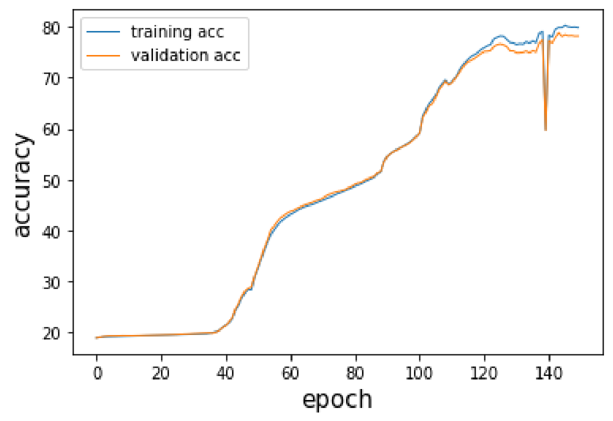
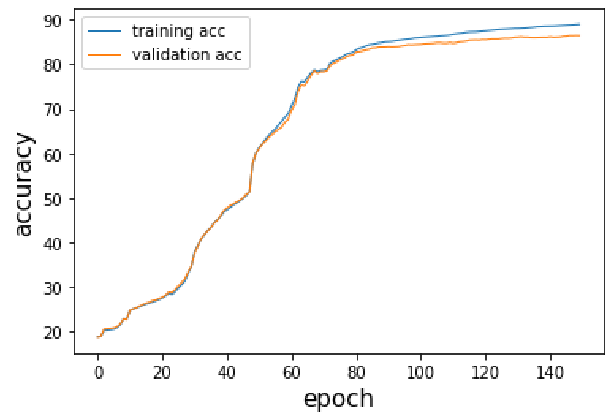
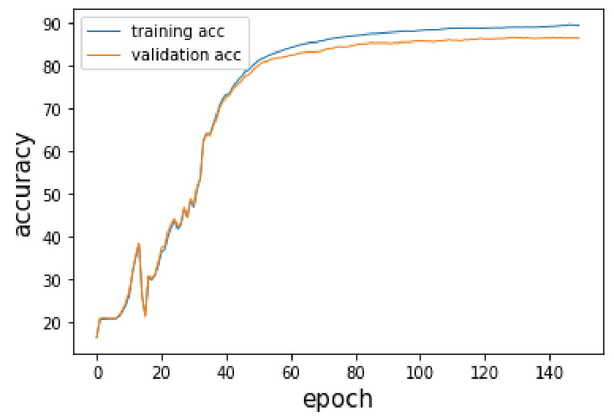
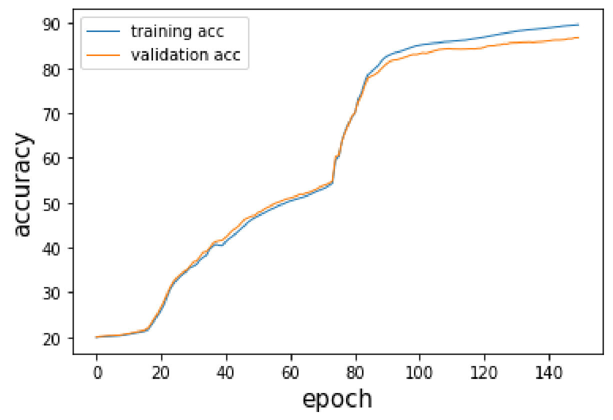
The result of changing the epochs on the initial training.

 **max_acc training: 67**	 **max_acc for training: 90**	 **max_acc for training: 93**

- As we can see from the graph the accuracy improved from increasing epoch from 50 to 150. At epoch 150 the accuracy is 90% that is good.
- But after increasing the epoch further our model overfit as the gap between validation accuracy and training accuracy increased.

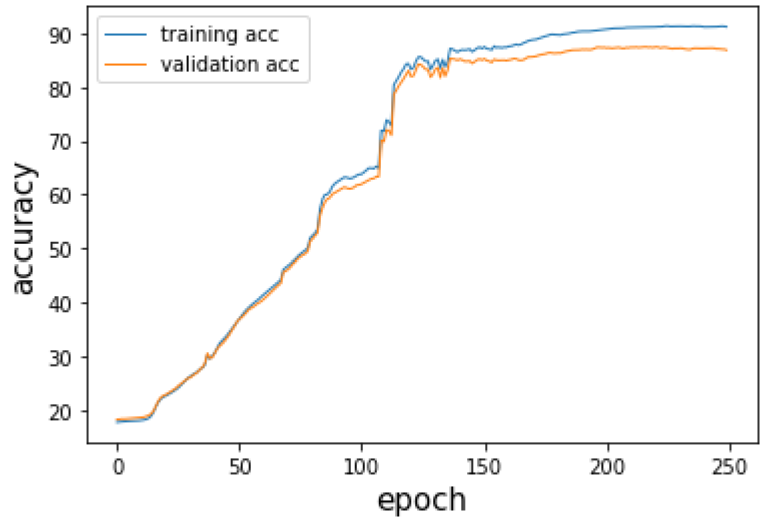
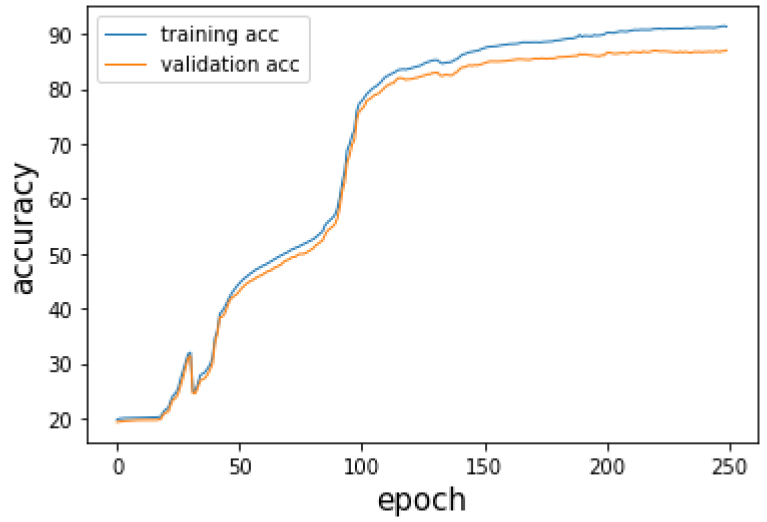
Model

All other hyperparameters are similar to the initial model.

1 - [784, 128, 64, 10]	2 - [784, 392, 64, 10]	3 - [784, 392, 32, 10]
 **testing accuracy: 89**	 **testing accuracy: 81.5**	 **testing accuracy: 77**
4 - [784, 128, 32, 10]	**5 - [784, 128, 128, 10]**	**6 - [784, 256, 64, 10]**
 **testing accuracy: 86.8**	 **testing accuracy: 86.7**	 **testing accuracy: 86.4**

- It can be seen that models 1, 3, 4, and 5 have already been overfitted so there is no need to train them further.
- Model 2 and 3 have further learning capacity so, we can increase their epoch to 250 and see the results.

- **Training Model 2 and 3 Further because they are underfitted.**
- In these two models, the learning rate is increased to **250**.

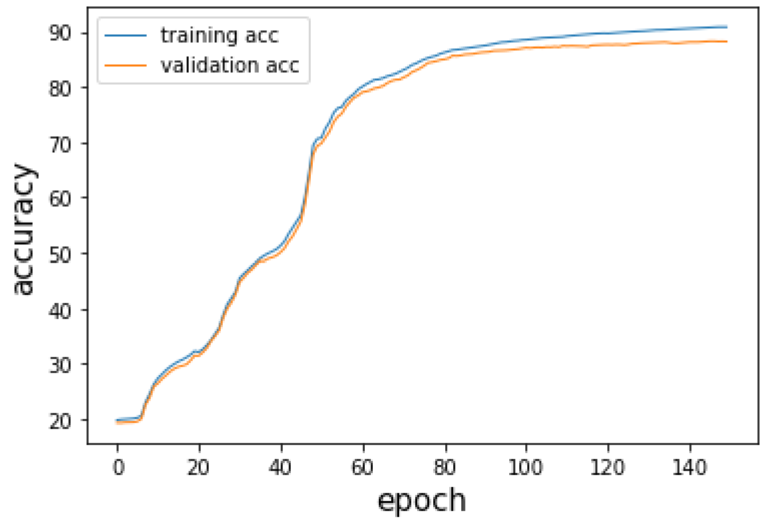
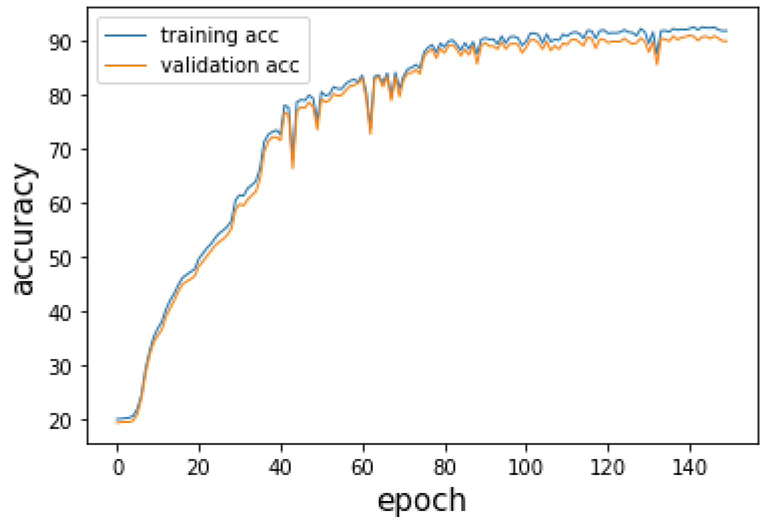
[784, 392, 32, 10]	**[784, 392, 64, 10]**
 **testing accuracy: 87.54**	 **testing accuracy: 87.02**
There is an increase in the test accuracy however that accuracy was already achieved by the simpler models. So the final model selected for further processing is s The issue that occurred after 120 iterations because the learning rate(0.01) was high due to which the is high fluctuation in the accuracy.	

-
- The learning Rate selected was 0.01, and there is no need to test the model with different learning rates because we have already reached a point where the model starts to overfit in a feasible time.

- The model selected for the next experiment is **[784, 128, 64, 10]** because it has maximum testing accuracy. Secondly, this is a simple model have fewer neurons which means less computation cost. It also shows that simple models can also perform better on the given data, so it also depends on the data distribution and complexity.
- Our model can **further be improved by comparing results with different numbers of layers.** But there is some issue with my code, so couldn't test it out.

Activation Functions with these parameters

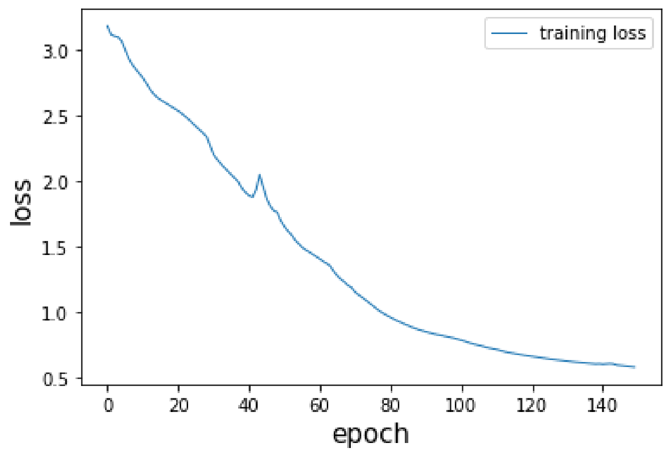
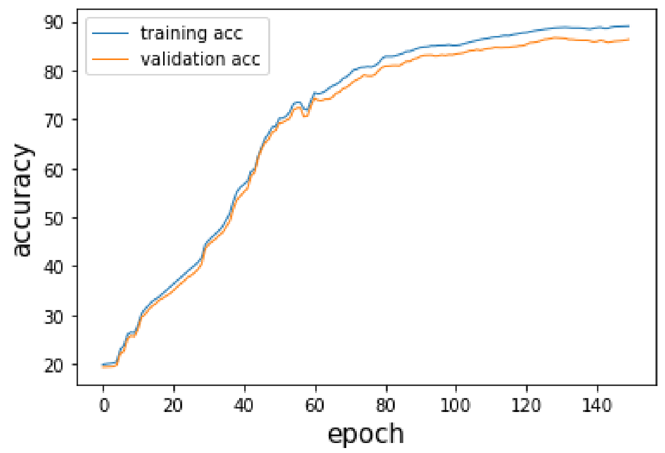
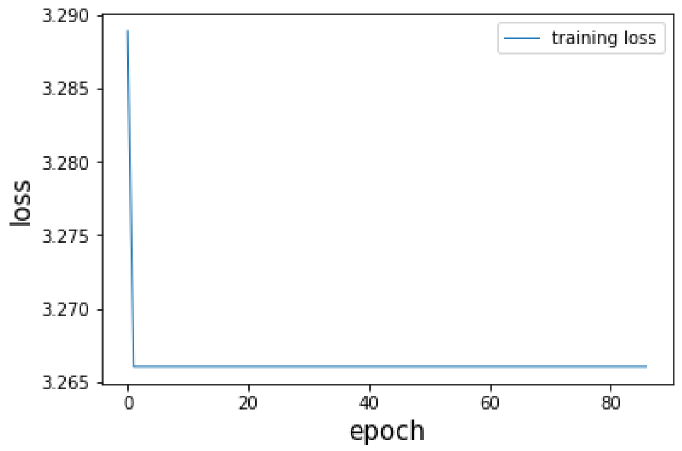
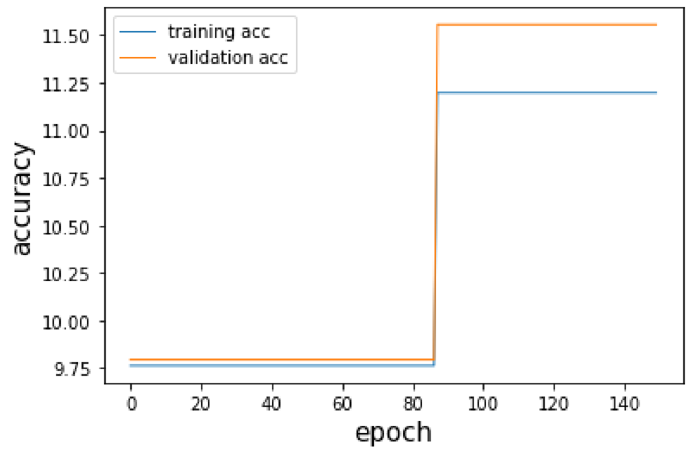
Model = [784, 128, 64, 10], Learning Rate = 0.01, Epochs = 250

[sigmoid, sigmoid, softmax]	**[relu, sigmoid, softmax]**
 **Testing accuracy: 89**	 **Testing accuracy: 91**

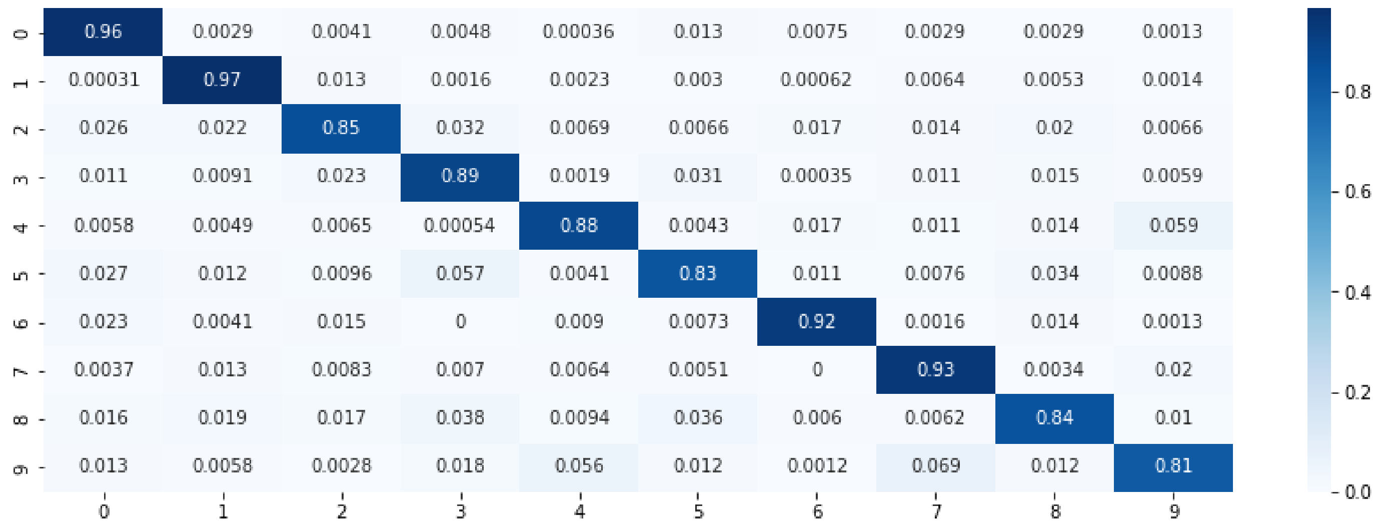
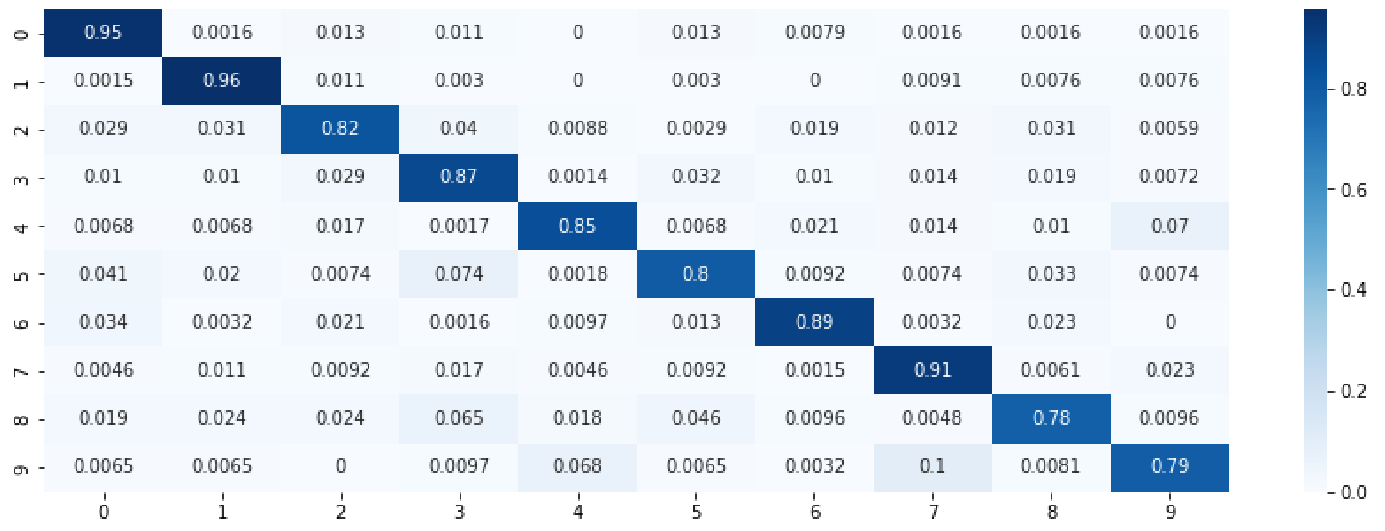
- The behavior with each activation function is different and notable.
  - With sigmoid, the network learns gradually.
  - With relu, there is a noise in the learning, this is because data is distributed from **-1 to 1** and the output of relu is 0 for -1. This may have caused the noise. There is the overflow encountered in exp **(return 1/(1+np.exp(-s)).**

Best Model

Model = [784, 128, 64, 10], Learning Rate = 0.01, Epochs = 150, Activations = [sigmoid, sigmoid, softmax]

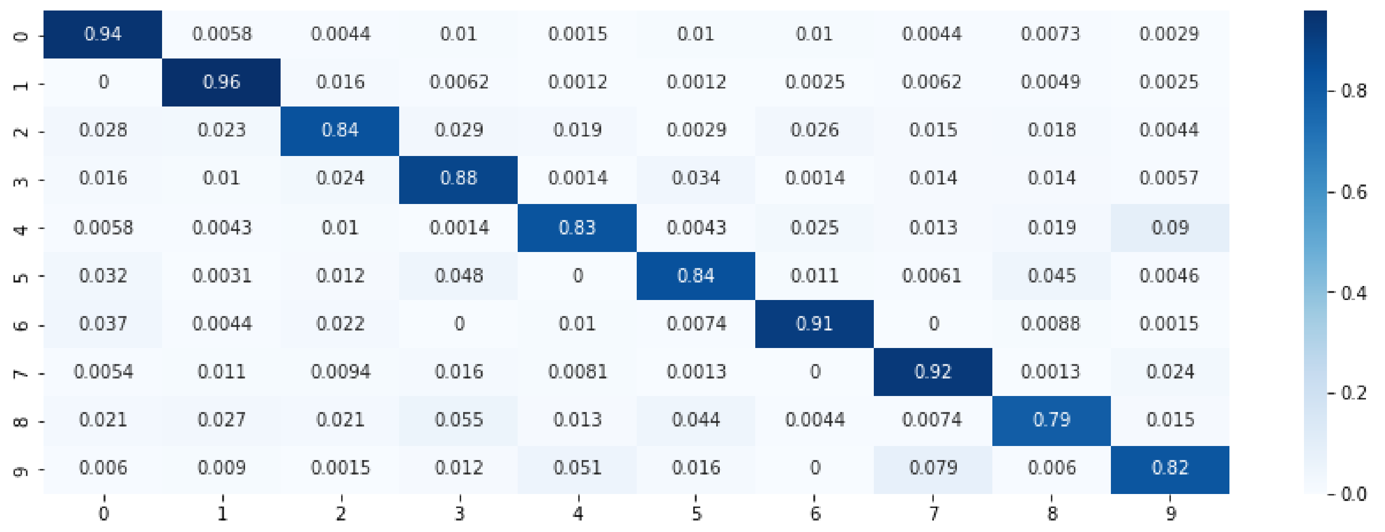
Loss Curve (With Mean Subtraction)	**Accuracy Curve (With Mean Subtraction)**
	
Loss Curve (Without Mean Subtraction)	**Accuracy Curve (Without Mean Subtraction)**
	

Confusion Matrix Results

Data	Confusion Matrix
Training Data	
Validation Data	

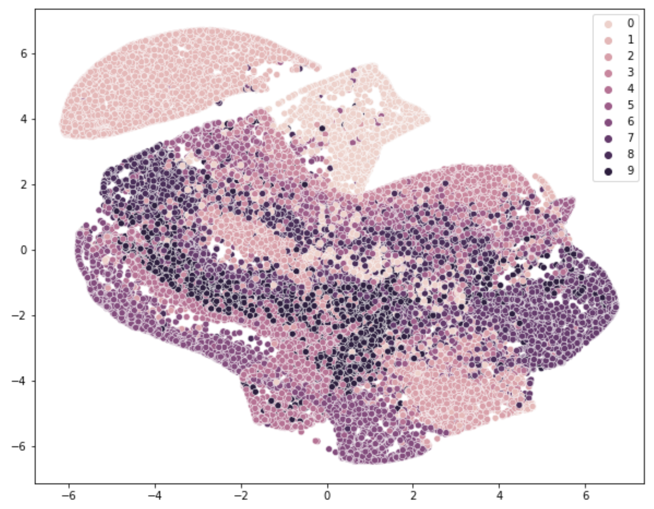
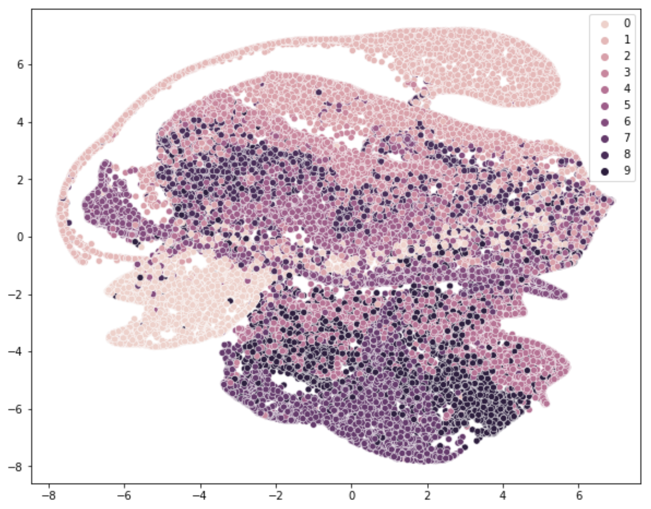
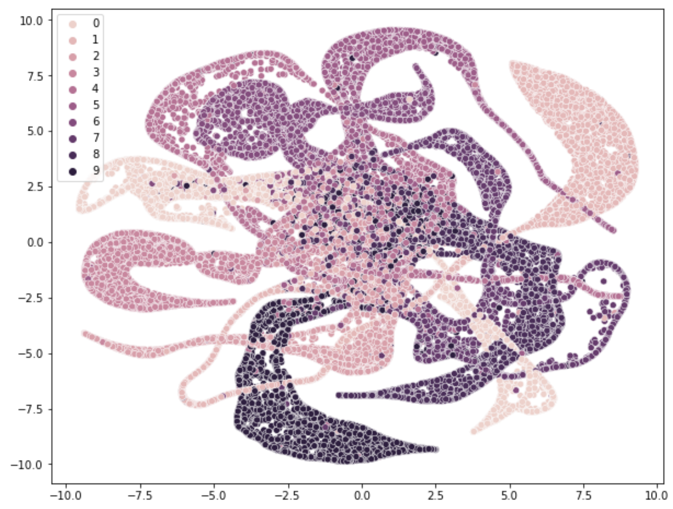
	0	1	2	3	4	5	6	7	8	9
0	0.96	0.0029	0.0041	0.0048	0.00036	0.013	0.0075	0.0029	0.0029	0.0013
1	0.00031	0.97	0.013	0.0016	0.0023	0.003	0.00062	0.0064	0.0053	0.0014
2	0.026	0.022	0.85	0.032	0.0069	0.0066	0.017	0.014	0.02	0.0066
3	0.011	0.0091	0.023	0.89	0.0019	0.031	0.00035	0.011	0.015	0.0059
4	0.0058	0.0049	0.0065	0.00054	0.88	0.0043	0.017	0.011	0.014	0.059
5	0.027	0.012	0.0096	0.057	0.0041	0.83	0.011	0.0076	0.034	0.0088
6	0.023	0.0041	0.015	0	0.009	0.0073	0.92	0.0016	0.014	0.0013
7	0.0037	0.013	0.0083	0.007	0.0064	0.0051	0	0.93	0.0034	0.02
8	0.016	0.019	0.017	0.038	0.0094	0.036	0.006	0.0062	0.84	0.01
9	0.013	0.0058	0.0028	0.018	0.056	0.012	0.0012	0.069	0.012	0.81

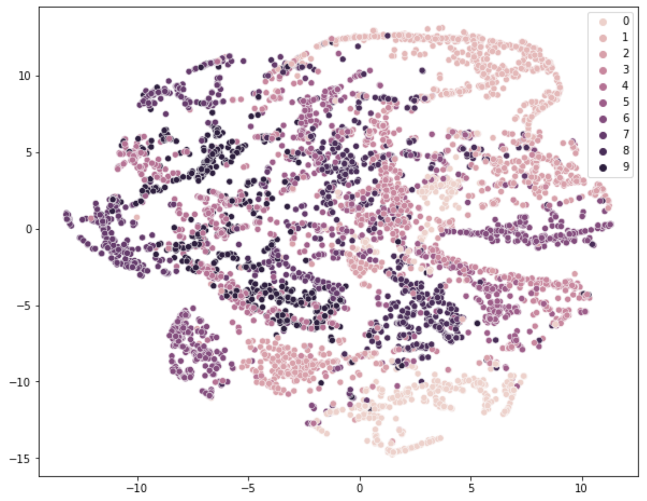
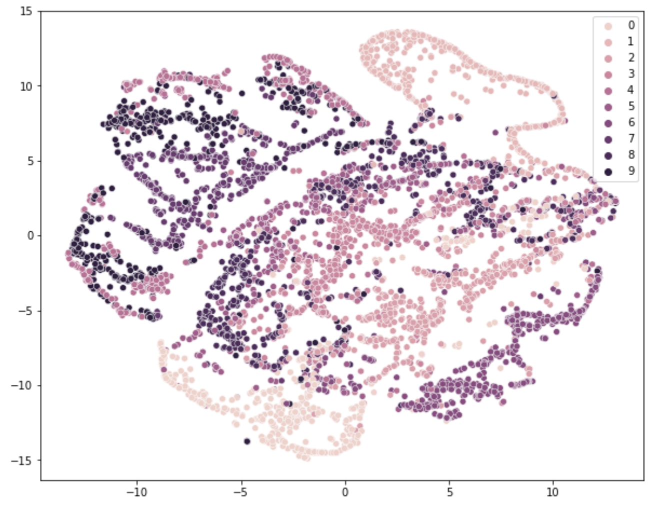
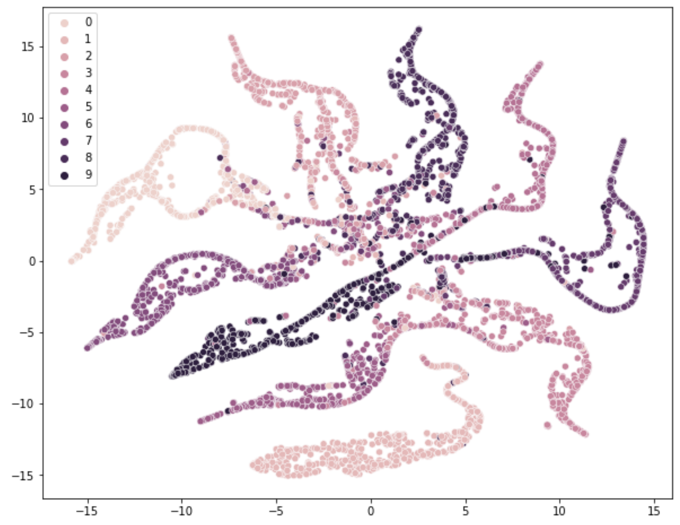
	0	1	2	3	4	5	6	7	8	9
0	0.95	0.0016	0.013	0.011	0	0.013	0.0079	0.0016	0.0016	0.0016
1	0.0015	0.96	0.011	0.003	0	0.003	0	0.0091	0.0076	0.0076
2	0.029	0.031	0.82	0.04	0.0088	0.0029	0.019	0.012	0.031	0.0059
3	0.01	0.01	0.029	0.87	0.0014	0.032	0.01	0.014	0.019	0.0072
4	0.0068	0.0068	0.017	0.0017	0.85	0.0068	0.021	0.014	0.01	0.07
5	0.041	0.02	0.0074	0.074	0.0018	0.8	0.0092	0.0074	0.033	0.0074
6	0.034	0.0032	0.021	0.0016	0.0097	0.013	0.89	0.0032	0.023	0
7	0.0046	0.011	0.0092	0.017	0.0046	0.0092	0.0015	0.91	0.0061	0.023
8	0.019	0.024	0.024	0.065	0.018	0.046	0.0096	0.0048	0.78	0.0096
9	0.0065	0.0065	0	0.0097	0.068	0.0065	0.0032	0.1	0.0081	0.79

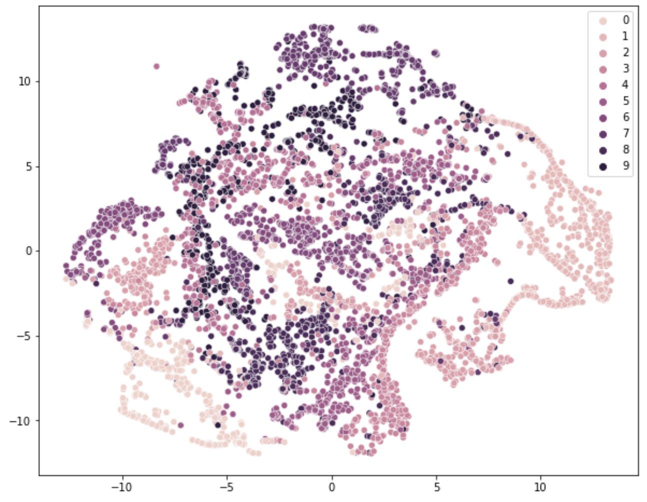
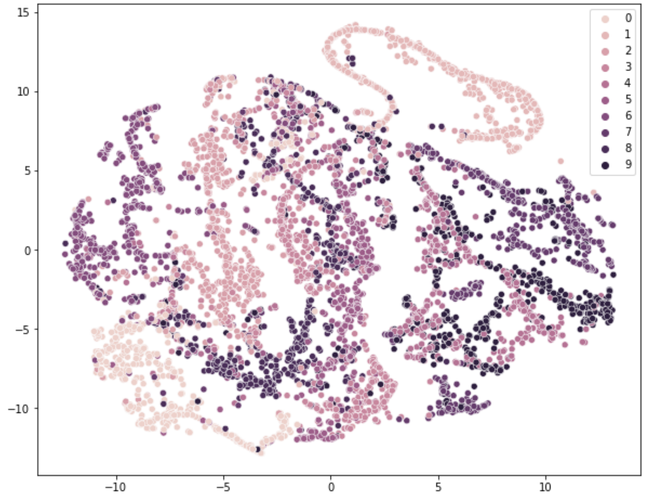
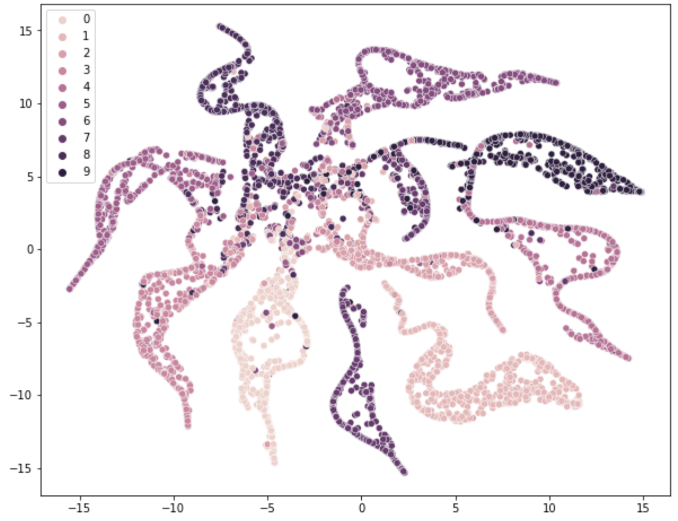
Testing Data	

	0	1	2	3	4	5	6	7	8	9
0	0.94	0.0058	0.0044	0.01	0.0015	0.01	0.01	0.0044	0.0073	0.0029
1	0	0.96	0.016	0.0062	0.0012	0.0012	0.0025	0.0062	0.0049	0.0025
2	0.028	0.023	0.84	0.029	0.019	0.0029	0.026	0.015	0.018	0.0044
3	0.016	0.01	0.024	0.88	0.0014	0.034	0.0014	0.014	0.014	0.0057
4	0.0058	0.0043	0.01	0.0014	0.83	0.0043	0.025	0.013	0.019	0.09
5	0.032	0.0031	0.012	0.048	0	0.84	0.011	0.0061	0.045	0.0046
6	0.037	0.0044	0.022	0	0.01	0.0074	0.91	0	0.0088	0.0015
7	0.0054	0.011	0.0094	0.016	0.0081	0.0013	0	0.92	0.0013	0.024
8	0.021	0.027	0.021	0.055	0.013	0.044	0.0044	0.0074	0.79	0.015
9	0.006	0.009	0.0015	0.012	0.051	0.016	0	0.079	0.006	0.82

- For each class, the training accuracy is greater than the validation and testing accuracy.

Training Data t-sne output		
hidden layer 1 output	**hidden layer 2 output**	**final layer output**
		

Validation Data t-sne output		
hidden layer 1 output	**hidden layer 2 output**	**final layer output**
		

Training Data t-sne output		
hidden layer 1 output	**hidden layer 2 output**	**final layer output**
		

- t-sne basically show how separated each class is after each layer, so for each data, train, test, validation, the classes are well separated after the output layer.