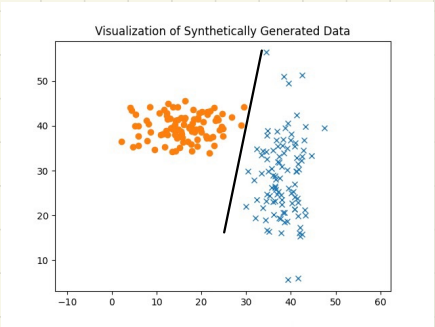
Part 1

A. Looking at the code above, we can't tell if the dataset will be linearly separable. To generate the dataset, we generate 2 Gaussian distributions with the mean of centers along each dimension chosen randomly and uniformly. In addition, each dimension is independent of each other. The means of the distributions could be close together and as a result the dataset might not be linearly separable. However, it is possible to get a linearly separable dataset.



We see that the generated data is linearly separable. It is possible to get a linearly separable dataset so we do expect it.

B. No, it is not linearly separable.

C. 1. Random weight initialization
2. Random train/test splits
3. Random batch shuffling

Part 2

A. $$\frac{\partial L}{\partial a_i^l} = \sum_{j=1}^{d^{(l)}} \frac{\partial L}{\partial a_j^{l+1}} \frac{\partial a_j^{l+1}}{\partial z_j^l} \frac{\partial z_j^l}{\partial a_i^l}$$

B. $$\frac{\partial L}{\partial z_j^l} = \frac{\partial L}{\partial a_j^{l+1}} \cdot \frac{\partial a_j^{l+1}}{\partial z_j^l}$$

C. $$\frac{\partial L}{\partial w_{ij}^l} = \frac{\partial L}{\partial a_j^{l+1}} \cdot \frac{\partial a_j^{l+1}}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{ij}^l} = \frac{\partial L}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{ij}^l}$$

$$\frac{\partial L}{\partial b_j^l} = \frac{\partial L}{\partial a_j^{l+1}} \cdot \frac{\partial a_j^{l+1}}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l} = \frac{\partial L}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l}$$

D. $$\frac{\partial a_j^{l+1}}{\partial z_j^l} = \theta'(z_j^l), \quad \frac{\partial z_j^l}{\partial a_i^l} = w_{ij}^l$$

$$\frac{\partial z_j^l}{\partial w_{ij}^l} = a_i^l, \quad \frac{\partial z_j^l}{\partial b_j^l} = 1$$

$$\Rightarrow \boxed{\begin{aligned} \frac{\partial L}{\partial z_j^l} &= \frac{\partial L}{\partial a_j^{l+1}} \cdot \theta'(z_j^l) \\ \frac{\partial L}{\partial w_{ij}^l} &= \frac{\partial L}{\partial a_j^{l+1}} \cdot \theta'(z_j^l) \cdot a_i^l \\ \frac{\partial L}{\partial b_j^l} &= \frac{\partial L}{\partial a_j^{l+1}} \cdot \theta'(z_j^l) \\ \frac{\partial L}{\partial a_i^l} &= \sum_{j=1}^{d^{(l)}} \frac{\partial L}{\partial a_j^{l+1}} \cdot \theta'(z_j^l) \cdot w_{ij}^l \end{aligned}}$$

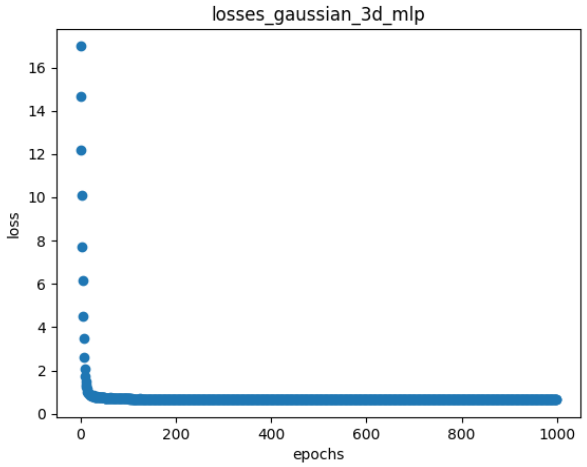
E. We formulate backpropagation in this manner because this algorithm makes it easier to train neural networks. Using the chain rule, we only have to compute the gradients for each parameter once and we can reuse these computations to compute gradients by propagating them backwards through the networks. As a result, this drastically reduces the time we would need to train these networks compared to if we had to recompute all of these gradients.
The two new terms we introduce are $\frac{\partial z_j^l}{\partial w_{ij}^l}$ and $\frac{\partial z_j^l}{\partial b_j^l}$. We know that $\frac{\partial z_j^l}{\partial b_j^l} = 1$ and $\frac{\partial z_j^l}{\partial w_{ij}^l} = a_i^l$, which makes the gradients $\frac{\partial L}{\partial w_{ij}^l}$ and $\frac{\partial L}{\partial b_j^l}$ easy to compute since we have all other terms.

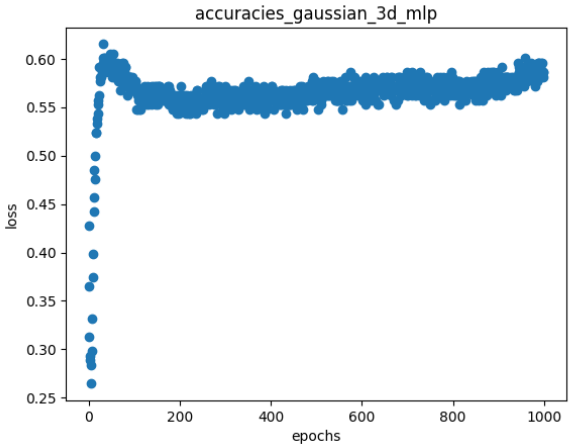
Part 3.

A. An MLP with 1 hidden layer would not suffice to correctly classify all points in the kCircles dataset. All the points of a class are in 1 circle, so the 2 classes should not be linearly separable since the classes are separated by a circle. An MLP with 1 hidden layer is essentially the same as a perceptron because there is no non-linearity, so it cannot learn non-linear functions.

B. No, an MLP with 1 hidden layer cannot correctly classify all points in the kIris dataset. 2/3 classes in the Iris dataset are not linearly separable from each other (from Iris dataset website), so an MLP with 1 hidden layer cannot correctly classify all points in the kIris dataset. The Iris dataset also has 3 target classes and an MLP with 1 hidden layer can only classify two targets correctly.

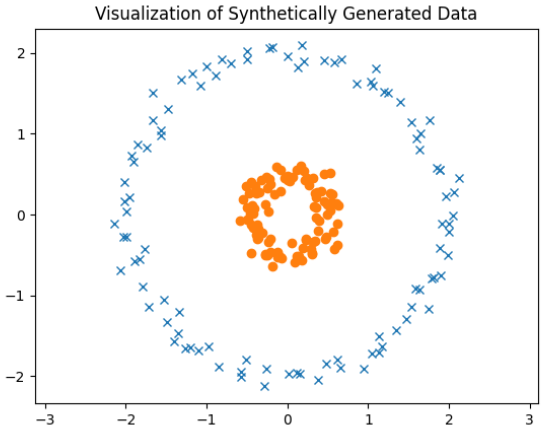
Part 2F Plots:

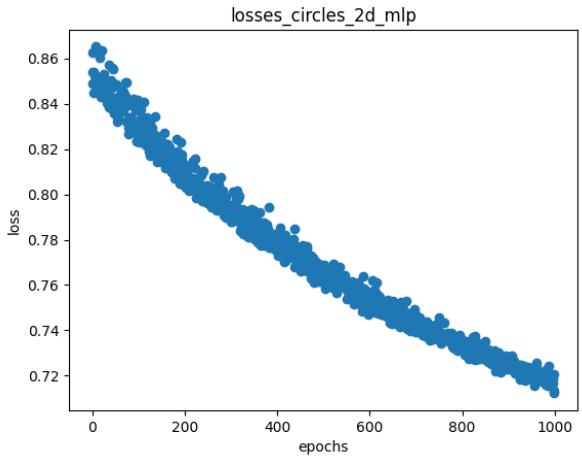


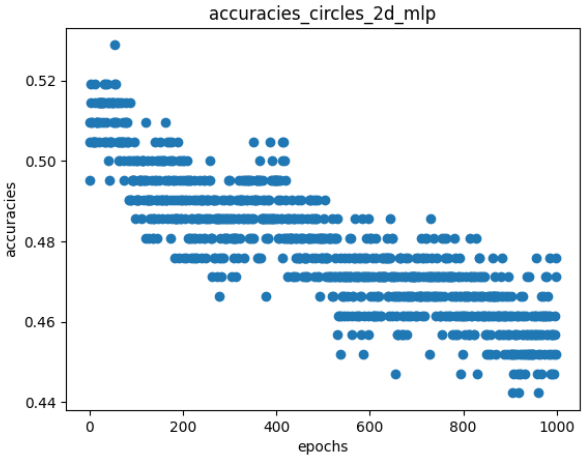


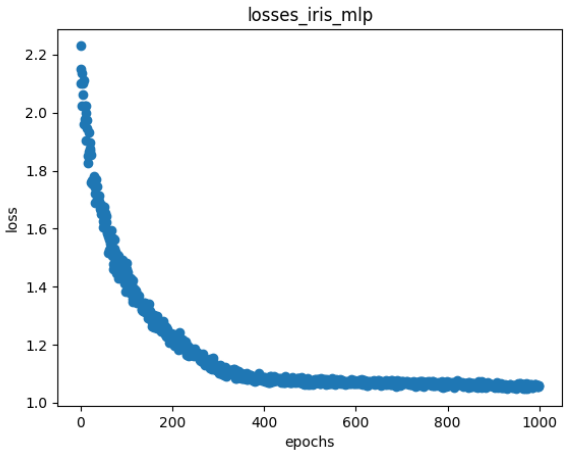
Part 3

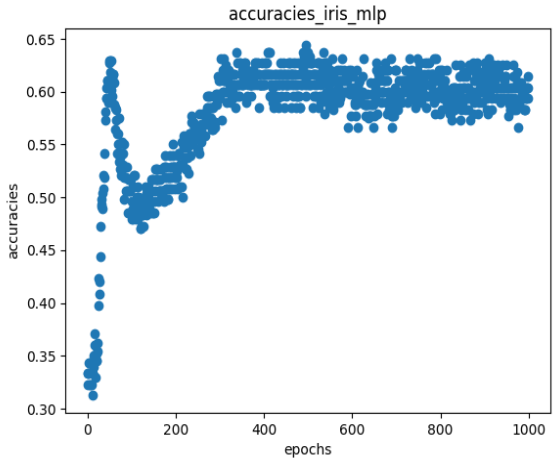
Plots:





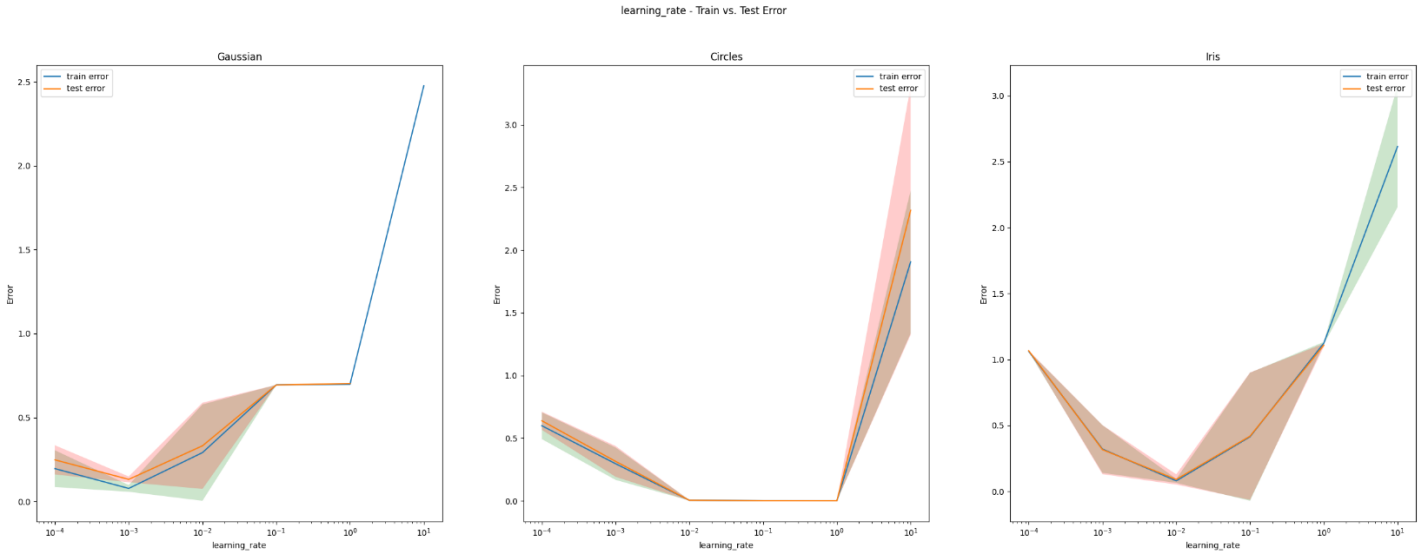






Part 4

4B. i) Visualize train and test loss against learning rate (5pts)



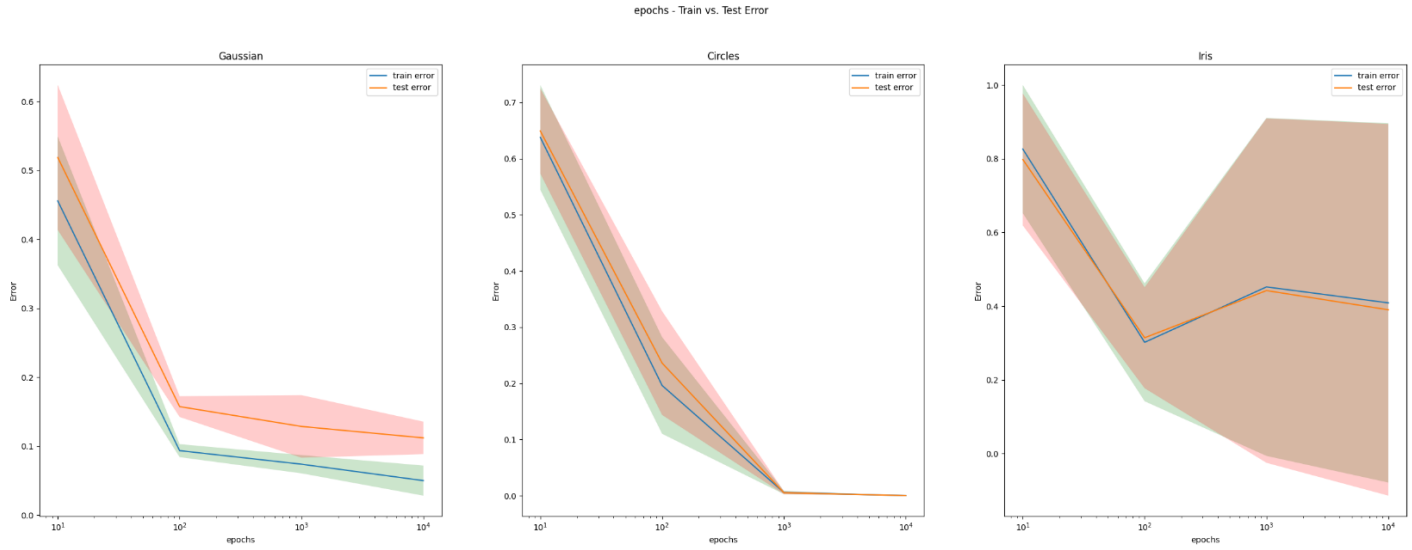
4B. ii) You may have noticed that for some experiments, the number of epochs was less than the default. This is because the loss either reached nan or infinity. Why does the loss explode when the learning rate is high? (2pts)

The loss explodes when the learning rate is high because high learning rates make the parameters update too drastically and diverge from the optimum, causing loss to explode.

4B. iii) What are some disadvantages when the learning rate is low (1pt)

When the learning rate is too low, the model will take a long time to converge because the parameters are updated with smaller magnitudes each epoch.

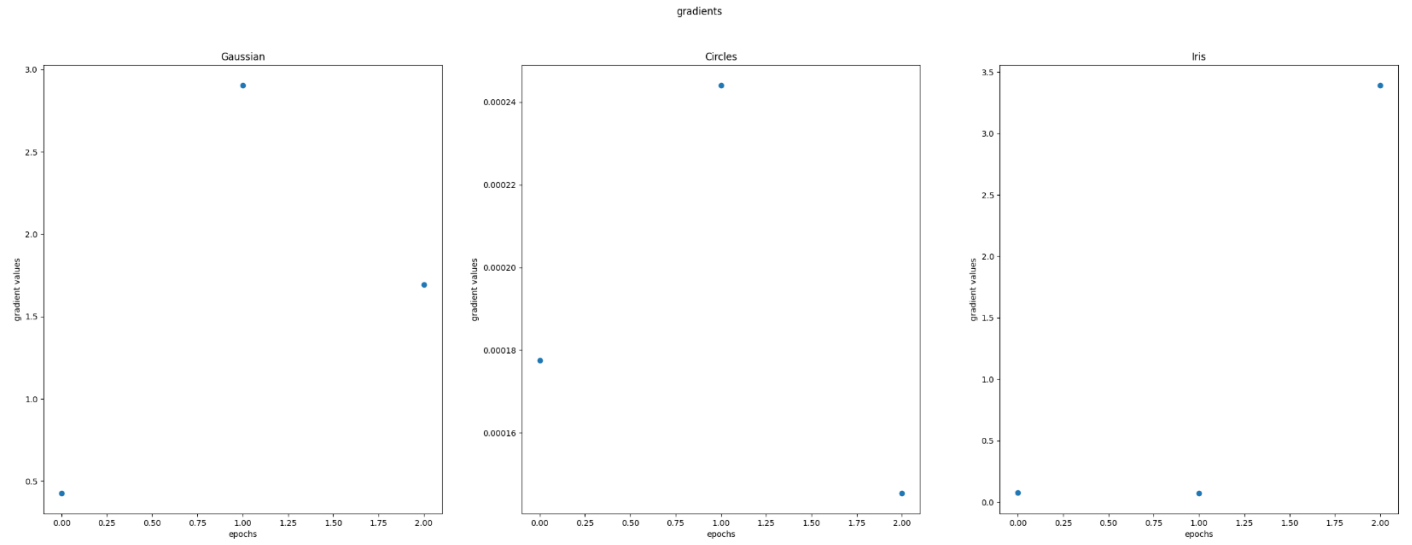
4C. i) Visualize train and test loss against number of epochs (x-axis should be logscale) (3pts)



4C. ii) When does the loss saturate for train? For test? (2pts)

	Train	**Test**
Gaussian	Does not saturate	Does not saturate
Circles	1000	1000
Iris	Does not saturate	Does not saturate

4C. iii) If the loss does not converge, comment on why there is no convergence. Plot the norm of the gradients over time to visualize (1pt)



The losses do not converge for Gaussian and Iris datasets because the MLP with 2 hidden layers is not complex enough to learn the non linear decision boundaries of the Gaussian and Iris datasets.

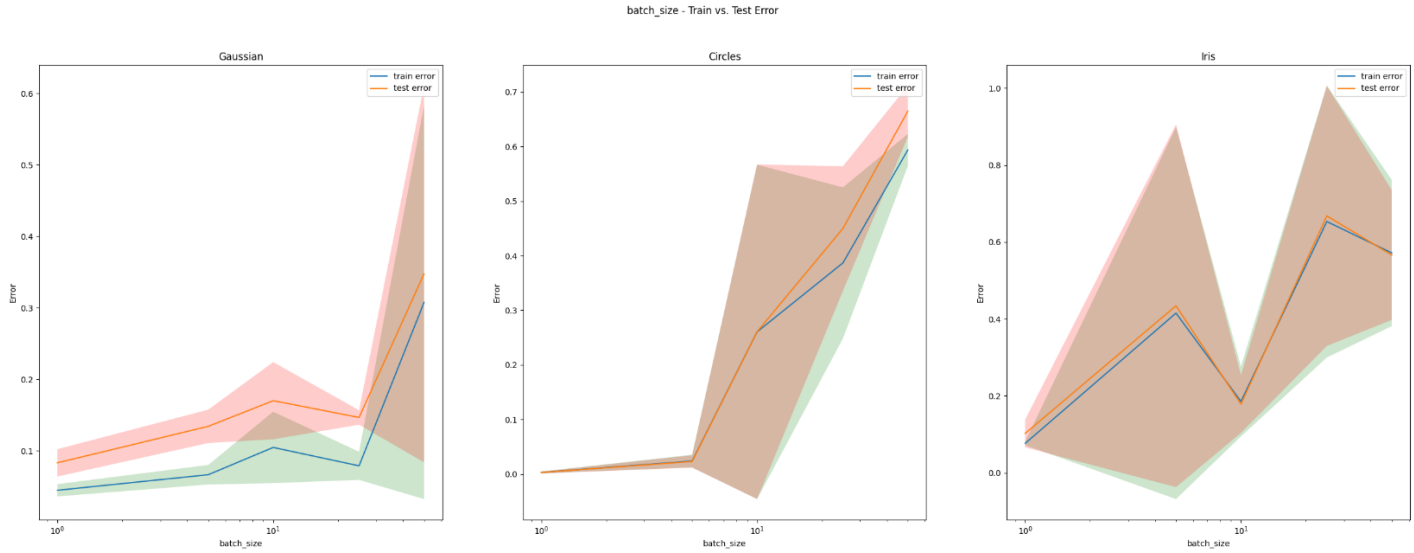
4C. iv) Do you observe overfitting? (1pt)

There could be slight overfitting for the Gaussian dataset from 1000 to 10000 epochs since the training error decreases slightly faster than the test error.

4C. v) What are some common techniques to prevent overfitting? List 3 (1pt)

1. *Early stopping*
2. *Regularization*
3. *Data augmentation*

4D. i) Set the batch size to [1,5,10,25,50] on all datasets. Plot the batch size vs. training error and testing error on the same plot. (3pts)



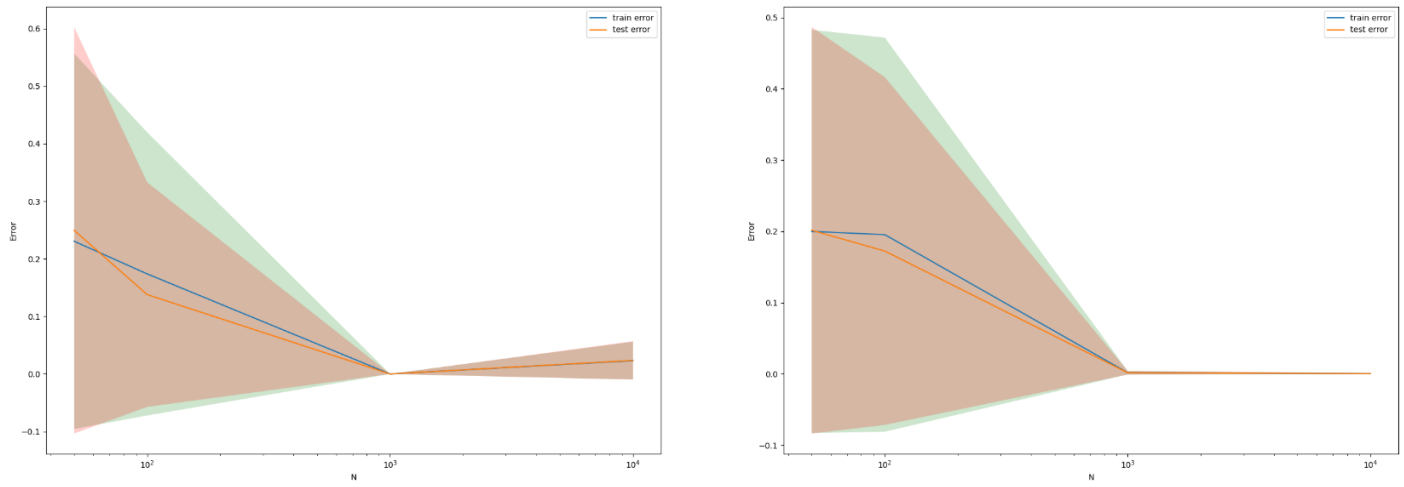
4D. ii) What effect does a larger batch size have on the errors? (2pts)

As batch size increases, both training and test error increases for all three datasets. This is what we expect as larger batch sizes should decrease the performance of the model.

4D. iii) What effect does a smaller batch size have on experiment runtime? (1pt)

A smaller batch size increases the experiment runtime because for every epoch, we have to make more forward and backward passes because there are more batches.

4E. i) Visualize train and test loss against the number of samples (x-log-scale plot) (3pts)



Left: Gaussian, Right: Circles

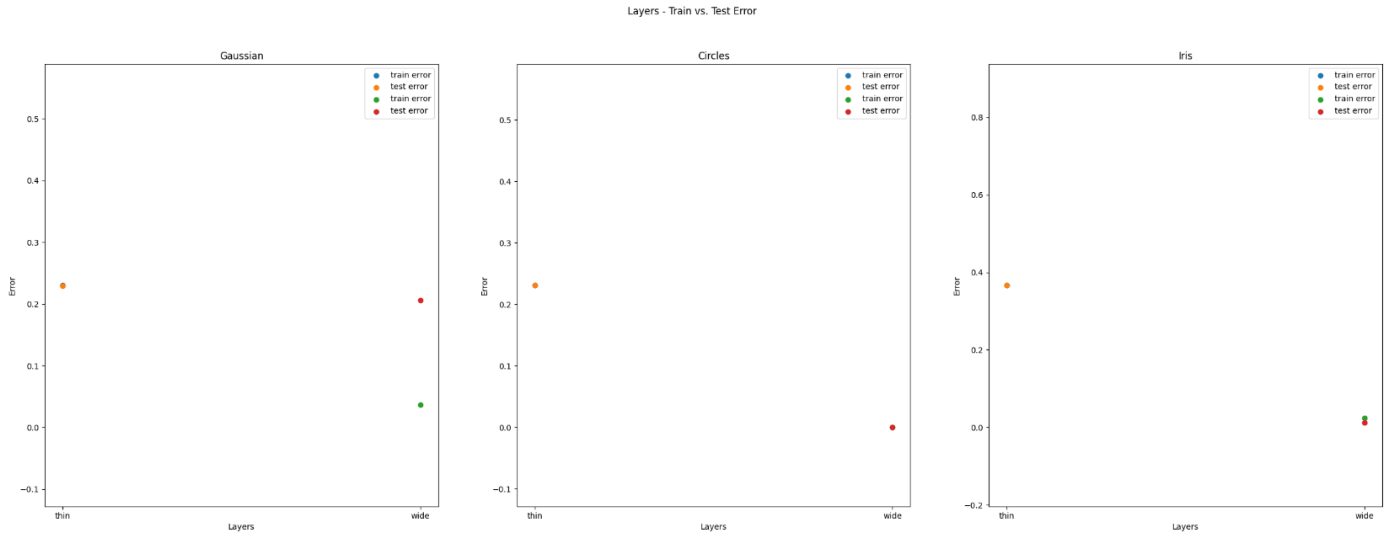
4E. ii) Comment on the result of the unbalanced dataset distributions [N=1000, M=100]. (2pts)

For the Gaussian dataset, the training and test error is higher with unbalanced dataset distributions compared to a balance of [N=1000, M=1000]. For the Circles dataset, we see that both training and test error is very low for the unbalanced dataset. It is possible that the loss is NaN or infinity for the Circles dataset since the loss is exactly at 0 and the variance is 0. This is what we expect for unbalanced dataset distributions since the model learns more from the class with more samples.

4E. iii) Comment on how the error changes for the different datasets. Why do you suppose the error changes as it does? (2pts)

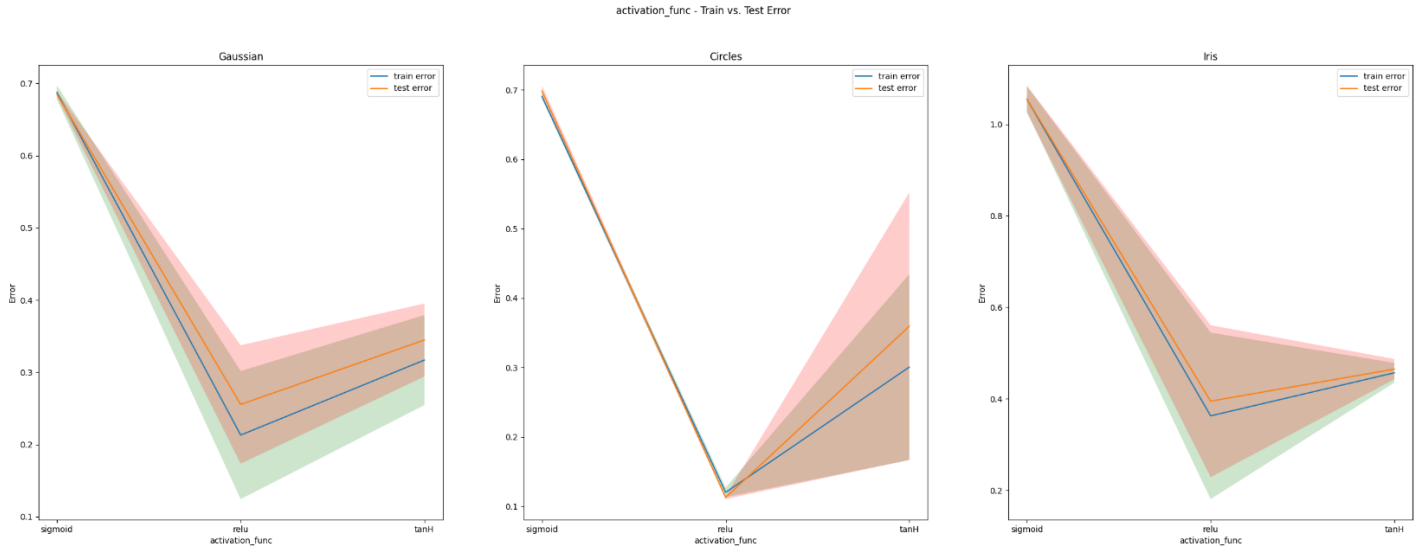
For both datasets, both training and test error decreases as the number of samples increases for balanced dataset distributions [10, 100, 1000] . As the number of samples increases, we have more batches so our network performs more updates each epoch. In most cases, this is what we expect.

4F. i) Comment on which network does better for Circles and the Iris Dataset. Comment on any intuition behind why that is. (5pt)



The wide network does better for both the Circles and Iris datasets. We see from the plots that both training and test error are lower with the wide network compared to the thin network. This is because with wider layers, we have more neurons per layer so we can learn more complex decision boundaries. Both the Circles and Iris datasets are not linearly separable, which means we need to learn non-linear decision boundaries so the wide network does better.

Part 5



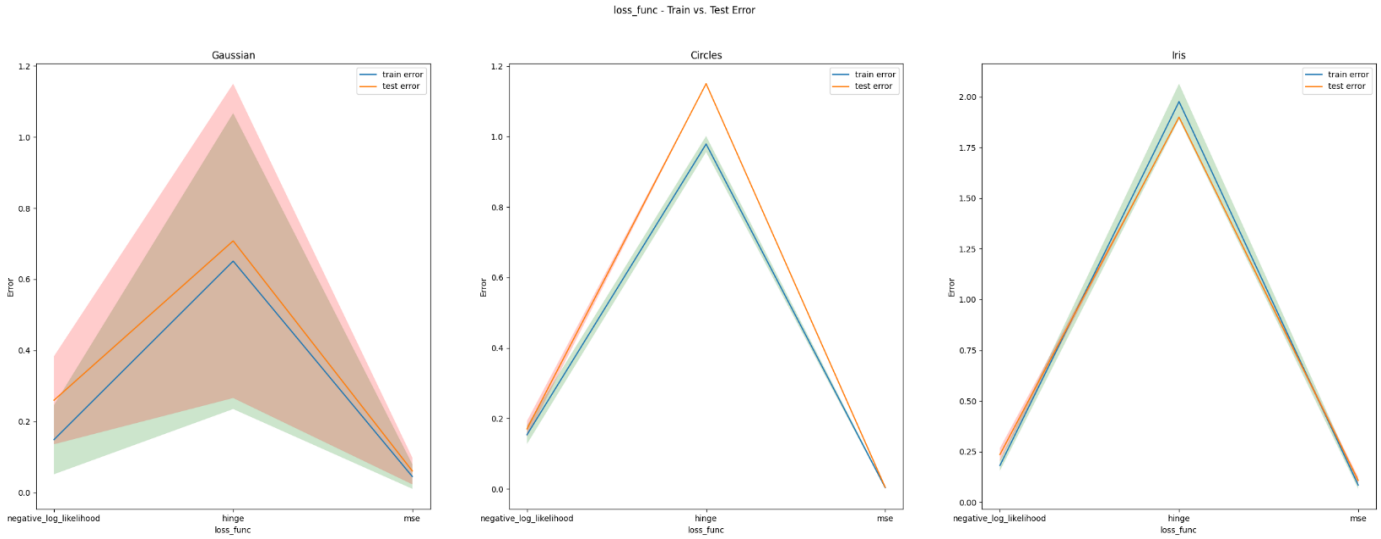
5A. Adjust the activation function to be a ReLU function. Comment on how/why performance adjusts. Plot the loss curves to compare using some of your plots from Part 5 to visualize this. (4pts)

When we change the loss function to ReLU, the training and test errors are much smaller on all datasets compared to when we use sigmoid activation. This is because sigmoid results in vanishing gradients while ReLU is non-saturating. Vanishing gradients means that weights cannot be updated so the network cannot learn, so performance is decreased.

5B. What happens when you replace the activation function to be tanh? (4pts)

When we use tanh we get better performance than sigmoid but is not better than ReLU for all datasets. The tanh function is still saturating, so we do not expect it to perform better than ReLU. The tanh function has larger derivatives than sigmoid so the network would converge faster to an optimum with tanh than sigmoid, which is why the loss is lower with tanh than sigmoid.

5C. Adjust the loss function from Negative Log Likelihood to Hinge loss and to MSE. Comment on why/how performance adjusts. (3pts)



Both training and test loss increases significantly with Hinge loss compared to Negative Log Likelihood and MSE. Hinge loss is not differentiable everywhere, which is a problem as we calculate gradients with respect to the loss and use these gradients to update the network parameters. If gradients are undefined, then the model will not learn so performance will be worse. For all datasets, both training and test loss are slightly lower with MSE compared to Negative Log Likelihood.