EE/CNS/CS 148 Topics in Computational Vision **April 21, 2023**

Problem Set 1

Name: Manuel Rodriguez **Late Submission:** No

Contents

1 Generate the Dataset

1.1 A

The data is going to be composed of points taken from two different, randomly-generated, Gaussian distributions. Without looking at the at plots, I would expect that the two classes are relatively different with the two having clearly different centers, but I would not expect the data to be linearly separable since it is unlikely that the centers are far apart and the uncertainties generated by both Gaussians are so small that there are no data points from the two distributions that are close together. Looking at the data, we can tell that there are two clear centers, but as expected, there is some overlapse in the middle of the graph.

1.2 B

For $dims = 3$, I think the data would be more likely to be linearly separable since there are more degrees of freedom for the points to be distributed across and thus there is less of a chance that they will disperse along the same axis or have nearby centers.

1.3 Controlling Randomness

1.3.1 C

An element of randomness that needs to be addressed for a perceptron is the set of points which are evaluated when determining error and updating weights. If these points are not chosen randomly, there would be a bias toward fitting for the first points in the model which could be fatal if the points are ordered, for example, by simple ascending order. Another element of randomness to consider is the initial values of all the weights, obviously they cannot be zero because then you face vanishing gradients, but also they must not be biased toward any particular target function, so a popular choice if initializing the weights to small, normally distributed values. Another element of randomness that could be addressed is the sampling of your data points. If your data points aren't actually sampled from the distribution you are trying to replicate (because of some bias in your system of sampling) the accuracy of your model on the test point will be poor and inaccurate.

1.3.2 D

Done :D

2 Compute Backprop + Code

2.1 A, B, C

(A) $\frac{\partial L}{\partial a_j^{l+1}} \overset{?}{\propto} \frac{\partial L}{\partial a_i^{l}}$ where $a_j^{l+1} = \sigma(z_j^l)$

$\boxed{\partial z_j^l}$ $\frac{\partial L}{\partial z_j^l}$ signal for jth node of layer l computing $\sigma()$ in

$$\boxed{\frac{\partial L}{\partial a_i^l} = \sum_{\forall a_j^{l+1}} \frac{\partial L}{\partial a_j^{l+1}} \cdot \frac{\partial a_j^{l+1}}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial a_i^l}}$$

(B) $$\boxed{\frac{\partial L}{\partial z_j^l} = \frac{\partial L}{\partial a_j^{l+1}} \cdot \frac{\partial a_j^{l+1}}{\partial z_j^l} \text{ (the first two terms of A.)}}$$

(C) $$\boxed{\frac{\partial L}{\partial w_{ij}^l} = \frac{\partial L}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{ij}}}$$

$$\boxed{\frac{\partial L}{\partial b_j^l} = \frac{\partial L}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l}}$$

$$\frac{\partial L}{\partial z_{j}^{l+1}} \frac{\partial z_{j}^{l+1}}{\partial a_j^{l+1}} \cdot \frac{\partial a_j^{l+1}}{\partial z_j^l}$$

$$\frac{\partial L}{\partial z_j^{l+1}} w_{ij}^{l+1} \cdot \sigma'(z_j^l)$$

2.2 D

Ⓓ $$\frac{\partial L}{\partial z_j^\ell} = \frac{\partial L}{\partial a_j^{\ell+1}} \cdot \frac{\partial a_j^{\ell+1}}{\partial z_j^\ell}$$

$$\Rightarrow \boxed{\frac{\partial L}{\partial z_j^\ell} = \frac{\partial L}{\partial a_j^{\ell+1}} \cdot \sigma'(z_j^\ell)}$$

$$\frac{\partial L}{\partial a_i^\ell} = \sum_{\forall j \text{ in layer } \ell} \frac{\partial L}{\partial z_j^\ell} \cdot \frac{\partial z_j^\ell}{\partial a_i^\ell}$$

Let $\frac{\partial L}{\partial a_i^\ell} = \Omega_i^\ell$ and Note $z_j = \sum_i (w_{ij}^\ell \cdot a_i^\ell) + b_j^\ell$

$$\Rightarrow \boxed{\frac{\partial L}{\partial a_i^\ell} = \sum_{j \in [d^{\ell+1}]} \Omega_j^{\ell+1} \cdot \sigma'(z_j^\ell) \cdot w_{ij}^\ell = \sum_{j \in [d^{\ell+1}]} \frac{\partial L}{\partial z_j^\ell} \cdot w_{ij}^\ell}$$

$$\frac{\partial L}{\partial w_{ij}^\ell} = \frac{\partial L}{\partial a_j^{\ell+1}} \cdot \frac{\partial a_j^{\ell+1}}{\partial z_j^\ell} \quad \frac{\partial z_j^\ell}{\partial w_{ij}}$$

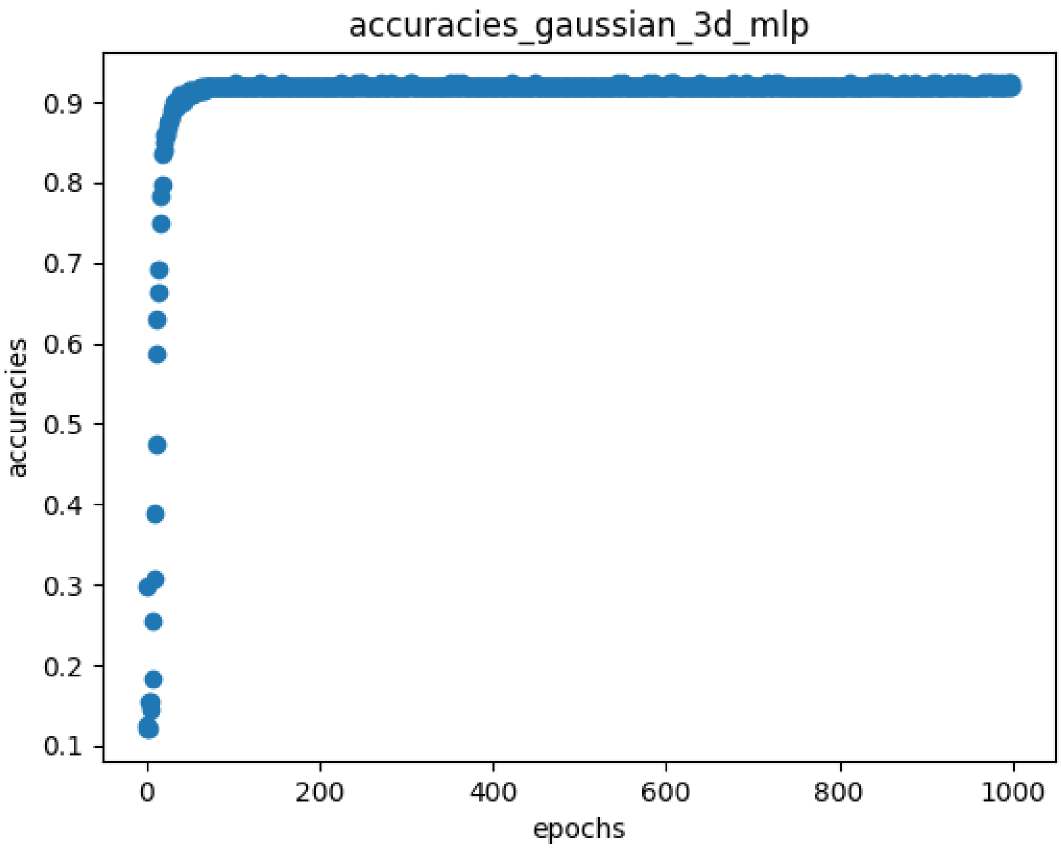
$$\Rightarrow \boxed{\frac{\partial L}{\partial w_{ij}^\ell} = \Omega_j^{\ell+1} \cdot \sigma'(z_j^\ell) \cdot a_i^\ell = \frac{\partial L}{\partial z_j^\ell} \cdot a_i^\ell}$$

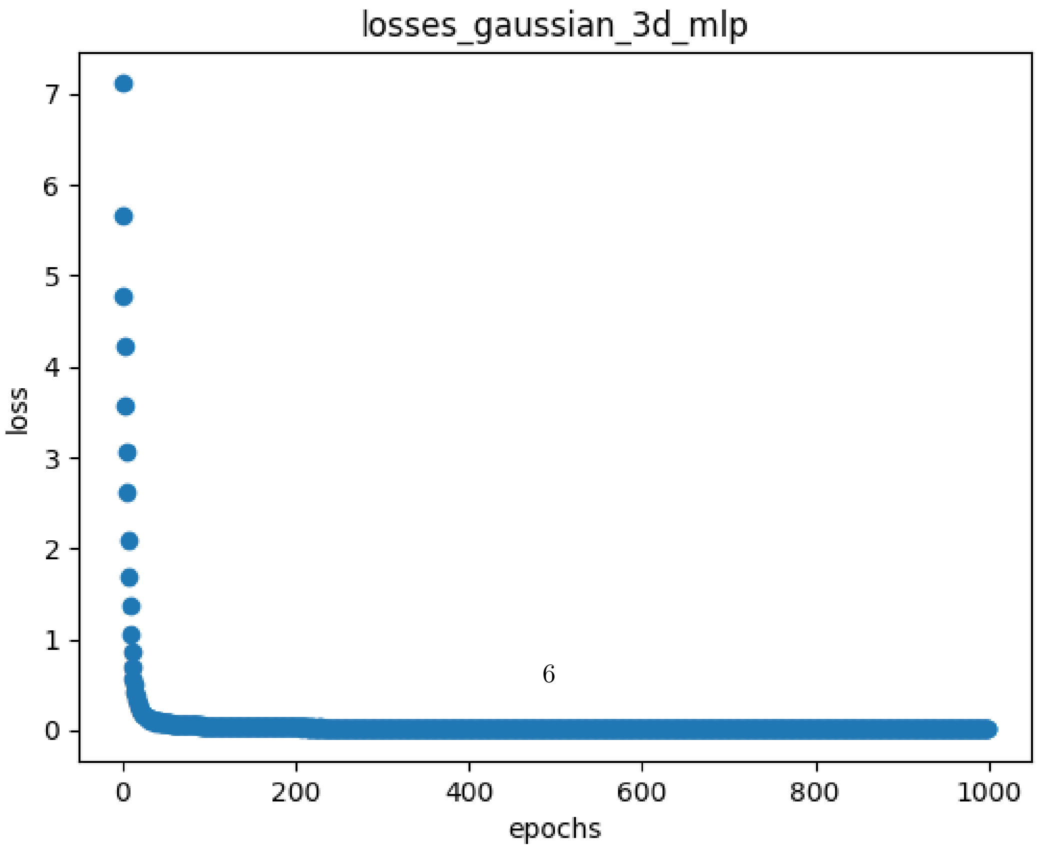
$$\Rightarrow \boxed{\frac{\partial L}{\partial b_j^\ell} = \Omega_j^{\ell+1} \cdot \sigma'(z_j^\ell)}$$

2.3 E

We choose to formulate backpropagation in this way because it lends itself to a lot of repeated subproblems. The derivatives for later layers appear in the calculations for previous layers and this makes it so that if we store our derivatives for activations and signals for later layers, then we can calculate in linear time each single activation of the current layer. In fact, each derivative is only dependent on the layer immediately after it.

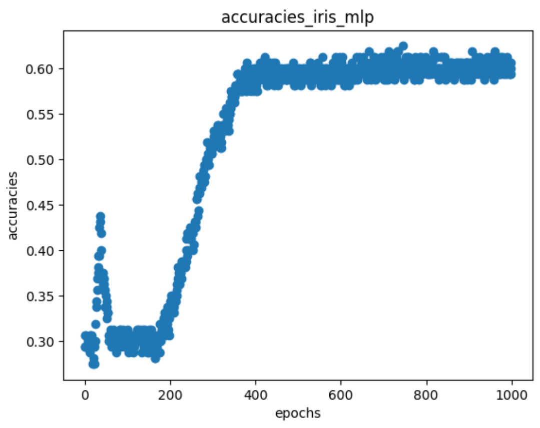
2.4 F

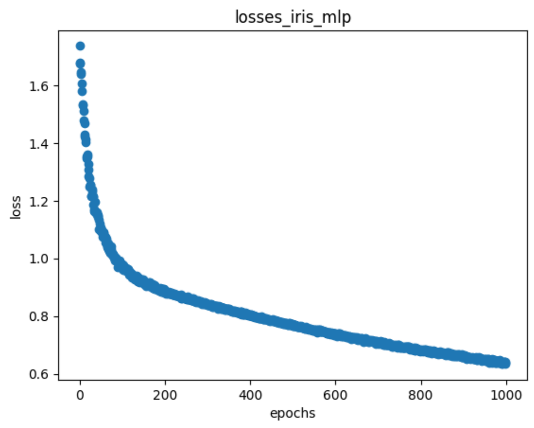

accuracies_gaussian_3d_mlp
accuracies
epochs
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
200
400
600
800
1000

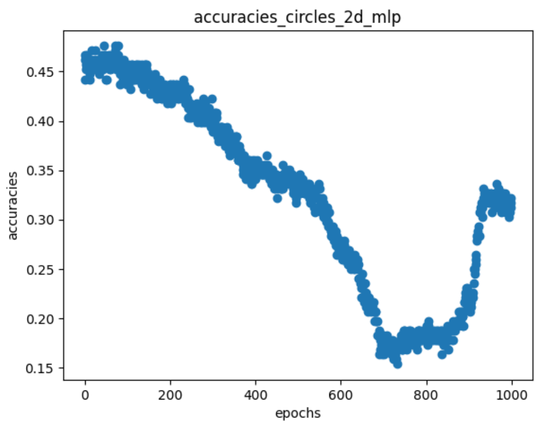

losses_gaussian_3d_mlp
loss
epochs
0
1
2
3
4
5
6
7
0
200
400
600
800
1000

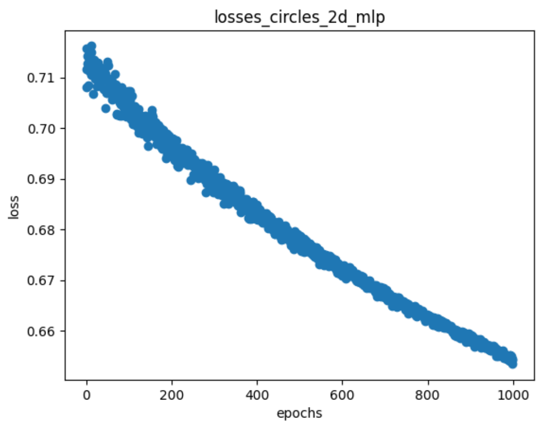
3 Compare Dataset Distributions

Here are the accuracies and losses for the concentric circles and iris datasets.









3.1 A

An MLP with one hidden layer is incapable of solving this kind of classification problem because each MLP layer can only learn lines. To do this classification, the MLP would need another layer that can serve as an AND that can combine all these lines together to make a bounding box around the spherical data in the middle which is not linearly separable.

3.2 B

Because the Iris dataset is a multiclass problem, it is unlikely that the data will be linearly separable and will need a more complicated model similarly to the kCircles dataset so that it can identify which sections of the feature space correspond to which classes.

4 Plots + Hyperparameter Sweeps and Effects

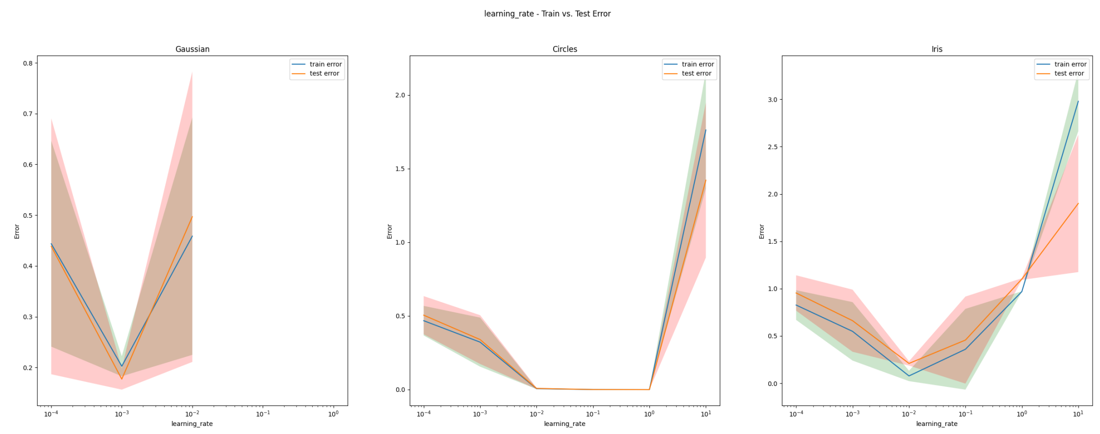
4.1 A

Done.

4.2 B Learning Rate

4.2.1 i

Below we can see that for the Gaussian, the best learning rate was around 10^{-3}, for the kCircles it was around 10^{-1}, and for the kIris it was around 10^{-2}. The kCircles is able to get close to zero test error for three values of the learning rate which suggests that there is a clear 'sweet spot' in terms of finding the minimum of the loss function.



4.2.2 ii

The loss explodes for high learning rates because the weights are just jumping around the Loss function at different points, completely overshooting the local minimum of the function.

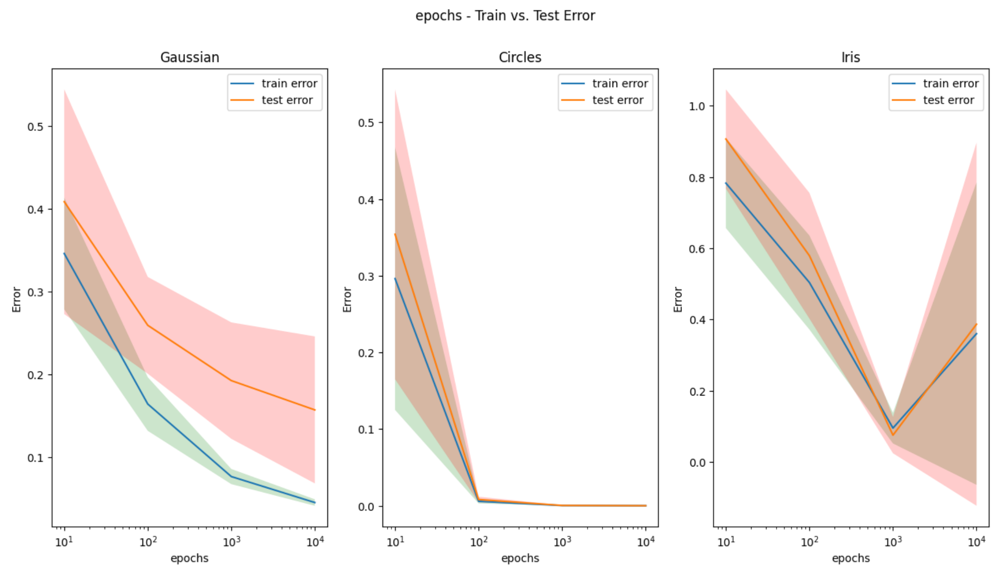
4.2.3 iii

When the learning rate is low, it takes too long to learn anything and the weights of the network barely change even if the loss is terrible and gradient is large.

4.3 C Epochs

4.3.1 i

Below we see that 10^4 epochs gives a clear best performance for the Gaussian distribution but for the Circles and Iris datasets 10^2 and 10^3 respectively give very low test error and with the Iris dataset the test error actually increases at 10^4 epochs. This means that Iris is asked the simplest task in some way such that training for too long leads to overfitting whereas on the other extreme, the Gaussian data needs the most amount of time to learn and benefits from more epochs.

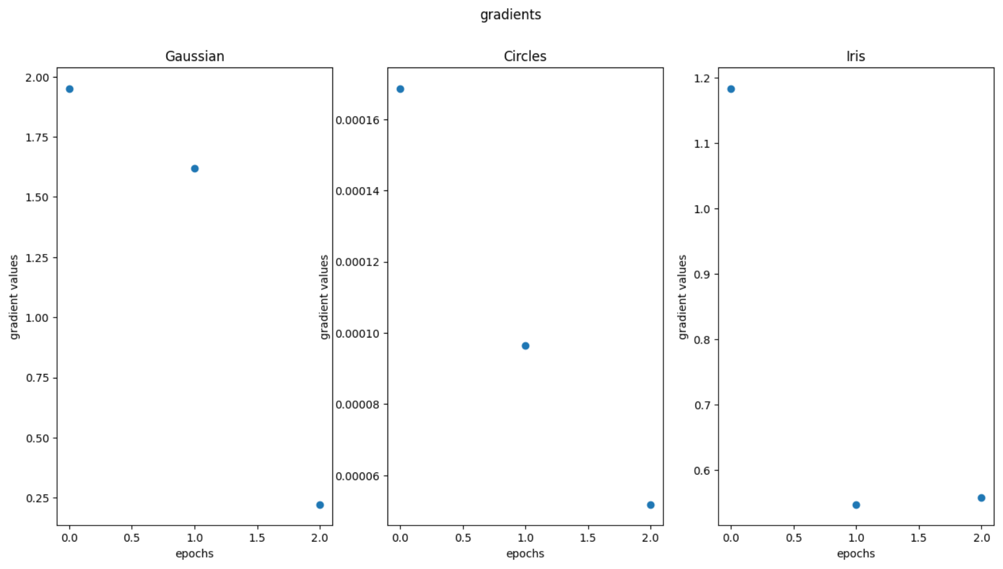


4.3.2 ii

$$Gaussian|Test : Never - Train : Never$$
$$Circles|Test : 10^2 - Train : 10^2$$
$$Iris|Test : 10^3 - Train : 10^3$$

4.3.3 iii

In the Gaussian dataset there is no convergence because the loss has not yet gotten close to a minimum and seems to be approaching lower and lower loss values but has not yet reached a plateau. Over more epochs, it is likely that the Gaussian dataset will converge.



4.3.4 iv

In the Iris dataset we see clear overfitting happening when we go to 10^4 epochs since the loss shoots up as compared to the local minimum that was seen at 10^3 epochs.

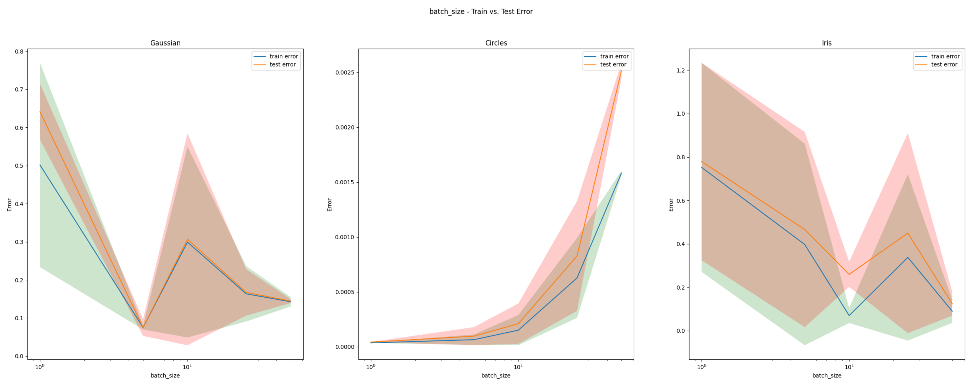
4.3.5 v

Three common techniques to prevent overfitting are to reduce model complexity, use regularization, and using dropout. They each can be optimal choices and through testing and Hyperparameter sweeps one can determine which give the best results for a given problem.

4.4 Batch Size

4.4.1 i

The best batch sizes for the Gaussian data was around 5, for the Circles data was around 1-2, and for the Iris dataset it was around 10.



4.4.2 ii

It can be noted that there is no general rule to decrease the loss according to batch size, however, in general, a greater batch size seems to make the test error follow the training error more tightly.

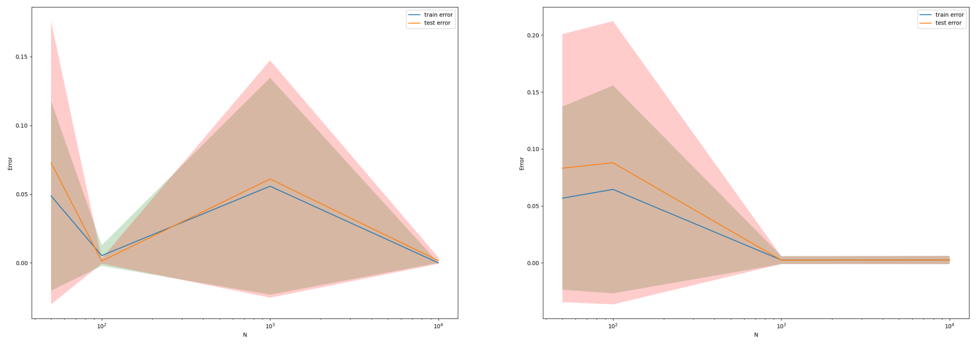
4.4.3 iii

A smaller batch size will make your experiment run quicker since runtime is directly proportional to the number of elements in a batch. For larger batches, it will take longer to calculate gradients and finish each epoch.

4.5 Training Samples

4.5.1 i

We note that for the Gaussian dataset, we observe a minimum in the test error at around 100 sample points whereas for the more complicated task of the Circles dataset, we get a minimum at around 1000 sample points.



4.5.2 ii

Unbalanced dataset distributions are a tradeoff between training power and reliability of results. If there is a fixed amount of data, using a greater training split is desirable because it will better train the model, potentially at the cost of overfitting, but will give you less reliable results for the estimate of the true accuracy/error.

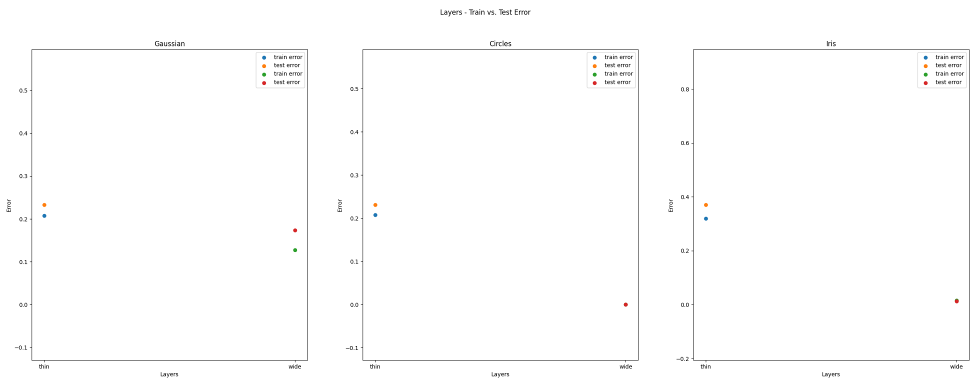
4.5.3 iii

Error changes across the different datasets because the complexity of classifying each task determines the amount of training data that is suitable for each network. According to the graphs, the Circles dataset has a more complex task because it requires more training data to get the same low error that the Gaussian dataset gets for smaller training data.

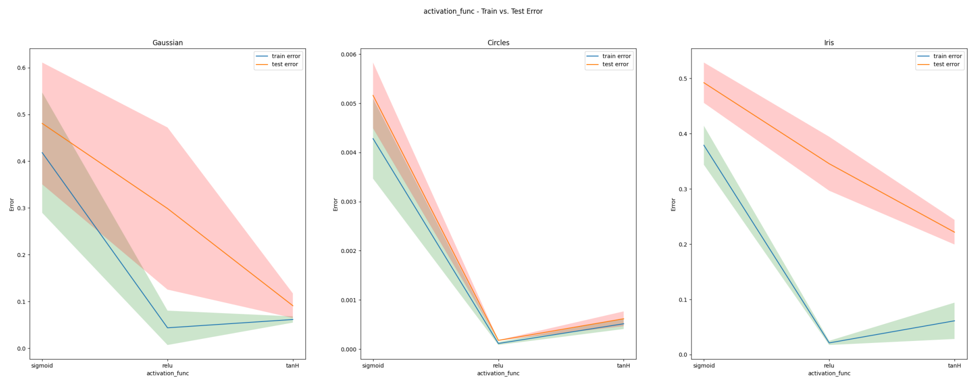
4.6 Layer Depth vs Width

4.6.1 i

For the Gaussian dataset, both the thin and wide network perform about the same, but for the Iris and Circles datasets, the wide network performs significantly better. This might be due to the idea that a thin layer in a network acts as a compression of information while a wide layer acts as an interpretation of the dataset. So when the complicated tasks of Iris and Circles get passed through the thin network, not much useful information is passed through any one layer leading to loss of information which makes it difficult for the network to learn. On the other hand, in the wide network, the model can break these tasks down into subproblems which can then each be solved building on each other, such as creating a bounding box from many perceptrons.



5 Intuition for Activation and Loss Functions

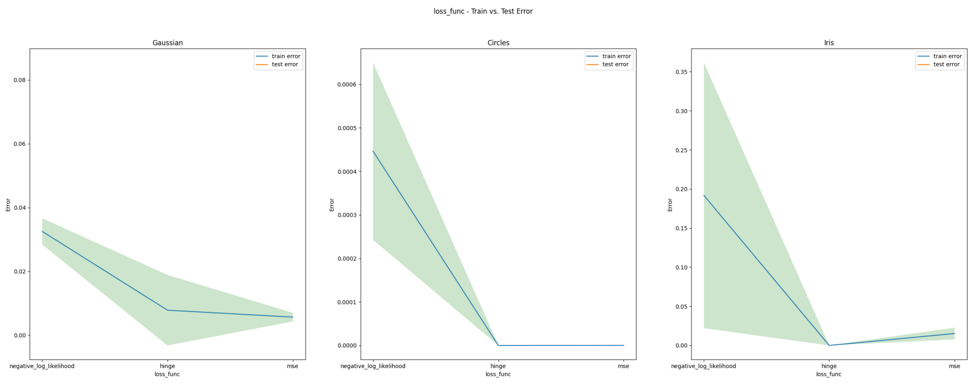


5.1 A

Changing the activation function to ReLU, we get almost a 50% drop in our error. This is likely due to the fact that ReLU gets rid of the vanishing gradients problem that is prominent to the sigmoid activation function due to its tangents to $y = 1$ and $y = -1$ when $\|x\|$ is large. This provides diminishing returns in our backpropagation for values that should really be very influential because they are unexpected.

5.2 B

Replacing the activation function to tanH, we get another almost 50% drop in our error. This could be due to ReLU skipping around too much to find the local minimum. tanH is slower than ReLU to converge and thus may have stumbled onto a local minimum faster than ReLU, because ReLU's steps were too large.



5.3 C

Changing the loss function to Hinge Loss and Mean Squared Error greatly decreased the error. Though these numbers look smaller, they are not necessarily better since all of these loss functions' error values only make sense in relation to themselves and comparison to itself. However, it is possible that changing the loss function can be very beneficial if that new loss function is more suitable to calculating the error of your measurements. For example, if you are trying to train a linear classifier on a set of linearly separable points, then using something like negative log likelihood will likely not give you as good-looking of a line as just using MSE.