CS 4033 Project 1

Alexander Greus, Spencer Smith

1 Design Choices Made by the Assignment (Steps 2-3)

The following choices were made by the assignment specification:

- **Learning paradigm:** Supervised learning, as labeled class data is provided.
- **Task type:** Binary classification — each sample belongs to one of two classes.
- **Architecture type:** Feedforward neural network with error backpropagation.
- **Loss signal:** Classification error between predicted and true class label.
- **Datasets:** Nine fixed datasets across Gaussian 2D, Gaussian 3D, and crescent moon distributions.

2 Design Choices For Us to Consider (Steps 4-5)

The following choices were considered, as well as what we what we plan on doing:

- **From Scratch:** Do it from scratch in C. Does not seem like we need something complex for this.
- **Hidden layers:** One hidden layer, as the universal approximation theorem guarantees a single hidden layer is sufficient to approximate any continuous function given enough neurons. Also simplifies things.
- **Hidden neurons:** 8 neurons per hidden layer.
- **Activation function:** Sigmoid, $\sigma(x) = \frac{1}{1+e^{-x}}$, natural probabilistic interpretation for binary classification. Also simple derivative for backprop.
- **Learning rate:** $\eta = 0.1$ Just a guess
- **Weight initialization:** Uniform random in $[-1, 1]$. Starting small.

3 Design Choices After Implementation (Steps 7-8)

- **From Scratch:** Yup, did it from scratch. Didn't end up being too difficult.
- **Hidden layers:** Yup, ended up doing 1 hidden layer since its simple. Made it easy to hard-code backprop formulas.
- **Hidden neurons:** 32 neurons, actually, since it was easy to add more and in initial testing it seemed to improve things.
- **Activation function:** Sigmoid, since again it was simple for backprop.
- **Learning rate:** $\eta = 0.1$ Seems a good rate to have chosen.
- **Weight initialization:** Uniform random in $[-1, 1]$. Didn't really experiment too much with this, but seems fine.
- **Epoch Count:** Decided to go with baseline of 10,000 but can stop early if validation shows no loss improvement after 500 epochs.
- **Train/Validate/Test Division:** Decided to go with a 60/20/20 split since this seemed to be better than others we tried like 75/15/15.

4 Data Collection (Step 9)

For each of the nine datasets, the network was trained over 20 independent runs, each with a different random weight initialization, keeping everything else the same. The data was, as mentioned before, split 60% training, 20% validation, and 20% test. The validation set was used to monitor overfitting, and we only tested after the entire training run was complete.

The following data was collected per run:

- Training loss per epoch, to observe convergence behavior.
- Training, validation, and test accuracy at the end of training.

We report mean and standard deviation of test accuracy across the runs. Since we randomly initialize the weights, we can't tell if our ANN is good or not from just a single run.

We also provide a graph of the loss across epochs from a randomly selected run.

We hope to determine if our ANN is any good from this data, specifically the mean and standard deviation across runs. As well we hope to see whether our ANN converges quickly, or in a consistent manner.

5 Results

5.1 Gaussian 2D Wide

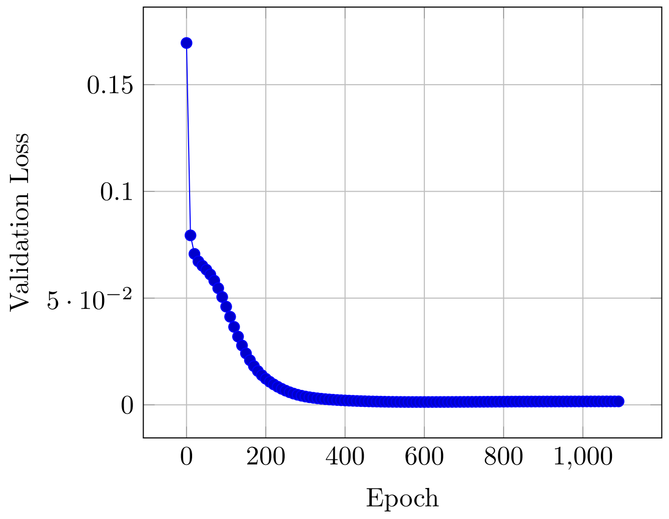

Figure 1: Validation loss over training epochs, Gaussian 3D Wide

We can observe that validation set loss converges and minimizes quite quickly. We could probably exit even earlier than our 500 epochs of no improvement check. This makes sense since the classes are distributed widely from each other, meaning its a relatively easy classification task.

Dataset	Class 0		Class 1		Overall	
	Mean	Std	Mean	Std	Mean	Std
Gaussian 2D Wide	1.00	0.00	1.00	0.00	1.00	0.00

Table 1: Test accuracy across all datasets, 20 runs each

We observe that across all runs we correctly classify our entire test set, since we have 0 variance. This seems expected since this distribution has a wide gap been classes.

Overall the ANN seems pretty effective at classifying this problem and dataset.

5.2 Gaussian 2D Narrow

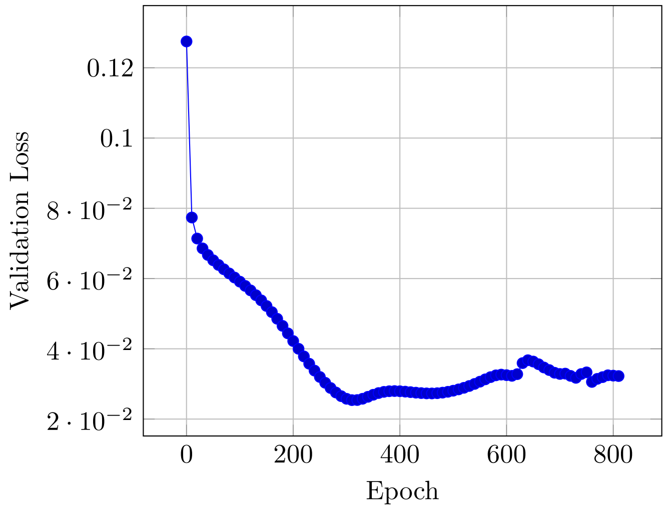

Figure 2: Validation loss over training epochs, Gaussian 3D Narrow

We can observe that validation set loss does not converge as quickly or as smoothly as in the Wide case. We actually reach a minimum loss around 300 epochs, but our early exit condition is too patient and only exits after 500 epochs of no improvement. Thus we actually and unfortunately start to increase loss, a type of overfitting. This is semi-expected since this is a harder problem, what with the classes being closer together.

Dataset	Class 0		Class 1		Overall	
	Mean	Std	Mean	Std	Mean	Std
Gaussian 2D Narrow	0.98	0.0292	0.9875	0.0217	0.9838	0.0182

Table 2: Test accuracy across all datasets, 20 runs each

We unfortunately do not always accurately classify all of our training set across runs. We don't seem much variance between class 0 and class 1, so that at least is good and means we are being consistent across classes. We also don't observe much variance at all, meaning the ANN is again relatively consistent.

Overall, the issue here for data seems to be over-fitting, and could be solved with some better checks for when that happens during validation. We do still achieve ¿98% accuracy, but we can always improve.

5.3 Gaussian 2D Overlap

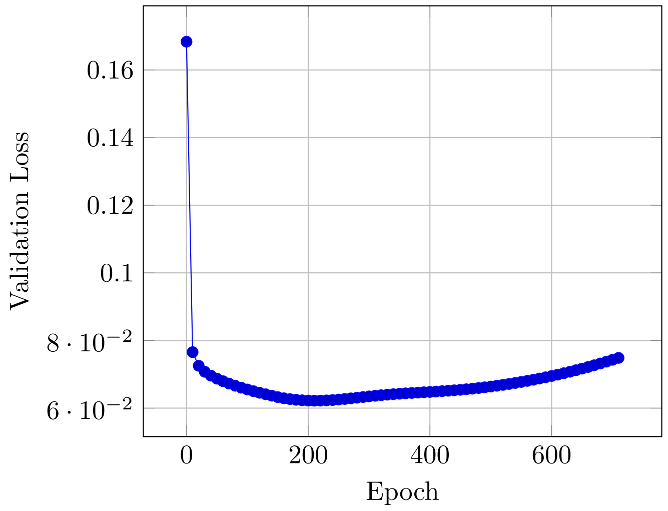

Figure 3: Validation loss over training epochs, Gaussian 3D Overlap

We again see not great behavior in our validation loss. We actually see a much steeper drop in loss than in the case of Wide, which is surprising since intuitively overlapping should be a harder problem. It is also more consistent than in Wide. Again though, we see the loss climb after reaching a local minimum around 200 this time. We can see that our overfitting checks are not really sufficient for this problem either.

Dataset	Class 0		Class 1		Overall	
	Mean	Std	Mean	Std	Mean	Std
Gaussian 2D Overlap	0.6975	0.0512	0.9450	0.0150	0.8213	0.0212

Table 3: Test accuracy across all datasets, 20 runs each

Again we see not great results. Class 1 looks alright, not perfect but not horrible. However, for Class 0, we see a pretty poor accuracy. We are not sure why there is a disparity here, but we assume it may be due to our previously demonstrated over fitting? Again there is not much variance between runs, so at least the poor performance for Class 0 is consistent.

5.4 Gaussian 3D Wide

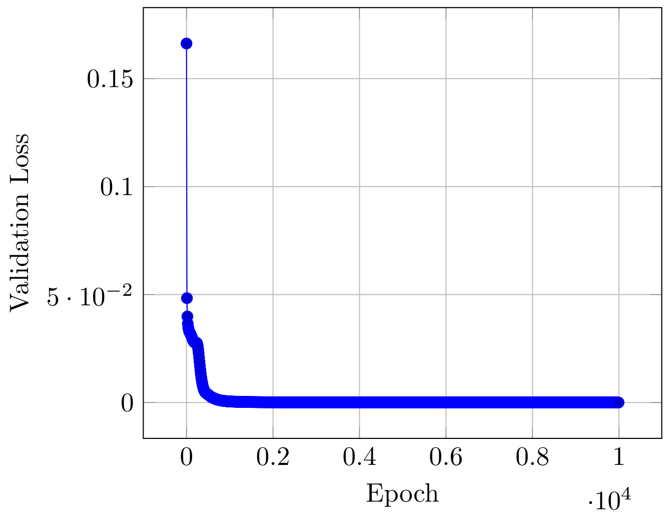

Figure 4: Validation loss over training epochs, Gaussian 3D Wide

Back to the easy data, we see very good performance, even better than in the 2D case surprisingly. The loss drops sharply and stays consistent thereafter. We believe the better performance is because it may be easier to classify with more information, where 2 points might overlap in 2 dimensions, the 3rd dimension might separate them.

Dataset	Class 0		Class 1		Overall	
	Mean	Std	Mean	Std	Mean	Std
Gaussian 3D Wide	1.00	0.00	1.00	0.00	1.00	0.00

Table 4: Test accuracy across all datasets, 20 runs each

Again, we see perfect performance across runs, all achieving 100% accuracy every time. As stated before, we think the extra useful info from the third dimension actually helps rather than hinders.

We observe that our ANN is very good at classifying this dataset.

5.5 Gaussian 3D Narrow

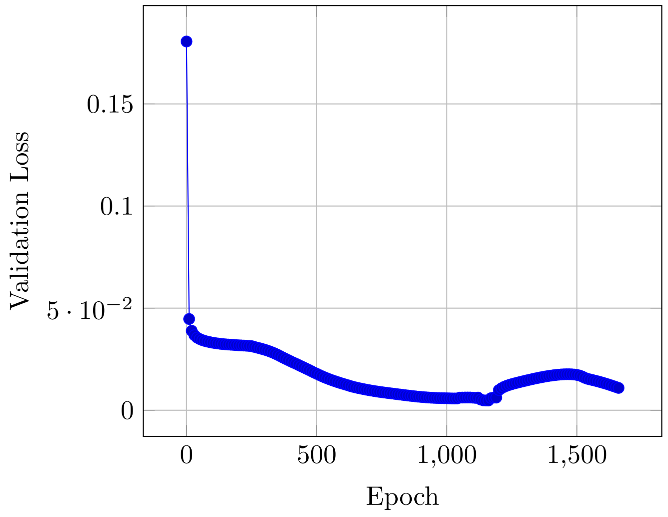

Figure 5: Validation loss over training epochs, Gaussian 3D Narrow

This is one is both surprising and unsurprising. We again see better performance than in the 2D case, but we see the same inconsistency/over-fitting. This time the minimum occurs around 1100 epochs. As well, the loss decreases in a more gradual manner after the initial drop. Overall, seems pretty good.

Dataset	Class 0		Class 1		Overall	
	Mean	Std	Mean	Std	Mean	Std
Gaussian 3D Narrow	0.9775	0.0370	0.9925	0.1785	0.9850	0.0184

Table 5: Test accuracy across all datasets, 20 runs each

Here, again we see better performance than in 2D. We see pretty consistent and accurate classification for both classes, with little variance between runs.

Our ANN seems pretty good at this problem.

5.6 Gaussian 3D Overlap

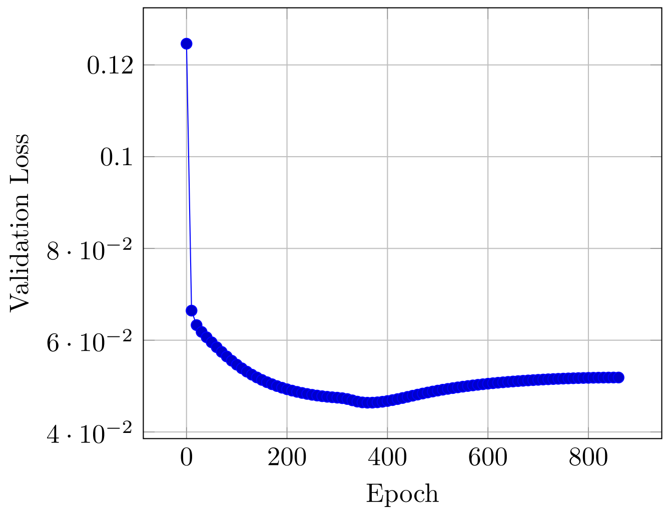

Figure 6: Validation loss over training epochs, Gaussian 3D Overlap

Yet again, we see better performance than the 2D case. Interestingly, we also see a sort of similar curve, in that it steadily decreases loss, reaches a minimum, and then begins to regress and increase loss. The minimum occurs around 350 epochs in this case though.

Dataset	Class 0		Class 1		Overall	
	Mean	Std	Mean	Std	Mean	Std
Gaussian 3D Overlap	0.9050	0.0150	0.9025	0.0249	0.9038	0.0119

Table 6: Test accuracy across all datasets, 20 runs each

We see much better accuracy for class 0 classification, and slightly worse for class 1, when compared to the 2D case. This is overall better since we performed very poorly on class 0, in the 2D case. Again its pretty consistent, with low standard deviation across runs.

Overall, we were pretty impressed with our performance for this problem.

5.7 Moon 2D Wide

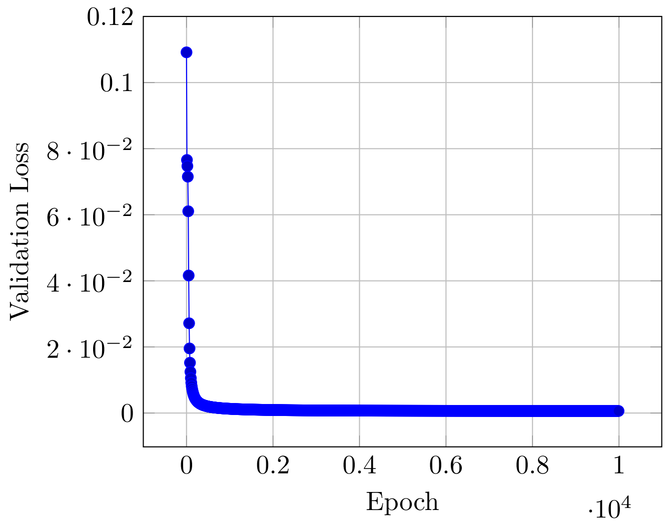

Figure 7: Validation loss over training epochs, Moon 2D Wide

Like with all wide datasets so far, we perform very well. A steep drop in loss, than flat-lining consistently. I think after looking at all the wide datasets, we attribute our performance mostly to just how easy the dataset is to classify, not that our ANN is too terrible hahaha.

Dataset	Class 0		Class 1		Overall	
	Mean	Std	Mean	Std	Mean	Std
Moon 2D Wide	1.00	0.00	1.00	0.00	1.00	0.00

Table 7: Test accuracy across all datasets, 20 runs each

Again, this is a pretty easy dataset, so our ANN never fails across all runs to get 100% accuracy. Again, we don't necessarily attribute this success to our ANN architecture, but mostly to just how good the dataset is for classification.

Overall, we perform very well on this dataset.

5.8 Moon 2D Narrow

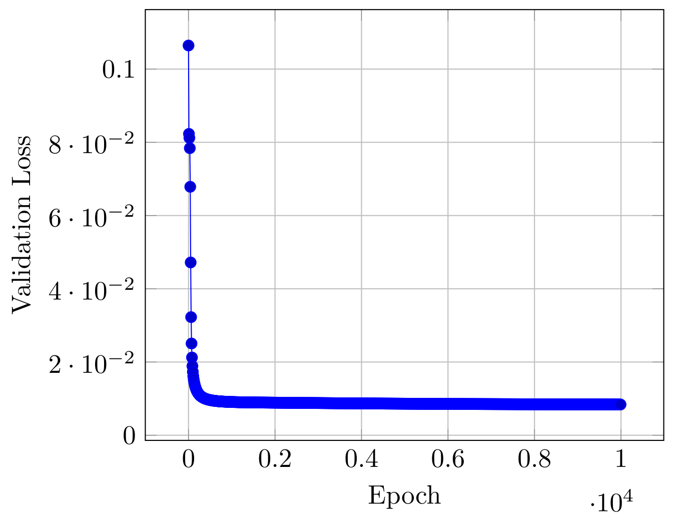

Figure 8: Validation loss over training epochs, Moon 2D Narrow

This one is slightly surprising, we expected to see the same degradation in performance as in previous cases of moving from wide to narrow. But here we see very little degradation. We see basically the same loss curve, but this time flat-lining a bit above zero. We wonder if this means that the crescent pattern is somehow easier to extract as a pattern than the Gaussian blobs?

Dataset	Class 0		Class 1		Overall	
	Mean	Std	Mean	Std	Mean	Std
Moon 2D Narrow	0.99	0.00	1.00	0.00	0.995	0.00

Table 8: Test accuracy across all datasets, 20 runs each

Again, not much performance degradation. Just 1 run fails to classify 1 class 0 sample. We are pretty satisfied with the performance in for this problem set, which we are surprised by. We expected the crescent to be harder to classify as its, at least visually, not linearly separable by one line. But apparently that 1 hidden layer is doing some heavy lifting here.

Overall, surprisingly good performance on this dataset.

5.9 Moon 2D Overlap

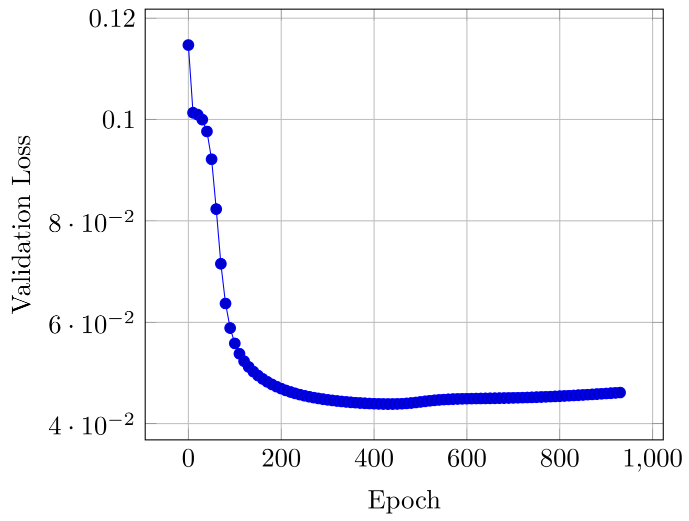

Figure 9: Validation loss over training epochs, Moon 2D Overlap

This one is pretty interesting. Pretty sharp decline, a minimum, and then gradual increase of loss. We see a sort of similar pattern with other overlapping datasets, which might suggest that with overlapping datasets over-fitting is a big issue. We could probably work on tuning our ANN to detect and avoid these issues.

Dataset	Class 0		Class 1		Overall	
	Mean	Std	Mean	Std	Mean	Std
Moon 2D Overlap	0.97	0.00	0.957	0.0056	0.9635	0.0028

Table 9: Test accuracy across all datasets, 20 runs each

One of the better performances for overlapping datasets. Looking at all the datasets now, we want to draw the conclusion that at least for our ANN the crescent pattern is easier to extract. As well, we can now pretty confidently say that while our model is not the best at everything, it is at least pretty consistent, as the std-dev of accuracy has always remained pretty low.

Overall, for making it from scratch, we are pretty happy with our ANN.