Write-Up

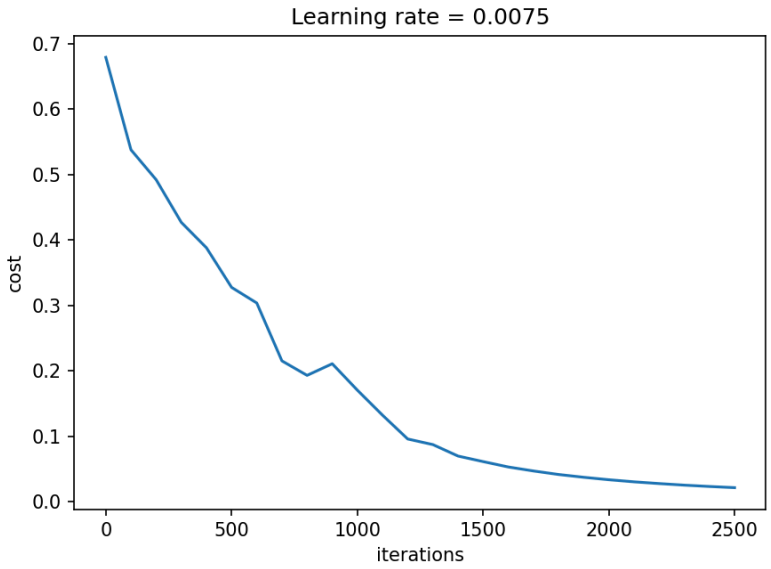
For this assignment I first ran the model exactly as it appears in Training.py, then modified the architecture and learning rate to see how performance would change.

Original Model

This is the configuration already present in the file: 12288 → 5 → 1, a single hidden layer of 5 units, learning rate 0.0075, 2500 iterations.

Metric	Train	Test
Accuracy	100%	66.0%
Precision	1.000	0.808
Recall	1.000	0.636
F1 score	1.000	0.712

The cost decreases fairly smoothly over training. There is a small bump around iteration 900 that I cannot fully explain, but the cost recovers immediately and continues downward:



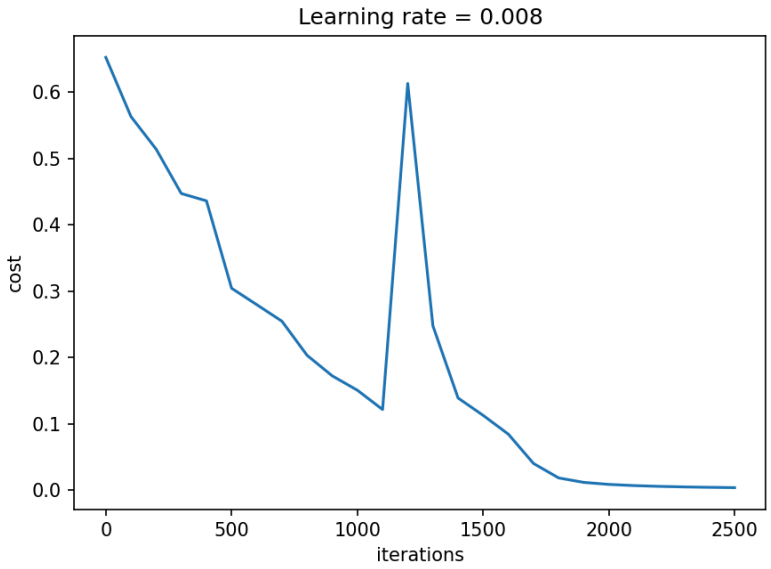
Misclassified test images:

Modified Model

For the second run I added four additional hidden layers (12288 → 40 → 20 → 10 → 5 → 1) and increased the learning rate slightly to 0.008. I kept the iteration count at 2500 so that at least one variable would remain consistent between the two runs.

Metric	Train	Test
Accuracy	100%	76.0%
Precision	1.000	0.862
Recall	1.000	0.758
F1 score	1.000	0.806

This cost curve is noticeably less smooth, with a spike around iteration 1200 that I discuss below:



Misclassified test images:

Questions

What factors made the model struggle?

With the original model, most of the errors are actual cats that the network failed to identify, rather than non-cat images mistakenly labeled as cats. Eight of the ten misclassified test images are cats, with only two exceptions: a butterfly and a bird silhouette, both labeled as cats.

Among the eight missed cat images, several are tightly cropped, showing only a face or a paw rather than the full body. One image also shows an all-white cat against a dark background, which stands out from most of the training photos and may have contributed to the error. This pattern makes sense given the model architecture: with only one small hidden layer and 209 training images, the network likely learned a fairly narrow notion of what a cat looks like, so any image that departs from that pattern is more likely to be missed.

How did changing the learning rate affect accuracy and F1?

Because I changed the learning rate and the architecture at the same time, I cannot fully isolate the effect of the learning rate alone. What is clear from the cost curve is that the modified run shows a distinct spike around iteration 1200, where the cost rises nearly back to its starting value before decreasing again. The original run shows only a minor bump around iteration 900 that resolves immediately. This spike is consistent with a learning rate large enough to overshoot a minimum during gradient descent before correcting itself.

Despite this instability, training still converged to a lower final cost than the original run, 0.0034 compared to 0.021. To evaluate the learning rate's effect on its own, I would need to hold the architecture constant and vary only the learning rate, which I did not do here.

How did changing the number of epochs affect accuracy and F1? Did more epochs always help the test set?

Both runs used 2500 iterations, so this comparison does not isolate the effect of epoch count between the two architectures. However, neither cost curve had flattened out by the end of training, meaning both models would likely continue to reduce training cost with additional iterations.

Training accuracy reached 100% for both models well before training concluded, so any additional iterations beyond that point are not correcting further training errors. The effect of continued training on the test set is not guaranteed to be positive; it depends on whether further optimization happens to move the decision boundary toward or away from better generalization. Based on this, I would not conclude that additional epochs always improve test performance, since it is possible to continue training well past the point where it is still beneficial.

How did increasing the number of nodes or layers affect performance? Do more complex models always do better?

Both networks reached 100% training accuracy, so there is no difference to observe on the training set itself. The difference appears on the test set: the shallow, single-layer model

achieved 66% accuracy and an F1 score of 0.712, while the deeper model achieved 76% accuracy and an F1 score of 0.806. In this case, the more complex model performed better on unseen data, which was not the outcome I initially expected.

A likely explanation is that with only 209 training images and 12288 input features, the model is almost certain to overfit regardless of depth, since the number of parameters vastly exceeds the number of training examples. If overfitting is effectively unavoidable in this setting, the additional layers may still provide some benefit by allowing the network to represent more structured features rather than relying purely on raw pixel values. That said, I would not conclude that added complexity is universally beneficial; the learning rate results above show that a larger model paired with a higher learning rate can just as easily destabilize training. The effect of model complexity appears to depend heavily on the specific conditions of the run rather than following a single consistent rule.