One of the hardest struggles I dealt with while training the Perceptron was avoiding overfitting. The dataset, at a staggering 208 inputs, was drastically low and imposed a large hurdle. I attempted multiple times to circumvent the idea of using a fixed number of epochs and opted for training until a certain accuracy threshold was achieved on the given training data, but even that didn’t yield very favorable results, and I wasn’t sure if this method was allowed.

After two weeks of tweaking values and trying to write scripts that would locate optimal values for me, I came up with a way to sidestep some of the problems with choosing the right learning rate, as using a coefficient that is too large in Newton’s method will have a harder time locating a minimum, but a coefficient that is too small will occasionally settle in local minimums rather than the absolute minimum. My solution to this problem was to start with a very large learning rate and then reduce that learning rate by a constant factor every epoch, such that the learning rate at any step *n* could be calculated as

The numbers I settled on were:

LearningRate0 = 0.5  
ReductionConstant = 0.9999  
Epochs = 50,000  
with an initial bias of 1

These parameters gave me an overall accuracy of 98% with an 88.5% accuracy on unseen data values, much higher than I ever saw using a constant learning rate.

As for choosing the training set, I originally set up a random number generator that would choose an input as part of the training set roughly 80% of the time, as a few online sources had cited that as a good split between training data (80%) and verification data (20%). However, I was still worried about making sure I had a good sample of the training data, so I settled on picking a fixed set by using a counter with modulus. I started with using a mod value of 5, then most recently switched to 6, resulting in a training set consisting of 83.33...% of the data, and a verification set consisting of the remaining 16.666...%.