Tweaking the Learning Rate

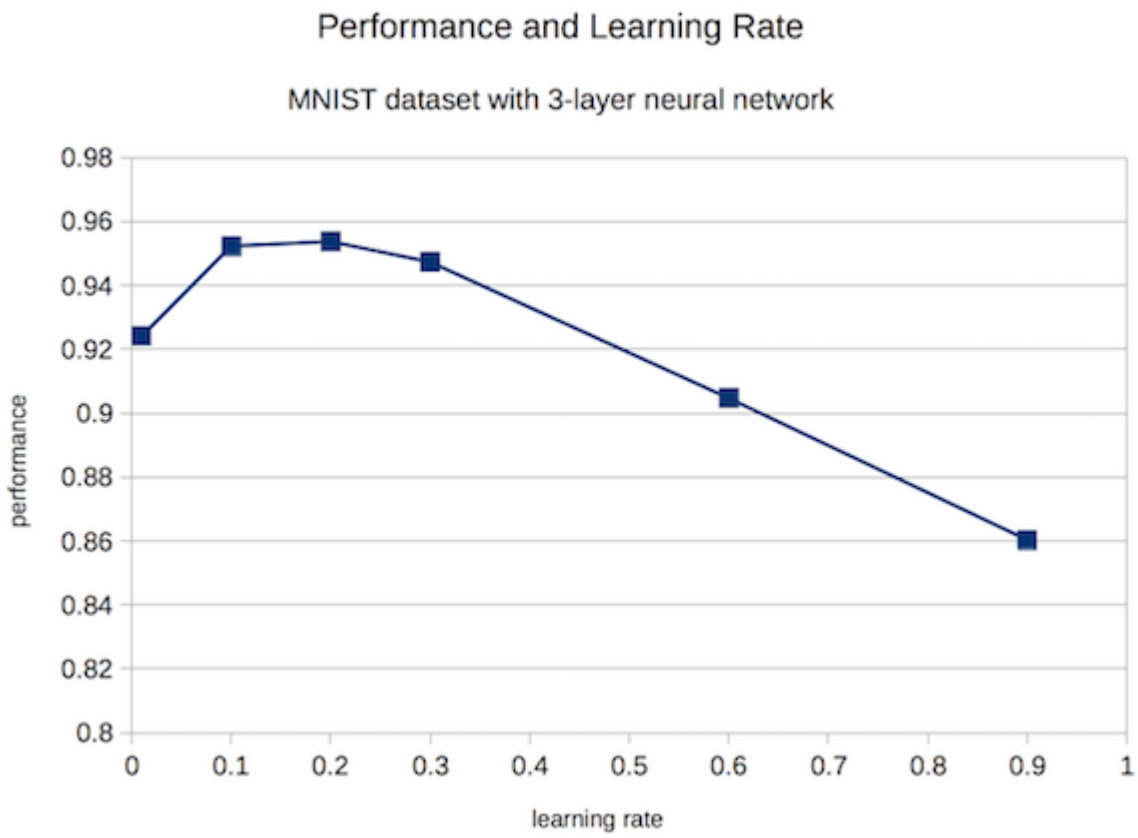
How adjusting the learning can bring improvement in the accuracy of result? Let's find out.

A 95% performance score on the MNIST dataset with our first neural network, using only simple ideas and simple Python is not bad at all, and if you wanted to stop here, you would be entirely justified. But let's see if we can make some easy improvements:

The first improvement we can try is to adjust the learning rate. We set it at 0.3 previously without really experimenting with different values. Let's try doubling it to 0.6, to see if a boost will actually be helpful or harmful to the overall network learning. If we run the code, we get a performance score of 0.9047. That's worse than before. So it looks like the larger learning rate leads to some bouncing around and overshooting during the gradient descent. Let's try again with a learning rate of 0.1. This time the performance is an improvement at 0.9523. It's similar in performance to one listed on that website which has 1000 hidden nodes. We're doing well with much less!

What happens if we keep going and set a learning rate of an even smaller 0.01 ? The performance isn't so good at 0.9241. So it seems to have too small a learning rate is damaging. This makes sense because we're limiting the speed at which gradient descent happens, we're making the steps too small.

The following plots a graph of these results. It's not a very scientific approach because we should really do these experiments many times to reduce the effect of randomness and bad journeys down the gradient descent, but it is still useful to see the general idea that there is a sweet spot for learning rate.



The plot suggested that between a learning rate of 0.1 and 0.3 there might be better performance, so let's try a learning rate of 0.2 The performance is 0.9537. That is indeed a tiny bit better than either 0.1 and 0.3. This idea of plotting graphs to get a better feel for what is going on is something you should consider in other scenarios too — pictures help us understand much better than a list of numbers!

So we'll stick with a learning rate of 0.2, which seems to be a sweet spot for the MNIST dataset and our neural network. By the way, if you run this code yourself, your own scores will be slightly different because the whole process is a little random. Your initial random weights won't be the same as my initial random weights, and so your own code will take a different route down the gradient descent than mine.