Neural Networks: Fundamentals & Applications
Week 1 Assignment

Richard Scalzo (CTDS)

1. Iris dataset with an additional hidden layer

I actually designed my code from the outset to be able to accommodate an arbitrary number of hidden layers (so I wouldn't have to keep fussing with it). The particular parameter choices follow Rohit's code, although I find I get much better performance with a learning rate of 0.2 and a MSE tolerance of 0.1. In principle this code could be extended to different types of activation functions and different objective functions, although the temptation would be to just use Keras after a certain point.

With topology [4, 6, 2], over 30 trials:
total training + eval time: 7.599 sec
final performance on 40 test instances: mse = 0.115 (+/- 0.006), acc = 89.3% (+/- 7.0%)

With topology [4, 6, 6, 2]:
total training + eval time: 14.657 sec
final performance on 40 test instances: mse = 0.114 (+/- 0.013), acc = 92.6% (+/- 3.4%)

The results don't seem dramatically different, except that the net with the additional hidden layer takes twice as long to train. Its performance seems slightly more consistent but not hugely better.

2. Iris dataset with multiple hidden layers

Let's keep adding layers and see what happens.

With topology [4, 6, 6, 6, 2]:
total training + eval time: 33.255 sec
final performance on 40 test instances: mse = 0.114 (+/- 0.010), acc = 93.5% (+/- 2.8%)

With topology [4, 6, 6, 6, 6, 2]:
total training + eval time: 112.971 sec
final performance on 40 test instances: mse = 0.110 (+/- 0.009), acc = 93.8% (+/- 2.7%)

It looks like we've maxed out our performance for this problem at two hidden layers, or three if we're being generous. Depends on how much that last 0.3% matters.

As things got longer I decided to start keeping track of how many iterations the algorithm went through. I would expect the execution time of each iteration to scale linearly with the number of layers, but the total number of iterations needed to reach a certain MSE is likely to increase as we go on. And indeed that's what I find.

With topology [4, 6, 6, 6, 6, 6, 2] — five hidden layers:
total training + eval time: 279.008 sec
final performance on 40 test instances: mse = 0.132 (+/- 0.045), acc = 88.1% (+/- 12.310) itc = 761.0% (+/- 163.949)

Here the accuracy actually starts to go down. I found that of the thirty iterations, four actually went over their allowed time (`maxIterations` = 1000) and gave up.

I should add that I haven't done anything clever here, like multiprocessing to make full use of my quad-core system. But it would be pretty straightforward to mini-batch the whole thing, so maybe I'll do that some other time.

3. Iris dataset with different network architectures

We can keep doing this all day! Since three hidden layers seems to get us most of the mileage with this dataset, I'll try reducing the number of neurons in the subsequent layers to speed things up a bit.

With topology [4, 6, 4, 3, 2]:
total training + eval time: 47.623 sec
final performance on 40 test instances: mse = 0.110 (+/- 0.008), acc = 94.7% (+/- 2.1)

Hm, took a bit longer but things became more accurate! Does that work if I shrink previous successful architectures?

With topology [4, 6, 4, 2]:
total training + eval time: 15.943 sec
final performance on 40 test instances: mse = 0.112 (+/- 0.011), acc = 94.0% (+/- 2.4)

With topology [4, 4, 3, 2]:
total training + eval time: 17.516 sec
final performance on 40 test instances: mse = 0.114 (+/- 0.009), acc = 94.0% (+/- 2.1)

Interestingly, it does. How few can we get away with?

With topology [4, 3, 3, 2]:
total training + eval time: 17.172 sec
final performance on 40 test instances: mse = 0.113 (+/- 0.008), acc = 93.5% (+/- 2.0)

With topology [4, 3, 2]:
total training + eval time: 8.565 sec
final performance on 40 test instances: mse = 0.116 (+/- 0.006), acc = 92.3% (+/- 3.7)

Starting to go back down again, so extra layers do add something.

Is it just as good to add more neurons in hidden layers?

With topology [4, 10, 2]:
total training + eval time: 6.126 sec
final performance on 40 test instances: mse = 0.115 (+/- 0.005), acc = 90.6% (+/- 4.4)

With topology [4, 20, 2]:
total training + eval time: 7.999 sec
final performance on 40 test instances: mse = 0.117 (+/- 0.012), acc = 87.7% (+/- 6.5)

With topology [4, 20, 10, 2]:
total training + eval time: 22.753 sec
final performance on 40 test instances: mse = 0.110 (+/- 0.011), acc = 91.7% (+/- 4.2)

Nope, those extra neurons don't help at all after the first few. The lion's share of the execution time seems to be taken up by making the network deeper; while numpy's fast matrix operations reduce overhead, the training time for a fully connected network seems to be exponential in the number of layers. Not to be taken lightly! So one good strategy for training might go something like this:

- Start out with a fairly broad, shallow network, to make sure you've captured all the features necessary for success.
- Add layers until performance improves to a standard you'd like to lock in.
- Prune back the layers to reduce the number of degrees of freedom to a workable minimum, improving the fit in the process.

Can that approach work generically? One way to find out.

4. Try at least one other dataset

Technically the problem asked for five extra datasets, but this problem set is already an hour overdue by now, so I'm going to do one and just hand it in, following the methodology I outlined above.

(a) Wine dataset (http://archive.ics.uci.edu/ml/datasets/Wine)

The dataset is in CSV form much like the iris dataset — I see now this is where Rohit must have gotten it from. It has 13 features and 3 classes, and is described as a well-posed, not very challenging dataset. Perfect for the night before.

I found quickly that what seemed like an arbitrary vectorization of the classification labels in the iris dataset is actually needed; trying to predict an unmodified integer class label (1, 2, 3) produced pretty catastrophic performance. By just coding the labels in binary — 1 = (0, 0), 2 = (0, 1) and 3 = (1, 0) — I got back on track, with a simple network topology [13, 6, 2] producing 99.4% accuracy on a random holdout set of 48 instances.