Practical 7: Introduction to Neural Networks

(Version 1.3)

1 Overview

This practical provides a hands-on introduction to neural networks. The idea is that you will try your hand at *training* a few different types of neural networks and see how well they do. The variations between the networks might depend on several things:

- Choice of network architecture (e.g., the number of hidden nodes)
- Choice of threshold function (also called "transfer function").
- Choice of learning rate
- Use, or otherwise, of momentum

You will work through some examples using `scikit-learn`, one using the venerable Iris dataset, another on a dataset from the UCI repository, and then you will try writing your own code to learn and then use simple neural networks.

2 A Neural Network for the Iris dataset

We start by looking at a very simple application of the the `scikit-learn` implementation of neural networks.

From KEATS download the file:

```
classify-iris-simple-nn.py
```

and take a look at it in your favourite editor. It should look familiar. That's because it is pretty much the same as the file that uses a decision tree on the Iris example. Aside from the comments, the only lines which are different are these:

```
iris_nn= MLPClassifier(solver='lbfgs', activation='logistic',
                       hidden_layer_sizes=(5, 2), random_state=1)
iris_nn.fit(X, y)
```

and even they look pretty similar to some of the commands we used with decision trees.

The first line creates an instance of the class `MLPClassifier`, for "Multi-Layer Perceptron", and the second line trains it on the dataset.

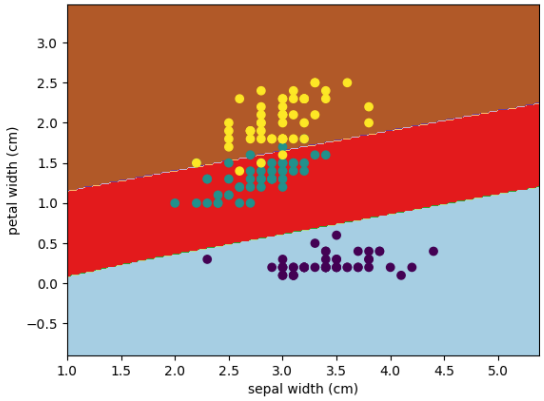

Figure 1: The output of a neural network trained on the iris dataset.

As you can see, you have some control over the neural network.[1] What you see here picks a logistic function as the activation function for the neurons, picks the LBFGS solver (the way that the weights are chosen[2]), fixes the seed of the random number generator (so that results are the same every time), and specifies two hidden layers, one with 5 neurons and one with 2.

Figure 1 shows how well this neural network performs.

Now:

1. Try using neural networks with different architectures (different numbers of neurons in the hidden layers, different numbers of hidden layers). How do these perform?
2. Try using the `sgd` solver (stochastic gradient descent). For that you will likely have to play with some of the learning rate parameters.

3 Classifying Seeds

Now you get to create your own neural network classifier using `scikit-learn`. Download:

`seeds_dataset.csv`

from KEATS. This is (a cleaned up version of) the seeds dataset from the UCI repository:

`https://archive.ics.uci.edu/ml/datasets/seeds`

Also download:

`classify-seeds-starter.py`

[1] As ever, there is a manual page for the class, `http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html`, and this specifies some other parameters that can be set.

[2] LBFGS "is a solver that approximates the Hessian matrix which represents the second-order partial derivative of a function. Further it approximates the inverse of the Hessian matrix to perform parameter updates. The implementation uses the Scipy version of L-BFGS." according to `http://scikit-learn.org/stable/modules/neural_networks_supervised.html`. It is a solver that works well on small data sets.

from KEATS. This loads the data from `seeds_dataset.csv`. Now:

1. Experiment to find a set of features which looks like it will allow a reasonable classification.

   Hint: If you plot the data from `seeds_dataset.csv` against pairs of feature values, you can look for groups of seeds that look as easy to separate as the Irises in the Iris dataset.

2. Experiment to find a neural network architecture that can produce a reasonable classification.

   Hint: Write code that tests different network architectures and evaluate them using cross-validation. For comparison, the best I could do was a classifier with an accuracy of 0.95.

4 Perceptrons for the Iris Dataset

Now it is time to dig into your own implementation of various approaches to training neural networks.

We'll start with a simple perceptron. Starting with another copy of

```
classify-iris-simple-nn.py
```

write code to train a single perceptron to classify the iris data using the error correction procedure. (That is the simplest procedure, so makes a good place to start.) Since you have a single perceptron, you won't be able to do three class classification. But you can classify one class from the other two. Start by learning weights to distinguish class 0 (0 is the value of `y`) from the other two. (This is setosa, the very distinct class that appears at the bottom of the plots). When you have cracked that, look to learn weights to distinguish class 2. (This is virginica, which appears at the top of the plots.)

Now write code to train a single perceptron using the generalised delta rule. (I started from a fresh copy of `classify-iris-simple-nn.py`.) Again, start by training it to classify class 0 from classes 1 and 2. Then train it to classify class 2 from classes 0 and 1.

Finally, build a single-layer multi-unit neural network that can classify the Iris dataset. If you train one perceptron to classify class 0 from classes 1 and 2, and another to classify class 2 form classes 0 and 1, you can combine the output to classify into three classes. I managed to train mine to an accuracy of 96%.

5 Backpropagation

You can guess what you need to do here. Write a multilayer neural network to classify the Iris data and then train it using backpropagation.

Two things to note here:

1. Since we can classify the Iris dataset pretty well with a single layer network (as in the previous exercise) using a multilayer network is clearly overkill. However, it is a useful exercise to learn how to do backpropagation.

2. Start by using something simple. It might be tempting to build a network with lots of hidden layers and lots of hidden units, but I would suggest something small.

Having said that, my experiments suggest that you probably need at least two output units . . . which also means I'll be impressed if anyone manages to get good results with less than two output units.

6 More

1. For a full set of training procedures, tackle the Iris dataset with a single-layer multi-unit network that is trained using the (non-generalised) Delta rule.
2. See how far you can get in classifying the seeds dataset using a single-layer neural network.
3. Now try a build-your-own neural network on the seeds dataset.

7 Version list

- Version 1.0, February 25th 2019.
- Version 1.1, February 28th 2019.
- Version 1.2, March 1st 2020.
- Version 1.3, February 9th 2021.