Assignment 7

7 December 2012

Due Monday 17 December at 1am

In this assignment, you will design and train a neural network to accomplish some classification task.

Choose a data set

The UCI Machine Learning Archive hosts various data sets suitable for testing learning algorithms. I suggest clicking on "View ALL Data Sets" on the right side of the page. That provides a nice interface in which you can filter by data type or area of interest.

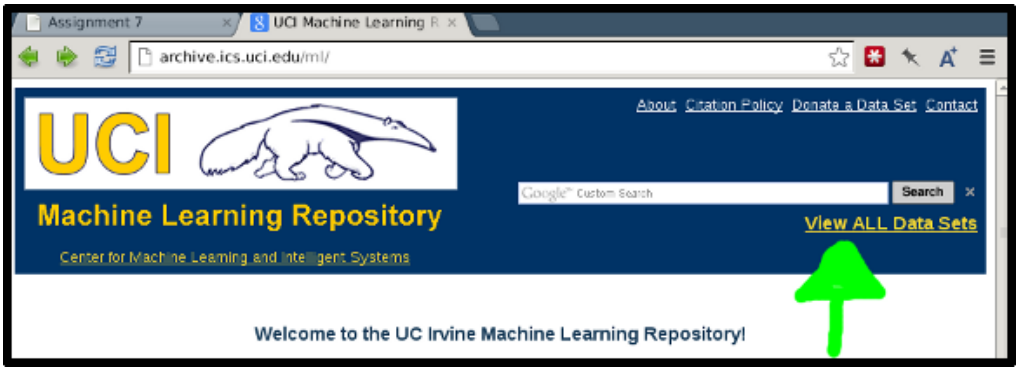

Figure 1:

The data should be suitable for a **classification** task, not clustering, recommendations, or regression. Neural networks support both categorical and numerical data, you'll just want to keep the number of attributes to less than 100, because we'll have to tune the way each attribute is presented to the network.

When you click on the data set, you'll see a description, citations, and details about the attributes. There are links near the top to the "Data Folder", and there you'll find a list of files ending in `.data` (the raw data) or `.names` (attribute descriptions).

Download the data and descriptions. We used the Mushroom data from the UCI repository previously (assignment 5), so I'll explore that – but you should choose something else for your project. The `.names` file contains:

```
1. Title: Mushroom Database

2. Sources:
   (a) Mushroom records drawn from The Audubon Society Field Guide to North
       American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred
```

```
        A. Knopf
    (b) Donor: Jeff Schlimmer (Jeffrey.Schlimmer@a.gp.cs.cmu.edu)
    (c) Date: 27 April 1987

3. Past Usage:
   1. Schlimmer,J.S. (1987). Concept Acquisition Through Representational
     Adjustment (Technical Report 87-19). Doctoral disseration, Department
     of Information and Computer Science, University of California, Irvine.
     --- STAGGER: asymptoted to 95% classification accuracy after reviewing
           1000 instances.

[etc.]
5. Number of Instances: 8124

6. Number of Attributes: 22 (all nominally valued)

7. Attribute Information: (classes: edible=e, poisonous=p)
     1. cap-shape:                bell=b,conical=c,convex=x,flat=f,
                                  knobbed=k,sunken=s
    2. cap-surface:             fibrous=f,grooves=g,scaly=y,smooth=s
   3. cap-color:           brown=n,buff=b,cinnamon=c,gray=g,green=r,
                             pink=p,purple=u,red=e,white=w,yellow=y

[etc.]
```

The `.data` file is a text file with *comma-separated values* (CSV), which can be imported easily into Excel or other spreadsheet applications:

```
p,x,s,n,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,k,s,u
e,x,s,y,t,a,f,c,b,k,e,c,s,s,w,w,p,w,o,p,n,n,g
e,b,s,w,t,l,f,c,b,n,e,c,s,s,w,w,p,w,o,p,n,n,m
p,x,y,w,t,p,f,c,n,n,e,e,s,s,w,w,p,w,o,p,k,s,u
e,x,s,g,f,n,f,w,b,k,t,e,s,s,w,w,p,w,o,e,n,a,g
[etc.]
```

Design your network

Your next task is to design your neural network architecture: how many neurons in each layer, and how to map neuronal activations to and from the data set?

Input layer

The number of input neurons will be *based* on the number of attributes in your data set, but it may not be a one-to-one match.

Generally, a *continuous* (real number) attribute can map directly to one neuron. There are no continuous attributes in the mushroom set, but the Heart Disease data contains a few, such as:

```
thalach: maximum heart rate achieved
```

which has values like 127, 154, or 166. It is helpful, however, to *normalize* these values to the range 0..1, so they are not terribly out of proportion to the inputs from other attributes. In the case of heart rate, we would find the minimum (60) and the maximum (182) in the data file. Then, to convert any value, we subtract the minimum and divide by the size of the range (182-60 = 122):

```
 Raw value    Normalized value
    60          0.0             = (60-60)/122
   127          0.549180327869  = (127-60)/122
   154          0.770491803279  = (154-60)/122
   166          0.868852459016  = (166-60)/122
   182          1.0             = (182-60)/122
```

A *discrete* (categorical) attribute must be translated in some way, usually using a binary encoding. Let's take the `cap-shape` of mushrooms as an example. These are the possible values:

```
bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s
```

Because there are 6 possible values, we can represent them in $\lceil \log_2(6) \rceil = 3$ input neurons, like this:

```
Code  Category  #  Binary  Input[0]  Input[1]  Input[2]
 b     bell     0   000      0.0       0.0       0.0
 c     conical  1   001      0.0       0.0       1.0
 x     convex   2   010      0.0       1.0       0.0
 f     flat     3   011      0.0       1.0       1.0
 k     knobbed  4   100      1.0       0.0       0.0
 s     sunken   5   101      1.0       0.0       1.0
```

Work through the attribute descriptions for your data set to determine the number of input neurons, the normalization parameters for continuous attributes, and the binary encoding for discrete attributes.

Hidden layer

You will have to decide how many neurons to use in the hidden layer. Too few, and the network will not be sophisticated enough to recognize the patterns in the data. Too many, and the network may take longer to converge on an acceptable solution.

I would recommend starting with the same number of hidden neurons as input neurons, and then experiment with reducing it.

Output layer

Most classifications will be discrete categories: poisonous/edible for mushrooms, or the diagnosis of heart disease in that data set:

```
num: diagnosis of heart disease (angiographic disease status)
-- Value 0: < 50% diameter narrowing
-- Value 1: > 50% diameter narrowing
```

You will want to have one output neuron for each possible classification, and use the "winner take all" strategy – the neuron with the highest activation determines the result. Here would be the expected outputs for the categories of mushrooms:

```
Code  Category  Output[0]  Output[1]
 e     edible    1.0        0.0
 p     poison    0.0        1.0
```

Implementation

After doing "Sync with git" from your `cs162` folder, you should see a sub-folder `a7` with my implementation of a neural network. Start in `neuro.h`, which defines the network dimensions:

```
const int INPUT_NEURONS = 57;   // Network dimensions
const int HIDDEN_NEURONS = 10;
const int OUTPUT_NEURONS = 2;
```

These are currently set up for the mushroom data, but you can alter them for your own network.

Next, in `main.cpp`, you will find a reference to the file name containing the data set:

```
    // Read data from file into two-dimensional vector
    read_csv("shrooms.data", data);
```

Save your `.data` file to the `a7` folder, and adjust the file name. Make sure the `read_csv` is capable of reading the data file. If it is, you should see a message like this upon running the program.

```
Read 8124 records x 23 attributes from shrooms.data
```

Next, we'll examine the two functions that map the data to the input and output neurons, respectively:

```
void set_network_inputs( vector<string>& row );
void set_desired_outputs( vector<string>& row, double out[OUTPUT_NEURONS] );
```

The implementations of these are just below the end of `main()`. Let's begin with `set_desired_outputs`. This is where we implement the winner-take-all. In the shrooms data file, the edible/poisonous classification is the first (0th) attribute, and it's a single character, 'e' or 'p'. When we access `row[0]`, we get the *string* value of the 0th attribute, and the extra `[0]` grabs the first (0th) *character* of that string.

```
switch(row[0][0]) {
case 'e':
    out[0] = 1;
    out[1] = 0;
    break;
case 'p':
    out[0] = 0;
    out[1] = 1;
    break;
default:
    cout << "Error: unexpected classification " << row[0][0] << "\n";
    abort();
}
```

The `default` case helps detect potential errors in parsing the file.

Your output interpretation will be similar, but it must be based on the format of your data and the number of output neurons.

As for the network inputs, take a look at the `set_network_inputs` function for the mushroom data in `main.cpp`:

```
int i = 0;
// 1. cap-shape: bell=b,conical=c,convex=x,flat=f,
// knobbed=k,sunken=s
switch(row[1][0]) {
case 'b': inputs[i++] = 0; inputs[i++] = 0; inputs[i++] = 0; break;
case 'c': inputs[i++] = 0; inputs[i++] = 0; inputs[i++] = 1; break;
case 'x': inputs[i++] = 0; inputs[i++] = 1; inputs[i++] = 0; break;
case 'f': inputs[i++] = 0; inputs[i++] = 1; inputs[i++] = 1; break;
case 'k': inputs[i++] = 1; inputs[i++] = 0; inputs[i++] = 0; break;
case 's': inputs[i++] = 1; inputs[i++] = 0; inputs[i++] = 1; break;
default:
    cout << "Error: unhandled cap-shape " << row[1][0] << "\n";
    abort();
}
```

This fragment illustrates converting a discrete value, represented by single characters, into bits in a binary encoding. You can see that the zeroes and ones assigned to `inputs[i++]` match the binary encodings in the previous table.

If your data set has real-valued inputs, this is where you would normalize them to the range 0.0–1.0.

Experiments

Once you have these input and output functions completed, you can experiment with training the network. Determine how many cycles it takes to converge to a solution with different proportions of training vs. test data. Then experiment with different numbers of hidden neurons.

Write up your observations. Notify me if you have trouble getting the network to converge at all, or if your input/output functions do not seem to work.