Today I will talk about another type of neural net software called ARTMAP, which stands for Adaptive Resonance theory, Mapping. This is an extension of the ART1 algorithm that I talked about last week. It still uses binary inputs and outputs. It uses two ART1 networks, one processes the top down expectations and one handles the bottom up inputs. It also adds the concept of a learning rate to the adjustment of long term memory weights.
This shows a further simplified block diagram of the ARTMAP algorithm. Note that ARTa processes the bottom up inputs and ARTb processes the top down expectations. The orienting systems serve to minimize the predictive error between the two.
Unlike my little play application last month, the classification algorithm I will talk about today is real. I will classify poisonous vs. edible mushrooms commonly found in North America. The classifications that it uses come from The Audubon Society field guide to North American Mushrooms, by Gary Lincoff, copyright 1981.
The computer program that I wrote in Visual Basic implements an algorithm based upon a paper written by Gail Carpenter, Stephen Grossberg and John Reynolds from the Center for Adaptive systems and department of Computer and Neural Systems at Boston University in 1990. Their research was supported by a DARPA contract. (The Defense Advanced Research Projects Agency).
Artificial Intelligence Part 7

2.3 Rule matching
The $i$th $F_j$ node is active if its net input exceeds a fixed threshold. Specifically,
\[ x_i = \begin{cases} 1 & \text{if } I_i + g_i + \sum_j z_{ij} > 1 + 2 \\ 0 & \text{otherwise} \end{cases} \quad (1) \]
where term $I_i$ is the binary $F_i \rightarrow F_j$ input, term $g_i$ is the binary non-specific $F_i$ gain control signal, term $\sum_j z_{ij}$ is the sum of $F_j \rightarrow F_i$ signals $y_j$ via pathways with adaptive weights $z_{ij}$, and 2 is a constant such that
\[ 0 < 2 < 1. \]
(2)

$F_j$ gain control
The $F_j$ gain control signal $g_j$ is defined by
\[ g_j = \begin{cases} 1 & \text{if } F_j \text{ is active and } F_j \text{ is inactive} \\ 0 & \text{otherwise} \end{cases} \quad (3) \]
Note that $F_j$ activity inhibits $F_j$ gain, as shown in Figure 5. These laws for $F_j$ activation imply that, if $F_j$ is inactive,
\[ x_i = \begin{cases} 1 & \text{if } I_i = 1 \\ 0 & \text{otherwise} \end{cases} \quad (4) \]
If exactly one $F_j$ node $J$ is active, the sum $\sum_j z_{ij}$ in eqn (1) reduces to the single term $z_{ij}$, so
\[ x_i = \begin{cases} 1 & \text{if } I_i = 1 \text{ and } z_{ij} = 2 \\ 0 & \text{otherwise} \end{cases} \quad (5) \]

4.2. $F_j$ Choice
Let $T_j$ denote the total input from $F_i$ to the $j$th $F_j$ node, given by
\[ T_j = \sum_{i=1}^{N} t_{ij} \]
(6)
where the $t_{ij}$ denote the $F_i \rightarrow F_j$ adaptive weights. If some $T_j > 0$, define the $F_j$ choice index $J$ by
\[ T_j = \max(T_j : j = 1 \ldots N) \]
(7)
In the typical case, $J$ is uniquely defined. Then the $F_j$ output vector $y = (y_1, \ldots, y_N)$ obeys
\[ y_i = \begin{cases} 1 & \text{if } j = J \\ 0 & \text{if } j \neq J \end{cases} \quad (8) \]
If two or more indices $j$ share maximal input, then they equally share the total activity. This case is not considered here.

A Real AI Application

The algorithm laid out in the paper is fairly complex but it is well suited for today’s multi processor computers.
$T_i = \sum_{i=1}^{u} x_i Z_{ij}$

where the $Z_{ij}$ denote the $F_1 \rightarrow F_2$ adaptive weights.

A Real AI Application

Any time you see a summation sign like this in an equation, and there are several of them in this algorithm, you can divide of the computations between several processors and it will run much faster. I have not done so explicitly in my program but W10 may do it automatically. We can tell this by seeing if it runs any faster on my i7 desktop with 8 virtual processors versus my i5 laptop with only 4 virtual processors.
As I said, this is a real application. In the field guide, Mr. Lincoff, characterizes north american mushrooms using 22 different features as shown here. Each of the features has from 2 to 12 values for a total of 126 different values. Thus, the feature values for each mushroom can be represented by a 126 element binary vector.
As you know, with any word problem in math, the trick is to fit the words into the math. Since Grossberg had at that time a very good algorithm to deal with binary inputs, he formulated this app to work with binary inputs.
Not only that, but some hard working graduate student must have worked for a semester or two compiling a database of 8,124 mushrooms and their feature values and put them in a CSV file that you can download. Using this as inputs, I can run my version of the ARTMAP algorithm and see if I can duplicate Mr. Grossbergs research results.
Shown here is a portion of that list. In the first column, it shows the classification of the mushroom as either poisonous (p), or edible (e). Followed by the values of each of the 22 features.
This doesn’t get you a 126 element binary vector. The first thing that must be done is to interpret the values and from them create a set of 126 element binary vectors. This I did using a macro in excel. The macro was written in Visual Basic for Applications (VBA) which is similar to Visual Basic used in Visual Studio except for times when it is not.
This results in a binary file a portion of which looks like the above. The binary file is 2Mb, while the file that holds the feature data was only 373Kb.
So, going back to the block diagram of the ARTMAP algorithm, the 126 element vectors on the mushroom features comes in at the bottom as the bottom up inputs, and the 2 bit poisonous/edible comes in at the top as the top down expectations.
Just a brief review of Artificial Neural networks. They model real neural networks as shown here. Electrical and chemical signals come in from the dendrites and synapses “downstream”. At the main body of the neuron, the soma, these signals are gathered together. If the net effect of these signals exceed the cells action potential, an electrical pulse is generated. This pulse is carried by the axon to the axon terminals which fire chemical signals via the synapse activating other neurons.
Here is a schematic of the computer analog to a neural network called an Artificial Neural Network or ANN.

Here the activation function is a step function. Note once again the summation equation which is well suited for a multiprocessor computer.
Again, here is the schematic version of ART1 which is at the heart of the ARTMAP algorithm. It’s “adaptive” because the long term weights in both the bottom up paths, $z_{ij}$, and the top down paths, $Z_{ji}$ (not shown) adapt to the inputs that are presented. It is resonant in that the bottom up predictions are compared to the top down expectations and if they don’t match closely enough, a new “choice” at layer F2 is made.
Again, we check to see if this new choice “resonates” with the bottom up predictions, if not, the F2 layer choice is reset and we generate a new choice. If after checking all active nodes in F2, that is all nodes that have been previously coded by input vectors, it is decided that this input vector is so novel that it needs its own F2 node and a new F2 node is activated.
If we have not created a model with a large enough F2 layer, we will run out and errors will be generated. To ensure that doesn’t happen, my model today has 100 nodes in the F2 in both ARTa and ARTb. The F1 layer has 126 nodes for the 126 inputs in ARTa and 2 nodes in ARTb.
The ARTMAP system uses both unsupervised learning and supervised learning. So far, I have only programmed the unsupervised learning part of the algorithm. In fact, I haven’t done anything on the MAP field programming. The algorithm is run in two steps: Training and Testing.
In the training portion, we process a number of input vectors from the mushroom database, put them in categories and constantly adjust the top-down and bottom-up weights between the F1a and F2a layers. I have also added programming to check to see if the categorization is correct. You have to train on at least 125 vectors to get sufficient accuracy.
Adaptive Resonance Theory

In the testing portion, you process a large number of vectors, that were not used in training to see how well the system works. During testing, the weights are not allowed to adapt or change and no new categories can be created. Mistakes between how the vector is classified and how it should be classified and a tally is kept in the “Mistakes” box. Mistakes occur when an input vector is so novel that it doesn’t match any of the current categories.
Let’s run a 125 vector training set. We see that the ART1a module has separated the mushrooms into 17 categories (Ca = 16, starting at 0). Under the “F2a Codes” box, we can see the 16 categories and how the algorithm has assigned the mushrooms to those categories. In the F2b Codes box, we can see what outputs, poisonous = 0, edible = 1, the vectors should have. This took about 6 seconds on my desktop, 7 and one-half seconds on my laptop. No mistakes occurred.
Demonstration

Even though we have 17 categories, there are only two possible output results, poisonous or edible. So we have several categories that are poisonous and even more that are edible. We know that this was a training run because there is no entries in the “Test Start” of “Test Num” boxes.
Following this, a 500 vector test was done. This took about 24 seconds on my desktop, 30 seconds on my laptop and generated 4 errors. This is a 99.2% success rate even without the MAP field operating to adjust the vigilance parameter.
Following this, a 1000 vector test was done. This took about 1 min and 21 seconds on my desktop, 1 min 30 seconds on my laptop and generated 30 errors. This is a 97% success rate even without the MAP field operating to adjust the vigilance parameter.
Run some more tests. Change the number of vectors in training and see how it affects the number of categories and the number of errors. Increase the vigilance parameter and see how it affects the results. Put in one vector and see if it checks out.
Next time I will demonstrate the entire ARTMAP algorithm and we will see how a changing and adapting vigilance parameter as well as changes in the learning rates of the long term weights affect the performance.
Break
Visual studio, is an “Integrated Development Environment” (or IDE) provided at no cost to you by Microsoft. It is built around .NET object libraries which you can only get from Microsoft and run only on Windows computers.
In the files on Barry’s website is a Word file of the source code for my ARTMAP program. To use this code, just cut and paste into an appropriate “Class” in VS2017. Please note that there is a lot of code there that is not used yet in preparation for future presentations.
Unusual in this program is that I am reading data from an external file. Please read the document also on Barry’s web site called “Reading CSV files” for more information. Part of the relevant code is shown above. You set up a “TextFieldParser” object using a comma as a delimiter and read fields from the mushrooms_vectors.csv file which contains all the mushroom feature data.
The ARTMAP Program

Using ReadInput As New Microsoft.VisualBasic.
    FileIO.TextFieldParser(
        "..\mushrooms_vectors.csv")
    ReadInput.TextFieldType = FileIO.FieldType.Delimited
    ReadInput.SetDelimiters("","")

Reading Data from Files

The next thing to notice about this code is that it reads the file from a relative location instead of absolute. The format "..\mushrooms_vectors.csv" says that the mushrooms file is in the same directory as this file is. That means it is in the "\bin" folder in your Visual Basics projects directory. (show this). If you use this relative referencing and put the file in the bin folder, you won’t have to change the reference when you move the program to a different computer.
The ARTMAP Program

Reading Data from Files

Here is the way it looks in my file system on my desktop
Here we see the code that reads the input data into a local array called `CurrentRow`. We do this sequentially for all the Training data.

```plaintext
The ARTMAP Program

If Train = True Then

    For vector_num = Train_Start To Train_N
        initialize_pres()
        Try
            CurrentRow = ReadInput.ReadFields()
```

Reading Data from Files
The ARTMAP Program

If Train = True Then

    For vector_num = Train_Start To Train_N
        initialize_pres()
        Try
            CurrentRow = ReadInput.ReadFields()

Reading Data from Files

This is a sequential read from the file. You can’t read it line by line. This why when you move from Training to testing, you must read the training vectors into “the bit bucket” before you start processing the test vectors.
As I mentioned last week, I normally program in Windows Forms. There is a file that tells you the properties of all the elements in a windows form but you are not supposed to use that file directly. You are supposed to create it manually. This is quite a task for a form that has a lot of elements like the one in this program but I will continue to look into ways to pass this on. In the meantime, the file that specifies these properties is on Barry’s website titled “ARTMAP form design”.
Here is a part of the ARTMAP1rormDesigner.vb file which you will find in the Solution explorer under the ARTMAP1form.vb tab.
Turn off Windows Spying after Every Update.

Settings – Update & Security – Advanced Options – Privacy Settings

Go through each option and turn off the spying.

W10 will reset your privacy settings after each update. Open Settings. Click on “Security & Updates”, then “Advanced Options” then “Privacy Settings” Go through each of the settings on the left side of the screen and reset them to what you want.

Now as we go out into the world, love one another, keep your anti-virus definitions updated and do your back-ups.