Ethan Criss
Eitan Romanoff
Mark Thill

Neural Networks - Project 2

Summary

For the this phase of the project we had three main goals to meet. The first was choosing which tool we wanted to use on the project. The second goal was choose and prepare three datasets. The third was to load our data sets into a format that is understood by the tool we selected.

We investigated different tools and decided on using Pybrain as for our tool. The Analysis section contains our findings on the different tools we looked at. We were successful in finding and preparing three datasets. More comments about each individual dataset is also in the Analysis section. Were also able to load our datasets into Pybrain's dataset module.

Requirements

Our understanding of the requirements of this project in particular is preparation of data for use during the training and research using neural networks. We believe that this section of the project is to ensure that we have appropriately selected datasets to use with our neural networks. This is important because improperly select data is not useful for reporting on our findings. If the dataset were to be too small then no conclusions could be drawn about the validity of our methodology or our results. It is equally important for this data to be normalized in order to not run into problems while training our neural network. Not normalizing our data can lead to strange results and inappropriate conclusions. The reason for this section of the project is to ensure that the data we are using is correct and that we will not run into issues with this data in the future.

Specification

For this project we would like to combine and modify the two Neural Evolution approaches we researched during the first phase of the project. We felt that the paper on predicting stock actions was the most appropriate for us to tackle during the quarter and offered the most promising results. The paper showed that their simple approach produced noticeable results and falls right in line with partial neural networks. We feel that it will be a good fit and already has a benchmark for us to test against.

We would like to modify the original project presented in the paper. The paper states that partial neural networks were created by setting up a fully connected neural network and assigning every link with a probability. A random number was chosen for each link to decide if that link would stay or be removed. This created purely random partial neural networks that were then not modified on their topology but only on the values of the weights for the links. We would like to modify this process by allowing breeding and mutation with the links as well. Populations would not only modify their weights but would also modify the topology of the network to try to find more fit individuals and gain a lower error rate.

Our proposal for achieving this only differs from the original paper on one point. We would create the partial networks initially in the same manner, by using random numbers. At this point we plan on creating a data structure to hold information about what links are active and what the weight value is for the link. This can be done using a simple array. During the process of modifying networks, we would use a genetic algorithm to pass connection and weight information from one network to another in order to create an offspring. This offspring would have characteristics of both parents and could potentially find combinations of links that were not created in the initial step. Genetic algorithms to pass information between parents as we have stated is a well documented obtainable goal that we feel can improve the results that were gotten from the original paper.

To measure our success we plan to measure the accuracy of our partially connected networks and compare it to the accuracy of a fully connected and trained neural network. We would like to compare and see what kind of accuracy's can be achieved by only using a partial network. Whether or not we are able to achieve relatively equal accuracy between fully connected and partially connected networks we will also measure the number of epochs each needs to stabilize. With this information we will be able to compare the networks and draw conclusions about the speed of training of each network. With this information we hope to draw conclusions about the relatively accuracy and training speed of partial networks and fully connected networks.

Feasibility and Analysis of Tools

Weka

One of the machine learning tools we looked at was the Waikato Environment for Knowledge Analysis known as WEKA. WEKA offers a large set of features that aid in handling machine learning problems. WEKA's library is divided into four main modules: Core, Classifiers, Filters and GUI.

The core module stores all of the data structures and classes needed to internally represent and manage the framework.

The Filters module gives support for a lot of the preprocessing commonly applied to a dataset. This may involve techniques for reducing noise in a dataset or normalizing between a set of values. Weka also gives support for attribute selection and manipulation with techniques such as principle components analysis or chi squared testing.

The GUI module gives a visual interface to access the functionality of WEKA. Although all the modules can be accessed through their java libraries, most operations supported by WEKA can be done completely through the GUI interface. Through the explorer interface datasets can be loaded and preprocessed. A classifier or clustering algorithm can then be run on the preprocessed dataset according to the parameters set. One of the powerful features of the GUI libraries is that there is support for visualising the data results of a classifier.

The classifier module contains a large set of classification algorithms that Weka has support for. Some of the types of classifiers supported are tree classifier like C4.5, probabilistic classifiers like variations on a Naive Bayes, and rule based classifiers to name a few. Most importantly there is support for Artificial Neural Networks with WEKA's multilayer perceptron classifier.

There are some customisable parameters in WEKA's multilayer perceptron. The number of hidden layers and the number of nodes in each layer must be provided. The learning rate, momentum, and decay during training can also be customised. There is some support for adding additional nodes to a network via a GUI interface, but it must be redefined for every networked trained. If any cross validation is used this can be tedious.

The focus of our group's research is on partially connected neural networks. Because partially connected networks use topologies that are quite different from the standard fully connected network, it is important that a tool we select has strong support for custom networks. Although WEKA has limited support for custom networks through the GUI tools, it is not readily available. This makes WEKA not the best fit for our area of research. If our project intended to use a feed forward fully connected neural network, the features provided by WEKA would have been a better fit.

Matlab

Matlab is a programming environment that allows for the easy manipulation of matrices and arrays. It allows for group operations to occur in every entry of an array or matrix with only a single command and supports iteration without the use of loops. Using these properties a neural network can easily and efficiently be represented in Matlab without the need of excess code or objects. It is a language that executes quickly and supports many graphing features to plot progress of a network as training is occurring.

The reason that this language is being looked at is because one of the original implementations of the NEAT algorithm was written in it. The project used Matlab object classes to hold information about the neurons, the weights, links, innovations, as well as all of the functionality for the necessary operations. This gave us a great amount of hope that we would be able to easily implement our own version of this program in the same language.

Matlab however is a rather obtuse system to work with. It has a very high initial learning curve because of its' differences from other programming languages. For many new users and programmers it is a difficult system to work with and understand. Many of the functions and properties of Matlab and its workings are not immediately clear and need to be researched. Furthermore many of the things that Matlab does is counterintuitive, and confusing. Although the documentation for the language is complete and readily available it is often just as difficult and hard to understand due to the highly technical nature of its writing.

Other knocks against Matlab include difficulties in understanding program flow and debugging. Working with large complex series of data and many objects in Matlab can be a painful experience for unskilled users. There are no easy, clear ways to trace through code in this situation and see how data is being passed around besides tracing using print lines. There are also some inherent properties that are not clearly stated for certain aspects of the system that make it all the more difficult.

PyBrain

PyBrain is an Artificial Neural Network framework and toolset that is built upon the open dynamically typed interpreted language of Python. Much like many other neural network frameworks and toolsets available, it provides a suite of quick-start tools that allow a user to create a neural network and apply it over datasets programmatically. PyBrain boasts its power in implementing cutting-edge algorithms to apply to its neural network structures, and its modular design.

Python - in particular, PyBrain - is an attractive tool to use for this project for multiple reasons. First and foremost, Python is a commonly used tool in Artificial Intelligence, and documentation is widely available. Dynamic typing allows a focus on design and process without being limited by obtuse datatypes. In addition to this, Python can run on any platform without hassle. Furthermore, the language supports easy object creation, first class functions, and has a large set of frameworks that make high-level design simple. Python is also a scripting language, so any automated testing or setup functionality can be scripted in the same environment.

Perhaps the most important aspect of PyBrain is its modular design, as it allows for a freedom of design with this project that is not so easily available in other platforms such as Weka. Perceptrons, Connections, and Networks are all separate python modules that can easily be decorated. This transparency and modularity allows for a level of detail necessary in implementing the dynamic addition, removal, and modification of Perceptron connections - the focal point of this project.

In addition to the modules provided by PyBrain, there is a very large collection of python libraries available for computation, including SciPy, NumPy, and Matplotlib for data representation and visualization, which will help any mathematical or scientific analysis conducted for this project. There are downsides to using the Python language and the platforms provided, however. Much of this functionality and ease of use comes at the expense of overhead, resulting in a performance hit by nature of the interpreted structure of the language. Because of these reasons, we chose to use Pybrain for this project.

Yeast Dataset

One of the data sets we choose is a data set for predicting protein localization sites for Yeast bacteria. This dataset was found in UCI Machine Learning Repository and is originally from the Institute of Molecular and Cellular Biology in Japan.

The properties of this data set are well suited for the area of research our group is looking at. the number of samples available are 1484. There are nine attributes, eight of which are numeric and one is categorical. The numeric attributes are already normalized to fall within the range of 0 to 1 so there is no additional normalization needed. The categorical attribute is a unique identifier for each sample and will add to creating a generalized classifier so this attribute can be ignored. The output of this data set is one of 10 classes. To make this dataset compatible with pybrain, the tool we have chosen, we had to modify the representation of the output. The categorical output is split into 10 numeric outputs, one representing each class. For each sample the output that represents its original output category is assigned a value of 1 while the other outputs have a value of 0.

The were a few motivating reasons for picking this data set. Primarily, our research focus is on using a genetic algorithm to create a partial connectivity neural network that competes with a traditional method like a fully connected network trained with backpropagation. This dataset is does not require much manipulation and feature extraction. It is pretty much ready to be inputted into a network. Also, the number of samples was a good range, it is not too big or too small of a dataset.

Abalone Dataset

Another one of the data sets we selected is a data set for classifying small sea snails called abalones. This dataset was found in UCI Machine Learning Repository and is originally from the University of Taroona’s Marine Research Laboratories.

This dataset suits the project well because, much like the yeast dataset, all of the data is numeric and is easy to centralize and normalize. Unlike the yeast dataset, however, there are only three classes that each instance in the dataset may fall into (male, female, or infant). There are 8 attributes in this dataset, including the classification label. Furthermore, the dataset is sizeable - 4177 instances, which allows us to select a large number of instances to train and test with for this relatively simple application.

In order to make this dataset work with PyBrain, it was necessary to change a few things. First, unlike the other datasets, the class attribute was listed first, rather than last. All other standard normalization processes were applied to the dataset. For the classifier, a series of 3 discrete values is used. A male, for example, is represented as a 1 in the male column and a 0 in the other class columns. This allows the attribute vector to be loaded into pybrain seamlessly.

Poker Dataset

The last of the data sets we chose to work with is a set of data that consists of poker cards and the hand value. This dataset was found in UCI Machine Learning Repository and is originally from Carlton University.

The properties of this data set are as follows. Each entry contains information about a possible five card hand in poker. The entries are each of a unique hand and the entire dataset contains well over a million entries. Each entry contains information about what the card value is, as well as the suit of the card. Each entry also contains class data as to what the hand represents. These classes are a runt, a pair, two pair, three of a kind, etc. All of this data is represent as numeric values. The value of the card being between 1 and 13, the suit between 1 and 4 and the hand class between 0 and 9.

The motivating reasons behind choosing this data set is an interest in all of the group members with poker. As soon as the group saw this particular dataset it was decided that it was one that needed to be used. There are also good economic reasons for choosing this set. In recent years online gambling has exploded into a huge business. The ability for a computer to recognize what kind of hand it is holding while playing a human is an important task. Although using neural networks to solve this problem may be overkill it is a fitting task for this assignment.

It should be noted that this dataset is extremely large - in total one million entries. We decided that it was sufficient to take a subset of this data as an example set, and that the full number of entries used will depend on the computation time regarding the cleaning and analysis over this information.

Implementation
As referenced above, the selected datasets were not initially configured to be used in a neural network, nor did any of the datasets contain a standardized manner of representation. For example, whereas the yeast and the poker datasets contained its classification attributes at the end of each row (taking up the last column), the abalone dataset had this reversed. Furthermore, whereas the abalone dataset and the poker dataset were represented as standard CSV files, the the yeast dataset contained an irregular number of spaces between each entry - making it difficult to create a standard manner of reading in the data.

One issue that all datasets contained lies in the representation of the classification attribute. PyBrain allows for a general manner of reading in data to put into a neural network structure, taking in both an input vector, as well as an output vector. The input vector is essentially a list of all the attributes against which the classification is found. The output vector is a vector that contains a series of values that are essentially the confidence values for the particular data entry in each classification. This means that each of the datasets need their classification attribute “split. In the case where 4 classes exist for a given dataset, this needs to turn into 4 separate “class” attributes, where only one class has a value of 1, and the rest are left at zero.

Beyond simple data cleaning and formatting, each script also normalizes the data using standard normalization techniques. After the data is formatted, the script computes the mean value for each of the attributes and centralizes each attribute by subtracting a particular value in a row by the mean. The script also computes the standard deviation, and each instance of every attribute is normalized by having the value of that instance divided by the standard deviation.

For the sake of consistency and simplicity, python was used for creating these scripts, and all operations use the language’s built-in libraries. For information on how to run the scripts, check the README files included in the zip.

Lastly, it should be noted that to run any code present in this project, a working installation of python 2.7 is required, along with a working installation of the PyBrain module.