Heriot-Watt University

Final Year Dissertation

Fast Algorithms for Hard Problems

Author:
Joseph Ray Davidson

Supervisor:
Dr. David Corne

January 13, 2011

Abstract

In computer science, there are many problems that can currently only be rigorously solved by using an exhaustive search method, brute force. For these problems, it is acceptable to instead only search for a solution that is a 'good enough' approximation for the task at hand. As a general rule, there is an accuracy – speed trade off for these methods.

In this dissertation, I shall investigate the usage of certain techniques to obtain more accurate results without sacrificing speed. **********

Contents

1 Introduction

2 Greedy algorithms

3 Artificial neural networks

For this part of the experiment, I implemented a framework to build and train artificial neural networks. This framework allows a user to build acyclic feedforward ANN of arbitrary size and makes use of an implementation of the backpropagation algorithm to train the network.

3.1 Architecture

The ANN as a whole has two classes: the Perceptron class[1] and the ANN class. External programs interact with the ANN class, which in turn uses the Perceptron class to create the sigmoid units which make up the network itself.

Sigmoid units were chosen over perceptrons becuause of the diffrent threshold functions of the two. Perceptrons calculate the threshold output as a linear combination of the inputs, which is only really useful in single layer neural networks. sigmoid units, on the other hand, calculate the output as a continious function of it's input as shown below.[Mit97] (1)

$$\sigma(y) = \frac{1}{1 + e^{-y}} \tag{1}$$

The ANN class keeps a record of all of these sigmoid units and each newly unit has an identifier, a list of input units, a randomised corresponding weight for each input (in the range -0.05 to 0.05), a list of output units as well as a number of variables that allow the ANN class to see if it has recently performed a calculation and how many inputs it need before it can make a calculation.

There are a number of conventions that the ANN framework uses to keep things in order:

- All sigmoid units have a constant input of 1. This input does have a random weight like all the other inputs however. By convention, this input comes from a unit with an id of -1.
- All of the inputs into the network are modelled as coming from units with negative ids. Because -1 is reserved, the inputs start at -2 and sucessively decrease with each new input.

[1] 'Perceptron' in this context is actually a misnomer, as the units involved in this implementation of the framework are not perceptrons but sigmoid units. The reason for this is a significant portion of the code referred to "Perceptrons" before I realised that sigmoid units were preferable when creating multilayer ANNs. A change at this point could have seriously damaged the code, introducing severe bugs. So I elected to keep it the same. For the rest of the paper, I shall use "sigmoid" when talking about the units and "Perceptron" when talking about the class.

- All of the units in the output layer give send their output to unit 0. This is to identify the output as the result of a finished computation so it can be collected with other results.

A calculation requires a set of input units and the corresponding values for each. In practice, this is modelled by a 2-dimentional array. The ANN class prepares each unit by giving it the input values it needs and then invokes the unit's fire() method. The unit then calculates its output value and returns the output along with a list of the units that the output is to go to.

The ANN uses this information to prepare and fire the other units in the network and continues to operate in this manner until all of the units have fired. The output from the network is collected and returned.

3.1.1 Other features

The ANN class has

1. For each unit in the network:
   (a) Reset.
2. While all of the units haven't fired:
   (a) For each unit in the network:
      i.

4 Kernelization

5 Results and conclusions

A Code

References

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1 edition, March 1997.