HERIOT-WATT UNIVERSITY

FINAL YEAR DISSERTATION

Fast Algorithms for Hard Problems

Author:
Joseph Ray DAVIDSON

Supervisor:
Dr. David CORNE

February 15, 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 Framework architecture

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

There were two choices for the computation unit of the ANN, sigmoid units and perceptrons. Sigmoid units were chosen over perceptrons because of the different threshold functions of the two; Perceptrons calculate the output as a linear combination of the inputs, if this combination is over a given threshold value, a value of 1 is outputted, otherwise -1. Such units are not useful for approximating complex functions.

Sigmoid units, on the other hand, calculate the output as a continuous function of it's input as shown below.[Mit97] (1) With this framework, I am making the implicit assumption that the vertex cover problem is a continuous function rather than a discontinuous one.

My reasoning for this is as follows: We can trivially see, for any given graph $G(V, E)$, that the minimum valid vertex cover VC with the lowest cardinality is $\mid VC \mid = 0$, this fact holds when $E = \emptyset$. Conversely, the minimum valid vertex cover VC with the highest cardinality is $\mid VC \mid = |V| - 1$. This is true when G is a complete graph (i.e. $|E| = \frac{|V|(|V|-1)}{2}$).

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

We can therefore see that there is correlation between the cardinality of the set of edges E and the cardinality of the minimum vertex cover VC associated with G. This correlation compels me to conclude that the vertex cover problem is an instance of a continuous function.

The ANN class keeps a record of all of these sigmoid units and each newly created 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 sigmoid units with no input list and a flag that indicates they are part of the input layer.
- All of the units in the output layer have no output list and a flag that indicates that they are in the output layer.

3.1.1 Calculations and training

A calculation requires a set of input units and the corresponding values for each. In practice, this is modelled by a 2-dimensional array. The ANN class prepares each unit by giving it the input values it needs and then invokes the units 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.

To train the network using the graph instances created by the test case generator, it was decided that the network would be given features of individual vertices and then it would output two probablities: the probability of the vertex being in the covering set, and the probability of it not. Backpropagation would then ensue, applying the correct result to the network and adjusting the weights.

******* Validation here *******

3.1.2 Other features

The ANN class has

3.2 ANN instance design

3.2.1 Feature selection

Feature selection was largely a case of thinking of a number of vertex properties and selecting those which would likely help in determinining whether a particular vertex was part of the covering set or not. These features were used to create an exhaustive data set (the details of which are explained in section 4.1).

Each instance in the set was then analysed by an implmentation of náive Bayes which trained itself on 80% of the data and attemped to correctly classify the remaining 20%

4 Test case generation framework

In order for an ANN to be an efficient solver of problems, it must first be trained. This training takes the form of supervised learning, where and input and a target output are given to the network. It processes the input, compares the calculated output to the provided one and adjusts its weights accordingly to produce more accurate results.

The observant reader will have already spotted a potential problem. If we're to train the ANN with any effectiveness, we'll have to create a supervisor that can not only produce problem instances, but can also solve them in order to pass the answer to the ANN. Therein lies the issue, we have to create non-trivial instances of the problem and then solve them.

4.1 Creating statistic datasets

5 Kernelization

6 Results and conclusions

A Code

References

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