11-2014

A Parallel Genetic Algorithm For Tuning Neural Networks

Nathan Chadderdon
Knox College

Ben Harsha
DePauw University

Steven Bogaerts
DePauw University

A Parallel Genetic Algorithm For Tuning Neural Networks

Nathan Chadderdon[2], Ben Harsha[1], Steven Bogaerts[1]

1. DePauw University Greencastle, IN
{benjaminharsha_2015, stevenbogaerts}@depauw.edu

2. Knox College, Galesburg, IL
nchadder@knox.edu

Abstract

One challenge in using artificial neural networks is how to determine appropriate parameters for network structure and learning. Often parameters such as learning rate or number of hidden units are set arbitrarily or with a general "intuition" as to what would be most effective. The goal of this project is to use a genetic algorithm to tune a population of neural networks to determine the best structure and parameters. This paper considers a genetic algorithm to tune the number of hidden units, learning rate, momentum, and number of examples viewed per weight update. Experiments and results are discussed for two domains with distinct properties, demonstrating the importance of careful tuning of network parameters and structure for best performance.

Setup

The first stage of this project was the creation of a multi-layer neural network, a common machine learning structure based on biological neurons and the connections between them. This network was designed to work with a variety of domains.

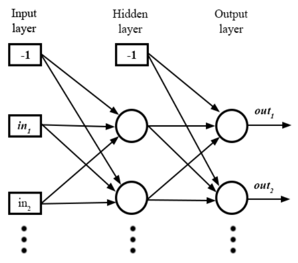

Fig 1. A multi-layer Neural Network

Next, a method was required to explore which network setups performed better. In this case a genetic algorithm, which mimics the process of evolution, was used. Tests were run by inputting certain restraints and allowing the genetic algorithm to optimize the networks. The best network from each test was recorded for analysis.

Parallelism

Due to the fact that both neural networks and genetic algorithms have long run times, parallel programming was implemented using OpenMP to increase the speed of computation.

```
#pragma omp parallel for if(hiddenParallel == true)
for(int i = 0; i < numHidden; i++){
    //update bias-hidden weights
    #pragma omp atomic
    tempbhWeights[i] += learningRate * -1.0 * hDeltas[i];
    for(int j = 0; j < numOutputs; j++){
        if(i == 0) {
            //update bias-output weights
            #pragma omp atomic
            tempboWeights[j] += learningRate * -1.0 * oDeltas[j];
        }
        //update hidden-output weights
        #pragma omp atomic
        temphoWeights[i][j] += learningRate * hActs[i] * oDeltas[j];
    }
}
```

Two sections were parallelized, the first being the training calculations for the network and the second being the creation of individuals in the genetic algorithm. Both methods produced similar speed increases.

Genes

Genetic algorithms use genes to determine the traits of the individuals neural networks in the population. In this experiment there were four genes describing the different network setups:

- **Hidden Units** - the number of neurons in the hidden layer of the neural network (Fig. 1)
- **Batch Size** - the number of examples to be considered during a weight update
- **Learning Rate** - the speed at which the network learns
- **Momentum** - the percentage of the previous weight update used in calculation for the new weight update

67	20	0.306	0.320
Hidden Units	Batch Size	Learning Rate	Momentum

Fig 2. A sample genome taken from the genetic algorithm

Learning Rate

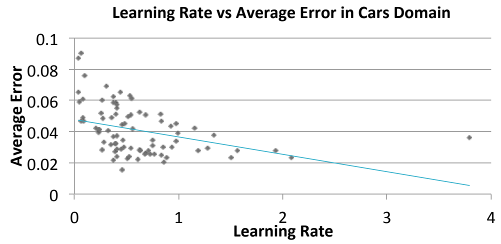

Fig 3. Effect of Learning Rate on Average Error

The Cars domain preferred to use higher learning rate. This contradicts previous assumptions that small learning rates around 0.1 or 0.3 were optimal. The Splice domain behaved as expected, and selected low learning rates in the 0.1-0.3 range.

Number of Hidden Units

Hidden Units Trends in Cars Domain

Max Hidden Units	Experiments Run	Below 50% of Max
30	46	34
50	8	7
100	17	13
150	8	7

Fig 4. Number of Experiments that settled below 50% of their maximum number of hidden units

The number of hidden units preferred to settle on low values relative to the given maximums. This trend continued in the Splice domain, although the range of values chosen in that domain was slightly larger.

Batch Size

Batch Size Trends in Cars Domain

Max Batch Size	Experiments Run	Below 50% of Max
50	54	40
100	17	17
150	8	7

Fig 5. Number of experiments that settled below 50% of their maximum batch size

Batch size had a tendency to settle on values that were low relative to their given maximums. This trend held true for both domains, with each giving very similar values for batch size.

Momentum

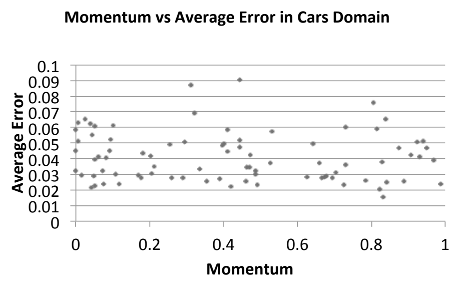

Fig 6. Effect of Momentum on Average Error

Momentum did not affect results for either domain. The values chosen were seemingly random and there was no trend that indicated a positive or negative effect on the average error.

Conclusions

- **Learning Rate** – Preference for learning rate depended strongly on the domain
- **Hidden Units** – Tended to settle on very low values for Cars domain, slightly higher values for Splice domain
- **Batch Size** – Tended to settle on low values for both domains
- **Momentum** – No effect on the results

Acknowledgement

This project was funded by the National Science Foundation.
Grant number: CNS-1156893