Texas A&M University

Summer Research Paper 2

Applications of Artificial Intelligence to Number Theoretic Problems: Artificial Neural Networks and the Integer Factorization Problem, Primality Testing, and the Greatest Common Divisor

Authors:
Stephen Capps
Sarah Sahibzada
Taylor Wilson

Supervisor:
Dr. Sara Pollock

July 5, 2015

Contents

1 Introduction

Artificial Intelligence (AI) is a growing area in computer science with a myriad of applications across disciplines: cars have moved from being directly human-controlled to all but driving themselves, and airplanes are now successfully taking off and landing on autopilot. AI is divided into subgroups of study with seven fundamental areas– reasoning, knowledge, planning, learning, communication/natural language processing, perception, moving and manipulating objects–as well as an eighth, the notion of distributed intelligence. (Murphy, Introduction to AI for Robotics). A similar term, machine learning, refers to the ability of a computer to study data, form patterns, and make predictions. Typical applications of artificial intelligence involve pattern-recognition problems robotic navigation; however, the uses of artificial intelligence and machine learning methods as tools to solve number-theoretic problems have not been thoroughly investigated. This investigation seeks to evaluate the effectiveness of an artificial neural network (ANN) in three key problems: classification of prime and composite numbers, integer factorization, and determining the greatest common divisor (GCD) of two numbers. Due to the scope of this investigation and the nature of an artificial neural network, the efficiency of the artificial neural network in each problem was not compared to the efficiency of existing algorithms to solve these problems.

2 Theoretical Analysis: Neural Networks

2.1 Artificial Neural Networks and their Architectures

At its core, an artificial neural network is an abstraction for the functionality of neurons and human cognition. The neural network is a collection of interconnected neurons, which are themselves abstractions of human neurons. (Russel and Norvig, Artificial Intelligence: a Modern Approach). The process of learning is achieved through the alterations of synaptic connections between these neurons, making the ANN an ideal choice for pattern-recognition problems: synaptic connections may be weighted in favor of "better" outputs.

2.2 The Neuron

The neuron may be represented as below: (((note: everything below this refers to a diagram.))) Input is provided to an input function; a link from

the input i to input j serves to propagate the activation from i to j. (Russel and Norvig, Artificial Intelligence: a Modern Approach). This input has an associated weight; each neuron will compute a weighted sum of its inputs and then apply an activation function to act as a threshold of sorts in order to derive an output. These thresholding functions can be as simple as logical operators AND, OR, and XOR, but can also be more complicated by using linear, sigmoid, or hyperbolic tangent functions. The function is generally chosen by the complexity of the problem, though can require some trial and error to determine the best fit. Neurons whose activation functions create hard thresholds are called perceptrons; neurons whose activation functions are logistic functions are known as sigmoid perceptrons. (Russel and Norvig, Artificial Intelligence: A Modern Approach) The perceptron maps input directly to output. It is relevant to note that in the early history of neural networks, single perceptrons were used to solve problems, albeit without a high degree of success.

2.3 Network Architectures

The neuron is the fundamental unit of the neural network; the interconnections of the neurons give rise to the properties of the network. There exist two principal ANN architectures: feed-forward networks, and recurrent networks. Due to the nonlinearity of the function that may be represented by the network, these allow for non-linear regression problems to be computed. (Russel and Norvig)

2.3.1 Feed-Forward Networks

A feed-forward network may be visualized as the directed, acyclic graph below ((diagram)). Note that the connections flow in one direction only. It maintains no internal state nor represents more than the input passed to the network. Due to the nature of the graph representation of the feed-forward network, the feed-forward network may be split into multiple layers. The first layer is the input layer. This is layer that handles the data entered into the network. This information is sent to the hidden layers where the real magic happens. They are called hidden layers because the building blocks are set up, but the computation is left out of reach. The hidden layers decide, based on a weighting system, the connections to make to the output layers. The weighting system is determined by functions that are chosen by the programmer and computed by the network. In general, this layer will not directly connect to the output layer. Due to the lack of internal state,

the feed-forward network lacks any kind of memory.

2.3.2 Recurrent Networks

A recurrent network may be visualized as below: ((diagram)). Note that the graph representation of this network permits cycles. This is because a recurrent network utilizes the notion of recursion: it will feed its outputs back into its own inputs, allowing for the network to maintain state and thus support some degree of short-term memory. (Russel and Norvig).

2.4 Learning

Learning through an ANN can be either supervised or unsupervised. Supervised learning is done through an external teacher. The network is told the desired response to certain input signals. Reinforcement learning is a form of supervised where a correct response is given a certain weight and the better the weight, the more correct the answer. Unsupervised learning, also known as self-organizing, is based entirely upon local information. The network is given a large amount of data and allowed to recognize its own patterns.

Implementation of any learning algorithm, supervised or unsupervised, is complicated by the presence of multiple outputs and hidden layers of the network: the fact that the hidden layer leaves its computations out of reach of the user means that any information obtained about errors at the hidden layer is all but useless to the user. Therefore, an algorithm for backpropagation must be used to back-propagate errors from the output layer to hidden layers. The algorithm is detailled below: algorithm detail here

3 Computational Approaches

3.1 Implementation Detail

The ANN was implemented in Python, using the Pybrain machine learning library. No other external libraries were used. All code was run on Canopy Python 2.7.6 in both the Canopy IDE for Mac and Windows, as well as a Mac Terminal. [everyone input your machine specs here] For

factorization, a class was defined containing prime factorizations of numbers 1 through 15. Data was not normalized, and was represented using the in-built Dataset type in Pybrain as a vector of one input and up to three outputs. For primality testing, a list of prime numbers up to 10,000 was obtained; the data were treated as a Pybrain classification data set, and the network was implemented accordingly. For the GCD problem, numbers were randomly generated from 0 to 100, and every possible combination of pairs tested. In all cases, feed-forward networks with backpropagation training were used; varying proportions of the initial data were used as training data.

4 Results

In the previous project, several algorithms that calculated the greatest common divisor (gcd) were tested for efficiency. The next logical step was to determine whether the efficiency could be improved upon. AI was chosen as a possible way to do this. Surely if a computer can figure out how to drive a car, it can learn the relationship between a set of two numbers and finding the greatest common divisor. The network was set up as a reinforced backpropagation system using Pybrain in Python. The teaching dataset used was combinations of (x,y), with x ranging from 2-5 and y ranging from 1-20. The input was a pair of x and y combined with the desired output of the gcd. It was set up to randomly go through $10,000$ iterations of the training dataset and then attempt to guess the output for each of the teaching entries. For the majority of the runs, there were 2 hidden layers activated. After many executions of the network, several things were determined to have an effect on the output. When the learning rate was increased from 0.05 to 0.1, the results improved slightly, though not drastically. Momentum was determined to work best at the value 0.025 , much higher and it caused the program to crash, however if it was too low, the program would get stuck in a trough. Surprisingly, the organization of the training data had some of the most interesting results. You could almost feed the computer already established patterns to recognize. Many different combinations were tried with the organization of the output gcds. Generally, whichever two were listed first in the training dataset would be guessed exactly correct. After those two, it would begin to go wrong very quickly. Though there were many different ways to produce varying results, none of them were able to produce anything close to the desired results.

5 Discussion

There are several explanations that could be applied to why the network was unable to produce the desired results. First, the training dataset could have been too small. There was never more than 60 inputs within which to find a pattern for the factorization problem. Sometimes it is difficult to find a pattern in a small section of data. Though the patterns may have been visible to the human eye, the computer has no prior knowledge of divisors to rely on when trying to find the pattern. Secondly, it could be related to the fact that prime numbers do not follow a well-defined pattern. They converge to a distribution, however it is not uniform, therefore it would take an extremely sophisticated machine to recognize that pattern. This would mean that a pattern would be difficult to find for a pair of numbers where one of them is prime. Another relevant factor to consider is the fact that only feed-forward networks were used, and not all potential functions for activation were tested: current literature indicates that hyperbolic tangent functions, one of the activation functions used, are ineffective for the factorization problem.

The scope of this investigation was certainly a non-trivial contributing factor to the fact that the network was unable to produce the desired results. While various parameters of the network were tested, very little was tested in the way of data representation beyond an elementary normalization of GCD inputs and outputs, a fundamental issue in problems of artificial intelligence (Murphy, Introduction to AI for Robotics). Future work will consider alternative representations of data that are more conducive to this problem and the architecture of the chosen network.

Finally, it could be that a neural network in the implementation used for our project is simply not well suited for this type of problem. There have already been well established algorithms, as demonstrated in the previous project, which rely on more concrete building blocks than just recognizing patterns that may not even be there. It is possible that other methods of artificial intelligence, such as support vector machines, might have more success at one or all of the problems investigated.

6 Individual Contributions

6.1 Stephen Capps

Did background research in Neural Networks and explored different application areas. Also covered main Latex document editing and formatting.

6.2 Sarah Sahibzada

Investigated the use of artificial intelligence in number theoretic problems such as factorization. Using Pybrain, implemented a feed-forward neural network and tested it against factorization data, as well as normalized GCD data.

6.3 Taylor Wilson

7 References