Table of Contents

Quantum and Evolutionary Approaches to Computational Learning

A Dissertation
Presented to the
Department of Computer Science
Brigham Young University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

by
Dan Ventura

August 1998

This dissertation by Dan Ventura is accepted in its present form by the Department of Computer Science of Brigham Young University as satisfying the dissertation requirements for the degree of Doctor of Philosophy.

Tony R. Martinez, Committee Chairman

Jean-Francois Van Huele, Committee Member

Parris K. Egbert, Committee Member

Bryan S. Morse, Committee Member

John C. Higgins, Committee Member

Date

Scott N. Woodfield, Graduate Coordinator

???

Acknowledgments

First and foremost I would like to thank my Heavenly Father for sustaining and providing for me and my family throughout the time required to complete this work. We have been the recipient of so many blessings.

Of course, a good deal of the credit for my progress and production goes to my advisor, Tony Martinez, who taught me, most importantly, how to think and who has been a great friend with whom I have thoroughly enjoyed working.

There have been many other faculty members and colleagues who have contributed in one way or another to my education, providing encouragement, insight, and helpful criticisms. For fear of excluding someone, I will not attempt to name them here; nevertheless I am grateful to each one.

I am also very grateful for the financial support that I received from various agencies during my research. In particular, Novell Inc. and WordPerfect Corp. provided early support, and the Computer Science Department has generously provided both financial support and access to excellent resources.

Finally, I am so thankful for the love and support of my wonderful wife, Marcia, who has put up with a lot for the past few years and has been more help to me than she will ever know.

Part I

Background and Motivation

The most exciting phrase to hear in science, the one that heralds new discoveries, is not "Eureka!" but "That's funny..."
-- Isaac Asimov

The problem of computational intelligence in general and computational learning more specifically is by nature extremely complex, multidimensional, and of necessity multidisciplinary. One particularly difficult aspect that presents itself in many forms is the seeming unavoidability of what has been termed "the curse of dimensionality". In essence what this term is meant to describe is the fact that even modest sized learning problems seem to embody a tremendous amount of information, and that information grows at an exponential rate as the number of features associated with the problem increases linearly. Thus, computational learning approaches are usually faced with a difficult choice: ignore the vast majority of the information inherent in the problem or face an exponential slowdown in performance as problem size grows.

This work considers both sides of the coin, with Part II making use of concepts from evolutionary computation in an attempt to avoid looking at most of the information available while being smart about what information is made use of. Part III then takes the opposite tack by considering an approach that has the capability to process an exponential amount of information in polynomial time -- quantum computation. Part I introduces these ideas and motivates the work. Chapter 1 briefly introduces the ideas of evolutionary and quantum computation and discusses the approach taken in the rest of the dissertation. Chapter 2 goes into more detail in presenting related work on evolutionary/neural hybrid approaches to learning, quantum computation, and quantum computational learning. Chapter 3 provides a high-level overview of many of the key issues involved in computational learning in general. It has been published under the following reference.

Ventura, Dan and Tony Martinez, "Concerning a General Framework for the Development of Intelligent Systems", *Proceedings of the International Conference on Artificial Intelligence, Expert Systems and Neural Networks*, Honolulu, Hawaii, pp. 44-47, 1996.

Chapter 1

Introduction

It has always seemed to me that the most difficult part of building a bridge would be the start.

-- Robert Benchley

The goal of automated problem solving has been approached in a myriad of ways. Often, the strengths of one approach will address those areas that are weaknesses of another. Therefore, it becomes desirable to discover combinations in which the strengths of several approaches can be combined to form an eclectic hybrid that takes advantage of the strengths of the various methods while avoiding their weaknesses. In this spirit, the work described in this dissertation draws on results from several areas; in particular evolutionary computation and quantum computation require some introduction.

1. Computational Learning

Computational learning is a broad term meant to describe the goal of developing computational systems that have the ability to adapt their behavior according to the information available to them. The field of computational learning is actually a fractured collection of work from many different subdisciplines including neural networks [Hay94] [Was93], machine learning [Sha90] [Mit97], and symbolic artificial intelligence [Lug97] [Rus95] to name a few. Researchers within the field often will not even agree on a basic goal, but I will suggest that one appropriate goal is the discovery of computational systems that exhibit problem solving capability. One particular problem that is often addressed in computational learning research is that of learning from a set of examples. In other words, given a set of examples, the goal first of all might be to produce a system that can simply recall the examples. Second, it would be desirable if the system were able to recall the examples from incomplete or noisy data, and third we would like the system to behave in a reasonable fashion for new situations for which it has not encountered an example. These three goals are often referred to as memorization, association and generalization. This dissertation will address all three of these, although as is typically the case in computational learning, the first goal is implicitly assumed as the minimum requirement for a system and as such is not explicitly considered.

2. Evolutionary Computation

Evolutionary computation seeks to imitate the evolutionary processes found in nature. The basic idea of natural evolution is that over the course of time a selection mechanism discovers traits that are in some sense favorable for the species in question. Evolutionary computation seeks to mimic this behavior with algorithms designed to maximize a fitness function with a search mechanism that employs genetic operators such as crossover and mutation [Gol89] [Spe93].

One of the strengths of evolutionary computation is its ability to guide a search through immense state spaces and find a reasonable candidate for maximizing some fitness function. The down side to this approach is that it is often quite slow. However, in many cases, the alternatives are slower yet and worse still may not find an acceptable candidate at all. Evolutionary computation is often used in conjunction with other computational learning methods, in an attempt to lend their excellent search capabilities to the strength of another approach. Some examples of such hybrid systems are discussed in Chapter 2.

3. Quantum Computation

The field of quantum computation was introduced in the early and mid-eighties when physicists began thinking about computation as a physical process [Ben82] [Deu85] [Fey86]. A very good introduction to the field may be found in [Bar96]. The idea of physical computation is to consider the initial state of a physical system and then to observe the outcome of its evolution through time, perhaps as it is affected by various outside influences. The smallest physical systems obey the laws of quantum mechanics, which for the most part are extremely nonintuitive. The field of quantum computation first addresses the question of whether or not quantum mechanical systems can perform *interesting* computation, and this question has been resolved in the affirmative. In fact, quantum systems can process an exponential amount of information in linear space and time. However, most of that information is unavailable outside of the quantum system. In the general case, no more information is available than that obtainable by classical means. This brings up the next question of whether or not we can control the systems to produce *useful* computation. The amazing aspect of quantum computation is that sometimes we can. Even though an exponential amount of information is never directly available, it is possible to extract extra global information about the system that is not available classically. Therefore, the trick is to recast classical computational problems in a form amenable to this quantum approach such that the global information can be used to advantage. Finally, there is the question of whether or not this is technologically possible, and this question is also beginning to be answered in the affirmative as exciting experimental results have begun to be produced [Chu98] [Jon98]. Some of the most important quantum algorithms produced to date will also be discussed in greater detail in Chapter 2.

4. Thesis Statement

Computational learning problems typically have enormous state spaces that present any algorithm with two choices: do not consider the vast majority of the space or slow down exponentially as the number of features associated with the problem increases linearly. Of course, in general, any useful algorithm must scale to large problems, so the second choice is not realistic. The real challenge for any useful algorithm is to somehow avoid most of the state space while still performing useful computation. This dissertation answers the challenge in two ways. The first is a hybrid approach for combining the thorough search capabilities of evolutionary computation with the speed and generalization of neural computation. The evolutionary part of the hybrid is used to sample the state space "intelligently" so that the neural network can then produce good generalization. The second approach to the computational learning problem is a theory of quantum computational learning that takes advantage of the unique characteristics of quantum systems. The quantum learning algorithms developed here provide a third alternative for dealing with the "curse of dimensionality" -- take advantage of the massive state space of quantum systems to process information in a way that is classically impossible.

5. Overview of Dissertation

This dissertation is divided into four parts. Part I consists of three chapters, the first of which (this one) serves as an introduction and covers necessary background and motivation for the work. Chapter 2 discusses work done by others that is pertinent to the research produced here. Chapter 3 presents a high-level survey of many hybrid approaches to computational intelligence and attempts to distill from them some key ingredients for developing computationally intelligent systems. This chapter also serves as a high-level motivation for the rest of the work.

Part II is dedicated to one particular hybrid approach to the problem of learning from examples. The basic hybrid system introduced in Chapters 4-7 combines the thorough search capabilities of evolutionary computation with the generalization and rapid execution of neural networks. The first chapter considers the problem, often overlooked in computational learning, of how to obtain the data needed to train a learning system. Chapters 5 and 6 then consider how the neural/evolutionary hybrid can be used as a learning optimization system. The last chapter in this part, Chapter 7, extends the idea to the domain of automatic control, demonstrating how the hybrid may be used to produce a successful neurocontroller.

Part III attacks the problem of computational learning from another angle, introducing ideas from the field of quantum computation. This part of the dissertation should be considered the most important research and the greatest contribution to computational learning. Chapters 8 and 9 are early attempts at integrating ideas from the two fields and produce important insights into what types of approaches may be taken. Chapter 10 presents a fundamental contribution to both the fields of quantum computation and computational learning by demonstrating how a set of classical examples can be represented in a quantum computational system. This result is a key ingredient for integrating the fields of quantum computation and computational learning, suggesting the promising new field of quantum computational learning. Chapter 11 builds on the work of the earlier chapters, particularly Chapters 9 and 10, as well as on the work of other researchers in quantum computation to produce a physically realizable quantum associative memory with a capacity far greater than that of classical models. This use of the term "physically realizable" is not meant to imply that this dissertation contains work dealing with such a physical realization of the algorithms developed here. It does not. The term simply refers to the fact that the algorithm is carefully constructed in order to obey all of the quantum mechanical laws necessary to make a physical realization *possible*. Finally, Chapter 12 introduces a new quantum computational inductive learning algorithm, revisiting the problem of learning from examples. It presents a Fourier-based quantum learning model that makes use of the unique characteristics of quantum computation to find Fourier basis functions that are well correlated with the known data. The algorithm presented here achieves significant speedup over any known classical algorithms. Further, it provides an important contribution to quantum computation because it does not assume *a priori* knowledge of a function.

Part IV consists of a single chapter that concludes the dissertation by summing up the contributions made, making some concluding remarks and suggesting areas for future research.

6. Publications

Chapters 3-12 are all papers that have either been published in or submitted for publication in refereed journals and conferences. The following is a list of references for these publications, broken up according to the parts of the dissertation in which they appear (or to which they are related). The reference for each paper is repeated at the beginning of the appropriate chapter.

I. Background and Motivation

Ventura, Dan and Tony Martinez, "Concerning a General Framework for the Development of Intelligent Systems", *Proceedings of the International Conference on Artificial Intelligence, Expert Systems and Neural Networks*, Honolulu, Hawaii pp. 44-47, 1996.

II. A Neural/Evolutionary Hybrid System

Ventura, Dan, Tim Andersen and Tony Martinez, "Using Evolutionary Computation to Generate Training Set Data for Neural Networks", *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, Ecole des Mines d'Ales, France, pp. 468-471, 1995.

Ventura, Dan and Tony Martinez, "A General Evolutionary/Neural Hybrid Approach to Learning Optimization Problems", *Proceedings of the World Congress on Neural Networks*, San Diego, California, pp. 1091-1095, 1996.

Ventura, Dan and Tony Martinez, "Robust Optimization Using Training Set Evolution", *Proceedings of the International Conference on Neural Networks*, Washington D.C., pp. 524-528, 1996.

Ventura, Dan and Tony Martinez, "Using Evolutionary Computation to Facilitate Development of Neurocontrol", *Proceedings of the International Workshop on Neural Networks and Neurocontrol*, Playa del Carmen, Mexico, 1997.

Ventura, Dan and Tony Martinez, "Optimal Control Using a Neural/Evolutionary Hybrid System", *Proceedings of the International Joint Conference on Neural Networks*, pp. 1036-1041, Anchorage, Alaska, 1998.

III. Quantum Computational Learning

Ventura, Dan and Tony Martinez, "An Artificial Neuron with Quantum Mechanical Properties", *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, Norwich, England,1997.

Ventura, Dan and Tony Martinez, "Quantum Associative Memory with Exponential Capacity", *Proceedings of the International Joint Conference on Neural Networks*, Anchorage, Alaska, pp. 509-513, 1998.

Ventura, Dan and Tony Martinez, "Initializing the Amplitude Distribution of a Quantum State", submitted to *Physical Review Letters*, June 1998.

Ventura, Dan and Tony Martinez, "A Quantum Associative Memory Based on Grover's Algorithm", submitted to the *International Conference on Artificial Neural Networks and Genetic Algorithms*, Portoroz, Slovenia, April 1999.

Ventura, Dan and Tony Martinez, "Quantum Associative Memory", submitted to *IEEE Transactions on Neural Networks*, June 1998.

Ventura, Dan and Tony Martinez, "A Quantum Computational Learning Algorithm", submitted to *Journal of Artificial Intelligence Research*, July 1998.

References

[Bar96] Barenco, Adriano, "Quantum Physics and Computers", *Contemporary Physics*, vol. 37 no. 5, pp. 375-89, 1996.

[Ben82] Benioff, Paul, "Quantum Mechanical Hamiltonian Models of Turing Machines", *Journal of Statistical Physics*, vol. 29 no. 3, pp. 515-546, 1982.

[Chu98] Chuang, Isaac, Neil Gershenfeld and Mark Kubinec, "Experimental Implementation of Fast Quantum Searching", *Physical Review Letters*, vol. 80 no. 15, pp. 3408-11, April 13, 1998.

[Deu85] Deutsch, David, "Quantum Theory, The Church-Turing Principle and the Universal Quantum Computer", *Proceedings of the Royal Society*, London A, vol. 400, pp. 97-117, 1985.

[Fey86] Feynman, Richard P., "Quantum Mechanical Computers", *Foundations of Physics*, vol. 16 no. 6, pp. 507-531, 1986.

[Gol89] Goldberg, David E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[Hay94] Haykin, Simon, *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing Company, 1994.

[Jon98] Jones, J. A. and M. Mosca, "Implementation of a Quantum Algorithm to Solve Deutsch's Problem on a Nuclear Magnetic Resonance Quantum Computer", *Journal of Chemical Physics*, to appear, 1998.

[Lug97] Luger, George F. and William A. Stubblefield, *Artificial Intelligence*, Addison Wesley Longman, 1997.

[Mit97] Mitchell, Tom M., *Machine Learning*, McGraw-Hill, 1997.

[Rus95] Russell, Stuart and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, 1995.

[Sha90] Shavlik, Jude W. and Thomas G. Diettrich (eds.), *Readings in Machine Learning*, Morgan Kaufmann, 1990.

[Spe93] Spears, W. M., K. A. Dejong, T. Baeck, D. Fogel, H. de Garis, "An Overview of Evolutionary Computation", *European Conference on Machine Learning (ECML-93)*, 1993.

[Was93] Wasserman, Philip D., *Advanced Methods in Neural Computing*, Van Nostrand Reinhold, New York, 1993.

Chapter 2

Related Work

The man who does not read good books has no advantage over the man who cannot read them.

-- Mark Twain

The work presented here builds, either directly or indirectly on research done by others. This chapter briefly discusses the most important of these foundational ideas. Most of the work discussed here is also mentioned in the introductory section of at least one of the chapters; however, it will be beneficial to collect all such related work here in a somewhat more cohesive fashion. Therefore, having read this section, the reader may give less attention to the related works discussed in the individual papers (most of the chapters do not have a specific entire section dedicated to related works -- this is usually just done in the introduction). However, there are references in the various chapters that are very specific and thus are not appropriate for mention here. Of course, justification cannot be done to any of these works in this brief account, and therefore numerous references are provided for the interested reader.

1. Neural Networks

The field of artificial neural networks if usually traced back to the seminal work by McCulloch and Pitts on modeling the dynamics of the biological neuron [McC43]. The result was the first example of an artificial neuron that to some extent models the dynamics of brain function. The basic features of the biological neuron that the McCulloch-Pitts neuron attempted to capture include multiple synaptic inputs (dendrites) into a cell body that computes an action potential and sends it out a single synaptic output (axon). Therefore, the architecture of the simple artificial model consists of a set of inputs, with an associated set of weights, a summation and thresholding unit and a single output. See Figure 1 for a 3-input example. The inputs x_i are multiplied by their corresponding weights w_i. The results are summed and if the total is greater than some threshold θ, then the output z is set to 1; otherwise it is set to 0. These simple artificial neurons can be combined into networks reminiscent of biological neural systems.

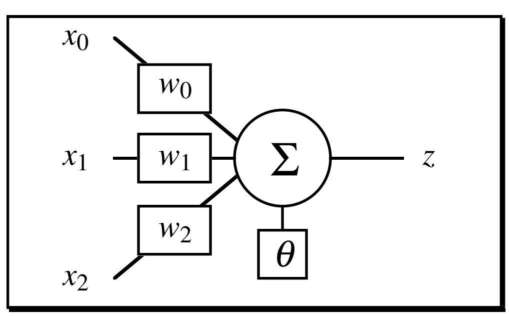

Figure 1. Architecture of the perceptron

The goal for an artificial neuron or network of neurons is, given an input to respond with the appropriate output, and the setting of the weights is the critical aspect of designing an artificial neural network to meet this goal. Since setting the weights by hand will be extremely difficult for any but the simplest problems, an important next advance was the development of learning algorithms for the perceptron. A learning algorithm is an automated mechanism for changing the weights to reflect (and store) the information presented to the perceptron from its environment. Perhaps the most intuitive of these first efforts is attributed to Hebb [Heb49]. The basic idea behind Hebbian learning is that if two neurons are active (outputting a 1) simultaneously, then the weight associated with their connection should be increased in order to reinforce this correlation.

In the 1960s, Rosenblatt developed an improved model, the perceptron, which was proven to converge on a set of desirable actions. In other words, using Rosenblatt's algorithm, a perceptron will eventually memorize the appropriate responses to a set of input examples. The basic idea for perceptron learning is that if a perceptron responds appropriately to a particular input, then nothing is done. In other words, if the system is already working, why change it? However, if it does not respond appropriately, then the weights are changed "a little bit" towards fixing the problem. The reason for making only small, incremental changes to the weights is to avoid unlearning those correct actions that are already encoded in the network. A quantum version of a perceptron is presented in Chapter 8 of this dissertation.

In 1969 a book by Minsky and Papert effectively killed artificial neural network research [Min69]. What they showed in their book was that the artificial neural networks developed to that point could only be effective on a very small set of simple problems. However, the field was rejuvenated by work done by Rumelhart, *et al.* which resulted in the well-known backpropagation learning algorithm [Rum88]. This algorithm is much more complicated than earlier efforts and thus overcame the objections of Minsky and Papert. However, the price for this is that the algorithm can not be guaranteed to converge, and it is also quite slow. The algorithm also requires the manual setting of several sensitive parameters. Since that time, much research has been done to overcome these difficulties (see for example [Mar90] [Fah90] [Red93] [Møl93]).

A variation on the artificial neural networks discussed above is the associative memory. The goal for this type of neural network is associative pattern completion. In other words, given a noisy or incomplete pattern, the network is expected to fill in the missing information. The best known classical associative memories are the Hopfield network and the bidirectional associative memory and background on these networks may be found in [Hop82] and [Kos88] respectively. The biggest drawback to traditional associative neural networks is their limited storage capacity, which is usually quite poor -- usually some fraction of the number of inputs. Since the number of possible patterns is exponential in the number of inputs, improving this would be greatly desirable and some work has been done in this area [Hai88] [Bak88]. The goal of improving the capacity of an associative memory is the motivation for Chapters 9 and 11 of this dissertation in which the idea of the associative memory is extended into the quantum domain.

2. Evolutionary Computation

The idea of using a model of natural evolutionary processes for computational purposes is usually attributed to John Holland [Hol75], although Fogel *et al*. also did pioneering work in the field [Fog66]. Holland attempted both to rigorously explain the mathematical processes behind natural evolutionary systems and to develop artificial evolutionary systems that work on the same principles and could be useful for computation.

Since evolution is a mechanism for selecting traits most useful for a species survival, it can be thought of as performing an optimization, and this is the principle computational problem to which it is applied. Given a model representation, the idea is to find the best model possible, under some measure of model fitness. The model representation is considered as a genome, and evolutionary operators modeled after natural evolutionary processes are applied to a population of different genomes. Many different operators have been proposed, but by far the most popular are crossover and mutation, both of which are based on fairly well-understood natural processes. Crossover involves two different genomes each splitting in to two pieces and exchanging one of the pieces, thus creating two new genomes. Mutation involves a single genome experiencing a small random change. Figure 2 represents these two basic functions for the simple case of binary strings being the genomes.

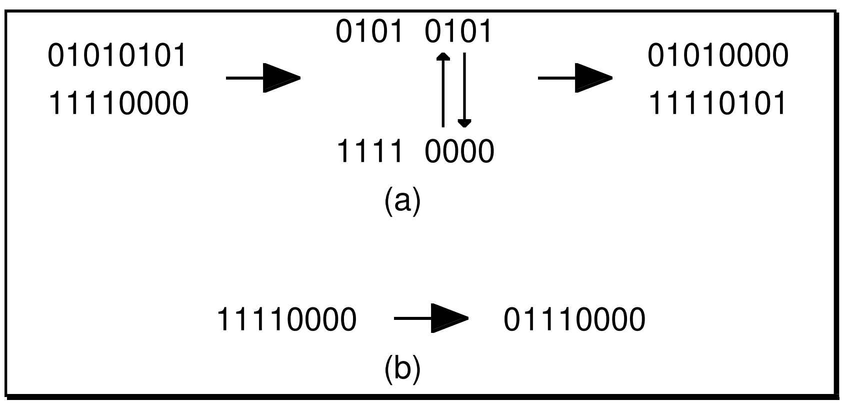

Figure 2. The genetic operators (a) crossover and (b) mutation

The fitness of each genome is measured and those genomes that have good fitness are selected for further evolution while those with poor fitness are allowed to "die". If the initial population in sufficiently large and diverse, then after some number of generations, a genome that is relatively close to optimum is usually found. Why this process works as well as it does is still not well understood, but a good treatment of the main issues can be found in [Gol89].

Evolutionary computation covers a broad range of algorithms all based on the ideas of natural selection. These include genetic algorithms, evolutionary strategies, evolutionary algorithms, and genetic programming. The discrimination between methods depends upon representation issues and choice of operators among other things, and there are no clear cut rules for classifying the different approaches. For a good overview of the differences between these variations see [Spe93]. The work in Part I of this dissertation uses an evolutionary technique that would probably be classified as an evolutionary strategy.

3. Neural/Evolutionary Hybrids

It has become popular to pursue hybrid approaches to computational learning, the idea being to combine the strengths of several models in an advantageous way. In fact, many conferences and volumes are now dedicated entirely to hybrid approaches. See for example [Goo95], [Hon94], [Sun95] and [Hin91]. One particular hybrid approach that receives quite a bit of attention is that of combining some form of evolutionary computation with some form of neural network. The typical approach here is to use the evolutionary computation to perform a search for a good architecture and/or set of weights for the neural network rather than training the network with a gradient descent or other standard learning algorithm. Because neural networks have a tendency to become trapped in local minima or are often very slow in converging to a solution, this hybrid approach sometimes succeeds in improving the chances of discovering a network suitable for the problem at hand. Examples of this kind of evolutionary/neural combination include Caudill's evolutionary neural networks [Cau91], Harp's work on designing application specific neural networks using genetic algorithms [Har90], and Saunders' work on evolving recurrent neural networks [Sau94].

In contrast, the work presented in Chapters 4-7 of this dissertation presupposes some neural architecture and focuses on using evolutionary computation to develop a training set suitable for training the neural network. Though much less common, some work similar in flavor to this approach does exist. [Mia96] presents a training algorithm for a neural network that is a combination of standard gradient descent and evolutionary techniques, and in [Mon89] a similar approach to training neural networks using genetic algorithms is presented. Romaniuk's evolutionary growth perceptrons [Rom93] are perhaps the closest to the work in this dissertation. However, he still assumes a training set *a priori*, and uses a genetic algorithm to select different subsets of the training set to train different nodes in a neural network.

Finally, [Wie90] should be mentioned for its use of evolutionary computation for discovering neural networks capable of acting as neurocontrollers. As mentioned above, however, the evolutionary computation is used to discover the architecture and weight settings for the neural network. In contrast, the work in Chapter 7 uses evolutionary computation to discover a training set that represents the control problem, and then standard neural network training is used to develop the neurocontroller. [Bar83] should also be mentioned for providing the differential equations that describe the control problem used in Chapter 7 (as well as in the Wieland paper).

4. Quantum Computation

The field of quantum computation, which applies ideas from quantum mechanics to the study of computation, was introduced in the mid-1980s [Ben82] [Deu85] [Fey86]. (For a review of the basic ideas necessary for quantum computation see Chapter 11 Section 2 of this dissertation.) The field is still in its infancy and very theoretical but offers exciting possibilities for the field of computer science -- the most important quantum algorithms discovered to date all perform tasks for which there are no classical equivalents. For example, Deutsch's algorithm [Deu92] is designed to solve the problem of identifying whether a binary function is constant (function values are either all 1 or all 0) or balanced (the function takes an equal number of 0 and 1 values). Deutsch's algorithm accomplishes the task in order $O(n)$ time, while classical methods require $O(2^n)$ time. Simon's algorithm [Sim97] is constructed for finding the periodicity in a 2-1 binary function that is guaranteed to possess a periodic element. Here again an exponential speedup is achieved. Admittedly, both these algorithms have been designed for artificial, somewhat contrived problems. Grover's algorithm [Gro96], on the other hand, provides a method for searching an unordered quantum database in time $O(\sqrt{n})$, compared to the classical bound of $O(n)$. This is a real-world problem for which quantum computation provides performance that is classically impossible (though the speedup is less dramatic than exponential). Grover's algorithm is used in the quantum associative memory developed in Chapter 11. The most well-known and perhaps the most important quantum algorithm discovered so far is Shor's algorithm for prime factorization [Sho97]. This algorithm finds the factors of very large prime numbers in polynomial time, whereas the best known classical algorithms require exponential time. Other recently developed quantum algorithms include two other variations on quantum search [Hog96] [Ter98].

One technology for physical realization of a quantum computer that has been recently suggested is nuclear magnetic resonance, a technology that has been around since the 1950s [Cor97]. In fact, very recently several groups have produced exciting experimental results by successfully implementing quantum algorithms on small-scale nuclear magnetic resonance quantum computers (see for example [Jon98] and [Chu98]). These advances in quantum technology indicate that it is not unreasonable to assume that larger quantum computers which can perform computation that is classically impossible will be possible in the near future.

4.1 Quantum computational learning

To date, very little has been done in combining the fields of quantum mechanics and neural networks. However, a few notable exceptions do exist. Perus has published an interesting set of mathematical analogies between the quantum formalism and neural network theory [Per96]. Menneer and Narayanan have proposed a model weakly inspired (their term) by the "many worlds" interpretation of quantum mechanics in which a network is trained for each instance in the training set and the final network is a superposition of these [Men95]. Finally, Behrman *et al.* have developed a novel approach to implementing a quantum neural network using quantum dots [Beh96]. It uses a quantum dot for each input and the system is allowed to evolve quantum mechanically through time while being observed at fixed time intervals. Interestingly, the different time slices act as the neurons in a hidden layer of a neural network -- the more time slices, the more hidden layer neurons. Some other quantum computational algorithms that should be mentioned here include work on using quantum systems to evaluate decision trees [Far98] and quantum Bayesian nets [Tuc95].

Some authors have put forth theories suggesting that quantum effects may actually play a crucial role in human cognition [Pen94] [Ham94]. While this has little bearing on the research presented here, it nevertheless can be considered as being in the same spirit, just as research in biological neural systems helps inspire work in artificial neural networks.

Chapter 12 is inspired by a Fourier-based machine learning algorithm due to Kushilevitz and Mansour [Kus93] and Jackson [Jac97]. Jackson's work presents a very nice theoretical result in computational learning, showing how to learn the function class DNF (disjunctive normal form) if blackbox access (called a membership oracle) to the function is available. In practice, however, this membership oracle will probably never be available. In fact, Bshouty and Jackson investigated a quantum computational approach to learning, which they hoped might avoid this reliance on membership oracles [Bsh95]. Again, however, their approach turned out to be interesting theoretically but not useful in practice. While [Ber97] is actually dedicated to quantum computational theory in general rather that quantum computational learning, it should be mentioned for its quantum Fourier sampling algorithm, which contains elements similar to the quantum encoding of examples of Chapter 11 and the Fourier method that is the key to Chapter 12. The major difference between their work and that presented here, and this is very significant, is the fact that they assume *a priori* knowledge of a function, whereas the work here assumes that the function in question is completely unknown except for a relatively few sample points (which is the typical case in computational learning).

References

[Bak88] Bak, Chan S. and Michael J. Little, "Memory Capacity of Artificial Neural Networks with High Order Node Connections", *IEEE International Conference on Neural Networks*, pp. 207-16, 1988.

[Bar83] Barto, Andrew G., Richard S. Sutton and Charles W. Anderson, "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. smc-13, no. 5, September/October, 1983.

[Beh96] Behrman, E.C., J. Niemel, J. E. Steck and S. R. Skinner, "A Quantum Dot Neural Network", *Proceedings of the Workshop on Physics of Computation*, Boston, pp. 22-4, November 1996.

[Ben82] Benioff, Paul, "Quantum Mechanical Hamiltonian Models of Turing Machines", *Journal of Statistical Physics*, vol. 29 no. 3, pp. 515-546, 1982.

[Ber97] Bernstein, Ethan and Umesh Vazirani, "Quantum Complexity Theory", *SIAM Journal of Computing*, vol. 26 no. 5, pp. 1411-1473, 1997.

[Bsh95] N. H. Bshouty and J. Jackson, in *Proceedings of the 8th Annual Conference on Computational Learning Theory*, *Santa Cruz, 1995*, edited by Wolfgang Maass (ACM Press, 1995).

[Cau89] Caudell, T. P. and C. P. Dolan, "Parametric Connectivity: Training of Constrained Networks using Genetic Algorithms", *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[Cau91] Caudill, Maureen, "Evolutionary Neural Networks", *AI Expert*, vol. 6, no. 3, pp. 28-33, March 1991.

[Cor97] Cory, David G., Mark D. Price, Amr F. Fahmy and Timothy F. Havel, “Nuclear Magnetic Resonance Spectroscopy: An Experimentally Accessible Paradigm for Quantum Computing”, *Physica D*, to appear, (accepted Aug. 27, 1997).

[Chu98] Chuang, Isaac, Neil Gershenfeld and Mark Kubinec, “Experimental Implementation of Fast Quantum Searching”, *Physical Review Letters*, vol. 80 no. 15, pp. 3408-11, April 13, 1998.

[Deu92] Deutsch, David and Richard Jozsa, “Rapid Solution of Problems by Quantum Computation”, *Proceedings of the Royal Society, London A*, vol. 439, pp. 553-8, 1992.

[Deu85] Deutsch, David, “Quantum Theory, The Church-Turing Principle and the Universal Quantum Computer”, *Proceedings of the Royal Society*, London A, vol. 400, pp. 97-117, 1985.

[Fah90] Fahlman, Scott E. and Christian Lebiere, “The Cascade Correlation Learning Architecture”, *Advances in Neural Information Processing Systems 2*, Morgan Kaufmannn, 1990.

[Far98] Farhi, Edward and Sam Gutmann, “Quantum Computation and Decision Trees“, *Physical Review A*, to appear, August 1998.

[Fey86] Feynman, Richard P., “Quantum Mechanical Computers”, *Foundations of Physics*, vol. 16 no. 6, pp. 507-531, 1986.

[Fog66] Fogel, L. J., A. J. Owens and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*, John Wiley, 1966.

[Goo95] Goonatilake, Suran and Sukhdev Khebbal (eds.), *Intelligent Hybrid Systems*, John Wiley & Sons, 1995.

[Gro96] Grover, L., “A Fast Quantum Mechanical Algorithm for Database Search”, *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, ACM, New York, pp. 212-19, 1996.

[Hai88] Haines, K. and R. Hecht-Nielson, “A BAM with Increased Information Storage Capacity”, *IEEE International Conference on Neural Networks*, pp. 181-90, 1988.

[Ham94] Hameroff, Stuart R., “Quantum Coherence in Microtubules: A Neural Basis for Emergent Consciousness?”, *Journal of Consciousness Studies*, vol. 1 no. 1, pp. 91-118, 1994.

[Har90] Harp, S. A., T. Samad and A. Guha, “Designing Application-Specific Neural Networks Using the Genetic Algorithm”, *NIPS-89 Proceedings*, 1990.

[Heb49] Hebb, D. O., *The Organization of Behavior*, John Wiley & Sons, 1949.

[Hin91] Hinton, G. E. (ed.), *Connectionist Symbol Processing*, MIT Press, 1991.

[Hog96] Hogg, Tad, “Quantum Computing and Phase Transitions in Combinatorial Search“ *Journal of Artificial Intelligence Research*, vol. 4, pp. 91-128, 1996.

[Hol75] Holland, John, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[Hon94] Honavar, Vasant and Leonard Uhr (eds.), *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration*, Academic Press, Inc., San Diego, California, 1994.

[Hop82] Hopfield, J. J., “Neural Networks and Physical Systems with Emergent Collective Computational Abilities”, *Proceedings of the National Academy of Scientists*, vol. 79, pp. 2554-2558, 1982.

[Jac97] Jackson, Jeffrey C., “An Efficient Membership-query Algorithm for Learning DNF with Respect to the Uniform Distribution”, *Journal of Computer and System Sciences*, vol. 55 no. 3, pp. 414-40, 1997.

[Jon98] Jones, J. A. and M. Mosca, “Implementation of a Quantum Algorithm to Solve Deutsch’s Problem on a Nuclear Magnetic Resonance Quantum Computer”, *Journal of Chemical Physics*, to appear, 1998.

[Kus93] Kushilevitz, E. and Y. Mansour, *SIAM Journal of Computation*, vol. 22 no. 6, pp. 1331-48, 1993.

[Kos88] Kosko, Bart, “Bidirectional Associative Memories”, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, pp. 49-60, 1988.

[Mar90] Martinez, Tony R., “Adaptive Self-Organizing Concurrent Systems”, *Progress in Neural Networks*, vol. 1, pp. 105-26, 1990.

[McC43] McCulloch, W. S. and W. Pitts, “A Logical Calculus of the Ideas Immanent in Nervous Activity”, *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.

[Men95] Menneer, Tammy and Ajit Narayanan, “Quantum-inspired Neural Networks”, technical report R329, Department of Computer Science, University of Exeter, Exeter, United Kingdom, 1995.

[Mia96] Miagkikh, V. V., A. P. Topchy and O. A. Lebedko, “Fast Learning in Multilayered Networks by means of Hybrid Evolutionary and Gradient Algorithms”, *Proceedings of the International Conference on Evolutionary Computation and it Applications*, pp. 390-8, June 1996.

[Min69] Minsky, M. L. and S. L. Papert, *Perceptrons*, MIT Press, 1969.

[Møl93] Møller, Martin F., “A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning”, *Neural Networks*, vol. 6, pp. 525-33, 1993.

[Mon89] Montana, D. J. and L. Davis, “Training Feedforward Neural Networks Using Genetic Algorithms”, *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[Pen94] Penrose, Roger, *Shadows of the Mind*, Oxford University Press, 1994.

[Per96] Perus, Mitja, “Neuro-Quantum Parallelism in Brain-Mind and Computers”, *Informatica*, vol. 20, pp. 173-183, 1996.

[Red93] Redding, Nicholas J., Adam Kowalczyk and Tom Downs, “Constructive Higher-Order Network Algorithm that is Polynomial Time”, *Neural Networks*, vol. 6, pp. 997-1010, 1993.

[Rom93] Romaniuk, Steve G., “Evolutionary Growth Perceptrons”, *Genetic Algorithms: 5th International Conference (ICGA-93)*, S. Forrest (ed.), Morgan Kaufmannn, 1993.

[Rum88] Rumelhart, David E., James L. McClelland and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, 1986.

[Sau94] Saunders, Gregory M., Peter J. Angeline and Jordan B. Pollack “Structural and Behavioral Evolution of Recurrent Neural Networks”, *Advances in Neural Information Processing Systems*, vol. 6, pp. 88-95, Morgan Kaufmannn Publishers Inc., 1994.

[Sim97] Simon, D., “On the Power of Quantum Computation”, *SIAM Journal of Computation*, vol. 26 no. 5, pp. 1474-83, 1997.

[Sho97] Shor, P. W. “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer”, *SIAM Journal of Computing*, vol. 26 no. 5, pp. 1484-1509, 1997.

[Spe93] Spears, W. M., K. A. Dejong, T. Baeck, D. Fogel and H. de Garis, “An Overview of Evolutionary Computation”, *European Conference on Machine Learning (ECML-93)*, 1993.

[Sun95] Sun, Ron and Lawrence Bookman (eds.), *Computational Architectures Integrating Neural and Symbolic Processes: A Perspective on the State of the Art*, Kluwer Academic, 1995.

[Ter98] Terhal, Barbara. M. and John. A. Smolin, “Single Quantum Querying of a Database”, *Physical Review A* , to appear, September 1998.

[Tuc95] Tucci, Robert R., “Quantum Bayesian Nets”, *International Journal of Modern Physics* vol. 9, pp. 295-337, 1995.

[Wie90] Wieland, Alexis P., “Evolving Controls for Unstable Systems”, *Proceedings of the 1990 Connectionist Models Summer School*, pp. 91-102, 1990.

Chapter 3

Concerning a General Framework for the Development of Intelligent Systems

When someone tells you something defies description, you can be pretty sure he's going to have a go at it anyway.
-- Clyde B. Aster

Proceedings of the International Conference on Artificial Intelligence, Expert Systems and Neural Networks, pp. 44-47, 1996.

Abstract

There exists on-going debate between Connectionism and Symbolism as to the nature of and approaches to cognition. Many viewpoints exist and various issues seen as important have been raised. This paper suggests that a combination of these methodologies will lead to a better overall model. The paper reviews and assimilates the opinions and viewpoints of these diverse fields and provides a cohesive list of issues thought to be critical to the modeling of intelligence. Further, this list results in a framework for the development of a general, unified theory of cognition.

1. Introduction

Developing artificial models of cognition is a challenging field that has roots in such diverse fields as cognitive psychology, computer science, linguistics, neurophysiology, and mathematics. Due at least in part to this fact, vastly differing theories for explaining and/or imitating cognitive capabilities have been developed including Connectionism [1] [2], Symbolic Artificial Intelligence [3], Fuzzy Logic [4], Machine Learning [5] and Genetic Algorithms [6], to name a few. Each of these approaches to modeling cognition (or, more accurately, some aspect thereof) has various strengths and weaknesses that are becoming better understood, and each claims some fairly impressive successes as well as some disheartening failures.

For the most part, relations among the various paradigms have been indifferent at best and often have been almost hostile. Though this is true of all the approaches mentioned above to some extent, the majority of the contrasts, comparisons, and conflicts are embodied in the Symbolism vs. Connectionism debate that has raged continuously over the last ten years.

However, some researchers have begun to recognize the apparent complimentary nature of these two paradigms. They believe that neither approach alone can produce adequate models of cognition but that in combination richer models with greater functionality can be realized. Efforts in this direction have variously been referred to as Hybrid Systems, High-Level Connectionism, Symbolic Connectionism, Symbol Processing Connectionist Systems and the like. Though work in this area has been on-going since the resurgence of

Connectionism in the late eighties [7] [8], a proliferation of new books on the subject ([9] [10] [11] [12], for example) indicates its increasing importance.

To date, the field of Hybrid Systems has produced some interesting applications [9] [10] [11] [12] [13], and some ground work has been laid concerning the proper integration of various approaches to modeling cognition; however, the critical next step must be the development of a formal theory for the integration of Connectionism with Symbolism and thus for a general theory of (artificial) intelligence. A promising approach, in our opinion, is to view Connectionism and Symbolism as complimentary approaches and to attempt to extract from each the primitives that embody them. Another way of looking at this is to realize that both Connectionism and Symbolism can be modeled by a Turing machine, and therefore our job is to identify a simple subset of Turing-computable functions that embody the primitives necessary for a model of intelligence.

This paper is an attempt to outline a framework for the development of such a general theory. The approach we take is to first review the differences and similarities, and the strengths and weaknesses of Connectionism and Symbolism. Second, a list of critical issues pertaining to intelligence is discussed. Since everyone has a differing opinion of what the critical issues are, we are attempting to distill the many ideas that *could* be considered into the essence of what *must* be considered. Third, a brief discussion of how these issues can guide in the development of a general theory of intelligence is presented.

2. Connectionism and Symbolism

Whatever their differences may be, Connectionism and Symbolism both share the assumption that at some level cognition can be functionally modeled as a computational process. Other than this extremely basic commonality, there is little else involving the issue of Connectionism and Symbolism that enjoys any kind of majority consent. Some researchers argue that Connectionism is fatally flawed as a model of cognition [14] or that while Connectionism may play a minor though important role, Symbolism represents the heart of any realistic model [15]. On the other hand, cases are presented for the desirability of Connectionism over Symbolism [16] and that the eventual dominance of Connectionism is only a matter of time [17]. Some see the two as complimentary or dualistic in nature and are optimistic that their combination will bear productive fruit [18] [19] [20] [21] [22], while others argue for their equivalence and cite performance issues as the main concern [23] [10]. Still others are more pessimistic as how intelligent a Connectionist-Symbolic hybrid will ever be [24], and some claim that the entire idea of modeling cognition as computation is hopeless [25].

Many people agree that cognition can and should be described at different levels and that care should be taken to compare and contrast only those explanations/models/theories that exist on the same level [23] [14] [16]. However, others feel that an explanation of cognition should not or can not be completely differentiated from the biological implementation of cognition that we know as the brain [22][24] [17].

Tables 1 and 2 attempt to assimilate these views from two different angles. One common thread that runs through any discussion of Connectionism vs. Symbolism, either implicitly or explicitly is the dual nature of the two. People refer to this duality in different ways, and Table 1 presents the computational dichotomy that these two approaches represent. For example, Connectionist networks are characterized by being spatially distributed while Symbolic approaches can be thought of as being temporally distributed. In related fashion, Connectionist networks perform processing in parallel over distributed representations but Symbolic models process serially over local ones. Connectionist models perform statistical approximations over continuous numerical data, and Symbolic models apply concrete logical operations to discrete symbolic data. Finally, on one hand, Connectionism is associated with cognitive tasks that are referred to as low-level, subconceptual, unconscious, or subdeliberative; on the other, Symbolism is associated with tasks that may be described as high-level, conceptual, conscious, or deliberative.

These are all prototypical generalizations, of course, and as is pointed out in, for example [21], the boundary dividing Connectionist models and Symbolic models is ill-defined at best. In fact Oden suggests that models that should be taken seriously will almost always exist in this fuzzy boundary region.

Table 1. The Connectionist-Symbolic Dichotomy

Connectionism vs.	Symbolism
Space	Time
Parallel	Serial
Distributed	Localist
Statistical	Logical
Continuous	Discrete
Numerical	Symbolic
Low-level	High-level
Subconceptual	Conceptual
Unconscious	Conscious
Subdeliberative	Deliberative

The second approach to looking at this is to examine the relative strengths and weaknesses of the two approaches. People see this particular debate in many different lights. There is somewhat of a consensus as to what the respective strengths and weaknesses are. However much disagreement exists over their relative importance. Table 2 is a compilation of general strengths and weaknesses for Connectionism and Symbolism. Again, this is an attempt to assimilate as many viewpoints as possible, though doubtless some will be excluded or treated unsatisfactorily.

Attributes of Connectionist networks usually cited as strong points include *robustness*, the ability to *fine tune* knowledge through experience, automatic *acquisition of knowledge*, *fault tolerance*, and *adaptivity*. Robustness is the ability to respond appropriately to inputs that are noisy, novel, or unanticipated. Fine tuning of knowledge expresses the idea that the system can slightly alter its representations to account for new inputs without drastically affecting the representations gained from past experiences. Automatic knowledge acquisition is the ability to extract knowledge from the environment. Fault tolerance allows for graceful degradation of performance as the system fails. Finally, adaptivity deals both with a wide range of applicability of the model as well as with the ability of a given instantiation of the model to change with its environment.

Table 2. Strengths and Weaknesses of Connectionism and Symbolism

Connectionism	Symbolism
Strengths	
robustness	handle complex structures
fine tuning	domain
knowledge acquisition	explanation
fault tolerant	serial computation
adaptive	
inherent parallelism	
Weaknesses	
difficult to interpret	brittleness
slow learning	symbol grounding
homogeneous stucture	knowledge acquisition
not naturally serial	narrow domain
scaling	scaling

Symbolism, on the other hand, is usually considered to have as strengths the ability to represent and manipulate *complex structures*, *rapid learning* of concepts, *explanation*, and the ability to naturally perform *serial* computation. Complex structures are defined recursively as a set of atomic symbols together with those molecular symbols that can be created by combining atomic symbols according to a set of syntactic rules. These structures possess a semantic structure that is closely related to their syntactic one. Rapid learning refers to the ability to quickly and drastically change representations. Explanation refers to the ability of a system to explain its decision in understandable terms. Some people hold that the explanation of a system's behavior (not the behavior itself) is the real object of interest. Serial computation is argued to be necessary for some kinds of cognition including ordered problem solving, logical reasoning, generalized thinking involving variable binding, and certain types of structural aspects of language [20].

Not surprisingly, the weaknesses of the respective approaches often seem to be the negation of a strength of the opposing method. For example, the difficulty in interpretation of Connectionist networks is in direct opposition to the strong explanation abilities of Symbolism. Likewise, Connectionism's slow learning rate, homogeneity of structure and lack of natural serial computation ability starkly contrast with Symbolism's strengths of use of domain knowledge, complex structure, and natural serial computation respectively. Similarly, Symbolism has its own weaknesses: brittleness, symbol grounding (or lack thereof), difficulty in acquiring knowledge, and narrow application domain (each system must be tailored to a specific domain). Brittleness results from a lack of robustness and fault tolerance, difficulty in acquiring knowledge contrasts with automatic knowledge acquisition, and a narrowness of domain results from lack of adaptivity. But robustness, fault tolerance, automatic knowledge acquisition, and adaptivity are exactly the strengths of Connectionism. The symbol grounding problem is interesting in that it does not directly oppose a strength of Connectionism, per se, (though Connectionism does not suffer from it and this may be thought of as a strength) and is treated in [15]. There are those, of course, who will point to one or another of one of the methods' weaknesses and argue that it is insurmountable and thus that the method is untenable. A good example of this is Fodor and Pylyshyn's denunciation of Connectionism on the basis that it lacks compositionality and systematicity (basically, the ability to represent complex structures and the existence of structure sensitive operations, respectively) [14].

3. Issues of Intelligence

Accepting the premise of Artificial Intelligence that cognition, at some level, can be modeled as computation and also taking the viewpoint that Connectionism and Symbolism represent a complementary dichotomy of computation, both poles of which explain certain aspects of intelligence, we would like to employ these assumptions to develop a framework for the development of a general theory of (artificial) intelligence. This framework will explicitly describe what such a theory must do in order to be a candidate for the explanation of cognition.

3.1. Formal primitives

A minimal set of formal primitives (a subset of the Turing computable functions) that are necessary and sufficient for modeling the building blocks of intelligence must be developed. Minimality is important for reasons of parsimony and elegance. Formality is a necessity for clarity, for testing hypotheses, and for implementation issues, among other things. Further, these primitives must adequately address the following issues: knowledge, complex symbol structure, symbol grounding, learning, robustness, fault tolerance, adaptivity, duality of intelligence, multiple levels of explanation, and scaling.

3.2. Knowledge

Possessing knowledge and rationally making use of it is a major component of intelligence and there are at least three facets to the knowledge problem: acquisition, representation, and explanation. Any cognitive system must be able to obtain information from its environment, internalize and manipulate that information, and respond appropriately to its environment.

3.2.1. Acquisition

A model of intelligence must provide a method for actively extracting pertinent information from the environment as well as allow for passive assimilation of information provided explicitly by an authority.

3.2.2. Representation

The model must internally represent knowledge both in a way that is useful to it and in a way that exhibits some kind of relationship with the external environment from which it was derived. This obviously relates to the issue of compositionality and to the issue of grounding.

3.2.3. Explanation

If for no other reason than for the practical one of trust, it is necessary that any intelligent system be capable of providing an explanation of its decisions. This is especially true of an artificial intelligence acting in any type of critical system (such as air traffic control or medicine). Some would claim that the explanation of a system's behavior is more important and more valuable (especially in the context of understanding intelligent behavior) than the behavior itself.

3.3. Complex symbols

For the staunch Classicists, the ability (or lack thereof) to represent and manipulate complex symbols is the crux of the matter. They believe that humans use symbol representations and manipulations in so many high-level cognitive tasks that this an absolute necessity. Connectionists are not convinced of this. It may certainly be argued that language syntax and semantics are based on this ability, and in fact, this ability is derived, for the most part, from a linguistic theory of mind. This concept of a complex symbolic representational system is so pervasive that any general model must either include provision for it (or a functional equivalent) or convincingly demonstrate it to be unnecessary.

3.3.1. Compositionality

This principle demands the ability to represent a theoretically infinite number of symbols (this is known as productivity) using a finite set of atomic symbols and a set of syntactic rules for their combination. This theoretical infinitude is limited in practice by finite resource constraints such as memory, time, etc. The important thing about these symbols is that their semantic interpretation is a semantic interpretation of their constituent parts along with the relationship that exists among these parts.

3.3.2. Systematicity

Systematicity complements compositionality in that it requires the existence of operations that are sensitive to syntactic structure. For example, in deductive logic a rule exists for transforming any structure of the form $A \wedge B$ to the form A. Because semantic structure is closely related to syntactic structure (by compositionality), this reflects the truism that if A and B exist (are true), then certainly A exists (is true).

3.4. Grounding

This consideration is due to Harnad [15] and addresses the problem in pure symbol systems of the lack of *intrinsic* meaning in the system. Though such a system possesses a rich symbol structure, that structure is purely internal and completely arbitrary. It is not grounded to anything in reality. Harnad likens this problem to that of trying to learn Chinese as a first language with only Chinese literature as an instruction source. All the Chinese symbols are related to one another in complex ways, but without so much as a notion of language (remember we are learning this as a *first* language), there is no way to connect any of those symbols to anything real. Symbolic representation is practically worthless to a system trying to function in an environment in which none of its symbols are grounded. Harnad suggests that the solution to this quandary is to begin with an elementary set of nonsymbolic representations such as various input patterns that can be both discriminated and identified (classified) correctly. The classifications of these patterns will then be abstract symbols, but they will be grounded to the input patterns from the environment that produced them. These can then be further combined into more complex symbols as discussed above, and the grounded meanings of the atomic symbols will be inherited by any higher level symbols of which they are constituents (by the principle of compositionality).

3.5. Learning

Knowledge acquisition is one method for providing a system with its initial knowledge; other approaches such as inductive methods also may be employed. The ability to alter that knowledge and accordingly to alter behavior due to changes in the environment is what allows a system to intelligently function in a dynamic, realistic environment. This is the ability to learn. Any intelligent system should, over time, become more adept at performing tasks (not just in the sense of speed-up learning) and interacting with its environment through this process. Both the rapid assimilation of concepts necessary for immediately critical behavioral changes and the gradual refining of representations involved in capturing exceptional behaviors and concept drift must be supported.

3.6. Robustness, fault tolerance, and adaptivity

The arguments for these capabilities are almost platitudes. Independent of method it is obvious that the abilities to perform in noisy, novel or adverse conditions, to maintain reasonable levels of performance despite failures in the system (graceful degradation), and to adapt to a changing environment and/or unique types of problems are clearly desirable properties that any intelligence or model of intelligence will possess.

3.7. Duality

As discussed in Section 2, there appears to be a dual nature to intelligence. It is multi-faceted and each of these has diametrically opposed poles that are both present (or at least seem to be) in intelligence. The most important of these dualities is briefly discussed below.

3.7.1. Continuous and discrete

Most of the real world is a continuum. Therefore it is critical that any model of intelligent behavior account for handling continuous data. On the other hand, we often process in discrete quantities as well, content to know that it is a cool day rather than that it is 63.7° F. Discretization, though far from satisfactory, is one approach to reconciling the two [26].

3.7.2. Numerical and symbolic

The argument here is related to the previous one. Clearly, we explicitly and consciously perform symbolic computation at least in language. Perhaps we perform some numerical computation in mathematical endeavors, though mathematics is by its very nature symbolic. More probably any numerical processing we perform occurs at an unconscious level, and there are numerous unconscious tasks that we perform (particularly of the pattern recognition sort) that can, so far, only be explained numerically.

3.7.3. Distributed and localist

A distributed system represents concepts as patterns over its units. A localist system represents a single concept with a single unit. Therefore a distributed system will be able to represent many more patterns than a localist one with similar resources. However, a fully distributed system typically can represent only a single concept at a time because all the units of the system are involved in the representation. On the other hand, a localist system can simultaneously represent as many concepts as it has units. The requirement here is to achieve the simultaneity of a localist system and the capacity of a distributed system. Other issues that could be addressed include those involving fault tolerance, performance issues, etc.

3.7.4. Parallel and serial

Many processing tasks such as pattern matching, image recognition, memory recall, and natural language understanding require huge amounts of processing and yet they are performed by biological systems in a few hundred milliseconds. This suggests massive amounts of parallel computation. However, other tasks such as logic inference, problem solving, and the like have a very serial nature to them in which one computation cannot be accomplished until after another is complete. How do we reconcile two such computational approaches in the same model?

3.7.5. Statistics and logic

In a related vein, it is often necessary to perform a nearest match or a closest approximation or a degree of membership type operation. Certainly any completely novel experience will require such improvisation for the production of behavior. At other times, strict all or nothing matching or yes/no reasoning is required. Both the “soft” constraints of statistics derived from the environment (as per Connectionism) and the “hard” constraints of logic (as per Symbolism) are required.

3.8. Multiple levels

The duality issues of Section 3.7 all suggest that an explanation of multiple levels of cognition is required. In every case one half of the dual was associated with high-level or conscious tasks such as reasoning while the other half was associated with low-level or unconscious tasks such as perception. It is well accepted that there exist multiple levels of *explanation* for cognitive phenomena and that there are at least the two levels (high-level and low-level processes) of *extant* cognitive phenomena. Therefore, any model of intelligence may exist at or take into account multiple levels of description; however, a model of intelligence *must* account for at least the high and low levels implicated in the computational dichotomy of intelligence.

3.9. Scaling

Neither Connectionism nor Symbolism have really succeeded in scaling their proposed explanations of cognition to real-world levels. Not only is a theory that only works on toy problems of no practical use, it is soon of very little use intellectually as well, for no theory that cannot account for the handling of real-world problems can come close to modeling intelligence. We must eventually succeed in scaling solutions up to realistic problems or finally abandon our efforts to explain cognition.

3.10. What and how

Are the brain and the mind inextricably associated one with another or is the functionality of mind completely separable from the implementation of the brain? There are those that will argue both ways. Among those that hold to the separability view, some will argue for a top-down approach in which we must understand *what* is happening before we can understand *how* it is happening, while others will argue the opposite way. We are of the opinion that mind and brain cannot be wholly separated and thus some attempt must be made to understand both at once. However, whatever the viewpoint, both the *how* and the *what* must eventually be addressed. The difference comes in the order they are tackled.

4. Toward a General Theory

As the issues above are confronted and resolved, a general, unified theory of (artificial) intelligence will emerge, and some good preliminary efforts have been made. For example, both Arbib's Schema Theory [27] [28] and Michalski's Multistrategy Task-adaptive Learning [29] give good accounts of many of the aspects of knowledge, complex symbols and learning. Though it is possible that the emergence of a theory will come as a revolutionary breakthrough, it is more likely that it will evolve much more slowly as an eclectic result of such research performed in many different disciplines. For this to have a chance to succeed, not only must argumentation and hostility between various camps be avoided, but also interdisciplinary cooperation should be fostered to a greater extent. For this to succeed, as McKenna points out, "we must resist the temptation to critique one discipline by the specialized criteria of another" [17]. It should be reiterated that although the theory may evolve in an eclectic manner, the final product cannot be a hodge-podge of techniques, partial models and explanations. It must be cohesive, formalized, and elegant and it must address the issues discussed here whether by subsumption or refutation. Though the issues presented may not be sufficient for a general theory of cognition they are surely necessary.

References

[1] Rumelhart, David E., James L. McClelland and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, 1986.

[2] Wasserman, Philip D., *Advanced Methods in Neural Computing*, Van Nostrand Reinhold, 1993.

[3] Russel, Stuart and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, 1995.

[4] Zadeh, L. A., "Fuzzy Logic", *IEEE Computer*, 83-93, 1988.

[5] Shavlik, Jude W. and Thomas G. Diettrich (eds.), *Readings in Machine Learning*, Morgan Kaufmannn, 1990.

[6] Goldberg, David E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[7] Barndan, John A. and J. B. Pollack (eds.), *Advances in Connectionist and Neural Computation Theory 1, High-level Connectionist Models*, Ablex, 1991.

[8] Hinton, G. E. (ed.), *Connectionist Symbol Processing*, MIT Press, 1991.

[9] Goonatilake, Suran and Sukhddev Khebbal (eds.), *Intelligent Hybrid Systems*, John Wiley & Sons, 1995.

[10] Honavar, Vasant, and Leonard Uhr (eds.), *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration*, Academic Press, 1994.

[11] Levine, Daniel S. and Manuel Aparicio (eds.), *Neural Networks for Knowledge Representation and Inference*, Lawrence Erlbaum Associates, 1994.

[12] Sun, Ron and Lawrence Bookman (eds.), *Computational Architectures Integrating Neural and Symbolic Processes: A Perspective on the State of the Art*, Kluwer Academic, 1995.

[13] Shibata, Takanori and Toshio Fukuda, "Hierarchical Intelligent Control for Robotic Motion", *IEEE Transactions on Neural Networks*, vol. 5, no. 5, 823-32, 1994.

[14] Fodor, J. A. and Z. W. Pylyshyn, "Connectionism and Cognitive Architecture: A Critical Analysis", *Cognition*, vol 28, 3-71, 1988.

[15] Harnad, Stevan, "The Symbol Grounding Problem", *Physica D*, vol. 42, 335-46, 1990.

[16] Barnden, John A., "On Using Analogy to Reconcile Connections and Symbols", *Neural Networks for Knowledge Representation and Inference*, eds. Daniel S. Levine and Manuel Aparicio, Lawrence Erlbaum, 27-64, 1994.

[17] McKenna, Thomas M., "The Role of Interdisciplinary Research Involving Neuroscience in the Development of Intelligent Systems", *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration*, eds. Vasant Honavar and Leonard Uhr, Academic Press, 75-92, 1994.

[18] Goonatilake, Suran and Sukhdev Khebbal, "Intelligent Hybrid Systems: Issues, Classifications and Future Directions", *Intelligent Hybrid Systems*, eds. Suran Goontalilake and Sukhdev Khebbal, John Wiley & Sons, 1-20, 1995.

[19] Sun, Ron, "An Introduction: On Symbolic Processing in Neural Networks", *Computational Architectures Integrating Neural and Symbolic Processes: A Perspective on the State of the Art*, eds. Ron Sun and Lawrence A. Bookman, Kluwer Academic, 1-18, 1995.

[20] Boden, Margaret, "Horses of a Different Colour?", *Aritificial Intelligence and Neural Networks: Steps Toward Principled Integration*, eds. Vasant Honavar and Leonard Uhr, Academic Press, 3-19, 1994.

[21] Oden, Gregg C., "Why the Difference between Connectionism and Anything Else Is More Than You Might Think but Less Than You Might Hope", *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration*, eds. Vasant Honavar and Leonard Uhr, Academic Press Inc., 93-103, 1994.

[22] Aparicio, Manuel and Daniel S. Levine, "Why are Neural Networks Relevant to Higher Cognitive Function?", *Neural Networks for Knowledge Representation and Inference*, eds. Daniel S. Levine and Manuel Aparicio, Lawrence Erlbaum, 1-26, 1994.

[23] Honavar, Vasant, "Symbolic Artificial Intelligence and Numeric Artificial Neural Networks: Towards a Resolution of the Dichotomy", *Computational Architectures Integrating Neural and Symbolic Processes: A Perspective on the State of the Art*, eds. Ron Sun and Lawrence A. Bookman, Kluwer Academic, 351-88,1995.

[24] Chandrasekaran, B. and Susan G. Josephson, "Architecture of Intelligence: The Problems and Current Approaches to Solutions", *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration*, eds. Vasant Honavar and Leonard Uhr, Academic Press, 21-50,1994.

[25] Penrose, Roger, *Shadows of the Mind*, Oxford University Press, 1994.

[26] Ventura, Dan and Tony Martinez, Tony, "An Empirical Comparison of Discretization Methods", *Proceedings of the Tenth International Symposium on Computer and Information Sciences*, 443-50, 1995.

[27] Arbib, Michael, "Schema Theory: Cooperative Computation for Brain Theory and Distributed AI", *Aritificial Intelligence and Neural Networks: Steps Toward Principled Integration*, eds. Vasant Honavar and Leonard Uhr, Academic Press, 51-74, 1994.

[28] Arbib, M. A., "Schema Theory", *The Encyclopedia of Artificial Intelligence*, 2nd edition, ed. S. Shapiro, Wiley-Interscience, 1427-43, 1992.

[29] Michalski, Ryszard S., "Toward a Unified Theory of Learning: Multistrategy Task-adaptive Learning", *Readings in Knowledge Acquisition and Learning*, eds. B. G. Buchanan and D. C. Wilkins, Morgan Kaufmannn, 1993.

Part II

A Neural/Evolutionary Hybrid System

I must create a system or be enslaved by another man's.
-- William Blake

A crucial aspect of computational learning that is often overlooked as trivial is that of obtaining training data for the learning algorithm. In fact, far from being trivial, this can often be the most difficult part of a computational learning approach to problem solving. This question of how to obtain training data motivates the research detailed in Chapters 4-7. Chapter 4 introduces an approach to generating data for training a neural network that utilizes evolutionary computation to ensure that the data generated will be useful. It has been published under the following reference.

Ventura, Dan, Tim Andersen and Tony Martinez, "Using Evolutionary Computation to Generate Training Set Data for Neural Networks", *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, Ecole des Mines d'Ales, France, pp. 468-471, 1995.

Chapters 5 and 6 are a natural progression of the work in Chapter 4. The approach for generating training data presented in Chapter 5 can be thought of in another light as an optimization problem with the goal being to find the very best example data. This viewpoint is used as the basis for adapting the research to the task of function optimization. Chapter 5 demonstrates how the evolutionary/neural hybrid can be thought of as a general approach to optimization, and Chapter 6 attempts to empirically evaluate its effectiveness with hundreds of simulation experiments. The results are encouraging in suggesting that the approach is indeed a robust approach to optimization. The references for the two chapters are as follows.

Ventura, Dan and Tony Martinez, "A General Evolutionary/Neural Hybrid Approach to Learning Optimization Problems", *Proceedings of the World Congress on Neural Networks*, San Diego, California, pp. 1091-1095, 1996.

Ventura, Dan and Tony Martinez, "Robust Optimization Using Training Set Evolution", *Proceedings of the International Conference on Neural Networks*, pp. 524-528, Washington D.C., 1996.

Given a method for optimization, it is natural to next consider the case of optimization through time, or in other words, control. Chapter 7 extends the hybrid approach into the domain of automatic closed-loop control for unstable systems. Early work on this idea was published in the following.

Ventura, Dan and Tony Martinez, "Using Evolutionary Computation to Facilitate Development of Neurocontrol", *Proceedings of the International Workshop on Neural Networks and Neurocontrol*, Playa del Carmen, Mexico, 1997.

Chapter 7 presents a mature approach to using the evolutionary/neural hybrid for neurocontrol and includes good empirical results for the classic problem of balancing a pole on a cart. It may be referenced as follows.

Ventura, Dan and Tony Martinez, "Optimal Control Using a Neural/Evolutionary Hybrid System", *Proceedings of the International Joint Conference on Neural Networks*, Anchorage, Alaska, pp. 1036-1041, 1998.

A certain amount of redundancy is inherent in the collection-of-papers format followed here. This redundancy is necessary as each chapter is a self-contained paper that may be read without reference to the rest of the dissertation. However, to aid in the case where the reader is considering the entire dissertation as a cohesive unit, suggestions for judiciously skipping through the chapters are offered here. Chapter 4 should be read in its entirety to get a feel for the basic approach to combining evolutionary and neural computation. For the following Chapters (5-7), Sections 2 and 3 of each may be skipped entirely as they are very similar to those in Chapter 4, discussing this basic approach. In addition, the introductory sections are somewhat similar, though each approaches the concepts in a different manner; therefore the reader may find it useful to skim Section 1 of each of the chapters to understand the motivations for the different applications of the hybrid approach.

Chapter 4

Using Evolutionary Computation To Generate Training Set Data For Neural Networks

Train up a child in the way he should go: and when he is old, he will not depart from it.

-- Proverbs 22: 6

Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, Ecole des Mines d'Ales, France, pp. 468-471, 1995.

Abstract

Most neural networks require a set of training examples in order to attempt to approximate a problem function. For many real-world problems, however, such a set of examples is unavailable. Such a problem involving feedback optimization of a computer network routing system has motivated a general method of generating artificial training sets using evolutionary computation. This paper describes the method and demonstrates its utility by presenting promising results from applying it to an artificial problem similar to a real-world network routing optimization problem.

1. Introduction

Many inductive learning algorithms based on neural networks, machine learning, and other approaches have been developed and have been shown to perform reasonably well on a variety of problems [2][4]. Typically, neural networks (NN) perform inductive learning through the presentation of preclassified examples; however, one of the largest obstacles faced in applying these algorithms to real-world problems is the lack of such a set of training examples. Many times collecting data for a training set is the most difficult facet of a problem.

This paper presents such a real-world problem -- one for which no training data exists and for which gathering such data is at best extremely expensive both in time and in resources. To remedy the lack of training set data, a method using evolutionary computation (EC) [3][8] is described in which the survivors of the evolution become the training examples for a neural network. The synthesis of EC with NN provides both initial unsupervised random exploration of the solution space as well as supervised generalization on those initial solutions. Work involving a combination of EC and NN is becoming more prevalent; the reader is referred to [1][5][6][7] for examples.

2. Problem Description

Although we describe here a specific problem, the approach described in the next section is general enough to apply to a large class of optimization/feedback problems. We are specifically concerned, however, with a computer networking application and the constant setting/adjusting of certain control variables depending on the values of certain status variables. More formally, given a network, Θ, the state of Θ may be described at time t by a vector of status variables, $\boldsymbol{s}^t$. Control of the network is effected by the setting of variables in a control vector, $\boldsymbol{c}$. That is, given a network Θ at time t described by vector $\boldsymbol{s}^t$, the setting of the values of the vector $\boldsymbol{c}$ will result in a different network Θ' at time $t+\delta$ described by the vector $\boldsymbol{s}^{t+\delta}$. The problem is, given a status vector, $\boldsymbol{s}^t$, what modifications should be made to the control vector $\boldsymbol{c}$ such that $\boldsymbol{s}^{t+\delta}$ describes a better network, if possible, than $\boldsymbol{s}^t$? Here, the best network is defined as the network that maximizes some function of its status variables, $f(\boldsymbol{s})$. The operation of Θ is continuous and feedback from Θ in the form of $\boldsymbol{s}$ is continuous as well. A neural network is expected to monitor the values of $\boldsymbol{s}$ either continuously or periodically and to update the values of $\boldsymbol{c}$, the goal being to maximize the performance of Θ over time. In this particular problem 21 status variables (e.g. AverageRoundTripTime, TotalPacketsSent, TotalPacketsReceived, etc.) and 11 control variables (e.g. SendWindowSize, RetransmitTimeInterval, MaxRetransmitCount, etc.) have been identified.

3. Generating the Training Set

Assuming that the status variables are defined over even a modest range, it is obvious that Θ may be described by any of an extremely large number of unique state vectors. In other words, the search space defined by the number of unique networks is enormous. Even if we limit ourselves for a moment to considering a single network by "freezing" the values of $\boldsymbol{s}$, we are still faced with a similar problem in choosing values for $\boldsymbol{c}$. With the intractable search spaces involved, evolutionary computation suggests itself for the exploration of those spaces. Actually, we limit ourselves to exploring, via evolutionary computation, the solution space for $\boldsymbol{c}$ only.

From the space defined by $\boldsymbol{s}$ that describes Θ we choose a representative set of network states by choosing n initial status vectors. We denote these $\boldsymbol{s}_i$, $0<i\leq n$ and refer to the network state described by $\boldsymbol{s}_i$ as Θ_i, $0<i\leq n$. These choices could of course be biased by any a priori heuristics as to what constitutes a realistic network. In choosing this set of status vectors, $\boldsymbol{s}_i$, $0<i\leq n$, we have chosen the left hand sides of our training instances. We now use evolutionary computation to discover "good" right hand sides yielding training instances of the form $\boldsymbol{s}_i^{t=0} \rightarrow \boldsymbol{c}_k$.

Assume a fitness function f that takes as input a status vector $\boldsymbol{s}$ and returns a real-valued fitness measure. Now for each $\boldsymbol{s}_i^{t=0}$, randomly initialize a population of m control vectors, denoted $\boldsymbol{c}_k$, $0<k\leq m$. Evaluate the initial population by simulating the workings of $\Theta_i^{t=0}$ for each $\boldsymbol{c}_k$ for δ time steps, where δ time steps are sufficient for Θ_i to stabilize, and then applying fitness function f to $\boldsymbol{s}_i^{t=\delta}$. Next choose parents and use genetic operators to produce m offspring. Now evaluate the children and select m survivors from amongst the parents and children. The algorithm is sketched in Figure 1.

evolve()
  generate $\boldsymbol{s}_i$
  for each $\boldsymbol{s}_i^{t=0}$
    initialize population $\boldsymbol{c}_k$, $0<k\leq m$.
    evaluate($\boldsymbol{c}_k$,Θ_i)
    until(done)
      select parents from $\boldsymbol{c}_k$
      apply genetic operators to parents
      evaluate(children,Θ_i)
      $\boldsymbol{c}_k \leftarrow$ choose m survivors from $\boldsymbol{c}_k$ and children

evaluate($\boldsymbol{c}$,Θ_i)
  run $\Theta_i^{t=0}$ for δ time steps with control vector $\boldsymbol{c}$
  return $f(\boldsymbol{s}_i^{t=\delta})$

Figure 1. Sketch of algorithm for evolving training set

Finally, choose j individuals from each of the n populations and build a set of jn training examples of the form $\boldsymbol{s}_i^{t=0} \rightarrow \boldsymbol{c}_k$. (To avoid ambiguity in the training set we could set $j=1$.)

We now return to the infinite space defined by $\boldsymbol{s}$. Since we have only chosen a finite number of seed points from this space, our evolutionary computation has found approximate solutions for only these n points in the space and can say nothing about any other points, many of which we are likely to encounter during normal execution of Θ. Therefore it becomes necessary to generalize on this relatively small set of approximate solutions. Using this set of approximate solutions as training examples, an NN model can be trained to develop a general hypothesis over the entire space defined by $\boldsymbol{s}$.

4. Simulation

In order to establish proof of concept, simulations using artificially generated problems were run. The simulation process includes the following steps:

1. Generate a problem definition
2. Create a training set using the algorithm
3. Train an NN with the training set
4. Create a test set
5. Test the NN network on the training set

An artificial problem generation/simulation program was used for two reasons. First, it is much easier to work with in terms of analysis, reproduction of results, etc. Second, it is possible to create a test set which can be used to show how well the NN is performing in relation to optimum, and thus to establish (to some extent) the quality of the EC-generated training set.

4.1. Generating a problem definition

In the networking problem, $\boldsymbol{s}$ is a vector of p status variables and $\boldsymbol{c}$ is a vector of q control variables. At any time t, $\boldsymbol{s}^t = g(\boldsymbol{s}^{t-\delta}, \boldsymbol{c}^{t-\delta})$, where g represents the operation of the network for time δ. Unfortunately, because of the network's asynchronous nature, it is impossible to determine the length of δ necessary for the network to stabilize or to accurately define g.

Therefore, we develop an artificial problem that is a simple analog of the network routing problem that preserves the relationship among $\boldsymbol{s}^t$, $\boldsymbol{s}^{t-1}$, and $\boldsymbol{c}^{t-1}$ but that operates in definite time steps and employs a simpler function H. Given a vector $\boldsymbol{s}$ of p integer variables (this vector is the analog of the status vector so we abuse notation and call it $\boldsymbol{s}$) and a vector $\boldsymbol{c}$ of q integer variables (the analog of the control vector), define the relationship $\boldsymbol{s}^t = H(\boldsymbol{s}^{t-1}, \boldsymbol{c}^{t-1})$ where H is in turn defined as a vector of functions $h_1, h_2, \ldots, h_p$. Now, to generate a problem, generate a $p \times q$ matrix G whose elements are randomly selected from {+,-}. The function h_i calculates the value of s_i^t from s_i^{t-1} and $\boldsymbol{c}^{t-1}$ as follows

$$s_i^t = h_i\left(s_i^{t-1}, \boldsymbol{c}^{t-1}\right) = \sum_{k=1}^{q} G_{ik}\left(s_i^{t-1} - c_k^{t-1}\right).$$

Thus, a problem is completely defined by H and G, which together represent a simple analog for the operation of a network. Once H and G have been defined, the combination of EC (to create a training set) and NN (to generalize on that training set) can attempt to learn the problem. To generate a new and different problem, randomly regenerate the matrix G; this allows for the generation of $\binom{2^q}{p}$ unique problems. Note that $\boldsymbol{s}_i^t$ ($\boldsymbol{c}_i^t$) has been used to indicate the *i*th vector at time t, while s_i^t (c_i^t) is now used to indicate the *ith element* of a vector $\boldsymbol{s}$ ($\boldsymbol{c}$) at time t, a slight abuse of notation.

As an example, suppose p=3 and q=3, that is, there are 3 "status" variables and 3 "control" variables. In order to generate a problem, randomly generate a 3×3 matrix G. One possibility is

$$G = \begin{bmatrix} + & - & - \\ + & + & + \\ - & + & - \end{bmatrix}$$

The system of equations that define the problem is then:

$$\begin{aligned} s_1^t &= +\left(s_1^{t-1} - c_1^{t-1}\right) - \left(s_1^{t-1} - c_2^{t-1}\right) - \left(s_1^{t-1} - c_3^{t-1}\right) \\ s_2^t &= +\left(s_2^{t-1} - c_1^{t-1}\right) + \left(s_2^{t-1} - c_2^{t-1}\right) + \left(s_2^{t-1} - c_3^{t-1}\right) \\ s_3^t &= -\left(s_3^{t-1} - c_1^{t-1}\right) + \left(s_3^{t-1} - c_2^{t-1}\right) - \left(s_3^{t-1} - c_3^{t-1}\right). \end{aligned}$$

4.2. Creating a training set

Once the problem has been defined, the evolutionary algorithm is employed to generate a training set. The function H and matrix G are substituted for the step "run $\Theta_i^{t=0}$ for δ time steps with control vector $\boldsymbol{c}$" in the *evaluate* procedure. The fitness function may be arbitrary, and the one used in these simulations was simply

$$f(\boldsymbol{s}) = \sum_{i=0}^{p} s_i$$

All simulations used training sets of size 1000.

4.3. Training the NN

The PDP group's software implementation of the backpropagation algorithm [4] was used for all simulation results. For the simulations we defined 5 integer "control" variables and 10 integer "status" variables on the range [0, 100). Note that even though we limit the problem to integer values, the search spaces are extremely large: 100^5 for "control" vectors and 100^{10} for "status" vectors. The inputs to the backprop (the "status" variables) were binary encoded, while the outputs were simple localist nodes, one for each control variable. Fifteen hidden units were used so that the entire network consisted of 70 input nodes, 15 hidden units, and 5 output nodes. The output nodes produce values in the range [0, 1) and these are simply multiplied by 100 to produce a "control" vector in the desired range. All simulations were run with the default settings and training was only allowed to proceed for 500 epochs.

4.4. Creating a test set

The main advantage of having an explicitly defined function H representing our problem is that given a status vector $\boldsymbol{s}^t$, the optimal control vector $\boldsymbol{c}$ (the one that optimizes $\boldsymbol{s}^{t+1}$) can be calculated. The simplest method is by brute force, though this can be significantly improved in the case of this particular function H. To create a test set, randomly generate a vector $\boldsymbol{s}$ and then determine the optimal vector $\boldsymbol{c}$. Add the instance $\boldsymbol{s} \rightarrow \boldsymbol{c}$ to the training set and repeat for as many instances as desired. All simulations used a test set of size 50.

4.5. Testing the NN network

Typically, when evaluating how well the NN performs on a given training set we determine the number of correctly classified instances from the test set. However, because this is an optimization problem and because the output nodes are localist and expected to produce outputs in the range [0, 1) (rather that just being active or not), the NN is not expected to produce the optimum vector $\boldsymbol{c}_{opt}$ but rather a good approximation of it, $\boldsymbol{c}_{approx}$. Thus, given a status vector $\boldsymbol{s}$ the NN generates $\boldsymbol{c}_{approx}$, while $\boldsymbol{c}_{opt}$ and $\boldsymbol{c}_{worst}$ (needed for normalization) may be found by brute force (or some more efficient method). A measure of the correctness (normalized % of optimum) of $\boldsymbol{c}_{approx}$ is

$$\frac{f(\boldsymbol{s}_{approx}) - f(\boldsymbol{s}_{worst})}{f(\boldsymbol{s}_{opt}) - f(\boldsymbol{s}_{worst})}$$

where $\boldsymbol{s}_{approx}=H(\boldsymbol{s},\boldsymbol{c}_{approx})$, $\boldsymbol{s}_{worst}=H(\boldsymbol{s},\boldsymbol{c}_{worst})$, and $\boldsymbol{s}_{opt}=H(\boldsymbol{s},\boldsymbol{c}_{opt})$.

5. Empirical Results

The simulation process as described in the previous section was performed for genetic population sizes (the upper bound m in the *evolve* algorithm) of 20, 100, and 200. Ten different simulations were run for each value of m and the results are reported in Table 1.

The % of optimum is the correctness measure discussed above where $\boldsymbol{c}_{approx}$ is produced by either the GA (training set) or the NN (test set). The average pss is the average sum-squared error per pattern.

Table 1. Simulation results for different population sizes

	Population Size (m)					
	20		100		200	
	mean	std dev	mean	std dev	mean	std dev
% opt. (train)	73.8	8.1	87.6	5.5	89.9	4.6
% opt. (test)	70.8	10.3	80.9	6.8	85.1	8.6
avg pss (train)	.237	.037	.128	.023	.103	.023
avg pss (test)	.667	.145	.364	.102	.327	.102

A number of interesting observations may be made from the table. First, the values for percent of optimum for the training set indicate that the GA is finding artificial instances that seem reasonable in the sense that they are a significant percentage of optimum. Second, the values for percent of optimum for the test set indicate that the training set produced by the GA enabled the NN to learn the problem fairly well, since it performs almost as well as the GA. Even more significant is the fact that as we increase the population size, both the GA and the NN perform closer to optimum with smaller standard deviations. Also, as the population size is increased, the sum-squared error on both the training set and test set decreases indicating that the training set more closely represents the true function.

6. Conclusion

Empirical simulation results suggest that at least for some problems, evolutionary computation is indeed capable of generating a training set that closely represents the actual underlying function. The key to this process is the assumption that an appropriate fitness function f can be defined. We conjecture that in many cases this will indeed be the case (e.g. in the networking problem f would attempt to maximize throughput while minimizing resource usage). This paper has developed proof-of-concept on a relatively simple problem. Current research involves application of this method to the real network optimization problem and other difficult real-world applications. Unfortunately, ascertaining what percentage of optimum the approximate solutions achieve will not be possible (since the function g will be unknown). A measure of performance will have to be obtained by comparing this approach with current methods for solving these problems. As a final note, although generating the training set and training the NN is potentially time consuming, this need be done only once (or very infrequently), and it is conceivable that further training could be ongoing in the background using results obtained during execution. The actual execution would be extremely fast.

References

[1] Caudell, T. P. and C. P. Dolan, "Parametric Connectivity: Training of Constrained Networks using Genetic Algorithms", *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[2] Falhman, S. E. and F. Lebiere, "The Cascade-Correlation Learning Architecture", *Advances in Neural Information Processing 2*, D. S. Touretzky (ed.), Morgan Kaufmannn, 1990.

[3] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning,* Addison-Wesley Publishing, 1989.

[4] McClelland, James L. and David E. Rumelhart, *Explorations in Parallel Distributed Processing*, MIT Press, Cambridge, Massachusetts, 1988.

[5] Harp, S. A., T. Samad and A. Guha, "Designing Application-Specific Neural Networks Using the Genetic Algorithm", *NIPS-89 Proceedings*, 1990.

[6] Montana, D. J. and L. Davis, "Training Feedforward Neural Networks Using Genetic Algorithms", *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[7] Romaniuk, Steve G., "Evolutionary Growth Perceptrons", *Genetic Algorithms: 5th International Conference (ICGA-93)*, S. Forrest (ed.), Morgan Kaufmannn, 1993.

[8] Spears, W. M., K. A. Dejong, T. Baeck, D. Fogel and H. de Garis, "An Overview of Evolutionary Computation", *European Conference on Machine Learning (ECML-93)*, 1993.

Chapter 5

A General Evolutionary/Neural Hybrid Approach to Learning Optimization Problems

Don't tell your problems to people; eighty percent don't care,
and the other twenty percent are glad you have them.
-- Lou Holtz

Proceedings of the World Congress on Neural Networks, San Diego, California, pp. 1091-1095, 1996.

Abstract

A method combining the parallel search capabilities of Evolutionary Computation (EC) with the generalization of Neural Networks (NN) for solving learning optimization problems is presented. Assuming a fitness function for potential solutions can be found, EC can be used to explore the solution space, and the survivors of the evolution can be used as a training set for the NN which then generalizes over the entire space. Because the training set is generated by EC using a fitness function, this hybrid approach allows explicit control of training set quality.

1. Introduction

Efforts to develop mechanisms that can be considered in some way intelligent have resulted in many problem solving techniques from various fields including Machine Learning, Neural Networks, Evolutionary Computation, Fuzzy Logic, and Symbolic Artificial Intelligence. Approaches from these different fields each have their strengths and their weaknesses. Recent research suggests that further progression may depend upon the synergistic combination of several complementary approaches from these different fields [4][6][11]. This new synergistic approach to machine intelligence is sometimes referred to as Hybrid Systems. One such possibility is the combination of Evolutionary Computation (EC) [3][10] with Neural Networks (NN) [9][13]. For some examples of such combinations, the reader is referred to [1][5][7][8].

This paper extends work on one such hybrid method, first presented in [12], that combines the broad, parallel search capabilities of EC with the generalization and execution speed of NN. Given a function to be learned, under certain reasonable assumptions the EC can be used to solve the problem at selected points in the input space; the survivors of the evolution become the instances in a training set for the NN, which then generalizes the optimization for the entire input space.

Section 2 of the paper presents a generalized formal description of the problem to be solved. Section 3 then discusses the combination of NN with EC as a general approach to solving the problem. As a proof-of-concept for the general approach, Section 4 describes a specific learning optimization problem and presents empirical results from simulations run on that problem. Finally, Section 5 presents conclusions and directions for ongoing research.

2. Problem Description

Given a system, Θ, the state of Θ may be described at time t by a vector of status variables, $\boldsymbol{s}^t$. Control of the system is effected by Γ which applies a control vector, $\boldsymbol{c}$, to Θ. That is, given a system Θ at time t described by vector $\boldsymbol{s}^t$, the setting of the values of the vector $\boldsymbol{c}$ will result in a different system Θ' at time $t+\delta$ described by the vector $\boldsymbol{s}^{t+\delta}$. The problem is how should Γ be constructed so that given a status vector, $\boldsymbol{s}^t$, Γ outputs a control vector $\boldsymbol{c}$ such that $\boldsymbol{s}^{t+\delta}$ describes a better system, if possible, than $\boldsymbol{s}^t$? We assume the existence of some evaluation function, f, that will determine whether or not one system is better than another. The operation of Θ may be either continuous or discrete and Γ knows nothing about Θ except for the value of $\boldsymbol{s}^t$. Given $\boldsymbol{s}^t$, Γ is expected to output values for $\boldsymbol{c}$, the goal being to maximize f for any given instance (status) of Θ. The problem may be a single-step optimization; alternatively, though this is not pursued here, the problem may include feedback with the process becoming iterative (see Figure 1).

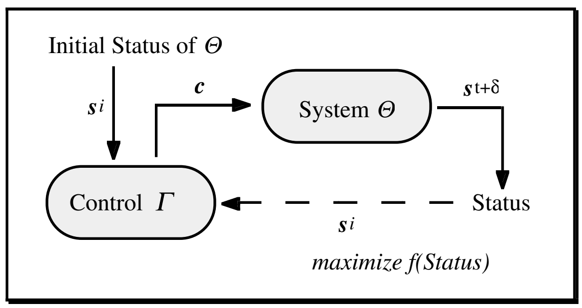

Figure 1. Problem Description

3. Combining Evolutionary Computation with Neural Computation

Assuming that the status and control variables are defined over even a modest range, it is obvious that the input (status) and output (control) spaces will be extremely large. Evolutionary computation lends itself well to the exploration of large spaces. However, such evolutionary exploration is often slow. If we assume that the mapping $\boldsymbol{s} \rightarrow \boldsymbol{c}$ is non random and in some sense generalizeable, then a (hopefully) representative set of points may be discovered via EC, and those points then used to train a NN which may then generalize over the rest of the function. The goal of the EC/NN synergism is to obtain the accuracy of evolutionary search and the speed of neural execution.

From the space defined by $\boldsymbol{s}$ that describes Θ we choose a representative set of system states by choosing n initial status vectors. We denote these $\boldsymbol{s}_i^{t=0}$, $0<i\leq n$ and refer to the system state described by $\boldsymbol{s}_i^{t=0}$ as $\Theta_i^{t=0}$, $0<i\leq n$. These choices could of course be biased by any a priori heuristics as to what constitutes a realistic system (see below and Section 5). The goal is to know, given one of these $\boldsymbol{s}_i^{t=0}$, what a “good” $\boldsymbol{c}$ vector would be. For each of the $\boldsymbol{s}_i^{t=0}$, EC is used to discover such a $\boldsymbol{c}$ in the following way.

```
evolve ()
    generate s_i
    for each s_i^{t=0}
      initialize population c_k, 0<k≤m.
      evaluate (c_k,Θ_i)
      until(done)
        select parents from c_k
        apply genetic operators to parents
        evaluate (children,Θ_i)
        c_k ← choose m survivors from c_k and children

evaluate (c,Θ_i)
    run Θ_i^{t=0} for δ time steps with control vector c
    return f(s_i^{t=δ})
```

Figure 2. Algorithm for evolving training set

Assume a fitness function f that takes as input a status vector $\boldsymbol{s}$ and returns a real-valued fitness measure. Now for each $\boldsymbol{s}_i^{t=0}$, randomly initialize a population of m control vectors, denoted $\boldsymbol{c}_k$, $0<k\leq m$. Evaluate the initial population by simulating the workings of $\Theta_i^{t=0}$ for δ time steps (where δ time steps are sufficient for Θ_i to stabilize) for each $\boldsymbol{c}_k$, and then applying fitness function f to $\boldsymbol{s}_i^{t=\delta}$. Next, until some stopping criterion is reached (a maximum number of generations, for example) choose parents and use genetic operators (crossover, mutation, etc.) to produce m offspring, evaluate the children and select m survivors from amongst the parents and children. The algorithm is sketched in Figure 2. Finally, for each of the n populations, choose the individual, $\boldsymbol{c}_{max_i}$, (the individual with the highest fitness) and build a set of n training examples of the form $\boldsymbol{s}_i^{t=0} \rightarrow \boldsymbol{c}_{max_i}$.

The EC has now found "good" approximate solutions for n points from the input (status) space but can say nothing about any other points, many of which we are likely to encounter during normal execution of Θ. Obviously, one solution to the problem defined in Section 2 would be to employ the evolutionary scheme discussed above as the control Γ. However, this would likely be unacceptable in terms of execution speed. Therefore, the NN is employed to generalize over the entire space defined by $\boldsymbol{s}$ using the relatively small set of approximate solutions as a training set. While the initial training of the network maybe somewhat time consuming, depending on the network and training algorithm employed, the generalization during execution will be extremely fast. The synergistic combination of EC and NN is then employed as the control Γ as in Figure 3.

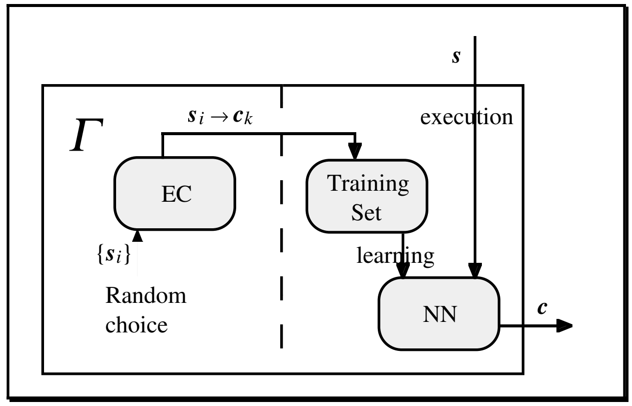

Figure 3. The control Γ

In experiments performed to date, the original set $s_i^{t=0}$ is simply chosen randomly. However, it is possible that techniques such as active learning [2] can be employed to choose the set $s_i^{t=0}$, leading to smaller training and/or better NN generalization.

The power of this EC/NN hybrid approach is its combination of the thoroughness of evolutionary search with the speed of neural generalization as well as its general applicability to any learning problem for which a fitness function can be found and for which "blackbox" access to the system to be learned (or a reasonable simulation thereof) is possible. In order to provide proof-of-concept, the next section presents a specific example of a problem and the results of applying the EC/NN hybrid to solve it.

4. Simulation

In the simulations, the EC/NN combination attempts to learn to use a simple gun to hit a target moving in 2-d space. An artificial problem was used for two reasons. First, it is much easier to work with in terms of analysis, reproduction of results, etc. Second, it is possible to create a test set which can be used to show how well the NN is performing in relation to optimum, and thus to establish (to some extent) both the quality of the training set generated as well as the quality of the NN's generalization.

The problem is difficult in several respects -- most notably in the fact that there are many optimal bullets (in the sense that they hit the target) for each target, and no attempt is made to constrain which of these is genetically chosen. However, it is unclear how a neural network trained with these two instances would generalize.

A second difficulty arises from the fact that only integer values are allowed for velocity values. This makes the optimization much more difficult than if real values were allowed, analogous to the difference in difficulty between the linear and integer programming problems.

4.1. The system Θ

The vector $\boldsymbol{s}$ describes the x and y components of the target's velocity and is therefore comprised of two elements: s_1 and s_2. The vector $\boldsymbol{c}$ describes the x and y components of the bullet's velocity and so consists of two elements: c_1 and c_2. The target vector is subject to a gravitational constant, but the bullet vector is not (it is assumed negligible). The system Θ describes the trajectories of both bullet and target as in Figure 4.

```
trajectories (targetx, targety, bulletx, bullety)
    targetx = targetx + targetxvelocity
    targety = targety + targetyvelocity
    bulletx = bulletx + bulletxvelocity
    bullety = bullety + bulletyvelocity
    if(mindistance > ((targetx-bulletx)+(targety-bullety)^2)^1/2)
        mindistance = ((targetx-bulletx)+(targety-bullety)^2)^1/2
    targetyvelocity = targetyvelocity - gravity
return(mindistance)
```

Figure 4. The target/bullet system

Notice that the function *trajectories*() returns the minimum distance between bullet and target in the course of their trajectories. It therefore acts as the system simulator and the fitness function. Optimizing this system entails minimizing the Euclidean distance between target and bullet at their nearest point. The fitness function therefore is simply

$$\textit{fitness}(\boldsymbol{s}) = 1 - \text{mindistance}(\boldsymbol{s},\boldsymbol{c})$$

which is to be maximized. The default fitness is the distance between the initial bullet position and the initial target position. Obviously, the ideal fitness is 0, and thus *fitness*() is defined over the range [0,default fitness].

4.2. Creating a training set

Training sets were comprised of 250 instances obtained using the evolutionary method described above. For each randomly generated target (left hand side), a population of bullets was allowed to evolve and the bullet that came closest to hitting the target was selected as the right hand side. The function *trajectories* is substituted for the step "run $\Theta_i^{t=0}$ for δ time steps with control vector $\boldsymbol{c}$" in the *evaluate* procedure, and

$$\textit{fitness}(\text{bullet}) = \text{mindistance}(\text{bullet, target}).$$

4.3. Training the neural network

The PDP group's software implementation of the backpropagation algorithm [9] was used for all simulation results. For the simulations, 2 integer "status" variables and 2 integer "control" variables were defined that describe the x and y velocities of the target and the bullet respectively. The inputs to the backpropagation NN (the "status" variables) were binary encoded, while the outputs were simple localist nodes, one for each control variable. Ten hidden units were used so that the entire network consisted of 12 input nodes, 10 hidden units (number of hidden nodes was determined empirically), and 2 output nodes. The output nodes produce values in the range (0, 1) and these are scaled to produce an integer "control" vector in the desired range. All NN simulations were run with the default settings and training was allowed to proceed for 1000 epochs.

4.4. Creating a test set

Creating a test set for this problem is extremely simple. Obviously, the optimum fitness for any bullet is 0. Therefore, to create a test set, simply randomly generate targets (left hand sides) that have not been seen by the NN before. Right hand sides are not necessary as explained below. All test sets generated contained 20 instances.

4.5. Testing the neural network

For this problem, the NN is expected to produce ***bullet***$_{approx}$, which represents the x and y velocities of a bullet that should come close to (hopefully hit) the target. Thus, given a target vector ***target*** the NN generates the bullet vector ***bullet***$_{approx}$. *Trajectories(***target, *bullet***$_{approx}$*)* is then used to determine how close ***bullet***$_{approx}$ comes to the target. Even though both ***bullet***$_{opt}$ and ***bullet***$_{worst}$ are unknown, *trajectories(**target, bullet**$_{opt}$)* = 0 and *trajectories(**target, bullet**$_{worst}$)* = default fitness (the default minimum distance between the target origin point and the bullet origin point in the simulation). A measure of the correctness (normalized % of optimum) of ***bullet***$_{approx}$ is therefore

$$\frac{trajectories(target,\ bullet_{worst}) - trajectories(target,\ bullet_{approx})}{trajectories(target,\ bullet_{worst}) - trajectories(target,\ bullet_{opt})},$$

which reduces to

$$1 - \frac{trajectories(target,\ bullet_{approx})}{\text{default fitness}}.$$

4.6. Empirical Results

All simulations were run 15 times and the results averaged. Initially, the evolutionary computation was able in every case to generate a perfect training set in the sense that all the examples had the maximum fitness (all were direct hits). This seems to indicate that the EC has generated a training set that faithfully represents the problem and the results of the NN's generalization accuracy bear this out.

Next, varying levels of noise were introduced into the training sets. The majority of instances in each training set were still direct hits, but each training set also contained a few instances with very poor fitness (the bullet did not come anywhere near the target). One of the most difficult aspects of learning with NN is obtaining a good training set. In contrast, the EC/NN hybrid approach described here allows for explicit control of training set quality because of the fact that a fitness function exists. Therefore, the fitness or quality of each individual example as well as the overall fitness of the training set (taken as the average) is known. Any examples whose fitness is below a certain threshold can simply be thrown out.

Table 1 presents the results of training a backpropagation NN with 10 hidden nodes on the genetically generated training sets and then testing it on test sets that it had not seen before. The first column of results is from simulations run with no noise in which the EC was always able to find training sets comprised entirely of examples with maximum fitness (direct hits). The remaining three columns are from simulations run with varying degrees of noise injected into the training sets, with each column representing the results from a different noise level as labeled. Each is further divided in half with the left half representing the results of training the NN on the noisy training set and the right half the results of training the NN on the smaller training set obtained by eliminating the noisy example using the fitness function.

Table 1. Results of simulation

Noise level	0%	15%		25%		35%	
Average training set fitness (EC)	1.00	.958	1.00	.915	1.00	.866	1.00
Average NN generalization	.943	.876	.920	.804	.834	.775	.910
Best NN generalization	.991	.944	.996	.925	.895	.887	.999
Worst NN generalization	.851	.757	.842	.632	.741	.653	.809
Average number of direct hits	10.8	5.5	10.2	4.0	4.4	2.7	11.3
Most direct hits (of 20)	16	9	16	7	6	5	19
Least direct hits (of 20)	3	2	3	2	0	1	4

The first row indicates the average training set fitness over all 15 runs. Recall, this is *not* the performance of the NN on the training set, but rather a measure of how good the training set (generated by the EC) is compared to the theoretical optimum of all instances being direct hits. As mentioned earlier, in the case of 0% noise, the EC was always able to create a training set with maximum fitness. This is an indication that the training set is a good representation of the problem. However, this was not the case with the noisy simulations. Still, the majority of the 250 instances in all the training sets were direct hits; however, in each of the three scenarios, a few instances of each training set in each run were always very poor. The second row indicates how well, on average, the NN generalized on a test set after being trained on one of the training sets. Since each of the 15 runs themselves represent averages over 250 training instances and 20 test instances, the first two rows of the table are really averages of averages.

The third row shows the best NN generalization over any of the test sets and the fourth row shows the corresponding worst generalization. The fifth row shows the average number of times (out of 20 test instances) that the NN actually hit the test target (a hit entails the minimum distance between bullet and target being 0). The sixth row shows the most hits in a single test set, and the last row shows the least hits in a single test set. The results are encouraging because they show that the EC can generate a good training set that allows the NN to generalize well over the entire problem space. From another perspective, the EC/NN hybrid has potential for effectively approximating a system.

For example, the best NN generalization in six of the seven scenarios (the right half of the 25% noise level column being the exception) is very close to the training set fitness. As for the average NN generalization under noisy conditions, the better the training set fitness (quality), the better the NN generalization. Obviously, the noisier the training set, the poorer the generalization (see the left halves of the three noisy columns). This is certainly no surprise. However, as discussed earlier, the EC/NN hybrid can be used to explicitly control training set quality by detecting and eliminating any training example whose fitness falls below an arbitrary threshold. This ability allows the hybrid to perform well even under noisy conditions (see the right halves of the three noisy columns). Also, averages of 10+ direct hits out of 20 test instances is respectable (again, the exception being the 25% column), given the EC/NN hybrid's complete lack of *a priori* knowledge for the problem, and the maximum number of direct hits for the 35% noise column (19) is excellent. (The one miss was within 0.00041 of being a hit.)

The results of the 25% noise column are a little less encouraging. In comparison to the other two noisy columns, detection and removal of noisy instances results in much less improvement in NN performance (still, *some* improvement is gained). This is probably due to the EC choosing somewhat unrepresentative points in the input space. Due to the nature of EC, this situation cannot always be avoided but increasing initial population size (if feasible) can minimize these effects.

5. Conclusion

Empirical simulation results suggest that for a broad class of problems (any problem for which a fitness function can be found to evaluate possible solutions), a combination of evolutionary and neural computation is indeed capable of efficiently learning to control a system by combining the thorough search capabilities of EC with the generalization and processing speed of NN. This is possible because the EC is able to generate a training set that closely represents the actual underlying function and the NN is then able to generalize from the training set. Further, the hybrid is capable of explicitly controlling the quality of the training set. The key to this process is the assumption that an appropriate fitness function f can be defined. We conjecture that in many cases this will indeed be the case (e.g. in a network control problem f would attempt to maximize throughput while minimizing resource usage). Although it is very difficult to solve optimization-type problems, determining the relative worth of one solution over another (via some fitness measure) appears, in general, to be a much simpler endeavor. This paper has developed proof-of-concept on a relatively simple problem.

There are many fruitful areas for future research including active learning of the training set, testing the hybrid on feedback problems, using other search techniques (other evolutionary approaches, simulated annealing, etc.), using other function approximation techniques (machine learning, fuzzy logic, etc.), the trade-off between a training set with few optimal examples vs. one with many sub-optimal examples, and determining better stopping criteria for the evolution and for the NN training. Also, application to real-world problems is an important next step. Unfortunately, ascertaining what percentage of optimum the approximate solutions achieve will not be possible (since the underlying function describing the system to be optimized will be unknown); however, a measure of performance can be obtained by comparing this approach with current methods for solving these problems.

References

[1] Caudell, T. P. and C. P. Dolan, “Parametric Connectivity: Training of Constrained Networks using Genetic Algorithms”, *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[2] Cohn, David, Les Atlas and Richard Ladner, “Improving generalization with active learning” *Machine Learning*, 15:201-21, May 1994.

[3] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing, 1989.

[4] Goonatilake, Suran and Sukhdev Khebbal (eds.), *Intelligent Hybrid Systems*, John Wiley & Sons, 1995.

[5] Harp, S. A., T. Samad and A. Guha, “Designing Application-Specific Neural Networks Using the Genetic Algorithm”, *NIPS-89 Proceedings*, 1990.

[6] Honavar, Vasant, and Leonard Uhr (eds.), *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration*, Academic Press, Inc., San Diego, California, 1994.

[7] Montana, D. J. and L. Davis, “Training Feedforward Neural Networks Using Genetic Algorithms”, *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[8] Romaniuk, Steve G., “Evolutionary Growth Perceptrons”, *Genetic Algorithms: 5th International Conference (ICGA-93)*, S. Forrest (ed.), Morgan Kaufmannn, 1993.

[9] Rumelhart, David E., James L. McClelland and the PDP Research Group, *Parallel Distributed Processing*, MIT Press, Massachusetts, 1988.

[10] Spears, W. M., K. A. Dejong, T. Baeck, D. Fogel and H. de Garis, “An Overview of Evolutionary Computation”, *European Conference on Machine Learning (ECML-93)*, 1993.

[11] Sun, Ron and Lawrence Bookman (eds.), *Computational Architectures Integrating Neural and Symbolic Processes: A Perspective on the State of the Art*, Kluwer Academic Publishers, Massachusetts, 1995.

[12] Ventura, Dan, Tim Andersen and Tony Martinez, "Using Evolutionary Computation to Generate Training Set Data for Neural Networks", *Proceedings of the International Conference on Neural Networks and Genetic Algorithms*, pp. 468-471, 1995.

[13] Wasserman, Philip D., *Advanced Methods in Neural Computing*, Van Nostrand Reinhold, New York, 1993.

Chapter 6

Robust Optimization Using Training Set Evolution

A complex system that works is invariably found to have evolved from a simple system that works.

-- John Gaule

Proceedings of the International Conference on Neural Networks, Washington D.C, pp. 524-528, 1996.

Abstract

Training Set Evolution is an eclectic optimization technique that combines evolutionary computation (EC) with neural networks (NN). The synthesis of EC with NN provides both initial unsupervised random exploration of the solution space as well as supervised generalization on those initial solutions. An assimilation of a large amount of data obtained over many simulations provides encouraging empirical evidence for the robustness of Evolutionary Training Sets as an optimization technique for feedback and control problems.

1. Introduction

Neural networks (NN) have been successfully applied to a variety of problems [2][5]. Also, work involving a combination of evolutionary computation (EC) and NN is becoming more prevalent [1][4][6][7]. One class of problems to which NN are often applied is that of optimization. The NN is responsible for optimizing a system based upon some criteria. Evolutionary Training Sets is an optimization technique that employs evolutionary computation [3][8] as a preprocessor that creates a training set for the neural network. The only requirements for the optimization technique are access to the system to be optimized and *a priori* knowledge of a fitness function that describes the desired optimization. The synthesis of EC with NN provides both initial unsupervised random exploration of the solution space as well as supervised generalization on those initial solutions.

Evolutionary Training Sets are introduced in [9] and [10]. Two artificial problems have been designed to explore the usefulness of this optimization technique. This paper extends previous work by presenting results obtained from running hundreds of new simulations in order to study the effects of varying periods of evolution and training set size on the effectiveness of optimization. Empirical results provide encouraging evidence for the robustness and general usefulness of Evolutionary Training Sets.

Section 2 of the paper presents a generalized formal description of the problem to be solved -- optimization of a system (or equivalently, function approximation), either statically or dynamically (control problems). Section 3 then briefly discusses the combination of a neural network with evolutionary computation as a general approach to solving the problem of Section 2. Section 4 presents data (collected over many empirical simulations) in condensed graphical form and discusses its implications for Evolutionary

Training Set optimization. Finally, Section 5 presents conclusions and directions for ongoing research.

2. Problem Description

Given a system, Θ, the state of Θ may be described at time t by a vector of status variables, $\boldsymbol{s}^t$. Suppose that control of the system is effected by the setting of variables in a control vector, $\boldsymbol{c}$. That is, given a system Θ at time t described by vector $\boldsymbol{s}^t$, the setting of the values of the vector $\boldsymbol{c}$ will result in a different system Θ' at time $t+\delta$ described by the vector $\boldsymbol{s}^{t+\delta}$. The problem is, given a status vector, $\boldsymbol{s}^t$, what modifications should be made to the control vector $\boldsymbol{c}$ such that $\boldsymbol{s}^{t+\delta}$ describes a better system, if possible, than $\boldsymbol{s}^t$? Obviously, some evaluation or fitness function, f, is necessary in order to determine whether or not one system is better than another.

The operation of Θ may be either continuous or discrete. A neural network is expected to detect the values of $\boldsymbol{s}^t$ and to output values for $\boldsymbol{c}$, the goal being to maximize f for any given instance (status) of Θ. If the problem is an optimization problem, this is a single iteration process; if the problem is an optimization/feedback problem then the process becomes an ongoing series of iterations.

3. Combining Evolutionary Computation with Neural Computation

From the space defined by $\boldsymbol{s}$ that describes Θ we choose a representative set of system states by choosing n initial status vectors. We denote these $\boldsymbol{s}_i^{t=0}$, $0<i\leq n$ and refer to the system state described by $\boldsymbol{s}_i^{t=0}$ as $\Theta_i^{t=0}$, $0<i\leq n$. These choices could of course be biased by any a priori heuristics as to what constitutes a realistic system. In choosing this set of status vectors, $\boldsymbol{s}_i^{t=0}$, $0<i\leq n$, we have chosen the left hand sides of the training instances. We now use evolutionary computation to discover "good" right hand sides, yielding training instances of the form $\boldsymbol{s}_i^{t=0} \rightarrow \boldsymbol{c}_k$.

Assume a fitness function f that takes as input a status vector $\boldsymbol{s}$ and returns a real-valued fitness measure. Now for each $\boldsymbol{s}_i^{t=0}$, randomly initialize a population of m control vectors, denoted $\boldsymbol{c}_k$, $0<k\leq m$. Evaluate the initial population by simulating the workings of $\Theta_i^{t=0}$ for δ time steps (where δ time steps are sufficient for Θ_i to stabilize) for each $\boldsymbol{c}_k$, and then applying fitness function f to $\boldsymbol{s}_i^{t=\delta}$. Next choose parents and use genetic operators to produce m offspring. Now evaluate the children and select m survivors from amongst the parents and children. Repeat this process until some stopping criterion is met (such as reaching a specified number of generations or finding an individual with a fitness higher than some threshold).

Finally, choose j individuals from each of the n populations and build a set of jn training examples of the form $\boldsymbol{s}_i^{t=0} \rightarrow \boldsymbol{c}_k$. (To avoid ambiguity in the training set we could set j=1.)

Since we have only chosen a finite number of seed points from this space, our evolutionary computation has found approximate solutions for only these n points in the space and can say nothing about any other points, many of which we are likely to encounter during normal execution of Θ. Therefore it becomes necessary to generalize on this relatively small set of approximate solutions. Using this set of approximate solutions as training examples, an NN model can be trained to develop a general hypothesis over the entire space defined by $\boldsymbol{s}$. (For a more thorough explanation of the algorithm, see [9] or [10]).

The power of this NN/EC hybrid approach is its general applicability to a wide class of problems including function approximation problems, optimization problems, feedback problems, and control problems.

4. Empirical Results

In order to study the probabilistic effects of varying length of evolution and training set size on quality of optimization, simulations using two artificially generated problems were run. The first entails solving a set of mathematical equations (described by a matrix) and the second requires the EC/NN combination to attempt to learn how to hit a target moving in 2-d space. Due to space constraints, neither problem is described in this paper; however, the first is described in [9] and the second in [10]. In general, both simulation processes include the following steps:

1. Generate a problem definition
2. Create a training set using evolutionary computation
3. Train an NN with the training set
4. Create a test set
5. Test the NN on the training set

Since each example in the training set has an associated fitness value (determined during creation of the training set by the evolutionary computation), an overall average fitness of the training set can easily be determined. We term this average fitness of the training set its *quality*. This research attempts to answer the following three questions regarding training set quality:

1) Does the evolutionary computation produce training sets of good quality?
2) Does training set quality correlate with NN generalization?
3) Does number of training examples or quality of training set more greatly affect NN generalization?

Artificial problem generation/simulation programs were used for several reasons. First, they are much easier to work with in terms of analysis, reproduction of results, etc. Second, it is possible to create a test set which can be used to show how well the NN is performing in relation to optimum, and thus to establish (to some extent) the quality of the optimization procedure. Third, fitness of individual training examples and therefore of entire training sets can also be measured against optimum. The training set quality then becomes a natural (though not a strict) upper bound on the NN's generalization accuracy and can therefore be used as a yardstick by which to measure the NN's performance.

All results are averages over ten runs, and all NN simulation was done with the PDP implementation of the back propagation algorithm [5].

4.1. Matrix problem

This problem involves solving a system of 10 equations with 15 variables for the 5 unknown variables, where the system of equations is represented by a matrix. The evolutionary computation produces a training set of hopefully good solutions for a small number of points (1000 examples) in the equation space; the NN then generalizes on the training set in order to approximate the function described by the matrix. Since the variables are defined on the range [0,100), the problem is difficult because of the huge search spaces (100^{10} for the space to be explored by the EC and 100^5 for the space to be generalized by the NN) involved.

Figure 1 shows the effects of evolution on training set quality (indicated by the filled squares). Here evolution is indicated on the *x*-axis and is a logarithmic function of both population size and number of generations. The more extended the evolution (that is, the larger the population and/or the more generations of evolution), the higher the training set quality. This is not surprising. The more interesting result is that the higher the training set quality, the better the NN generalization. The hollow squares indicate NN generalization using all 1000 examples in the training set, the solid circles indicate generalization on half of the training set (500 examples), and the hollow circles show generalization on one quarter of the training set (250 examples). Notice that generalization accuracy increases as training set quality increases. In other words, the evolutionary computation is producing a training set that faithfully represents the underlying function to be approximated.

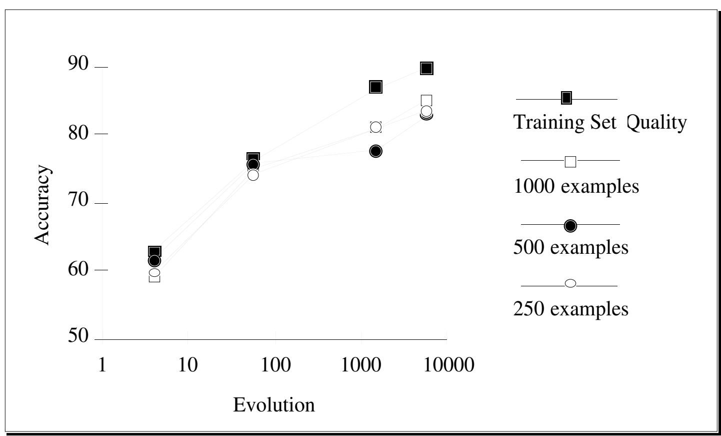

Figure 1. Training set quality and NN generalization accuracy for matrix problem

Figure 2 shows that as evolution time increases, the standard deviation in training set quality decreases. So with longer evolution, the probability of finding a good training set increases. Even more encouraging, Figure 2 also indicates that the standard deviation in NN generalization also decreases as training set quality goes up. Therefore, as evolution times are increased, the probability of finding a high quality training set increases; and as the quality of training set increases, the probability of good generalization accuracy increases as well.

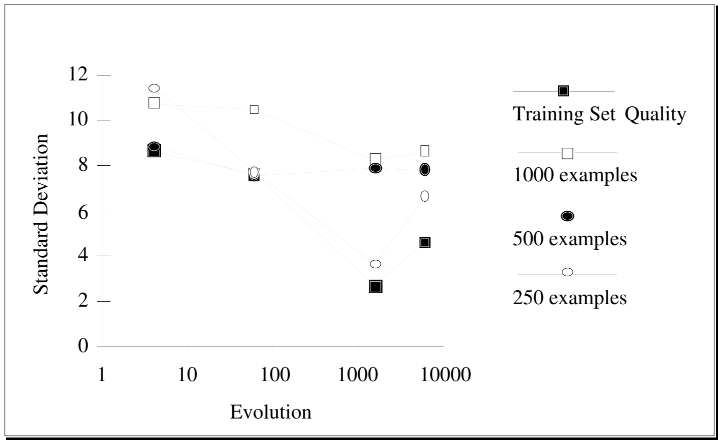

Figure 2. Standard deviation of training set quality and generalization accuracy for matrix problem

4.2. Target problem

This problem involves the NN trying to learn to hit a target moving in 2-d space with a simple gun, where both bullet and target are subject to the effects of gravity. The evolutionary computation produces a training set of hopefully good bullets for a small number of points (250 examples) in the target space; the NN then generalizes on the training set in order to attempt to learn how to hit any target. The search spaces are much smaller in this problem (35^2 for the EC and 45^2 for the NN); the difficulty this time arises from the fact that the gun is placed in front of the target origin so that some targets cannot be hit at all. This has the effect of introducing noise into the generated training set.

Figures 3 and 4 are analogs to Figures 1 and 2 for the target problem. Figure 3 shows again both that training set quality increases with time and that NN generalization accuracy increases with training set quality. Further, Figure 4 reiterates the idea that confidence in training set quality increases with time of evolution and also that confidence in generalization accuracy increases with training set quality. Again, it is seen that as evolution time increases, the probability of producing a high quality training set also increases and that as the quality of the training set increases, the probability of good generalization accuracy increases as well.

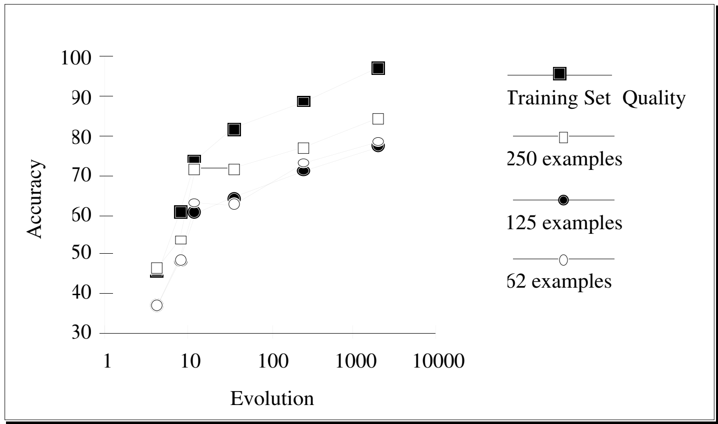

Figure 3. Training set quality and NN generalization accuracy for target problem

Another point of interest in both Figures 1 and 3 is the effect that training set quality vs. training set size has on NN generalization. Notice that with a 5% to 10% increase in training set quality the number of training examples required to maintain generalization accuracy is reduced by 50% to 75%. For example, in Figure 3, the fourth solid square from the left indicates a training set quality of 0.820. The resulting NN generalization accuracy on the entire training set (250 examples) indicated by the fourth hollow square is 0.718. However, when the training set accuracy is increased to 0.891 (the next solid square to the right), the fifth hollow circle from the left shows that only one quarter of the training set (62 examples) is required to maintain that generalization accuracy (actually it increases slightly to 0.736).

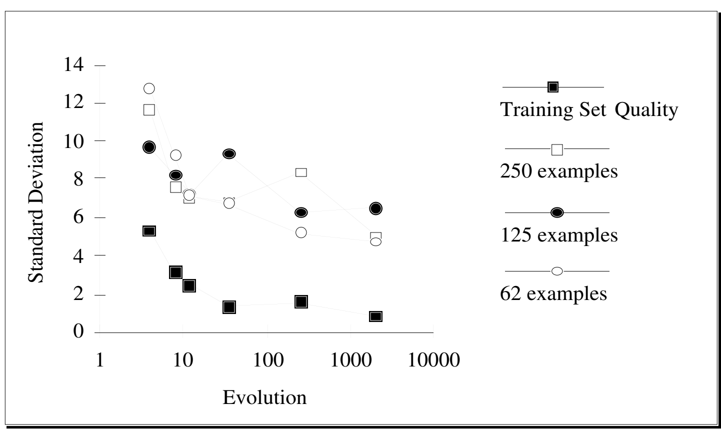

Figure 4. Standard deviation of training set quality and generalization accuracy for target problem

One difference between Figures 1 and 2 and Figures 3 and 4 is the stratification of training set quality from NN generalization accuracy. This is readily explained by the previously mentioned noise that is inherent in these training sets. Thus, even though the training set quality can be extremely high, NN generalization suffers somewhat because noise exists in the training set. Nevertheless, the principle of longer evolution producing a good training set that results in good NN generalization is still very much in evidence.

4.3. Training set quality vs. training set size

The effects of quality vs. size were further explored in this final set of simulations. The highest quality (0.978 average) training sets generated in the target problem were altered by removing the noisy instances and were then randomly and systematically reduced in size. This was done to investigate the value of individual examples for NN generalization. Figure 5 shows accuracy of generalization vs. number of training examples. With only 50 instances in the training set generalization accuracy is at 0.80. Since the number of possible targets in this problem is $35^2=1225$, this performance is attained after seeing only 4% of the possible targets. Another way of looking at this is to consider the generated training set as a set of exemplars. We have attained 80% of optimum performance while reducing the problem representation by 25 times.

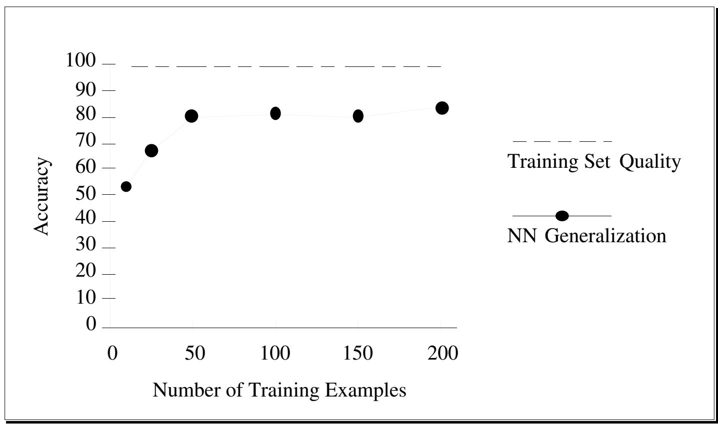

Figure 5. Generalization accuracy vs. number of training examples for target problem

Finally, Figure 6 is included to show the standard deviation in generalization accuracy as opposed to training set size. Even though standard deviation is somewhat higher with a training set size of only 50, it still indicates a confidence in the generalization ability with a training set of that size. Increasing to 100 examples lowers the standard deviation to 5%, indicating that even if the training set is as small as 8% of the total number of possible targets, 95% of the time generalization accuracy will be at least 70% of optimum.

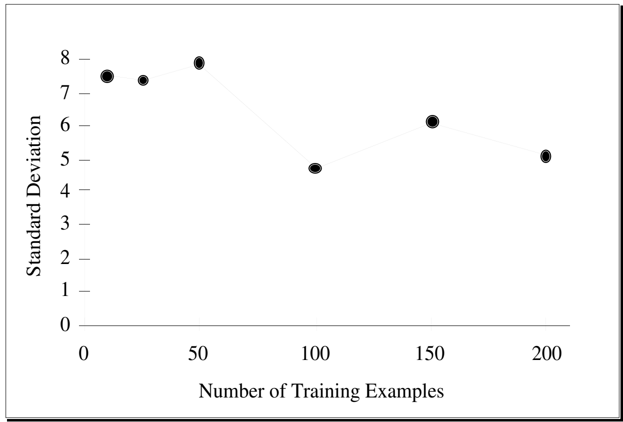

Figure 6. Standard deviation of accuracy vs. number of training examples for target problem

5. Conclusion and Future Work

The results in this paper are an assimilation of data collected over the course of hundreds of EC and NN simulations. They provide empirical evidence that

1) Evolutionary computation produces training sets of good quality.
2) The longer the evolution, the greater the confidence in the training set quality.
3) Training set quality correlates with NN generalization.
4) The higher the training set quality, the greater the confidence in the NN generalization.
5) Training set quality has a greater effect on NN generalization than does training set size.

Current research focuses on developing a theoretical basis for the empirical results discussed in this paper. Also, application of Evolutionary Training Sets to real world problems (such as real-time network control) is necessary to further validate this optimization technique.

References

[1] Caudell, T. P. and C. P. Dolan, "Parametric Connectivity: Training of Constrained Networks using Genetic Algorithms", *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[2] Falhman, S. E. and F. Lebiere, "The Cascade-Correlation Learning Architecture", *Advances in Neural Information Processing 2*, D. S. Touretzky (ed.), Morgan Kaufmann, 1990.

[3] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing, 1989.

[4] Harp, S. A., T. Samad and A. Guha, "Designing Application-Specific Neural Networks Using the Genetic Algorithm", *NIPS-89 Proceedings*, 1990.

[5] McClelland, James L. and David E. Rumelhart, *Explorations in Parallel Distributed Processing*, MIT Press, Cambridge, Massachusetts, 1988.

[6] Montana, D. J. and L. Davis, “Training Feedforward Neural Networks Using Genetic Algorithms”, *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[7] Romaniuk, Steve G., “Evolutionary Growth Perceptrons”, *Genetic Algorithms: 5th International Conference (ICGA-93)*, S. Forrest (ed.), Morgan Kaufmann, 1993.

[8] Spears, W. M., K. A. Dejong, T. Baeck, D. Fogel and H. de Garis, “An Overview of Evolutionary Computation”, *European Conference on Machine Learning (ECML-93)*, 1993.

[9] Ventura, Dan, Tim Andersen and Tony Martinez, “Using Evolutionary Computation to Generate Training Set Data for Neural Networks”, *Proceedings of the International Conference on Neural Networks and Genetic Algorithms*, pp. 468-471, 1995.

[10] Ventura, Dan, and Tony Martinez, “A General Evolutionary/Neural Hybrid Approach to Learning Optimization Problems”, submitted to the *World Congress on Neural Networks*, 1996.

Chapter 7

Optimal Control Using a Neural/Evolutionary Hybrid System

To suppose that the eye...could have been formed by natural selection seems, I confess, absurd in the highest degree.
-- Charles Darwin

Proceedings of the International Joint Conference on Neural Networks, Anchorage, Alaska, pp. 1036-1041 1998.

Early work on this subject was published in:
Proceedings of the International Workshop on Neural Networks and Neurocontrol, Playa del Carmen, Mexico, 1997.

Abstract

One of the biggest hurdles to developing neurocontrollers is the difficulty in establishing good training data for the neural network. We propose a hybrid approach to the development of neurocontrollers that employs both evolutionary computation (EC) and neural networks (NN). EC is used to discover appropriate control actions for specific plant states. The survivors of the evolutionary process are used to construct a training set for the NN. The NN learns the training set, is able to generalize to new plant states, and is then used for neurocontrol. Thus the EC/NN approach combines the broad, parallel search of EC with the rapid execution and generalization of NN to produce a viable solution to the control problem. This paper presents the EC/NN hybrid and demonstrates its utility in developing a neurocontroller that demonstrates stability, generalization, and optimality.

1. Introduction

Any system developed for control of an external plant should possess certain characteristics. For instance, the system should prevent the plant state from oscillating or "pinging", maintaining instead a smooth, stable trajectory for all plant state variables. Also, the control system will hopefully generalize not only to novel plant states, but also to reasonable changes in plant parameters. Finally, the controller should faithfully track the desired trajectory of plant output variables, maintaining plant operation at or near optimal levels according to some metric for performance level. To summarize, any controller should possess the following characteristics:

- Stability,
- Generality,
- Optimality.

Although neural networks (NN) possess great potential for the control of complex systems, the field of neurocontrol faces difficult problems as well. One of the most difficult involves proper training of NN for control of complex systems, which is a complicated endeavor when the system to be controlled is open-loop unstable. This paper extends and improves a method first proposed in [12] for combining evolutionary computation (EC) [3] with NN to control such a system. The broad, parallel search capabilities of the EC are employed to find regions of the system state space that are stable, and the survivors of the evolution constitute the training set used to train the NN for use as a neurocontroller. The goal of this paper is to show that controllers developed using this method do exhibit the desired characteristics listed above.

Combination of NN and EC technology is becoming more prevalent and usually focuses on using EC to develop the architecture (the weights, the topology, or both) of the NN. For example see [2], [4], [9], and [13]. On the other hand, the work presented here presupposes some NN architecture and focuses on using EC to develop a training set suitable for training the NN. It extends previous work done by the authors in which the combination of EC and NN was used for optimization. Though much less common, some work similar in flavor to this approach does exist including [5], [6], and [8].

Section 2 describes the problem of training an NN for system control and Section 3 then describes the hybrid EC/NN approach that is the main contribution of this paper. Section 4 discusses applying this approach to the well-known pole balancing problem and demonstrates empirically that the controller developed possesses the desired characteristics of stability, generality, and optimality. Finally, Section 5 provides conclusions and directions for future work.

2. Problem Description

Given a plant, Θ, the state of Θ may be described at time t by a vector of status variables, $\boldsymbol{s}^t$. Control of the plant is effected by Γ which applies a control vector, $\boldsymbol{c}$, to Θ. That is, given a plant state at time t described by vector $\boldsymbol{s}^t$, the setting of the values of the vector $\boldsymbol{c}$ will result in a different plant state at time $t+\delta$ described by the vector $\boldsymbol{s}^{t+\delta}$. The problem is how should Γ be constructed so that given a status vector, $\boldsymbol{s}^t$, Γ outputs a control vector $\boldsymbol{c}$ such that $\boldsymbol{s}^{t+\delta}$ describes a better plant state, if possible, than $\boldsymbol{s}^t$? We assume the existence of some evaluation function, f, that will determine whether or not one plant state is better than another. The operation of Θ may be either continuous or discrete and Γ has no information about the internal dynamics of Θ. The only information about Θ available to Γ is the value of $\boldsymbol{s}^t$. Given $\boldsymbol{s}^t$, Γ is expected to output values for $\boldsymbol{c}$, the goal being to maximize f for any given instance (state) of Θ. In earlier papers [10] [11], single iteration, open-loop optimization type problems have been considered.

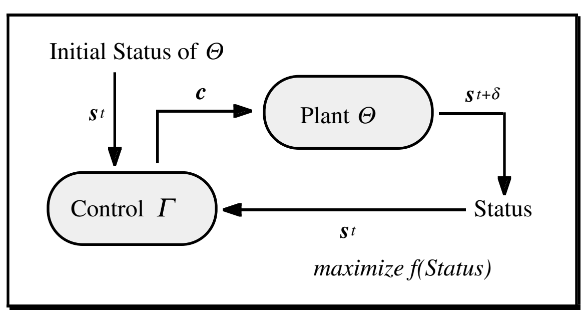

Figure 1. Control system with feedback

Here, we address the problem of continuous, closed loop control using feedback (see Figure 1). We assume that the plant Θ to be controlled is open-loop unstable, and the goal is to develop a neurocontroller for the Γ component of the system that will optimize (in some sense) the operation of the plant Θ. Since it is assumed that the internal dynamics of the plant are unknown (if they are known, a good controller may be developed using conventional control theory), using a neural network that learns to control the plant using only the externally available information (plant state $\boldsymbol{s}^t$) is a promising option. However, we are still faced with a difficult problem. Namely, neural networks usually learn inductively by repeated exposure to preclassified examples. What examples should be used for training the neurocontroller? The following section details the use of evolutionary computation and "black box" access to the plant (that is, only the plant state $\boldsymbol{s}^t$ is available) to develop a training set for the neurocontroller.

3. Using Evolutionary Computation to Produce a Training Set

Assuming that the status and control variables are defined over even a modest range, it is obvious that the input (status) and output (control) spaces will be extremely large. Evolutionary computation lends itself well to the exploration of large spaces. However, such evolutionary exploration is often slow. If we assume that the mapping $\boldsymbol{s} \rightarrow \boldsymbol{c}$ is nonrandom and in some sense generalizeable, then a representative set of points may be discovered via EC, and those points can then be used to train a NN which may then generalize over the rest of the function. The goal of the EC/NN synergism is to obtain the accuracy of evolutionary search and the speed of neural execution.

From the space spanned by $\boldsymbol{s}$ that describes Θ we choose a representative set of plant states by choosing n initial status vectors. We denote these $\boldsymbol{s}_i^{t=0}$, $0<i\leq n$ and refer to the plant described by state $\boldsymbol{s}_i^{t=0}$ as $\Theta_i^{t=0}$, $0<i\leq n$. These choices could be random or could be biased by any *a priori* heuristics as to what constitutes a realistic or desirable plant state (for example, see Section 4). The goal is to know, given one of these $\boldsymbol{s}_i^{t=0}$, what a "good" $\boldsymbol{c}$ vector would be. For each of the $\boldsymbol{s}_i^{t=0}$, EC is used to discover such a $\boldsymbol{c}$ in the following way.

Assume a fitness function f that takes as input a status vector $\boldsymbol{s}$ and returns a real-valued fitness measure. Now for each $\boldsymbol{s}_i^{t=0}$, randomly initialize a population of m control vectors, denoted $\boldsymbol{c}_k$, $0<k\leq m$. Evaluate the initial population by simulating the workings of $\Theta_i^{t=0}$ for δ time steps for each $\boldsymbol{c}_k$ and then applying fitness function f to $\boldsymbol{s}_i^{t=\delta}$. Next, until some stopping criterion is reached (a maximum number of generations or a threshold fitness value is achieved, for example), choose parents and use genetic operators (crossover, mutation, etc.) to produce m offspring, evaluate the children and select m survivors from amongst the parents and children. Finally, for each of the n populations, choose the individual, $\boldsymbol{c}_{max_i}$, (the individual with the highest fitness) and build a set of n training examples of the form $\boldsymbol{s}_i^{t=0} \rightarrow \boldsymbol{c}_{max_i}$. The algorithm is summarized in Figure 2.

evolve ()
  generate $\boldsymbol{s}_i$
  for each $\boldsymbol{s}_i^{t=0}$
    initialize population $\boldsymbol{c}_k$, $0<k\leq m$.
    evaluate($\boldsymbol{c}_k$,Θ_i)
    until(done)
      select parents from $\boldsymbol{c}_k$
      apply genetic operators to parents
      evaluate(children,Θ_i)
      $\boldsymbol{c}_k \leftarrow$ choose m survivors

evaluate($\boldsymbol{c}$,Θ_i)
  run $\Theta_i^{t=0}$ for δ time steps with control $\boldsymbol{c}$
  return $f(\boldsymbol{s}_i^{t=\delta})$

Figure 2. Algorithm for evolving training set

The EC has now found "good" approximate solutions for n points from the input (status) space but can say nothing about any other points, many of which are likely to be encountered during normal execution of Θ. Obviously, one solution to the problem defined in Section 2 would be to employ the evolutionary scheme discussed above as the control Γ. However, this would likely be unacceptable in terms of execution speed. Therefore, the NN is employed to generalize over the entire space defined by $\boldsymbol{s}$ using the relatively small set of approximate solutions as a training set. While the initial training of the network maybe somewhat time consuming, depending on the network and training algorithm employed, the generalization during execution will be extremely fast. The synergistic combination of EC and NN is then employed as the control Γ as in Figure 3.

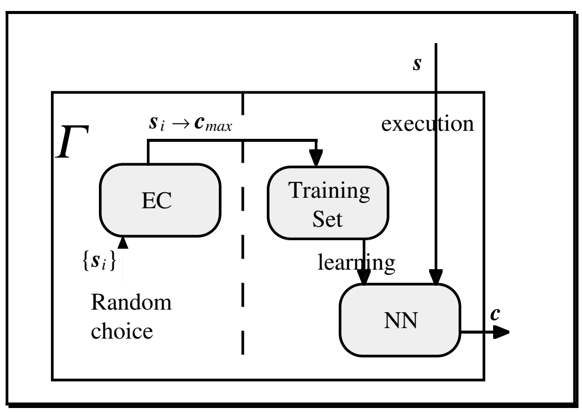

Figure 3. The control Γ

It should be noted that since many (if not most) interesting control problems involve plants that are open-loop unstable, care must be taken in the choice of the parameter δ. On the one hand, if δ is too short, the effect of applying the control to the system will not be readily apparent. On the other hand, if δ is too long, the instability of the plant will have destroyed any useful measure of how good the control vector was.

The power of this EC/NN hybrid approach is its combination of the thoroughness of evolutionary search with the speed and accuracy of neural generalization. Further, it is generally applicable to any control problem for which a fitness function can be found and for which "blackbox" access to the system to be learned (or to a reasonable simulation) is possible. In order to provide proof-of-concept, the next section discusses using the EC/NN approach to solve one such control problem.

4. The Pole Balancing Problem -- An Example

The pole balancing problem is a well known, textbook example of a complex control system. The problem exists in many variations, but perhaps the most common consists of a pole attached by a hinge to a wheeled cart that sits in a short track. The challenge is to apply force to the cart in order to keep the pole upright and at the same time not run the cart into the side of the track (since that would of course cause the pole to fall). Figure 4 gives a simple diagrammatic representation.

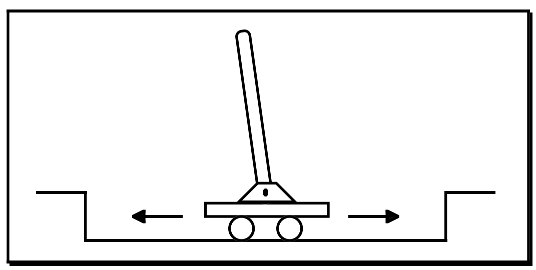

Figure 4. The pole balancing problem

The system is inherently unstable and the solution is, of course, a gentle oscillatory application of force in first one direction and then the other. Other variations on this theme include infinite length tracks, multi-jointed poles, and multiple poles on the cart.

This system was simulated using both Euler and Runge-Kutta approximations (with the integration time step equal to 0.005 seconds) for solving the second order differential equations that describe the pole/cart system. For this simple problem, the physics is well understood and the equations represent an extremely accurate simulation of the cart and pole motion, including accounting for friction between the pole and the cart and between the cart and the track (see Appendix A for details). A training set of 200 instances of the form $\left(\theta, \dot{\theta}, \ddot{\theta}\right) \rightarrow F$ was generated using the evolutionary method described in Section 3. The length of time, δ, allowed for system stabilization during the evolutionary evaluation was set to 3 time steps (0.015 seconds). As noted earlier, during training set generation, any knowledge of which plant states are more likely or more useful (stable) should be incorporated into the evolutionary process in order to concentrate on exploring those parts of the state-space that will be most helpful for neural network generalization. In the case of the pole balancer, we are most interested in the those states in which the pole angle and its derivatives are relatively small (since once any of these variables become too large the pole will fall for sure due to the system limitations on available force). Once the training set was generated, it was used to train a simple backpropagation [7] network with a single hidden node for 1000 epochs. The learning rate was set to 0.5, and a momentum term with a coefficient of 0.95 was also employed. To test the controller, the pole was given several initial non-zero angles and the system was expected to balance the pole. Tests were performed over varying time increments from 30 seconds up to 15 minutes. In all cases, once the controller stabilized the plant, there was no significant deviation off optimal, deadbeat control. Figures 5 and 6 show the first 10 seconds of the trajectory of the pole's angular vertical displacement, θ, for two such tests (initially θ = 0.4 radians ≈ 23° and θ = -0.78 radians ≈ 45° respectively).

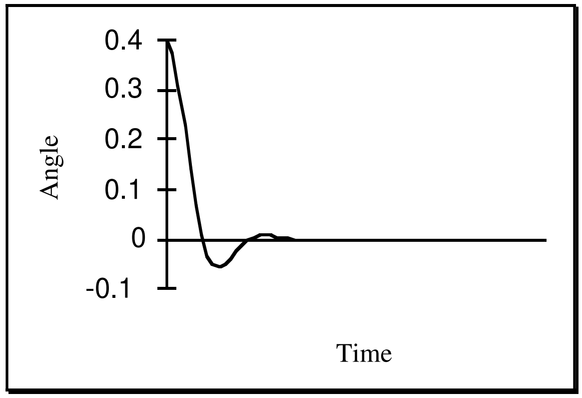

Figure 5. θ-trajectory over 10 seconds for initial pole angle of θ = 0.4 radians ≈ 23°

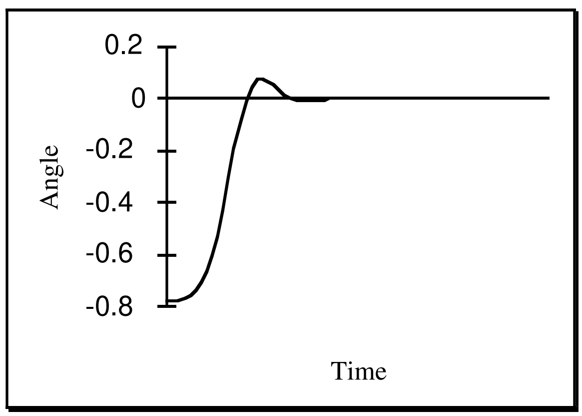

Figure 6. θ-trajectory over 10 seconds for initial pole angle of θ = -0.78 radians ≈ -45°

Both tests demonstrate nicely the controller's ability to stabilize the plant. Also, not only is the plant stable, but also it can in several senses be considered to be optimal. First, the number of initial oscillations required to stabilize is just three, the minimum for stabilizing a second order system. Second, the amplitude of those oscillations is very small (maximum overshoot of 4.3 degrees for the -45 degree case, for example). Third, once the plant is stable, it maintains "deadbeat" control right on the desired trajectory of θ = 0°. Finally, since neither of these initial plant states was encountered during training, the control system demonstrates an ability to generalize to previously unseen plant states. To further test the system's generalization ability, the plant parameter corresponding to length of pole (see Appendix A) was changed *after* training. In other words, the control was trained with a pole of one length (1 meter) and tested with poles of other lengths. Thus the control is required to generalize not just to novel plant states but to a plant that is different from the one it was trained to control. Figure 7 shows the first 20 seconds of the θ-trajectory for a 4 meter pole with an initial angle of θ = -0.78 radians ≈ 45°.

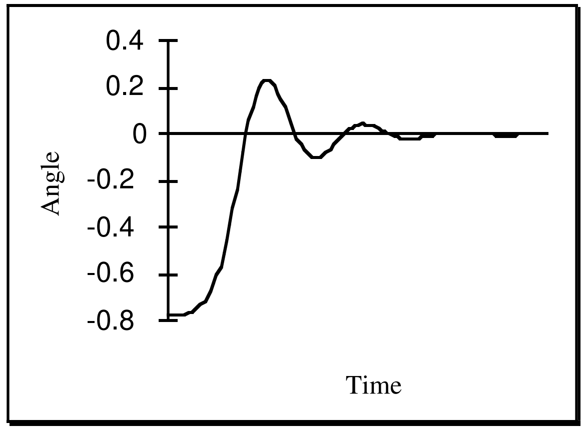

Figure 7. θ-trajectory for 4 meter pole with initial pole angle of θ = -0.78 radians ≈ -45°

Notice that the controller is still stable, though the response is not as good. This is, of course, to be expected, but the important point is that a controller trained on one plant is able to control another. Other experiments were conducted altering pole mass, both pole mass and pole length together, and the gravitational constant, etc. (see Appendix B). In the case of altering the gravitational constant, the controller performed very well from 0.5g up to 1.25g, with little drop-off in response. In other cases, some performance degradation is observed (as in Figure 7) as expected. Finally it should be mentioned that because of the architecture of the EC/NN hybrid, it is possible to continue generating training instances in the background and to then improve the controller by on-line learning. In fact, because of the existence of the fitness function, the controller could monitor itself and note the plant states for which its performance could be improved. The controller could then request the EC to generate training instances from regions of the state space near these plant states.

5. Conclusions

A new evolutionary/neural hybrid approach for the development of neurocontrol has been demonstrated. In particular, it has been shown how evolutionary computation and "black box" access to the plant can be used to generate a suitable training set for training a neural network to act as a controller of an inherently unstable system. Further, the resulting controller has been demonstrated to possess the desirable characteristics of

- Stability,
- Generalization, and
- Optimality.

Because many interesting control problems exist for which the dynamics of the plant are unknown, neural networks offer a viable approach to approximating desired control of a plant. However, because many plants are open-loop unstable, training the neural network for control is difficult. The use of evolutionary computation, along with a judicious choice of the stabilization parameter δ facilitates the development of neurocontrol for unstable systems. Currently, we are applying the methods described here to the real world problem of controlling the combination of engine and airframe for a high performance military aircraft. Future work includes quantifying the choice of δ, theoretical analysis and quantification of stability limits, generalization capabilities and optimality results, and combining these techniques with those that evolve the network architecture.

Acknowledgments

The authors would like to thank George Lendaris and Tad Shannon of Portland State University for many insightful comments and helpful suggestions, as well as for providing the code for the pole- cart simulation.

References

[1] Barto, Andrew G., Richard S. Sutton and Charles W. Anderson, "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. smc-13, no. 5, September/October, 1983.

[2] Caudill, Maureen, "Evolutionary Neural Networks", *AI Expert*, vol. 6, no. 3, pp. 28-33, March 1991.

[3] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing, 1989.

[4] Harp, S. A., T. Samad and A. Guha, "Designing Application-Specific Neural Networks Using the Genetic Algorithm", *NIPS-89 Proceedings*, 1990.

[5] Miagkikh, V. V., A. P. Topchy and O. A. Lebedko, "Fast Learning in Multilayered Networks by means of Hybrid Evolutionary and Gradient Algorithms", *Proceedings of the International Conference on Evolutionary Computation and it Applications*, pp. 390-8, June 1996.

[6] Montana, D. J. and L. Davis, "Training Feedforward Neural Networks Using Genetic Algorithms", *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[7] Rumelhart, David E., James L. McClelland and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, 1986.

[8] Romaniuk, Steve G., "Evolutionary Growth Perceptrons", *Genetic Algorithms: 5th International Conference (ICGA-93)*, S. Forrest (ed.), Morgan Kaufmannn, 1993.

[9] Saunders, Gregory M., Peter J. Angeline and Jordan B. Pollack "Structural and Behavioral Evolution of Recurrent Neural Networks", *Advances in Neural Information Processing Systems*, vol. 6, pp. 88-95, Morgan Kaufmannn Publishers Inc., 1994.

[10] Ventura, Dan and Tony Martinez, "Robust Optimization Using Training Set Evolution", *Proceedings of the International Conference on Neural Networks*, pp. 524-8, 1996.

[11] Ventura, Dan and Tony Martinez, "A General Evolutionary/Neural Hybrid Approach to Learning Optimization Problems", *Proceedings of the World Congress on Neural Networks*, pp. 1091-5, 1996.

[12] Ventura, Dan and Tony Martinez, "Using Evolutionary Computation to Generate Training Set Data for Neural Networks", *Proceedings of the International Conference on Artifical Neural Networks and Genetic Algorithms*, pp. 468-71, 1995.

[13] Wieland, Alexis P., "Evolving Controls for Unstable Systems", *Proceedings of the 1990 Connectionist Models Summer School*, pp. 91-102, 1990.

Appendix A: Dynamics of the pole balancing problem

The nonlinear differential equations describing the pole balancing system used in the simulations are taken from [1] and are as follows.

$$\ddot{x} = \frac{F + ml\left[\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta\right] - \mu_c \operatorname{sgn}(\dot{x})}{m_c + m}$$

$$\ddot{\theta} = \frac{g\sin\theta + \cos\theta\left[\dfrac{-F - ml\dot{\theta}^2 \sin\theta + \mu_c \operatorname{sgn}(\dot{x})}{m_c + m}\right] - \dfrac{\mu_p \dot{\theta}}{ml}}{l\left[\dfrac{4}{3} - \dfrac{m\cos^2\theta}{m_c + m}\right]}$$

Here x is the horizontal position of the cart, θ is the angle of the pole off of vertical, F is the force applied to the cart, m_c and m are the masses of the cart and pole respectively, l is the half length of the pole, μ_c and μ_p are the friction coefficients for the cart and pole respectively, and g is the gravitational constant. Table 1 gives the values used for the variables during the simulations.

Table 1. Values for the pole balancing simulation

Variable	Meaning	Value
x	horizontal cart position	± 10 m
θ	angle of pole off vertical	± 1.57 rad
F	force applied to cart	± 50 N
m_c	mass of cart	1 kg
m	mass of pole	.1 kg
l	half length of pole	.5 m
g	gravitational constant	-9.8 m/s^2
μ_c	friction of cart on track	.0005
μ_p	friction of pole on cart	.000002

Appendix B: Further experimental results

All results shown are for 10 seconds. Figure 8 shows results of using only the angle and its velocity (and not the angular acceleration) as inputs to the controller. Figure 9 shows the result of changing the gravitational constant to 0.5g, and Figure 10 shows the result for a pole with 4 times more massive than the pole used during training (0.4 kg for testing but 0.1 kg for training).

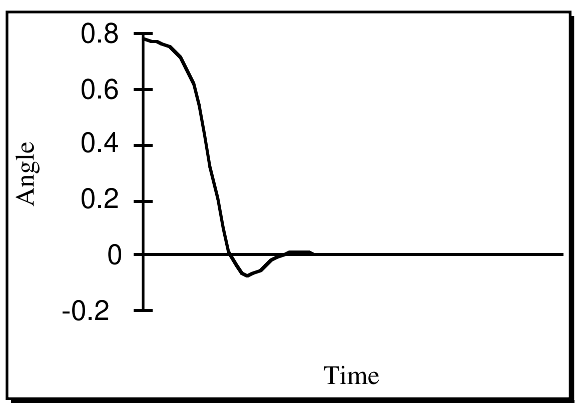

Figure 8. θ-trajectory using only 2 inputs with initial pole angle of $\theta = 0.78$ radians $\approx 45°$

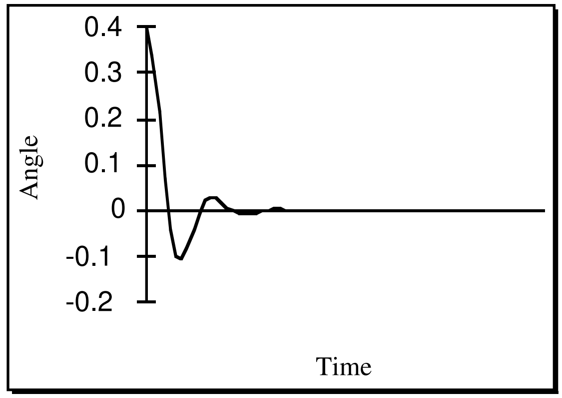

Figure 9. **θ-trajectory for 0.5g with initial pole angle of θ = 0.4 radians ≈ 23°**

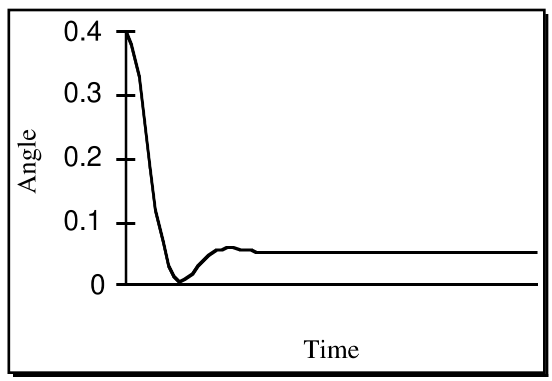

Figure 10. **θ-trajectory for 0.4 kg pole with initial pole angle of θ = 0.4 radians ≈ 23°**

Part III

Quantum Computational Learning

...And out of small things proceedeth that which is great.
-- D&C 64:33

This part of the dissertation may be considered the most important research direction and the most significant contribution to computational learning. It presents an entirely new approach to the problem, introducing, in essence, the field of quantum computational learning. The counterintuitive properties of quantum mechanical systems seem to provide, in some contexts, a possibility of defeating the "curse of dimensionality" not by circumventing the issue as the work of Part II (and in fact most of classical computational learning theory attempts to do) but rather by embracing it. Quantum computation provides us the ability to do the seemingly absurd: process an exponential amount of information in polynomial time. If this capability can be properly harnessed, then extremely powerful and unique computational learning mechanisms become a possibility. The work described in Part III is one of the first comprehensive attempts at a union of quantum computation with computational learning.

Chapter 8 is an initial, rather naive attempt at combining ideas from quantum mechanics with those from traditional feedforward neural networks, in particular the classic perceptron. Since the learning of the perceptron is encoded in its weights, it is natural to suggest that the most likely part of the neuron for useful extension to the quantum domain is the weight vector. This chapter considers creating a neuron with weights that exist in a quantum superposition and suggests a simple model for its implementation. The reference is as follows:

Ventura, Dan and Tony Martinez, "An Artificial Neuron with Quantum Mechanical Properties", *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, Norwich, England, 1997.

Chapter 9 looks at combining quantum ideas with ideas from another type of classical neural network, the associative memory. In particular, a particularly weak aspect of classical associative memories is their limited capacity. The fact that quantum systems exist in exponentially large spaces suggests that representing patterns as a quantum superposition may present a solution to the capacity problem. This chapter describes a quantum associative memory with a capacity that is exponential in the number of neurons -- a significant improvement over classical models. The ideas here are somewhat further developed than those of Chapter 8, but are still shy of the eventual goal of a true quantum computational learning system. This chapter may be referenced as follows:

Ventura, Dan and Tony Martinez, "Quantum Associative Memory with Exponential Capacity", to appear in the *Proceedings of the International Joint Conference on Neural Networks*, Anchorage, Alaska, pp. 509-513, 1998.

Chapter 10 provides a key contribution to the field of quantum computation itself by exhibiting for the first time a physically realizable, truly quantum algorithm for encoding training example information into a quantum system. As discussed in the introduction, this use of the term "physically realizable" does not imply work here on a physical implementation; rather, this work ensures that such an implementation is possible. Other quantum algorithms assume *a priori* knowledge of a function rather than knowledge of a relatively few sample points. Therefore, the result of Chapter 10 is significant because it provides a way of encoding example data and thus allows the development of useful quantum computational learning algorithms. Because the work described in this chapter describes a fundamental physical process, it has been submitted to a leading physics journal under the following reference:

Ventura, Dan and Tony Martinez, "Initializing the Amplitude Distribution of a Quantum State", submitted to *Physical Review Letters*, June 1998.

With the work of Chapter 10 as a foundation, Chapter 11 provides a mature approach to the idea, first introduced in Chapter 9, of a quantum associative memory. The ideas in Chapter 9, while important theoretically, are not directly implementable in a physical quantum system. In contrast, the quantum associative memory of Chapter 11 is based on actual quantum algorithms -- the algorithm of Chapter 10 for pattern storage and Grover's quantum search algorithm for pattern recall -- both of which are physically realizable. The initial idea of using Grover's algorithm as the basis for an associative recall mechanism was first introduced in the following paper:

Ventura, Dan and Tony Martinez, "A Quantum Associative Memory Based on Grover's Algorithm", submitted to the *International Conference on Artificial Neural Networks and Genetic Algorithms*, Portoroz, Slovenia, April 1999.

Chapter 11 builds on the theoretical ideas of Chapter 10 as well as on the foundational work in the reference above to produce a system that is physically realizable. In addition, good theoretical analysis of the system's capabilities is provided. This chapter has been submitted to the following leading neural networks journal:

Ventura, Dan and Tony Martinez, "Quantum Associative Memory", submitted to *IEEE Transactions on Neural Networks*, June 1998.

Finally, Chapter 12 considers a combination of quantum computation with classical machine learning. It considers the problem of learning functions that can be described in disjunctive normal form (DNF), and the result is a quantum computational learning algorithm that uses a Fourier-based mechanism to learn a function from a set of random examples. This compares favorably with the best classical algorithm for learning DNF, which requires the ability to choose which examples it sees. This chapter has been submitted for review in a leading machine learning journal under the following reference:

Ventura, Dan and Tony Martinez, "A Quantum Computational Learning Algorithm", submitted to *Journal of Artificial Intelligence Research*, July 1998.

As with the previous group of chapters, the chapters in Part III duplicate some introductory material. First of all, because it is assumed that the majority of readers will not be very familiar with the basic ideas of quantum mechanics, each paper contains a section (Section 2 in Chapters 8-11 and Section 3 in Chapter 12) that covers in varying degrees some concepts necessary for the discussion that follows. Therefore, the reader that is familiar with the basics of quantum mechanics is advised to skip these sections. On the other hand if a basic review is required, the most comprehensive introduction is given in Chapter 11, so 11.2 should be read before anything else and then the other chapters may be addressed in order, skipping Section 2 (Section 3 for Chapter 12) thereafter. Some redundancy is unavoidable in the introductory sections as well, but these should perhaps still be perused in order to understand the proper setting for each chapter. Finally, if Chapter 10 has been read, 11.3.2 should be skipped as it is just a review of the basic ideas of Chapter 10 that will be used in the quantum associative algorithm developed in Chapter 11.

Chapter 8

An Artificial Neuron with Quantum Mechanical Properties

The most dangerous thing in the world is to try to leap a chasm in 2 jumps.

-- David Lloyd George

Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, Norwich, England, 1997.

Abstract

Quantum computation uses microscopic quantum level effects to perform computational tasks and has produced results that in some cases are exponentially faster than their classical counterparts. Choosing the best weights for a neural network is a time consuming problem that makes the harnessing of this "quantum parallelism" appealing. This paper briefly covers necessary high-level quantum theory and introduces a model for a quantum neuron.

1. Introduction

The field of artificial neural networks has at least two important goals: (A) creation of powerful artificial problem solving systems and (B) furthering understanding of biological neural networks including the human brain. Much effort has been made in both areas and some progress has been realized. The field of quantum computation [1], which has been completely unrelated to that of neural networks until very recently, applies ideas from quantum mechanics to the study of computation and has made interesting progress. Most notably, quantum algorithms for prime factorization and discrete logarithms have recently been discovered that provide exponential improvement over the best known classical methods [2]. Recently some work has been done in the area of combining classical artificial neural networks with ideas from the field of quantum mechanics in pursuit of goal (A).

It is the purpose of this paper to further this pursuit of goal (A), that is, to show the usefulness of some ideas from the field of quantum mechanics (in particular those of linear superposition and coherence/decoherence) to that of artificial neural networks in order to improve neural networks' abilities as problem solving systems. Our approach is to introduce a mathematical model of an artificial neuron with quantum mechanical properties that allow it to discriminate linearly inseparable problems using only linear thresholding as its activation function.

2. Some Quantum Ideas

Quantum mechanics is in many ways extremely counterintuitive and yet it has provided us with perhaps the most accurate theory (in terms of predicting experimental results) ever devised by science. The theory is well established and is covered in its basic form by many textbooks (see for example [3]). Several ideas from this theory that are necessary for the following presentation must be briefly mentioned.

Linear superposition is closely related to the familiar mathematical principle of linear combination. For example, in a vector space with bases $\boldsymbol{x}$ and $\boldsymbol{y}$, any vector $\boldsymbol{v}$ can be defined as $\boldsymbol{v} = a\boldsymbol{x} + b\boldsymbol{y}$. In some sense $\boldsymbol{v}$ can be thought of as being both $\boldsymbol{x}$ and $\boldsymbol{y}$ at the same time. In quantum mechanics, this principle actually applies to physical variables in physical systems. The vector space is generalized to a Hilbert space whose bases are the classical values normally associated with the system. For example, the position of an electron orbiting a nucleus is usually a superposition of all possible positions in 3-d space, where each possible position is a basis state for the Hilbert space and has a finite probability of being the actual position. One of the most counterintuitive aspects of quantum theory is this -- at the quantum or microscopic level, the electron is not in any one position in an orbit, but it is in a superposition of all of them at once. In some sense it is in all positions at the same time. However, at the macroscopic or classical level, the location of the electron is a single definite position in 3-d space. This apparent contradiction is still not fully understood, but it is explained as follows. A quantum mechanical system remains in a superposition of its basis states until it interacts with its environment.

Coherence/decoherence is related to the idea of linear superposition. A quantum system is said to be *coherent* if it is in a linear superposition of its basis states. As mentioned above, a result of quantum mechanics is that if a system that is in a linear superposition of states is observed or interacts in any way with its environment, it must instantaneously choose one of those states and "collapse" into that state and that state only. This collapse is called *decoherence* and is governed by the wave function Ψ. Just as the basis states of the Hilbert space have a physical interpretation (as the classical values associated with a system), so too does the amplitude of the wave function Ψ describing the system. This wave function actually represents the probability amplitudes (in the complex domain) of all bases (possible positions, etc.) such that the probability for a given basis (position, etc.) is given by $|\Psi|^2$.

Operators are a mathematical formalism used to describe how one wave function is changed into another. They are analogous to matrices operating on vectors. Using operators, an eigenvalue equation can be written $\hat{A}\phi_i = a\phi_i$. The solutions ϕ_i to such an equation are called eigenstates and are the basis states of a Hilbert space. In the quantum formalism, all properties (position, momentum, spin, etc.) are represented as operators whose eigenstates are the classical values normally associated with that property.

3. Related Work

To date, very little has been done in combining the fields of quantum mechanics and neural networks. However, a few notable exceptions do exist. Perus has published an interesting set of mathematical analogies between the quantum formalism and neural network theory [4]. Menneer and Narayanan have proposed a model weakly inspired (their term) by the "many worlds" interpretation of quantum mechanics in which a network is trained for each instance in the training set and the final network is a superposition of these [5]. Finally, Behrman *et al.* have developed a novel approach to implementing a quantum neural network using quantum dots [6]. It uses a quantum dot for each input and the system is allowed to evolve quantum mechanically through time while being observed (and thus forced to decohere) at fixed time intervals. Interestingly, the different time slices act as the neurons in a hidden layer of a neural network -- the more time slices, the more hidden layer neurons. Perhaps most notably, Behrman *et al.* have actually implemented this quantum dot neural network and give results for its learning several two input Boolean functions.

4. Quantum Neuron

The simplest classical artificial neuron is the perceptron that takes as input n bipolar (or binary) values, $\{i_j\}$. It is defined as a weight vector $\boldsymbol{w} = (w_1, w_2, ..., w_n)^T$, a threshold θ and an output function f where

$$f = \begin{cases} 1 & \text{if } \sum_{j=1}^{n} w_j i_j > \theta \\ -1 & \text{otherwise} \end{cases}. \tag{1}$$

This neuron model is well understood, but it is mentioned for several reasons. First, it cannot solve problems that are not linearly separable, and second, though this classical perceptron is extremely simple and well understood, this will not be the case for its quantum counterpart and thus it is important to start with the very simplest concepts as quantum ideas are incorporated.

We now define a simple quantum analog to the perceptron, which also takes as input $\{i_j\}$. It is similar in all respects to the classic perceptron except that the single weight vector $\boldsymbol{w}$, is replaced by a wave function, $\Psi(\boldsymbol{w},t)$ in a Hilbert space whose basis states are the classical weight vectors. This wave function represents the probability amplitude (in general this is a complex wave as opposed to a real one) for all possible weight vectors in weight space together with the *normalization condition* that for any time t

$$\int_{-\infty}^{\infty} |\Psi|^2 \, d\boldsymbol{w} = 1. \tag{2}$$

Thus, the weight vector of the perceptron is replaced with a quantum superposition of many weight vectors which on interaction with its environment will decohere into one classical weight vector, according to the probabilities given by $|\Psi|^2$.

For example, consider the one-input, one-output bipolar function that inverts its input (NOT). For convenience the weights will be bounded such that

$$-\pi \leq w_j \leq \pi. \tag{3}$$

In order for a quantum neuron to learn this function, $\Psi(\boldsymbol{w},t)$ and θ must be found. We assume that Ψ is time-invariant and concentrate only on $\boldsymbol{w}$, which is in this case a one-dimensional vector. Because of the strict bounds given in (3), finding Ψ is equivalent to solving the one-dimensional rigid box problem common to most elementary treatments of quantum mechanics, whose solutions are of the form

$$\Psi(w_0) = A \sin\left(\frac{n\pi}{a} w_0\right). \tag{4}$$

Here A is a normalization constant that can be found using (2), $n = 1, 2, 3, ..., w_0$ is the single element of $\boldsymbol{w}$, and a is the width of the box (in general this is 2π, but the width may be altered to be smaller).

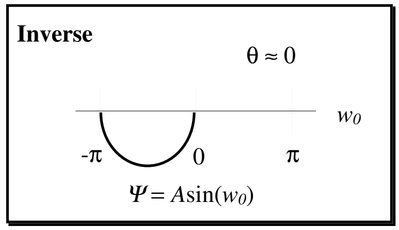

Figure 1. Solutions for one-dimensional NOT function

Figure 1 shows a graph of one Ψ for the one-input bipolar NOT function, and for comparison Figure 2 shows a solution for another one-input bipolar function, TRUE (the extra π term shifts the function, which is necessary since the general solution (4) assumes for convenience that the left hand side of the box occurs at 0). To understand what these graphs represent, it must be realized that they are graphs in weight space and that Ψ is the probability amplitude for a given weight vector $\boldsymbol{w}$. In a quantum neuron the weight vectors exist in coherent superposition of all possible classical weight vectors in weight space with non-zero probability amplitude.

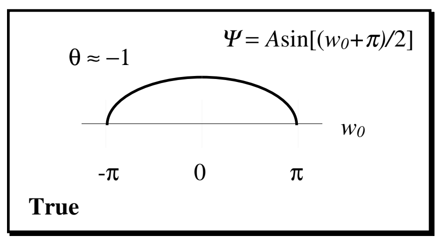

Figure 2. Solutions for one-dimensional TRUE function

When the superposition of weight vectors interacts with its environment (for example when it encounters an input) it must decohere into one basis state -- a classical weight vector within the bounds enforced in (3) -- and this decoherence occurs with probability governed by $|\Psi|^2$. Note that in the case of these simple functions, no matter what weight vector is chosen at decoherence, it will result in the correct output.

Now consider the more complicated two-input XOR problem, which is of course, not linearly separable. Using a similar argument for that of the one-dimensional case, it is not surprising to find that the solutions to problems with two-dimensional weight spaces are equivalent to those of the two dimensional rigid box

$$\Psi(w_0, w_1) = A\sin\left(\frac{n_{w0}\pi}{a}w_0\right)\sin\left(\frac{n_{w1}\pi}{a}w_1\right), \tag{5}$$

where the variables and constants have the same meanings as in (4) with the added note that now there are two different n variables, one for each weight.

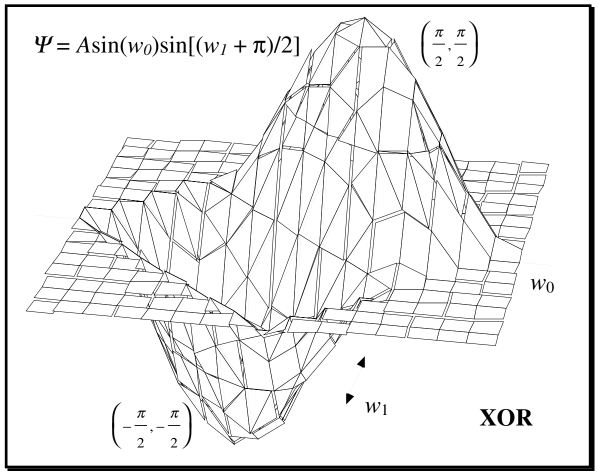

Figure 3. Plot of Ψ for the XOR problem

Figure 3 shows a contour plot of one such solution, rotated by 45°. Note again that this is a graph of Ψ, the probability amplitude, whereas the probability, $|\Psi|^2$, is what is really important.

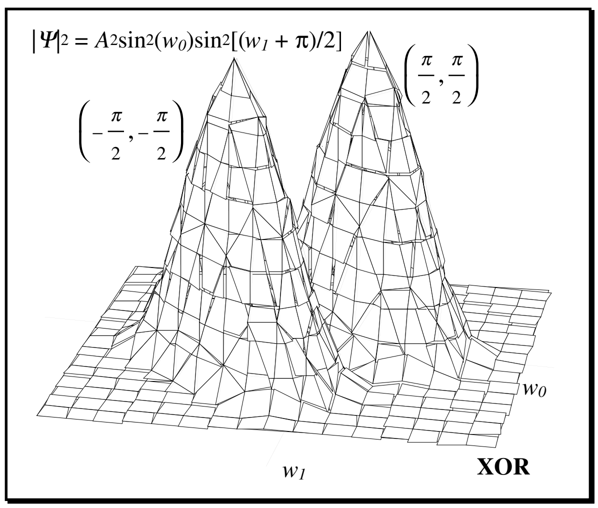

Figure 4. Plot of $|\Psi|^2$ for the XOR problem

Also notice that its contour plot (Figure 4) gives two solutions with maximal probability, at $(\pi/2, \pi/2)$ and at $(-\pi/2, -\pi/2)$. It is also important to note that the nodal line that exists at $w_0 = -w_1$ in Figure 3 ensures that no solutions exist on this line which must be the case in order to learn the function. Finally note that since the quantum neuron maintains a coherent superposition of weights, it can solve non-linearly separable problems.

Of course, these examples are simple and their solutions are simple. Even in the general case however, the solutions will always be equivalent to those of an *n* dimensional rigid box so that Ψ will always be of the general form introduced above. This need not be true, however, if other physical models, such as the non-rigid box, simple harmonic oscillator, hydrogen atom, etc. are be considered. Further, the time variable has not been incorporated into the equations. Another important topic is training of the neuron, which entails changing the wave function Ψ. Since Ψ is governed by the (time-independent for now) Schrödinger equation,

$$\nabla^2 \Psi = \frac{2m}{\hbar^2}[U - E]\Psi, \tag{6}$$

note that the only variable that may be changed is the potential, U. Therefore Ψ may be changed by changing U.

The n's introduced in (4) and (5) are termed quantum numbers and play an important role in quantum mechanics. Further investigation of their role with regards to quantum neurons is needed. A series of classification operators that ensures appropriate wave function decoherence is also required. For example, in the XOR problem, two of the possible input patterns, (-1,1) and (1,-1), are correctly classified using any of the superposed weight vectors; however, for the other two input patterns, (-1,-1) and (1,1), this is not the case. The necessary operators for the quantum neuron will be analogical to a matrix form of the input vector. The operator associated with the input vector (-1,-1) must greatly decrease the probability amplitude (and thus the probability) for all the negative weight vectors and correspondingly increase the amplitudes for the positive weight vectors. The operator associated with the input vector (1,1) must do the opposite.

5. Concluding Remarks

Though many of these topics are beyond the scope of this paper, it has been demonstrated that the application of quantum mechanical ideas to the field of neural networks is a fertile area for further research. This includes development of a learning algorithm for the quantum neuron, further investigation of the classification operators, and theoretical analysis of the quantum neuron's capabilities. Further, the idea of linear superposition may be applied not only to the weight vector of a neuron, but also to its inputs, its output, and its activation function, among other things. See [7] for an application to the problem of choosing useful features from the exponentially large set of possibilities. Also, other quantum mechanical concepts such as the quantum nature of energy, spin, momentum, etc. and EPR phenomenon may find application as this field is explored further.

References

[1] Brassard, Gilles, "New Trends in Quantum Computation", *13th Symposium on Theoretical Aspects of Computer Science*, 1996.

[2] Shor, Peter W., "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", to appear in the *SIAM Journal of Computing*, 1996.

[3] Taylor, John R. and Zafiratos, Chris D., *Modern Physics for Scientists and Engineers*, Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[4] Perus, Mitja, "Neuro-Quantum Parallelism in Brain-Mind and Computers", *Informatica*, vol. 20, pp. 173-183, 1996.

[5] Menneer, Tammy and Narayanan, Ajit, "Quantum-inspired Neural Networks", technical report R329, Department of Computer Science, University of Exeter, Exeter, United Kingdom, 1995.

[6] Behrman, E. C., Steck, J. E., and Skinner, S. R., "A Quantum Dot Neural Network", preprint submitted to *Physical Review Letters*.

[7] Ventura, Dan and Martinez, Tony, “Application of Quantum Mechanical Properties to Machine Learning”, submitted to the *International Conference on Machine Learning*, 1997.

Chapter 9

Quantum Associative Memory with Exponential Capacity

This isn't right. This isn't even wrong.
-- Wolfgang Pauli

Proceedings of the International Joint Conference on Neural Networks, Anchorage, Alaska, pp. 509-513, 1998.

Abstract

Quantum computation uses microscopic quantum level effects to perform computational tasks and has produced results that in some cases are exponentially faster than their classical counterparts by taking advantage of quantum parallelism. The unique characteristics of quantum theory may also be used to create a quantum associative memory with a capacity exponential in the number of neurons. This paper covers necessary high-level quantum mechanical ideas and introduces a simple quantum associative memory. Further, it provides discussion, empirical results and directions for future work.

1. Introduction

We consider the problem of associative pattern completion -- given a set of patterns of the form $\vec{x}_i \rightarrow \vec{o}_i$, learn to produce the vector $\vec{o}_i$ given the vector $\vec{x}_i$. The trivial solution is simply to store the set of patterns as a lookup table. The problem becomes interesting with the added constraint of finding as small a representation as possible. This paper proposes a quantum associative memory (QuAM) with a storage capacity of $O(2^n)$, while using only n neurons.

The field of quantum computation, which applies ideas from quantum mechanics to the study of computation, was introduced in the mid-1980s [6]. For a readable introduction to quantum computation see [3]; for a more rigorous treatment see for example [1]. The field is still in its infancy and very theoretical but offers exciting possibilities for the field of computer science -- the most notable to date being the discovery of quantum computational algorithms for computing discrete logarithms and prime factorization in polynomial time, two problems for which no known classical polynomial time solutions exist [11]. These algorithms provide theoretical proof not only that interesting computation can be performed at the quantum level but also that it may in some cases have distinct advantages over its classical cousin. The quantum computing approach to solving these two problems takes advantage of the unique features of quantum systems to process an exponential number of possibilities simultaneously, thus providing exponential speedup over classical approaches.

This paper presents a unique reformulation of the pattern classification problem into the language of wave functions and operators. This reformulation may be generalized to a very large class of computational learning problems, opening up the possibility of employing the capabilities of quantum computational systems for the solution of computational learning problems. Here we examine one such problem, that of pattern completion, and show how it may be handled in the new formalism. Further, it is shown that the possibility of utilizing quantum effects such as *coherence* and *interference* (see Section 2) may have great advantage over classical computational learning methods.

Artificial neural networks (ANN) seek to provide ways for classical computers to learn rather than to be programmed. If quantum computers become a reality, then artificial neural network methods that are amenable to and take advantage of quantum mechanical properties will become possible. Further, the possibility of processing an exponential space in polynomial time is very appealing. In particular, can quantum mechanical properties be applied to ANNs for problems such as associative memory? Recently, some work has been done in the area of combining classical artificial neural networks with ideas from the field of quantum mechanics [2], [8], [10], [12]. This paper introduces some important ideas from quantum mechanics, proposes a quantum associative memory, presents analytical discussion, and produces empirical results that support the analysis.

2. Quantum Mechanics

Quantum mechanics is a theory that in many ways is counterintuitive. Yet it has provided us with perhaps the most accurate physical theory (in terms of predicting experimental results) ever devised by science. The theory is well-established and is covered in its basic form by many textbooks (see for example [7]). Several necessary ideas from this theory are briefly reviewed here.

Linear superposition is closely related to the familiar mathematical principle of linear combination of vectors. Quantum systems are described by a wave function ψ that exists in a Hilbert space. In the Hilbert space there exists a set of states, $|\phi_i\rangle$, that form a basis, and the system is described by a quantum state $|\psi\rangle$,

$$|\psi\rangle = \sum_i c_i |\phi_i\rangle. \tag{1}$$

$|\psi\rangle$ is said to be in a linear superposition of the basis states $|\phi_i\rangle$, and in the general case, the coefficients c_i may be complex. We use here the Dirac bracket notation where the ket $|\cdot\rangle$ is analogous to a column vector, and the bra $\langle\cdot|$ is analogous to the complex conjugate transpose of the ket. In quantum mechanics the Hilbert space and its basis have a physical interpretation, and this leads directly to perhaps the most counterintuitive aspect of the theory. The counter intuition is this -- at the microscopic or quantum level, the state of the system is described by the wave function ψ, that is, as a linear superposition of all basis states. However, at the macroscopic or classical level the system can be in only a single basis state. For example, at the quantum level an electron can be in a superposition of many energy levels; however, in the classical realm this obviously cannot be.

Coherence and *decoherence* are closely related to the idea of linear superposition. A quantum system is said to be coherent if it is in a linear superposition of its basis states. A strange result of quantum mechanics is that if a system that is in a linear superposition of states interacts in any way with its environment, the superposition is destroyed. This loss of coherence is called decoherence and is governed by the wave function ψ. The coefficients c_i are called probability amplitudes, and $|c_i|^2$ gives the probability of $|\psi\rangle$ collapsing into state $|\phi_i\rangle$ if it decoheres. In the Dirac notation, the probability that a quantum state $|\psi\rangle$ will collapse into an eigenstate $|\phi_i\rangle$ is written $|\langle\phi_i|\psi\rangle|^2$ and is analogous to the dot product (projection) of two vectors. Consider, for example, a discrete physical variable called spin. The simplest spin system is a two-state system, called a spin-1/2 system, whose basis states are usually represented as $|\uparrow\rangle$ (spin up) and $|\downarrow\rangle$ (spin down). In this simple system the wave function ψ is a distribution over two values (up and down) and a coherent state $|\psi\rangle$ is a linear superposition of $|\uparrow\rangle$ and $|\downarrow\rangle$. One such state might be

$$|\psi\rangle = \frac{2}{\sqrt{5}}|\uparrow\rangle + \frac{1}{\sqrt{5}}|\downarrow\rangle. \tag{2}$$

As long as the system maintains its quantum coherence it cannot be said to be either spin up or spin down. It is in some sense both at once. Under the classical interpretation, of course, it must be one or the other, and when this system decoheres the result is, for example, the $|\uparrow\rangle$ state with probability

$$\left|\langle\uparrow|\psi\rangle\right|^2 = \left(\frac{2}{\sqrt{5}}\right)^2 = 0.8. \tag{3}$$

Operators on a Hilbert space describe how one wave function is changed into another. Here they will be denoted by a capital letter with a hat, such as $\hat{A}$, and they may be represented as matrices acting on vectors. Using operators, an eigenvalue equation can be written

$$\hat{A}|\phi_i\rangle = a_i|\phi_i\rangle, \tag{4}$$

where a_i is the eigenvalue. The solutions $|\phi_i\rangle$ to such an equation are called eigenstates and can be used to construct the basis of a Hilbert space as discussed above. In the quantum formalism, all properties are represented as operators whose eigenstates are the basis for the Hilbert space associated with that property and whose eigenvalues are the quantum allowed values for that property. It is important to note that operators in quantum mechanics must be linear operators.

Interference is a familiar wave phenomenon. Wave peaks that are in phase interfere constructively (magnify each other's amplitude) while those that are out of phase interfere destructively (decrease or eliminate each other's amplitude). This is a phenomenon common to all kinds of wave mechanics from common water waves to optics. The well-known double slit experiment demonstrates empirically that at the quantum level interference also applies to the probability waves of quantum mechanics. Herein lies a truly non-classical property that gives computation by quantum means a unique advantage over classical approaches, including any stochastic methods [5].

Entanglement is the potential for quantum states to exhibit correlations that cannot be accounted for classically. Mathematically, states that cannot be factorized (as a tensor product) are said to be entangled, while those that can be factorized are not. There are different degrees of entanglement, and it is interesting to note from a computational standpoint that quantum states that are superpositions of *only* those basis states that are maximally far apart in terms of Hamming distance are those states with the greatest entanglement. For example, a superposition of only the states $|00\rangle$ and $|11\rangle$ is maximally entangled; on the other hand, a superposition of $|00\rangle$, $|01\rangle$ and $|11\rangle$ still exhibits entanglement, but to a lesser degree. Finally, it should be mentioned that while interference is a quantum property that has a classical cousin, entanglement is a completely quantum phenomenon for which there is no classical analog.

3. Quantum Associative Memory (QuAM)

The quantum associative memory proposed here consists of a set of spin systems, each of which may be thought of as the quantum analog of a simple 2 or 3 state artificial neuron. The patterns to be learned are viewed as operators that act on the spin systems. The following definitions are necessary for describing the QuAM:

$\vec{x}$ as a binary input vector of length n, whose elements can have the values 0 or 1,
$\vec{z}$ as a ternary feature vector of length n, whose elements can have the values 0, 1, or $*$, with $*$ representing a "don't care" state,
$\vec{o}$ as a binary output class vector of length s associated with an input vector $\vec{x}$,
$\vec{p}$ as a binary output class vector of length s associated with a feature vector $\vec{z}$, and
$\mathcal{T}$ as a set of patterns $\vec{x}_i \rightarrow \vec{o}_i$, $1 \leq i \leq q$ to be learned.

A feature vector $\vec{z}$ is said to match an input vector $\vec{x}$ if and only if $\forall j, z_j = x_j$ or $z_j = *$. Further, if the number of elements $z_j \neq *$ is r, then the feature vector $\vec{z}$ is of order r, or $\vec{z}$ is said to be an rth order feature. For example, given the input vector (101), the 1st order features that match it are $(1**)$, $(*0*)$, and $(**1)$. Additional necessary definitions include the matching functions m and w. Together these two functions compute a weighted hamming score for how closely a feature vector matches an input vector.

$$m(x,z) = \begin{cases} 1 & \text{if } z = * \\ \rho & \text{if } x = z \\ 0 & \text{otherwise.} \end{cases}$$

$$w(\vec{x},\vec{z}) = \begin{cases} 0 & \text{if } \exists j, m(x_j, z_j) = 0 \\ \prod_j m(x_j, z_j) & \text{otherwise.} \end{cases}$$

For computational purposes, the most commonly used quantum system is the spin-1/2 system introduced in Section 2, which is referred to as a quantum bit or *qubit*. Relabeling the eigenstates as $|0\rangle$ and $|1\rangle$, it is easy to see why this is so. One useful feature of quantum computation is that as long as a set of n qubits maintains its quantum coherence, it can be considered to be all 2^n possible binary patterns at once. Thus any operation performed on the qubits is performed on all 2^n patterns in parallel. This is termed *quantum parallelism*.

We propose the use of a slightly more complex spin-1 system (3 states) whose eigenstates are labeled $|0\rangle$, $|1\rangle$ and $|*\rangle$. This system may be thought of as a classical ternary variable --- when it decoheres it will have a value of either 0 or 1 or $*$. However, while coherence is maintained, it will exist as a superposition of all three. Thus, a coherent set of n such quantum systems would be represented as a superposition of all 3^n eigenstates. We now define the quantum system for learning associative pattern completion:

$|\psi\rangle = \sum_{\vec{z}\vec{p}} c_{\vec{z}\vec{p}} |\vec{z}\vec{p}\rangle$ as the quantum state of a set of n spin-1 systems and s spin-1/2 systems, with the former designated to represent the input vector and the latter designated to represent the output class vector.

Each spin-1 system can be thought of as a neuron. Notice that the eigenstates are labeled as a feature vector ($\vec{z}$) paired with an output vector ($\vec{p}$), corresponding to the different states in which the quantum system can be found. Each basis state corresponds to a possible pattern or partial pattern, and a superposition of states corresponds to a superposition of different patterns and partial patterns. Therefore, ψ represents a QuAM with the ability to represent $3^n \times 2^s$ different patterns. In this sense it is no different than a classical pattern associator. As we shall see, however, its storage capacity will be exponential in n. The patterns to be learned are considered as operators that operate on ψ, with each pattern $\vec{x}_i\vec{o}_i$ producing two operators: $\hat{A}_{\vec{x}_i\vec{o}_i}$ and $\hat{B}_{\vec{x}_i\vec{o}_i}$. Consider each $\hat{A}_{\vec{x}_i\vec{o}_i}$ operator as a matrix a that is indexed by column and row as $\vec{z}_1\vec{p}_1, \vec{z}_2\vec{p}_2$. Then the definition for the $\hat{A}_{\vec{x}_i\vec{o}_i}$ operators is

$$a_{\vec{z}_1\vec{p}_1,\vec{z}_2\vec{p}_2} = \begin{cases} 0 & \text{if } \vec{z}_1 \text{ does not match } \vec{x}_i \\ w(\vec{x}_i, \vec{z}_1) & \text{if } \vec{z}_1 = \vec{z}_2 \text{ and } \vec{o}_i = \vec{p}_1 = \vec{p}_2. \end{cases}$$

The $\hat{A}$ operators are designed to increase the probability amplitudes of those eigenstates in $|\psi\rangle$ associated with features that match the input vectors in the training set. An $\hat{A}$ operator increases the probability amplitude for only those eigenstates whose features match its associated input training vector and whose output class matches the output training class. The second set of operators, $\hat{B}_{\vec{x}_i\vec{o}_i}$, is defined similarly as

$$b_{\vec{z}_1\vec{p}_1,\vec{z}_2\vec{p}_2} = \begin{cases} 0 & \text{if } \vec{z}_1 \text{ does not match } \vec{x}_i \\ -w(\vec{x}_i, \vec{z}_1) & \text{if } \vec{z}_1 = \vec{z}_2 \text{ and } \vec{o}_i \neq \vec{p}_i \\ w(\vec{x}_i, \vec{z}_1) & \text{if } \vec{z}_1 = \vec{z}_2 \text{ and } \vec{o}_i = \vec{p}_i. \end{cases}$$

The $\hat{B}$ operators, on the other hand, are constructed to cause interference (constructive or destructive depending upon the output class) between the amplitudes of competing eigenstates. They increase the probability amplitudes (positively or negatively) of those eigenstates that correspond to features that match their associated input training vector regardless of output class. Finally, the pattern operators are combined as follows:

$\hat{A}$ as the operator $\sum_{i=1}^{q} \hat{A}_{\vec{x}_i\vec{o}_i}$,
$\hat{B}$ as the operator $\sum_{i=1}^{q} \hat{B}_{\vec{x}_i\vec{o}_i}$, and
$\hat{T}$ as the operator $\hat{A}\hat{B}$.

$\hat{T}$ combines the information in $\hat{A}$ and $\hat{B}$ to represent an exponential number of patterns in linear superposition for the purpose of pattern association. The learning of the patterns in $\mathcal{T}$ by the QuAM may be summarized as follows:

1. Prepare $|\psi\rangle = \sum_{\vec{z}\vec{p}} c_{\vec{z}\vec{p}} |\vec{z}\vec{p}\rangle$ such that the $c_{\vec{z}\vec{p}}$ represent a uniform distribution.
2. Calculate the operator $\hat{T}$.
3. Apply the operator $\hat{T}$ to $|\psi\rangle$.

In essence, the information learned from the training set is encoded in the coefficients $c_{\vec{z}\vec{p}}$. For the execution or pattern completion phase two more definitions are required.

$\vec{y}$ as a binary input vector to be classified, and
$\hat{Y}$ as an operator defined as the matrix

$$y_{\vec{z}_1\vec{p}_1,\vec{z}_2\vec{p}_2} = \begin{cases} 0 & \text{if } \vec{z}_1 \text{ does not match } \vec{y}_i \\ w(\vec{y}_i,\vec{z}_1) & \text{if } \vec{z}_1 = \vec{z}_2 \end{cases}$$

$\hat{Y}$ acts similarly to $\hat{B}$ in that it makes non-zero the probability amplitudes for those eigenstates whose features match the input vector to be classified; obviously output class cannot be considered. Given the input vector $\vec{y}$, pattern completion proceeds as follows:

4. Apply the operator $\hat{Y}$ to $|\psi\rangle$.
5. Observe the quantum spin-1/2 systems associated with the output vector.

Concisely, to complete the partial pattern described by an input vector $\vec{y}$ given a training set $\mathcal{T}$, the quantum state $|\psi\rangle$ is prepared to be in a superposition of all its eigenstates, and then

$$\hat{Y}\hat{T}|\psi\rangle \tag{5}$$

is calculated. Finally, the system $|\psi\rangle$ is observed causing it to choose a single eigenstate, and the values observed for the spin-1/2 systems associated with the output pattern are the completion of $\vec{y}$.

4. Discussion and Empirical Results

Three datasets from the UCI repository [9] were selected for empirical testing. Due to the exponential nature of simulating quantum systems on a classical computer, only small data sets (in the number of inputs) can be considered. These data sets were converted to binary representation before simulation. The *Lenses* data set represents patients and what kind of contact lense they should be prescribed (hard, soft, none). The *Hayes* data set is an abstract measure of concept formation used to test human psychological function. The *LED* data set is an artificially generated representation of the 10 digits of an LED display with 25% noise. Ten fold cross validation was used to obtain results, and since quantum mechanics is a probabilistic theory, ten trials for each data split were run to get a more accurate picture of the results.

Table 1. Pattern completion accuracy

	$\rho = 2$	$\rho = 4$	$\rho = 8$	$\rho = 16$	$\hat{A}$
Lenses	0.77	0.98	1.00	1.00	1.00
Hayes	-	-	0.90	0.91	0.91
LED	-	-	0.63	0.66	0.73

For the *Lenses* data set, the QuAM uses 5 neurons (3-state qubits) to represent 23 patterns; for the *Hayes* data set, the QuAM uses 8 neurons to represent 83 patterns; and for the *LED* data set, the QuAM uses 7 neurons to represent 200 (noisy) patterns. Table 1 shows the pattern completion rate on the data sets for different values of ρ (from the *m* function). The first four columns, labeled with their respective values for ρ, show that as ρ increases in value, pattern recall accuracy improves. This is due to the nature of the m function and the length of the input vector x. This improvement is most marked for the *Lenses* data set which achieves perfect recall of all 23 patterns using only 5 qubits. The results for the *Hayes* data set are equally encouraging -- 90% recall of 83 patterns using only 8 qubits. Finally notice that for the *LED* data set the results are not as good. This is due to the noise in the data set producing harmful interference patterns through operator $\hat{B}$. The fifth column shows recall accuracy using the $\hat{A}$ operator instead of the $\hat{T}$ operator. In this case, results for *LED* are also near optimum (note that due to noise the maximum accuracy for *LED* is 75%).

The ability to memorize and correctly recall this many patterns with so few neurons is classically not possible. It is the quantum effects of coherence, interference and entanglement that make the exponential storage capacity possible.

4.1. Generalization

Although the QuAM's primary purpose is to memorize a set of patterns for completion, it does have the ability to generalize over patterns not seen during learning. However, since this is not its main purpose and since the QuAM's bias is towards the highest order matching feature, its generalization ability is expected to be limited. The definition of $\hat{T}$ includes the $\hat{B}$ operator for the purpose of utilizing interference between contradictory features in order to improve generalization. This is accomplished by having features that are good predictors of the output vector reinforced and by having features that are bad predictors eliminated. However, the interference in $\hat{B}$ sometimes overcompensates and becomes detrimental to generalization by actually reinforcing bad features. Thus, while the $\hat{B}$ operator provides some generalization accuracy, a new operator that better describes the proper constructive and destructive interference between patterns will improve this accuracy significantly. Table 2 supports this by giving generalization results using the operator $\hat{A}$ alone (no interference) and the full operator $\hat{T}$. Note that on the *Lenses* and *Hayes* data sets, the addition of the interference inherent in $\hat{B}$ does improve generalization as expected. However, due to the noise in the *LED* data set, $\hat{B}$ actually hurts generalization accuracy, just as it did recall accuracy (Table 1). For reference, the third column, labeled “Other”, provides average generalization accuracy on these data sets for several well known learning algorithms [13]. As expected, the generalization accuracy of the QuAM may still be improved significantly.

Table 2. Generalization accuracy

	$\hat{A}$	$\hat{T}$	Other
Lenses	0.43	0.68	0.73
Hayes	0.45	0.48	0.66
LED	0.59	0.56	0.68

5. Concluding Remarks

A unique view of the pattern completion problem is presented that allows the proposal of a quantum associative memory with exponential storage capacity. It employs simple spin-1/2 and spin-1 quantum systems and represents training and input vectors as quantum operators. This approach demonstrates that learning techniques can be developed making use of the unique characteristics of quantum mechanics. Discussion on the physical realization of this system is beyond the scope of this paper. We acknowledge that the implementation is far from a trivial issue as investigation into the field of quantum computation will attest. By far the most difficult issue is the maintenance of quantum coherence throughout the computation. However, recently some excitement has been generated over the possible use of nuclear magnetic resonance (NMR) techniques to overcome many of these difficulties [4]. Finally, it may be noted that we have made use of only the very simplest of quantum systems and one of the very simplest of learning algorithms. Current research is focused on improving generalization capability for patterns not memorized during learning. Future work will include application of quantum theory to more complex learning systems, application of more complex quantum systems to the artificial neural network paradigm, and extension to continuously valued domains.

References

[1] Barenco, A., C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter, "Elementary Gates for Quantum Computation", *Physical Review A*, vol. 52, pp. 3457-3467, 1995.

[2] Behrman, E. C., J. Niemel, J. E. Steck and S. R. Skinner, "A Quantum Dot Neural Network", *IEEE Transactions on Neural Networks*, submitted, 1996.

[3] Brassard, G., "New Trends in Quantum Computation", *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science*, pp. 3-10, Grenoble, France, Springer, 1996.

[4] Cory, David G., Mark D. Price, Amr F. Fahmy and Timothy F. Havel, "Nuclear Magnetic Resonance Spectroscopy: An Experimentally Accessible Paradigm for Quantum Computing", *Physica D*, to appear, (accepted Aug. 27, 1997).

[5] Deutsch, D. and Richard Jozsa, "Rapid Solution of Problems by Quantum Computer", *Proceedings of the Royal Society, London A*, vol. 439, pp. 553-8, 1992.

[6] Deutsch, D., "Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer", *Proceedings of the Royal Society*, *London A*, vol. 400, pp. 97-117, 1985.

[7] Feynman, R. P., R. B. Leighton and M. Sands, *The Feynman Lectures on Physics*, vol. 3, Addison-Wesley Publishing Company: Reading Massachusetts, 1965.

[8] Menneer, T. and A. Narayanan, "Quantum-inspired Neural Networks", technical report R329, Department of Computer Science, University of Exeter, Exeter, United Kingdom, 1995.

[9] Mertz, C. J. and P. M. Murphy, *UCI Repository of Machine Learning Databases*, University of California at Irvine, Department of Information and Computer Science, http://www.ics.uci.edu/~mlearn/ MLRepository.html, 1996.

[10] Perus, M., "Neuro-Quantum Parallelism in Brain-Mind and Computers", *Informatica*, vol. 20, pp. 173-83, 1996.

[11] Shor, P. W. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", *SIAM Journal of Computing*, vol. 26 no. 5, pp. 1484-1509, 1997.

[12] Ventura, D. and T. Martinez, "An Artificial Neuron with Quantum Mechanical Properties", *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, to appear, 1997.

[13] Zarndt, Frederick, "A Comprehensive Case Study: An Examination of Machine Learning and Connectionist Algorithms", Masters Thesis, Brigham Young University, 1995.

Chapter 10

Initializing the Amplitude Distribution of a Quantum State

...and distribution was made unto every man according as he had need.

-- Acts 4:35

submitted to
Physical Review Letters, June 16 1998.

Abstract

To date, quantum computational algorithms have operated on a superposition of all basis states of a quantum system. Typically, this is because it is assumed that some function f is known and implementable as a unitary evolution. However, what if only some points of the function f are known? It then becomes important to be able to encode only the knowledge that we have about f. This paper presents an algorithm that requires a polynomial number of elementary operations for initializing a quantum system to represent only the m known points of a function f.

PACS numbers: 03.67.Lx, 03.67.-a, 89.80.+h, 03.65.Bz

The study of computation as a physical process has produced fascinating results in the form of new approaches to problem solving and information processing that exhibit impressive speed-up over classical approaches for some problems [1-6]. All of these algorithms operate on a superposition of all basis states, usually assuming prior knowledge of some function f that can be effected as a unitary evolution of the system. However, it is often the case in computational problem solving that a function f is not known but rather that some example points of the function are all the knowledge available. Such situations are particularly common in the field of computational learning, which typically deals with learning a function from examples. If quantum computation is to become a broadly applicable paradigm, it must be possible for quantum algorithms to function given only a limited set of functional points. The first major step toward this goal is developing the ability to encode a set of examples of a function as the state of a quantum system. If this can be achieved, then the possibility of quantum computational learning algorithms presents itself. For example, Fourier-based machine learning algorithms such as those by Kushilevitz and Mansour [7] and Jackson [8] are tempting candidates for generalization into the quantum realm. In fact, Bshouty and Jackson investigated a quantum computational approach to learning [9]. However, their algorithm lacks the capability proposed here (relying instead upon a *quantum example oracle* that somehow encodes the entire function f even though the function is not known *a priori*) and therefore, though theoretically interesting, is not a plausible quantum algorithm. This paper presents an algorithm for encoding, in polynomial time, a set of examples of a function as the state of a quantum system.

Given a set $\mathcal{T}$ of m examples of a function f, the goal is to produce

$$\left|\tilde{f}\right\rangle = \sum_{\bar{z}\in\mathcal{T}} f(\bar{z})|\bar{z}\rangle \tag{1}$$

as the quantum state of n qubits. It will be shown that the state $\left|\tilde{f}\right\rangle$ can be constructed using a polynomial number (in n and m) of elementary operations on one, two, or three qubits. In what follows, the specific qubits to which an operator is to be applied are indicated as subscripts on that operator. For simplicity, consideration will first be restricted to the case of $f:\bar{z}\to s$ with $\bar{z}\in\{0,1\}^n$ and $s\in\{\text{-}1,1\}$. First, define the set of 2-qubit operators,

$$\hat{S}^{s,p} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \sqrt{\frac{p-1}{p}} & \frac{-s}{\sqrt{p}} \\ 0 & 0 & \frac{s}{\sqrt{p}} & \sqrt{\frac{p-1}{p}} \end{bmatrix}, \tag{2}$$

where $s\in\{\text{-}1,1\}$ and $m \geq p \geq 1$. These operators form a set of conditional Hadamard-like transforms that will be used to incorporate the example set into a coherent quantum state. There will be a different $\hat{S}^{s,p}$ operator associated with each example in the set $\mathcal{T}$. Next, define

$$\hat{F} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \tag{3}$$

which flips the state of a qubit. Now, define the 2-qubit Control-NOT operator

$$\hat{F}^0 = \begin{bmatrix} \hat{F} & \hat{0} \\ \hat{0} & \hat{I}_2 \end{bmatrix}, \tag{4}$$

where $\hat{0}$ and $\hat{I}_2$ are the 2×2 zero and identity matrices respectively, which conditionally flips the second qubit if the first qubit is in the $|0\rangle$ state. Similarly define $\hat{F}^1$ which conditionally flips the second qubit if the first qubit is in the $|1\rangle$ state [the matrix representing $\hat{F}^1$ is the same as Equation (4) with $\hat{I}_2$ and $\hat{F}$ exchanged]. Finally, introduce four 3-qubit operators (really just four versions of the Fredkin gate [10]). These operators are used to identify specific states in a superposition, similar to Grover's identification of which state should be phase-rotated in his search algorithm [2]. The first of these,

$$\hat{A}^{00} = \begin{bmatrix} \hat{F} & \hat{0} \\ \hat{0} & \hat{I}_6 \end{bmatrix}, \tag{5}$$

where the $\hat{0}$ are 6×2 and 2×6 zero matrices and $\hat{I}_6$ is the 6×6 identity matrix, conditionally flips the third bit if and only if the first two are in the state $|00\rangle$. Note that this is actually equivalent to $\hat{F}^{00}$, where $\hat{F}^{00}$ is the 3-qubit generalization of Equation (4); however, this operator will be used with the third qubit always in the $|0\rangle$ state and thus can be thought of as performing a logical AND of the negation of the first two qubits, setting the third to $|1\rangle$ if and only if the first two are $|00\rangle$. The other 3-qubit operators, $\hat{A}^{01}$, $\hat{A}^{10}$, and $\hat{A}^{11}$, are variations of $\hat{A}^{00}$ in which $\hat{F}$ occurs in the other three possible locations along the main diagonal.

The algorithm for encoding $\mathcal{T}$ into a quantum system requires $n + (n - 1) + 2$ qubits, arranged in three quantum registers labeled x, g, and c. The quantum state of all three registers together is represented as $|x, g, c\rangle$, and the algorithm proceeds as in Figure 1.

Generate $|\tilde{f}\rangle = |x_1 \ldots x_n, g_1 \ldots g_{n-1}, c_1 c_2\rangle = |\overline{0}\rangle$
for $m \geq p \geq 1$
    for $1 \leq j \leq n$
        if $z_{pj} \neq z_{p+1j}$ (where $z_{m+1} = \{0\}^n$)
            $\hat{F}^{0}_{c_2 x_j}|\tilde{f}\rangle$
    $\hat{F}^{0}_{c_2 c_1}|\tilde{f}\rangle$
    $\hat{S}^{s_p, p}_{c_1 c_2}|\tilde{f}\rangle$
    $\hat{A}^{z_1 z_2}_{x_1 x_2 g_1}|\tilde{f}\rangle$
    for $3 \leq k \leq n$
        $\hat{A}^{z_k 1}_{x_k g_{k-2} g_{k-1}}|\tilde{f}\rangle$
    $\hat{F}^{1}_{g_{n-1} c_1}|\tilde{f}\rangle$
    for $n \geq k \geq 3$
        $\hat{A}^{z_k 1}_{x_k g_{k-2} g_{k-1}}|\tilde{f}\rangle$
    $\hat{A}^{z_1 z_2}_{x_1 x_2 g_1}|\tilde{f}\rangle$
$\hat{F}_{c_2}|\tilde{f}\rangle$

Figure 1. Initializing amplitude distribution in a quantum system

The x register will hold a superposition of the examples in the set $\mathcal{T}$ -- there are n qubits in the register, and the value of f for an example will be used as the coefficient for the state corresponding to that example. The g and c registers contain only ancillary qubits and are restored to the state $|\overline{0}\rangle$ by the end of the algorithm. A high-level intuitive description of the algorithm is as follows. The system is initially in the state $|\overline{0}\rangle$. The qubits in the x register are conditionally flipped so that their states correspond to the first example. The $\hat{S}^{s,p}$ operator corresponding to that example then changes the basis of the system such that one of the states has a coefficient that matches the example's value for f. This same state then is changed to one that exists outside of the subspace affected by the $\hat{S}^{s,p}$ operators, in effect making it permanent, and the process is repeated for each example. When all the examples have been processed, the result is a coherent superposition of states corresponding to the sample points, where the amplitudes of the states all have the same magnitude but have different phases according to the values of their corresponding examples. This is somewhat reminiscent of the first part of Bernstein and Vazirani's Fourier sampling mechanism [11]. However, they assume that the function f is known and calculable in polynomial time, whereas here we do not know the function *f*, but only a small set of examples drawn from *f*.

In analyzing the complexity of the algorithm it is assumed that line 1 can be done trivially. The loop of line 2 is repeated *m* times and consists of *n* (lines 3-5) + 3 (6-8) + *n*-2 (9-10) + 1 (11) + *n*-2 (12-13) + 1 (14) operations, and line 15 requires one more operation. Thus, the entire algorithm requires $m(n + 3 + n\text{-}2 + 1 + n\text{-}2 + 1) + 1 = m(3n+1)+1$ operations and is $O(mn)$. This is optimal in the sense that just reading each instance once cannot be done in any fewer than *mn* steps.

A concrete example of a simple 2-input function will help clarify the preceding discussion. Suppose that we are given the set $\mathcal{T} = \{f(01)=-1, f(10)=1, f(11)=-1\}$. The initial state is $|\bar{0}\rangle$, and the algorithm evolves the quantum state through the series of unitary operations described in Figure 1. For convenience lines 3-6, 8-10, and 12-14 of the algorithm are agglomerated as the compound operators $FLIP$, AND, and $AND^\dagger$ respectively and are treated in more detail in Equation (6-8) and Figures 2 and 3. First the qubit states of the *x* register are flipped to match the first example, and the *c* register is marked so the state will be affected by the $\hat{S}^{s,p}$ operator.

$$|00,0,00\rangle \xrightarrow{FLIP} |01,0,10\rangle$$

Next, the appropriate $\hat{S}^{s,p}$ operator (with *s* equal to the output class of the instance being processed, here -1, and *p* equal to the number of the instances, including the current one, yet to be processed, here 3) is applied, creating a new state in the superposition. This process will be referred to as *state generation* and corresponds to line 7 of Figure 2.

$$\xrightarrow{\hat{S}^{-1,3}} -\tfrac{1}{\sqrt{3}}|01,0,11\rangle + \sqrt{\tfrac{2}{3}}|01,0,10\rangle$$

The two states just affected by the $\hat{S}^{s,p}$ operator are marked in their *g* registers.

$$\xrightarrow{AND} -\tfrac{1}{\sqrt{3}}|01,1,11\rangle + \sqrt{\tfrac{2}{3}}|01,1,10\rangle$$

One of the marked states is made permanent by setting its *c* register to $|01\rangle$ and this state now represents the first example. The *c* register of the other state is returned to $|00\rangle$, and it is ready to generate a new state. This is performed by line 11 of the algorithm.

$$\xrightarrow{\hat{F}^{1}_{g_{n-1}c_1}} -\tfrac{1}{\sqrt{3}}|01,1,01\rangle + \sqrt{\tfrac{2}{3}}|01,1,00\rangle$$

Finally the work done in the *g* register is undone and at this point one pass through the loop of line 2 of the algorithm has been performed.

$$\xrightarrow{AND^\dagger} -\tfrac{1}{\sqrt{3}}|01,0,01\rangle + \sqrt{\tfrac{2}{3}}|01,0,00\rangle$$

Now, the entire process is repeated for the second example. Again, the *x* register of the appropriate state (that state whose c_2 qubit is in the $|0\rangle$ state) is selectively flipped to match the new example, and its *c* register is again marked. This time the selective qubit state flipping occurs for those qubits that correspond to bits in which the first and second examples differ -- both in this case. Notice that the state representing the first example is not affected by this operation, nor any of those that follow.

$$\xrightarrow{FLIP} -\tfrac{1}{\sqrt{3}}|01,0,01\rangle + \sqrt{\tfrac{2}{3}}|10,0,10\rangle$$

Next, another $\hat{S}^{s,p}$ operator is applied, this time with $s = 1$ and $p = 2$.

$$\xrightarrow{\hat{S}^{1,2}} -\tfrac{1}{\sqrt{3}}|01,0,01\rangle + \tfrac{1}{\sqrt{2}}\sqrt{\tfrac{2}{3}}|10,0,11\rangle + \sqrt{\tfrac{1}{2}}\sqrt{\tfrac{2}{3}}|10,0,10\rangle$$

The two states just affected by the $\hat{S}^{s,p}$ operator are marked in their *g* registers.

$$\xrightarrow{AND} -\tfrac{1}{\sqrt{3}}|01,0,01\rangle + \tfrac{1}{\sqrt{3}}|10,1,11\rangle + \sqrt{\tfrac{1}{3}}|10,1,10\rangle$$

The c register is used to save another state (this one representing the second example).

$$\xrightarrow{\hat{F}^1_{g_{n-1}c_1}} -\tfrac{1}{\sqrt{3}}|01,0,01\rangle + \tfrac{1}{\sqrt{3}}|10,1,01\rangle + \sqrt{\tfrac{1}{3}}|10,1,00\rangle$$

A final bit of house keeping resets the g register, preparing for the process to be repeated again.

$$\xrightarrow{AND^\dagger} -\tfrac{1}{\sqrt{3}}|01,0,01\rangle + \tfrac{1}{\sqrt{3}}|10,0,01\rangle + \sqrt{\tfrac{1}{3}}|10,0,00\rangle$$

Finally, the third example is considered. The x register of the generator state is again selectively flipped and its c register is marked. This time, only those qubits corresponding to bits that differ in the second and third examples are flipped, in this case just qubit x_2.

$$\xrightarrow{FLIP} -\tfrac{1}{\sqrt{3}}|01,0,01\rangle + \tfrac{1}{\sqrt{3}}|10,0,01\rangle + \sqrt{\tfrac{1}{3}}|11,0,10\rangle$$

Again a new state is generated to represent this third example. Notice that the generator state is now left with a magnitude of 0, because there are no more examples to process.

$$\xrightarrow{\hat{S}^{-1,1}} -\tfrac{1}{\sqrt{3}}|01,0,01\rangle + \tfrac{1}{\sqrt{3}}|10,0,01\rangle - \tfrac{1}{\sqrt{1}}\sqrt{\tfrac{1}{3}}|11,0,11\rangle + \sqrt{\tfrac{0}{1}}\sqrt{\tfrac{1}{3}}|11,0,10\rangle$$

Once again the house keeping steps are performed.

$$\xrightarrow{AND} -\tfrac{1}{\sqrt{3}}|01,0,01\rangle + \tfrac{1}{\sqrt{3}}|10,0,01\rangle - \tfrac{1}{\sqrt{3}}|11,1,11\rangle$$

$$\xrightarrow{\hat{F}^1_{g_m c_1}} -\tfrac{1}{\sqrt{3}}|01,0,01\rangle + \tfrac{1}{\sqrt{3}}|10,0,01\rangle - \tfrac{1}{\sqrt{3}}|11,1,01\rangle$$

$$\xrightarrow{AND^\dagger} -\tfrac{1}{\sqrt{3}}|01,0,01\rangle + \tfrac{1}{\sqrt{3}}|10,0,01\rangle - \tfrac{1}{\sqrt{3}}|11,0,01\rangle$$

Finally, line 15 performs one last operation, restoring all the ancillary qubits to their initial state.

$$\xrightarrow{F_{c_2}} -\tfrac{1}{\sqrt{3}}|01,0,00\rangle + \tfrac{1}{\sqrt{3}}|10,0,00\rangle - \tfrac{1}{\sqrt{3}}|11,0,00\rangle$$

At this point, the g and c registers are not entangled with the x register, and therefore the system can be simplified as

$$-\tfrac{1}{\sqrt{3}}|01\rangle + \tfrac{1}{\sqrt{3}}|10\rangle - \tfrac{1}{\sqrt{3}}|11\rangle,$$

and it may be seen that the partial function defined by the set $\mathcal{T}$ is now represented as a quantum superposition in the x register.

Here the complex operators *FLIP*, *AND*, and $AND^\dagger$ are specified in more detail. In operator notation,

$$FLIP = \hat{F}^0_{c_2 c_1}\, \hat{F}^0_{c_2 x_j}\left(n \geq j \geq 1,\ z_{pj} \neq z_{p+1j}\right). \tag{6}$$

Alternatively, *FLIP* may be represented as a quantum network as in [12]. Figure 2 shows one such network for a 3-input function. In the figure, a '?' indicates that the $\hat{F}^0$ operator is applied only if the value of the bit in question differs from the value of that bit for the previous example.

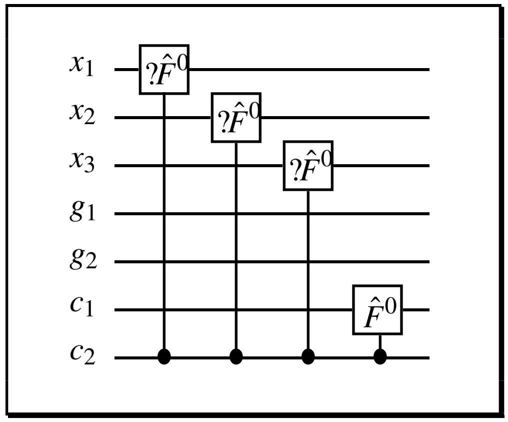

Figure 2. Quantum network for implementing *FLIP*

Also, in operator notation

$$AND = \hat{A}_{x_n g_{n-2} g_{n-1}}^{z_n 1} \cdots \hat{A}_{x_3 g_1 g_2}^{z_3 1} \hat{A}_{x_1 x_2 g_1}^{z_1 z_2} \tag{7}$$

and

$$AND^{\dagger} = \hat{A}_{x_1 x_2 g_1}^{z_1 z_2} \hat{A}_{x_3 g_1 g_2}^{z_3 1} \cdots \hat{A}_{x_n g_{n-2} g_{n-1}}^{z_n 1}. \tag{8}$$

AND and $AND^{\dagger}$ also may be represented as quantum networks, and corresponding example diagrams appear in Figures 3(a) and 3(b) respectively. Note that the only difference between the two is the order in which the qubits are operated upon since all the $\hat{A}$ operators are their own adjoints.

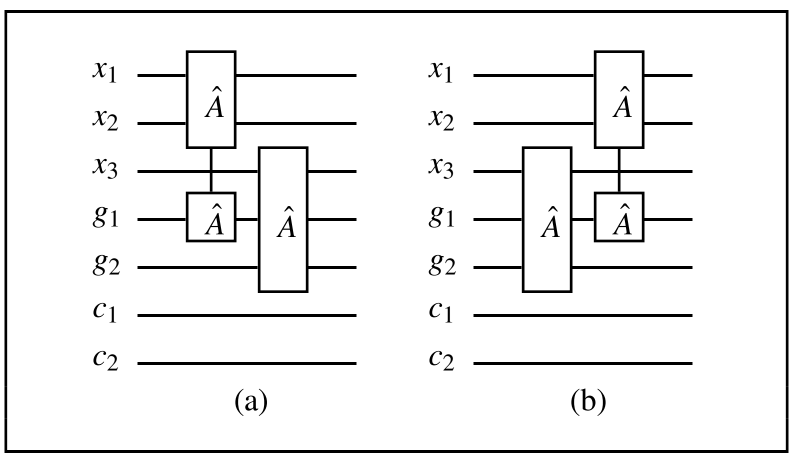

Figure 3. Quantum networks for implementing *AND* and *AND*†

The quantum network representation of the entire algorithm requires the addition of gates for performing the $\hat{S}$ and $\hat{F}$ operations, and the complete network is shown in Figure 4. The network shown will be repeated several times -- once for each instance in the training set. Each such repetition will make use of different $\hat{A}$ operators and flip different qubits using the $\hat{F}$ gates, according to the particular example being processed.

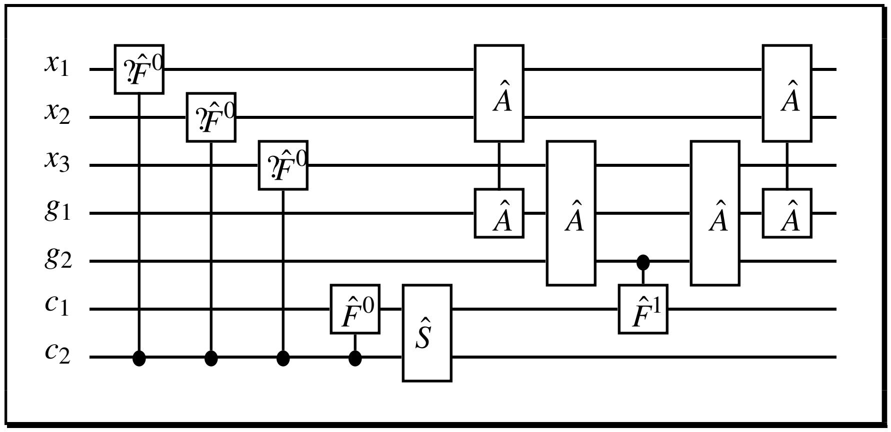

Figure 4. Quantum network for encoding a single function example

The trick that makes the algorithm work is the fact that whenever an $\hat{S}^{s,p}$ operator is applied, there are no states with the c register in the $|11\rangle$ state. This is crucial to the construction of $|\tilde{f}\rangle$ because it allows the operations to be unitary without generating "extraneous" states. Another key is the ability to identify a specific state in the superposition (the one just generated) using the *AND* and *AND*† operators; this provides the ability to mark that particular state and only that state so as not to operate on it again. And this is done reversibly so that no disruptive entanglement results.

The algorithm of Figure 1 can handle only binary functions. However, generalization to functions with N input/output values is straightforward. The only necessary changes are a generalization of the x register (and only the x register) to include systems with more than two states, generalization of some of the operators and the representation of more than two output values as points on the unit circle in the complex plane (representing a binary output with the values -1 and 1 is just a special case of this). The operator generalizations are as follows.

$$\hat{S}^{s,p} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \sqrt{\frac{p-1}{p}} & \frac{e^{-(s2\pi i/N)}}{\sqrt{p}} \\ 0 & 0 & \frac{e^{(s2\pi i/N)}}{\sqrt{p}} & \sqrt{\frac{p-1}{p}} \end{bmatrix}, \tag{9}$$

where now $s \in \{0, ..., N\text{-}1\}$ and $m \geq p \geq 1$. The unit circle in the complex plain is partitioned into N pie pieces, and the points on the pie piece borders correspond to the function values. By convention, one of the values is always on the positive real line.

$$^N\hat{F}^0 = \begin{bmatrix} \hat{C} & \hat{0} \\ \hat{0} & \hat{I}_N \end{bmatrix}, \tag{10}$$

where $\hat{0}$ and $\hat{I}_N$ are the $N \times N$ zero and identity matrices respectively, and $\hat{C}$ is the $N \times N$ circulant matrix obtained by shifting all rows of $\hat{I}_N$ down one, with the Nth row becoming the first. There is now a set of $\hat{F}^0$ operators, one for 2-state qubits, one for 3-state qubits, etc., and the $\hat{F}^0$ operator defined in Equation (4) is the special case for N=2. These operators act on a pair of qubits, the first of which is in the c register and thus is still just a

2-state system, and the second of which is in the x register and thus now has N states, as ${}^{N}\hat{F}^{0}|0q\rangle = |0(q+1) \bmod N\rangle$ and ${}^{N}\hat{F}^{0}|1q\rangle = |1q\rangle$, conditionally rotating the second qubit (labeled q) to the next highest value (but rotating the maximum value back to 0) if the first qubit is in the $|0\rangle$ state. Note that $\hat{F}^{1}$ does not need to be generalized since it only acts on qubits in the g and c registers, which are still only 2-state qubits.

$$ {}^{N}\hat{A}^{00} = \begin{bmatrix} \hat{F} & \hat{0} \\ \hat{0} & \hat{I}_a \end{bmatrix}, \tag{11} $$

where $a = 2N^2$-2, the $\hat{0}$ are 2×a and a×2 zero matrices, $\hat{I}_a$ is the a×a identity matrix, $\hat{F}$ is as defined in Equation (3) and the entire matrix is $2N^2$×$2N^2$. Again there is now a set of $\hat{A}^{00}$ operators and the $\hat{A}^{00}$ operator defined in Equation (5) is the special case for N=2. These operators act on three qubits, as ${}^{N}\hat{A}^{00}|00r\rangle = |00(r+1) \bmod 2\rangle$ and ${}^{N}\hat{A}^{00}|pqr\rangle = |pqr\rangle$ for $p,q \neq 0$, conditionally flipping the third qubit (labeled r), which will still be a 2-state qubit as it is in the g register, if the first two qubits are in the $|00\rangle$ state. Note that whereas the operator $\hat{A}^{00}$ defined in Equation (5) was really equivalent to a 3-qubit version of Equation (4), this is *not* the case with these generalized operators. In other words, Equation (11) is not equivalent to a 3-qubit generalization of Equation (10). This is because ${}^{N}\hat{F}^{0}$ now affects qubits (in the x register) with more than 2 states, therefore requiring a rotation through the various states rather than a simple flip between 2 possible states; whereas ${}^{N}\hat{A}^{00}$ still affects qubits with only 2 states and thus simply toggles between states. Since this operator will be used with the third bit always in the $|0\rangle$ state, it can still be thought of as performing a sort of logical AND (actually a generalized AND where the inputs can be greater than binary) on the values of the first two qubits, setting the third to $|1\rangle$ if and only if the first two are $|00\rangle$. The other $\hat{A}$ operators must also be generalized, a set for each possible pair of values that any of the qubits in the x and g registers can assume. In other words we need all the sets of operators ${}^{N}\hat{A}^{ij}$ for 0≤i,j≤N-1. However, these sets of operators are simple variations of the ${}^{N}\hat{A}^{00}$ set given in Equation (11) with the sub-matrix $\hat{F}$ occurring in different locations along the diagonal according to the values for i and j .

The algorithm for the more general case is the same as in Figure 1, except for two changes. First, the operators must be replaced with their generalizations defined in Equation (9-11), and second, the IF statement of line 4 must be changed to a WHILE statement. This is again because instead of simply flipping a qubit's state between two possibilities as the binary operator $\hat{F}^{0}$ defined in Equation (4) does, the generalized operator ${}^{N}\hat{F}^{0}$ defined in Equation (10) rotates a qubit's state through N possibilities in a specific order one at a time. Thus, the compound *FLIP* operation defined by lines 3-6 of the algorithm now may require up to (N-1)n operations instead of the $O(n)$ required for the simpler binary case. This brings the total number of operations required for the algorithm up to $(N+1)mn + m + 1$, which is still $O(mnN)$. This reduces to $O(mn)$ assuming $N<<m,n$.

In summary, this paper presents a polynomial-time algorithm for initializing a quantum system to represent m known points of a function f. The result is a quantum superposition with m non-zero coefficients -- the creation of which is a nontrivial task compared to creating a superposition of all basis states. It may be appropriate to mention here that very recently work as been done to analyze Grover's algorithm [2] for the case of arbitrary initial amplitude distributions [13]. Further, the paper suggests a new field to which quantum computation may be applied to advantage -- that of computational learning. In fact, it is the authors' opinion that this application of quantum computation will, in general, demonstrate much greater returns than its application to more traditional computational tasks (though Shor's algorithm is an obvious exception). We make this conjecture because results in both quantum computation and computational learning are by nature probabilistic and inexact, whereas most traditional computational tasks require precise and deterministic outcomes.

References

[1] P. Shor, SIAM J. Comput. **26**, 1484 (1997).

[2] L. Grover, *Proceedings of the 28th ACM Symposium on the Theory of Computing*, *Philadelphia*, *1996*, edited by Gary L. Miller (ACM Press, 1996).

[3] D. Simon, SIAM J. Comput. **26**, 1474 (1997).

[4] D. Deutsch and R. Jozsa, Proc. Roy. Soc. London Ser. A **439**, 553 (1992).

[5] T. Hogg, Journal of Artificial Intelligence Research **4**, 91 (1996).

[6] B. M. Terhal and J. A. Smolin, Phys. Rev. A (to be published).

[7] E. Kushilevitz and Y. Mansour, SIAM J. of Comput. **22**, 1331 (1993).

[8] J. Jackson, Journal of Computer and System Sciences **55**, 414 (1997).

[9] N. H. Bshouty and J. Jackson, in *Proceedings of the 8th Annual Conference on Computational Learning Theory*, *Santa Cruz*, *1995*, edited by Wolfgang Maass (ACM Press, 1995).

[10] E. Fredkin and T. Toffoli, Internat. J. Theoret. Phys. **21**, 219 (1982).

[11] E. Bernstein and U. Vazirani, SIAM J. Comput. **26**, 1411 (1997).

[12] D. Deutsch, Proc. Roy. Soc. London Ser. A **425**, 73 (1989).

[13] D. Biron *et al.*, in *Proceedings of the 1st NASA International Conference on Quantum Computing and Quantum Communications*, *Palm Springs*, *1998*, edited by Charles Bennett, (to be published).

Chapter 11

Quantum Associative Memory

When bad men combine, the good must associate...
-- Edmund Burke

submitted to
IEEE Transactions on Neural Networks, June 16 1998.

Early work on this subject has been submitted to
International Conference on Artificial Neural Networks and Genetic Algorithms,
Portoroz, Slovenia, April 1999.

Abstract

This paper combines quantum computation with classical neural network theory to produce a quantum computational learning algorithm. Quantum computation uses microscopic quantum level effects to perform computational tasks and has produced results that in some cases are exponentially faster than their classical counterparts. The unique characteristics of quantum theory may also be used to create a quantum associative memory with a capacity exponential in the number of neurons. This paper combines two quantum computational algorithms to produce such a quantum associative memory. The result is an exponential increase in the capacity of the memory when compared to traditional associative memories such as the Hopfield network. The paper covers necessary high-level quantum mechanical and quantum computational ideas and introduces a quantum associative memory. Theoretical analysis proves the utility of the memory, and it is noted that a small version should be physically realizable in the near future.

1. Introduction

The field of neural networks seeks, among other things, to develop algorithms for imitating in some sense the functionality of the brain. One particular area of interest is that of associative pattern recall. The field of quantum computation (QC) investigates the power of the unique characteristics of quantum systems used as computational machines. This paper combines results from both of these fields to produce a new quantum computational learning algorithm. This contributes significantly to both the field of quantum computation and to the field of neural networks. The field of neural networks benefits by the introduction of a quantum associative memory with a storage capacity exponential in the number of neurons. The contribution to QC is in the form of a new quantum algorithm capable of results that appear to be impossible using classical computational methods.

Assume a set $\wp$ of m binary patterns of length n. We consider the problem of associative pattern completion -- learning to produce one of the full patterns when presented with only a partial pattern. The trivial solution is simply to store the set of patterns as a lookup table or RAM. There are two reasons why this is not always the best solution. First, it requires that a unique address be associated with and remembered for each pattern. Second, the lookup table requires mn bits in order to store all the patterns. It is often desirable to be able to recall the patterns in an associative fashion, thus eliminating the need for explicit addressing. That is, given a partial pattern one would like to be able to "fill in" a reasonable guess as to the rest of the pattern. This may also be considered a form of generalization as the partial pattern may never have been seen during the learning of the pattern set $\wp$. Further, it would of course be beneficial if a smaller representation was possible.

To this end, various classical associative memory schemes have been proposed, perhaps the most well known being the Hopfield network [Hop82] and the bidirectional associative memory (BAM) [Kos88]. These neural approaches to the pattern completion problem allow for associative pattern recall, but suffer severe storage restrictions. Storing patterns of length n requires a network of n neurons, and the number of patterns, m, is then limited by $m \leq kn$, where typically $0.15 \leq k \leq 0.5$. This paper offers improvement by proposing a quantum associative memory that maintains the ability to recall patterns associatively while offering a storage capacity of $O(2^n)$ using only n neurons.

The field of quantum computation, which applies ideas from quantum mechanics to the study of computation, was introduced in the mid-1980s [Ben82] [Deu85] [Fey86]. For a readable introduction to quantum computation see [Bar96]. The field is still in its infancy and very theoretical but offers exciting possibilities for the field of computer science -- perhaps the most notable to date being the discovery of quantum computational algorithms for computing discrete logarithms and prime factorization in polynomial time, two problems for which no known classical polynomial time solutions exist [Sho97]. These algorithms provide theoretical proof not only that interesting computation can be performed at the quantum level but also that it may in some cases have distinct advantages over its classical cousin. Very recently several groups have produced exciting experimental results by successfully implementing quantum algorithms on small-scale nuclear magnetic resonance (NMR) quantum computers (see for example [Jon98] and [Chu98]).

Artificial neural networks (ANNs) seek to provide ways for classical computers to learn rather than to be programmed. As quantum computer technology continues to develop, artificial neural network methods that are amenable to and take advantage of quantum mechanical properties will become possible. In particular, can quantum mechanical properties be applied to ANNs for problems such as associative memory? Recently, work has been done in the area of combining classical artificial associative memory with ideas from the field of quantum mechanics. Perus details several interesting mathematical analogies between quantum theory and neural network theory [Per96] [Beh96]. [Ven98a] goes a step further by proposing an actual model for a quantum associative memory and [Ven98c] further develops this model by exhibiting a physically realizable quantum system for acting as an associative memory. The work here extends the work introduced in [Ven98c], by further developing the ideas, presenting examples and providing rigorous theoretical analysis.

This paper presents a unique reformulation of the pattern completion problem into the language of wave functions and operators. This reformulation may be generalized to a large class of computational learning problems, opening up the possibility of employing the capabilities of quantum computational systems for the solution of computational learning problems. Section 2 presents some basic ideas from quantum mechanics and introduces quantum computation and some of its early successes. Since neither of these subjects can be properly covered here, references for further study are provided. Section 3 discusses in some detail two quantum algorithms, one for storing a set of patterns in a quantum system and one for quantum search. The quantum associative memory that is the main result of this paper is presented in Section 4 along with theoretical analysis of the model, and the paper concludes with final remarks and directions for further research in Section 5.

2. Quantum Computation

Quantum computation is based upon physical principles from the theory of quantum mechanics (QM), which in many ways is counterintuitive. Yet it has provided us with perhaps the most accurate physical theory (in terms of predicting experimental results) ever devised by science. The theory is well-established and is covered in its basic form by many textbooks (see for example [Fey65]). Several necessary ideas that form the basis for the study of quantum computation are briefly reviewed here.

2.1. Linear superposition

Linear superposition is closely related to the familiar mathematical principle of linear combination of vectors. Quantum systems are described by a wave function ψ that exists in a Hilbert space [You88]. The Hilbert space has a set of states, $|\phi_i\rangle$, that form a basis, and the system is described by a quantum state,

$$|\psi\rangle = \sum_i c_i |\phi_i\rangle. \tag{1}$$

$|\psi\rangle$ is said to be in a linear superposition of the basis states $|\phi_i\rangle$, and in the general case, the coefficients c_i may be complex. Use is made here of the Dirac bracket notation, where the ket $|\cdot\rangle$ is analogous to a column vector, and the bra $\langle\cdot|$ is analogous to the complex conjugate transpose of the ket. In quantum mechanics the Hilbert space and its basis have a physical interpretation, and this leads directly to perhaps the most counterintuitive aspect of the theory. The counter intuition is this -- at the microscopic or quantum level, the state of the system is described by the wave function ψ, that is, as a linear superposition of all basis states (i.e. in some sense the system is in all basis states at once). However, at the macroscopic or classical level the system can be in only a single basis state. For example, at the quantum level an electron can be in a superposition of many different energies; however, in the classical realm this obviously cannot be.

2.2. Coherence and decoherence

Coherence and *decoherence* are closely related to the idea of linear superposition. A quantum system is said to be coherent if it is in a linear superposition of its basis states. A result of quantum mechanics is that if a system that is in a linear superposition of states interacts in any way with its environment, the superposition is destroyed. This loss of coherence is called decoherence and is governed by the wave function ψ. The coefficients c_i are called probability amplitudes, and $|c_i|^2$ gives the probability of $|\psi\rangle$ collapsing into state $|\phi_i\rangle$ if it decoheres. Note that the wave function ψ describes a real physical system that must collapse to exactly one basis state. Therefore, the probabilities governed by the amplitudes c_i must sum to unity. This necessary constraint is expressed as the unitarity condition

$$\sum_i |c_i|^2 = 1. \tag{2}$$

In the Dirac notation, the probability that a quantum state $|\psi\rangle$ will collapse into an eigenstate $|\phi_i\rangle$ is written $|\langle\phi_i|\psi\rangle|^2$ and is analogous to the dot product (projection) of two vectors. Consider, for example, a discrete physical variable called spin. The simplest spin system is a two-state system, called a spin-1/2 system, whose basis states are usually represented as $|\uparrow\rangle$ (spin up) and $|\downarrow\rangle$ (spin down). In this simple system the wave function ψ is a distribution over two values (up and down) and a coherent state $|\psi\rangle$ is a linear superposition of $|\uparrow\rangle$ and $|\downarrow\rangle$. One such state might be

$$|\psi\rangle = \tfrac{2}{\sqrt{5}}|\uparrow\rangle + \tfrac{1}{\sqrt{5}}|\downarrow\rangle. \tag{3}$$

As long as the system maintains its quantum coherence it cannot be said to be either spin up or spin down. It is in some sense both at once. Under the classical interpretation, of course, it must be one or the other, and when this system decoheres the result is, for example, the $|\uparrow\rangle$ state with probability

$$|\langle\uparrow|\psi\rangle|^2 = \left(\tfrac{2}{\sqrt{5}}\right)^2 = 0.8. \tag{4}$$

A simple two-state quantum system, such as the spin-1/2 system just introduced, is used as the basic unit of quantum computation. Such a system is referred to as a quantum bit or *qubit*, and renaming the two states $|0\rangle$ and $|1\rangle$ it is easy to see why this is so.

2.3. Operators

Operators on a Hilbert space describe how one wave function is changed into another. Here they will be denoted by a capital letter with a hat, such as $\hat{A}$, and they may be represented as matrices acting on vectors. Using operators, an eigenvalue equation can be written $\hat{A}|\phi_i\rangle = a_i|\phi_i\rangle$, where a_i is the eigenvalue. The solutions $|\phi_i\rangle$ to such an equation are called eigenstates and can be used to construct the basis of a Hilbert space as discussed in Section 2.1. In the quantum formalism, all properties are represented as operators whose eigenstates are the basis for the Hilbert space associated with that property and whose eigenvalues are the quantum allowed values for that property. It is important to note that operators that represent the time evolution of a system in quantum mechanics must be linear operators and further they must be unitary so that $\hat{A}^\dagger\hat{A} = \hat{A}\hat{A}^\dagger = \hat{I}$, where $\hat{I}$ is the identity operator, and $\hat{A}^\dagger$ is the complex conjugate transpose, or adjoint, of $\hat{A}$.

2.4. Interference

Interference is a familiar wave phenomenon. Wave peaks that are in phase interfere constructively (magnify each other's amplitude) while those that are out of phase interfere destructively (decrease or eliminate each other's amplitude). This is a phenomenon common to all kinds of wave mechanics from water waves to optics. The well-known double slit experiment demonstrates empirically that at the quantum level interference also applies to the probability waves of quantum mechanics. As a simple example, suppose that the wave function described in Equation (3) is represented in vector form as

$$|\psi\rangle = \frac{1}{\sqrt{5}}\begin{pmatrix}2\\1\end{pmatrix} \tag{5}$$

and suppose that it is operated upon by an operator $\hat{O}$ described by the following matrix:

$$\hat{O} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{6}$$

The result is

$$\hat{O}|\psi\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\frac{1}{\sqrt{5}}\begin{pmatrix} 2 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{10}}\begin{pmatrix} 3 \\ 1 \end{pmatrix}, \tag{7}$$

and therefore now

$$|\psi\rangle = \tfrac{3}{\sqrt{10}}|\uparrow\rangle + \tfrac{1}{\sqrt{10}}|\downarrow\rangle. \tag{8}$$

Notice that the amplitude of the $|\uparrow\rangle$ state has increased while the amplitude of the $|\downarrow\rangle$ state has decreased. This is due to the wave function interfering with itself through the action of the operator -- the different parts of the wave function interfere constructively or destructively according to their relative phases just like any other kind of wave.

2.5. Entanglement

Entanglement is the potential for quantum states to exhibit correlations that cannot be accounted for classically. From a computational standpoint, entanglement seems intuitive enough -- it is simply the fact that correlations can exist between different qubits -- for example if one qubit is in the $|1\rangle$ state, another will be in the $|1\rangle$ state. However, from a physical standpoint, entanglement is little understood. The questions of what exactly it is and how it works are still not resolved. What makes it so powerful (and so little understood) is the fact that since quantum states exist as superpositions, these correlations somehow exist in superposition as well. When the superposition is destroyed, the proper correlation is somehow communicated between the qubits, and it is this "communication" that is the crux of entanglement. Mathematically, entanglement may be described using the density matrix formalism. The density matrix ρ_ψ of a quantum state $|\psi\rangle$ is defined as

$$\rho_\psi = |\psi\rangle\langle\psi|. \tag{9}$$

For example, the quantum state

$$|\xi\rangle = \tfrac{1}{\sqrt{2}}|00\rangle + \tfrac{1}{\sqrt{2}}|01\rangle \tag{10}$$

appears in vector form as

$$|\xi\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \tag{11}$$

and it may also be represented as the density matrix

$$\rho_\xi = |\xi\rangle\langle\xi| = \frac{1}{2}\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \tag{12}$$

while the state

$$|\psi\rangle = \tfrac{1}{\sqrt{2}}|00\rangle + \tfrac{1}{\sqrt{2}}|11\rangle \tag{13}$$

is represented as

$$\rho_\psi = |\psi\rangle\langle\psi| = \frac{1}{2}\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \tag{14}$$

and the state $|\zeta\rangle = \tfrac{1}{\sqrt{3}}|00\rangle + \tfrac{1}{\sqrt{3}}|01\rangle + \tfrac{1}{\sqrt{3}}|11\rangle$ as

$$\rho_\zeta = |\zeta\rangle\langle\zeta| = \frac{1}{3}\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}, \tag{15}$$

where the matrices and vectors are indexed by the state labels 00, ..., 11. Now, notice that ρ_ξ can be factorized as

$$\rho_\xi = \frac{1}{\sqrt{2}}\left(\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}\right), \tag{16}$$

where $\otimes$ is the normal tensor product. On the other hand, ρ_ψ can not be factorized. States that can not be factorized are said to be entangled, while those that can be factorized are not. Notice that ρ_ζ can be partially factorized two different ways, one of which is

$$\rho_\zeta = \frac{1}{\sqrt{3}}\left(\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}\right) \tag{17}$$

(the other involves Equation (12) and a different remainder); however, in both cases the factorization is not complete. Therefore, ρ_ζ is also entangled, but not to the same degree as ρ_ψ (because ρ_ζ can be partially factorized but ρ_ψ cannot). Thus there are different degrees of entanglement and much work has been done on better understanding and quantifying it [Joz97] [Ved97]. It is interesting to note from a computational standpoint that quantum states that are superpositions of *only* basis states that are maximally far apart in terms of Hamming distance are those states with the greatest entanglement. For example, ρ_ψ is a superposition of only the states $|00\rangle$ and $|11\rangle$, which have a maximum Hamming spread, and therefore ρ_ψ is maximally entangled. Finally, it should be mentioned that while interference is a quantum property that has a classical cousin, entanglement is a completely quantum phenomenon for which there is no classical analog.

2.6. Quantum networks

Quantum networks [Deu89] are one of several theoretical models of quantum computation. Others include quantum Turing machines [Ben82], and quantum cellular automata [Grö88]. In the quantum network model, each unitary operator is modeled as a quantum logic gate that affects one, two, or more qubits. Schematically, this is represented as a set of quantum "wires" entering and leaving the quantum gates, reminiscent of classical logic networks. For example, Figure 1 shows a network that operates on three qubits, which are represented as lines.

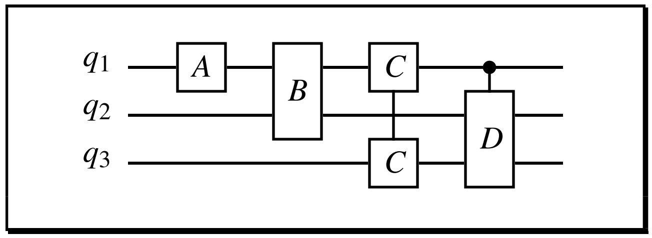

Figure 1. A network of quantum logic gates

By convention the logic flows from left to right. The gates are represented as boxes and labeled with the name of the operator that they represent. A dot on a quantum "wire" represents a conditional upon that qubit. Therefore, in the quantum network shown in Figure 1, $\hat{A}$ represents a single qubit quantum gate, $\hat{B}$ and $\hat{C}$ represent 2-qubit quantum gates, and $\hat{D}$ represents a conditional 3-qubit gate. Suppose that $\hat{A}$ is an operator that flips the state of a qubit, $\hat{B}$ is an operator that exchanges the states of two qubits, $\hat{C}$ is an operator that flips the states of two qubits if they are equal, and $\hat{D}$ is an operator that exchanges the states of two qubits if a third qubit is in the $|1\rangle$ state. When three qubits "enter" the quantum logic network, the one labeled q_1 first has its state flipped; then q_1 and q_2 exchange states, q_1 and q_3 have their states flipped if they are equal, and finally q_2 and q_3 exchange states if q_1 is in the state $|1\rangle$. Of course, if the qubits "entering" the logic array did not exist in a superposition of states, this would be no different than a classical logic sequence. However, the qubits *do* exist in a superposition of states; therefore, these gates or operations are applied to all the states in the superposition simultaneously, resulting in what has been called *quantum parallelism*. Recall that the quantum logic gate arrays are simply a schematic way to represent the time evolution of a quantum system. They are not meant to imply that quantum computation can be physically realized in a manner similar to classical logic networks. Alternatively, the network could be represented as a product of quantum operators. Since operators are applied right to left, the network of Figure 1 would be represented as the operator product $\hat{D}\hat{C}\hat{B}\hat{A}$. In what follows, both the network and the product of operators representations will be used.

2.7. Quantum algorithms

The field of quantum computation is just beginning to develop and offers exciting possibilities for the field of computer science -- the most important quantum algorithms discovered to date all perform tasks for which there are no classical equivalents. For example, Deutsch's algorithm [Deu92] is designed to solve the problem of identifying whether a binary function is constant (function values are either all 1 or all 0) or balanced (the function takes an equal number of 0 and 1 values). Deutsch's algorithm accomplishes the task in order $O(1)$ time, while classical methods require $O(2^n)$ time, where n is the number of binary inputs to the function. Simon's algorithm [Sim97] is constructed for finding the periodicity in a 2-1 binary function that is guaranteed to possess a periodic element. Here again an exponential speedup is achieved. Admittedly, both these algorithms have been designed for artificial, somewhat contrived problems. Grover's algorithm [Gro96], on the other hand, provides a method for searching an unordered quantum database in time $O(\sqrt{n})$, compared to the classical bound of $O(n)$. Here is a real-world problem for which quantum computation provides performance that is classically impossible (though the speedup is less dramatic than exponential). Finally, the most well-known and perhaps the most important quantum algorithm discovered so far is Shor's algorithm for prime factorization [Sho97]. This algorithm finds the prime factors of very large numbers in polynomial time, whereas the best known classical algorithms require exponential time. Obviously, the implications for the field of cryptography are profound. These quantum algorithms take advantage of the unique features of quantum systems to provide impressive speedup over classical approaches.

3. Storing and Recalling Patterns in a Quantum System

Implementation of an associative memory requires the ability to store patterns in the medium that is to act as a memory and the ability to recall those patterns at a later time. This section discusses two quantum algorithms for performing these tasks.

3.1. Grover's algorithm

Lov Grover has developed an algorithm for finding one item in an unsorted database, similar to finding the name that matches a telephone number in a telephone book. Classically, if there are N items in the database, this would require $O(N)$ queries to the database. However, Grover has shown how to do this using quantum computation with only $O(\sqrt{N})$ queries. In the quantum computational setting, finding the item in the database means measuring the system and having the system collapse with near certainty to the basis state which corresponds to the item in the database for which we are searching. The basic idea of Grover's algorithm is to invert the phase of the desired basis state and then to invert all the basis states about the average amplitude of all the states [Gro96] [Gro98]. This process produces an increase in the amplitude of the desired basis state to near unity followed by a corresponding decrease in the amplitude of the desired state back to its original magnitude. The process is cyclical with a period of $\frac{\pi}{4}\sqrt{N}$, and thus after $O(\sqrt{N})$ queries, the system may be observed in the desired state with near certainty (with probability at least $1-\frac{1}{N}$). Interestingly this implies that the larger the database, the greater the certainty of finding the desired state [Boy96]. Of course, if even greater certainty is required, the system may be sampled k times boosting the certainty of finding the desired state to $1-\frac{1}{N^k}$. Here we present the basic ideas of the algorithm and refer the reader to [Gro96] for details. Define the following operators:

$$\hat{I}_\phi = \text{identity matrix but with the element } i_{\phi\phi} = -1, \tag{18}$$

which simply inverts the phase of the basis state $|\phi\rangle$ and

$$\hat{W} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \tag{19}$$

which is often called the Walsh or Hadamard transform. This operator, when applied to a set of qubits, performs a special case of the discrete Fourier transform.

Now to perform the quantum search on a database of size $N = 2^n$, where n is the number of qubits, begin with the system in the $|\bar{0}\rangle$ state and apply the $\hat{W}$ operator. This initializes all the states to have the same amplitude, $\frac{1}{\sqrt{N}}$. Next apply the $\hat{I}_\tau$ operator, where $|\tau\rangle$ is the state being sought, to invert its phase. Finally, apply the operator

$$\hat{G} = -\hat{W}\hat{I}_{\bar{0}}\hat{W} \tag{20}$$

$\frac{\pi}{4}\sqrt{N}$ times and observe the system (see Figure 2). The $\hat{G}$ operator inverts all the states' amplitudes around the average amplitude of all states.

1. Generate the initial state $|\bar{0}\rangle$
2. $|\psi\rangle = \hat{W}|\bar{0}\rangle$
3. Repeat $\frac{\pi}{4}\sqrt{N}$ times
4. $\quad |\psi\rangle = \hat{I}_\tau|\psi\rangle$
5. $\quad |\psi\rangle = \hat{G}|\psi\rangle$
6. Observe the system

Figure 2. Grover's algorithm

3.1.1. An example of Grover's algorithm

Consider a simple example for the case $N = 16$. Suppose that we are looking for the state $|0110\rangle$, or in other words, we would like our quantum system to collapse to the state $|\tau\rangle = |0110\rangle$ when observed. In order to save space, instead of writing out the entire superposition of states, a transpose vector of coefficients will be used, where the vector is indexed by the 16 basis states $|0000\rangle,\cdots,|1111\rangle$. Step 1 of the algorithm results in the state

$$|\psi\rangle = (1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0).$$

In other words, the quantum system described by $|\psi\rangle$ is composed entirely of the single basis state $|0000\rangle$. Now applying the Walsh transform in step 2 to each qubit changes the state to

$$|\psi\rangle \xrightarrow{\hat{W}} |\psi\rangle = \tfrac{1}{4}(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),$$

that is a superposition of all 16 basis states, each with the same amplitude. The loop of step 3 is now executed $\frac{\pi}{4}\sqrt{N} \approx 3$ times. The first time through the loop, step 4 inverts the phase of the state $|\tau\rangle = |0110\rangle$ resulting in

$$|\psi\rangle \xrightarrow{\hat{I}_\tau} |\psi\rangle = \tfrac{1}{4}(1,1,1,1,1,1,-1,1,1,1,1,1,1,1,1,1),$$

and step 5 then rotates all the basis states about the average, which in this case is $\frac{7}{32}$, so

$$|\psi\rangle \xrightarrow{\hat{G}} |\psi\rangle = \tfrac{1}{16}(3,3,3,3,3,3,11,3,3,3,3,3,3,3,3,3).$$

The second time through the loop, step 4 again rotates the phase of the desired state giving

$$|\psi\rangle \xrightarrow{\hat{I}_\tau} |\psi\rangle = \tfrac{1}{16}(3,3,3,3,3,3,-11,3,3,3,3,3,3,3,3,3),$$

and then step 5 again rotates all the basis states about the average, which now is $\frac{17}{128}$ so that

$$|\psi\rangle \xrightarrow{\hat{G}} |\psi\rangle = \tfrac{1}{64}(5,5,5,5,5,5,61,5,5,5,5,5,5,5,5,5).$$

Repeating the process a third time results in

$$|\psi\rangle \xrightarrow{\hat{I}_\tau} |\psi\rangle = \tfrac{1}{64}(5,5,5,5,5,5,\text{-}61,5,5,5,5,5,5,5,5,5).$$

for step 4 and

$$|\psi\rangle \xrightarrow{\hat{G}} |\psi\rangle = \tfrac{1}{256}(\text{-}13,\text{-}13,\text{-}13,\text{-}13,\text{-}13,\text{-}13,251,\text{-}13,\text{-}13,\text{-}13,\text{-}13,\text{-}13,\text{-}13,\text{-}13,\text{-}13,\text{-}13)$$

for step 5. Squaring the coefficients gives the probability of collapsing into the corresponding state, and in this case the chance of collapsing into the $|\tau\rangle = |0110\rangle$ basis state is $0.98^2 \approx 96\%$. The chance of collapsing into one of the 15 basis states that is not the desired state is approximately $0.05^2 = 0.25\%$ for each state. In other words, there is only a $15*0.05^2 \approx 4\%$ probability of collapsing into an incorrect state. This chance of success is better than the bound $1-\frac{1}{N}$ given above and will be even better as N gets larger. For comparison, note that the chance for success after only two passes through the loop is approximately 91%, while after four passes through the loop it drops to 58%. This reveals the periodic nature of the algorithm and also demonstrates the fact that the first time that the probability for success is maximal is indeed after $\frac{\pi}{4}\sqrt{N}$ steps of the algorithm. Figure 3 represents Grover's algorithm as a quantum network. The ellipses indicate that the three operators to the right of the wavy line are those repeated $\frac{\pi}{4}\sqrt{N}$ times. Notice that the $\hat{I}$ operators require an ancillary bit.

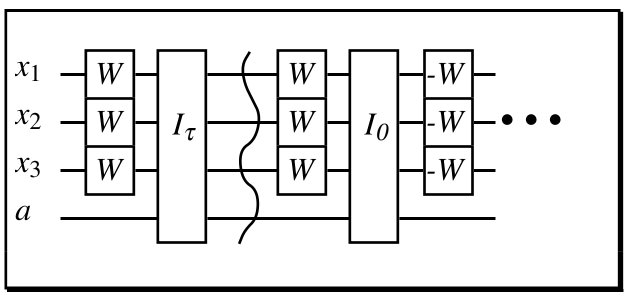

Figure 3. A quantum network for Grover's algorithm

3.2. Initializing the quantum state

[Ven98b] presents a polynomial-time quantum algorithm for constructing a quantum state over a set of qubits to represent the information in a training set. The algorithm is implemented using a polynomial number (in the length and number of patterns) of elementary operations on one, two, or three qubits. Here the necessary operators are presented briefly and the reader is referred to [Ven98b] for details.

$$\hat{S}^p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \sqrt{\frac{p-1}{p}} & \frac{-1}{\sqrt{p}} \\ 0 & 0 & \frac{1}{\sqrt{p}} & \sqrt{\frac{p-1}{p}} \end{bmatrix}, \tag{21}$$

where $m \geq p \geq 1$. These operators form a set of conditional transforms that will be used to incorporate the set of patterns into a coherent quantum state. There will be a different $\hat{S}^p$ operator associated with each pattern to be stored. The interested reader may note that this definition of the $\hat{S}^p$ operator is slightly different than the original. This is because in this context, we are considering associative pattern recall rather than pattern classification and therefore have no output class *per se*. Thus the phase of the coefficients becomes unimportant in this case.

$$\hat{F}^0 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{22}$$

conditionally flips the state of the second qubit if the first qubit is in the $|0\rangle$ state; another operator, $\hat{F}^1$, conditionally flips the second qubit if the first qubit is in the $|1\rangle$ state ($\hat{F}^1$ is the same as $\hat{F}^0$ except that the off-diagonal elements occur in the bottom right quadrant rather than in the top left). These operators are referred to elsewhere as Control-NOT because a logical NOT (state flip) is performed on the second qubit depending upon (or controlled by) the state of the first qubit. A complex combination of several of these operators is used to change basis states to correspond to patterns and will be termed *FLIP* in what follows (for details see [Ven98b]).

$$\hat{A}^{00} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{23}$$

conditionally flips the state of the third qubit if and only if the first two are in the state $|00\rangle$. Note that this operator is really just a Fredkin gate [Fre82] and can be thought of as performing a logical AND of the negation of the first two bits, writing a 1 in the third if and only if the first two are both 0. Three other operators, $\hat{A}^{01}$, $\hat{A}^{10}$ and $\hat{A}^{11}$, are variations of $\hat{A}^{00}$ in which the off diagonal elements occur in the other three possible locations along the main diagonal. $\hat{A}^{01}$ can be thought of as performing a logical AND of the first bit and the negation of the second, and so forth. A combination of these operators is used to identify specific states in a superposition and along with one $\hat{F}^1$ operator combine to form complex operator that will be called *SAVE* here (again for details see [Ven98b]).

Now given a set $\wp$ of m binary patterns of length n to be memorized, the quantum algorithm for storing the patterns requires a set of $2n+1$ qubits, the first n of which actually store the patterns and can be thought of analogously as n neurons in a quantum associative memory. For convenience, the qubits are arranged in three quantum registers labeled x, g, and c, and the quantum state of all three registers together is represented in the Dirac notation as $|x,g,c\rangle$. The algorithm proceeds as follows (see Figure 4).

The x register will hold a superposition of the patterns. There is one qubit in the register for each bit in the patterns to be stored, and therefore any possible pattern can be represented. The g register is a garbage register used only in identifying a particular state. It is restored to the state $|\bar{0}\rangle$ after every iteration. The c register contains two control qubits that indicate the status of each state at any given time and may also be restored to the $|\bar{0}\rangle$ state at the end of the algorithm. A high-level intuitive description of the algorithm is as follows. The system is initialized to the single basis state $|\bar{0}\rangle$. The qubits in the x register are selectively flipped so that their states correspond to the inputs of the first pattern. Then, the state in the superposition representing the pattern is "broken" into two "pieces" -- one "larger" and one "smaller" and the status of the smaller one is made permanent in the c register. Next, the x register of the larger piece is selectively flipped again to match the input of the second pattern, and the process is repeated for each pattern. When all the patterns have been "broken" off of the large "piece", then all that is left is a collection of small pieces, all the same size, that represent the patterns to be stored; in other words, a coherent superposition of states is created that corresponds to the patterns, where the amplitudes of the states in the superposition are all equal. The algorithm requires $O(mn)$ steps to encode the patterns as a quantum superposition over n quantum neurons. Note that this is optimal in the sense that just reading each instance once cannot be done any faster than $O(mn)$.

1. Generate the initial state $|\psi\rangle = |\bar{0}\rangle$
2. For $m \geq p \geq 1$
3. $\quad FLIP|\psi\rangle$
4. $\quad \hat{S}^p|\psi\rangle$
5. $\quad SAVE|\psi\rangle$

Figure 4. Quantum algorithm for storing patterns

3.2.1. An example of storing patterns in a quantum system

A concrete example for a set of binary patterns of length 2 will help clarify much of the preceding discussion. Suppose that we are given the pattern set $\mathcal{P} = \{01,10,11\}$. Recall that the x register is the important one that corresponds to the various patterns, that the g register is used as a temporary workspace to mark certain states and that the c register is a control register that is used to determine which states are affected by a particular operator. Now the initial state $|00,0,00\rangle$ is generated and the algorithm evolves the quantum state through the series of unitary operations described in Figure 4.

First, for any state whose c_2 qubit is in the state $|0\rangle$, the qubits in the x register corresponding to non-zero bits in the first pattern have their states flipped (in this case only the second x qubit's state is flipped) and then the c_1 qubit's state is flipped if the c_2 qubit's state is $|0\rangle$. This flipping of the c_1 qubit's state marks this state for being operated upon by an $\hat{S}^p$ operator in the next step. So far, there is only one state, the initial one, in the superposition, so things are pretty simple. This flipping is accomplished with the $FLIP$ operator of line 3 in Figure 4.

$$|00,0,00\rangle \xrightarrow{FLIP} |01,0,10\rangle$$

Next, any state in the superposition with the c register in the state $|10\rangle$ (and there will always be only one such state at this step) is operated upon by the appropriate $\hat{S}^p$ operator (with p equal to the number of patterns including the current one yet to be processed, in this case 3). This essentially "carves off" a small piece and creates a new state in the superposition. This operation corresponds to line 4 of Figure 4.

$$\xrightarrow{\hat{S}^3} \frac{1}{\sqrt{3}}|01,0,11\rangle + \sqrt{\frac{2}{3}}|01,0,10\rangle$$

Next, the two states affected by the $\hat{S}^p$ operator are processed by the $SAVE$ operator of line 5 of the algorithm. This makes the state with the smaller coefficient a permanent representation of the pattern being processed and resets the other to generate a new state for the next pattern. At this point one pass through the loop of line 2 of the algorithm has been performed.

$$\xrightarrow{SAVE} \frac{1}{\sqrt{3}}|01,0,01\rangle + \sqrt{\frac{2}{3}}|01,0,00\rangle$$

Now, the entire process is repeated for the second pattern. Again, the x register of the appropriate state (that state whose c_2 qubit is in the $|0\rangle$ state) is selectively flipped to match the new pattern. Notice that this time the generator state has its x register in a state corresponding to the pattern that was just processed. Therefore, the selective qubit state flipping occurs for those qubits that correspond to bits in which the first and second patterns differ -- both in this case.

$$\xrightarrow{FLIP} \frac{1}{\sqrt{3}}|01,0,01\rangle + \sqrt{\frac{2}{3}}|10,0,10\rangle$$

Next, another $\hat{S}^p$ operator is applied to generate a representative state for the new pattern.

$$\xrightarrow{\hat{S}^2} \frac{1}{\sqrt{3}}|01,0,01\rangle + \frac{1}{\sqrt{2}}\sqrt{\frac{2}{3}}|10,0,11\rangle + \sqrt{\frac{1}{2}}\sqrt{\frac{2}{3}}|10,0,10\rangle$$

Again, the two states just affected by the $\hat{S}^p$ operator are operated on by the $SAVE$ operator, the one being made permanent and the other being reset to generate a new state for the next pattern.

$$\xrightarrow{SAVE} \frac{1}{\sqrt{3}}|01,0,01\rangle + \frac{1}{\sqrt{3}}|10,0,01\rangle + \sqrt{\frac{1}{3}}|10,0,00\rangle$$

Finally, the third pattern is considered and the process is repeated a third time. The x register of the generator state is again selectively flipped. This time, only those qubits corresponding to bits that differ in the second and third patterns are flipped, in this case just qubit x_2.

$$\xrightarrow{FLIP} \frac{1}{\sqrt{3}}|01,0,01\rangle + \frac{1}{\sqrt{3}}|10,0,01\rangle + \sqrt{\frac{1}{3}}|11,0,10\rangle$$

Again a new state is generated to represent this third pattern.

$$\xrightarrow{\hat{S}^1} \frac{1}{\sqrt{3}}|01,0,01\rangle + \frac{1}{\sqrt{3}}|10,0,01\rangle + \frac{1}{\sqrt{1}}\sqrt{\frac{1}{3}}|11,0,11\rangle + \sqrt{\frac{0}{1}}\sqrt{\frac{1}{3}}|11,0,10\rangle$$

Finally, proceed once again with the $SAVE$ operation.

$$\xrightarrow{SAVE} \frac{1}{\sqrt{3}}|01,0,01\rangle + \frac{1}{\sqrt{3}}|10,0,01\rangle + \frac{1}{\sqrt{3}}|11,0,01\rangle$$

At this point, notice that the states of the g and c registers for all the states in the superposition are the same. This means that these registers are in no way entangled with the x register, and therefore since they are no longer needed they may be ignored without affecting the outcome of further operations on the x register. Thus, the simplified representation of the quantum state of the system is

$$-\frac{1}{\sqrt{3}}|01\rangle+\frac{1}{\sqrt{3}}|10\rangle-\frac{1}{\sqrt{3}}|11\rangle,$$

and it may be seen that the set of patterns $\wp$ is now represented as a quantum superposition in the x register.

The quantum network representation of the algorithm for a single pattern is shown in Figure 5. The *FLIP* operator is composed of the $\hat{F}^0$ operators left of the $\hat{S}^p$ and the question marks signify that the operator is applied only if the qubit's state differs from the value of the corresponding bit in the pattern being processed. The *SAVE* operator is composed of the $\hat{A}$ operators and the $\hat{F}^1$ to the right of $\hat{S}^p$. The network shown is simply repeated for additional patterns.

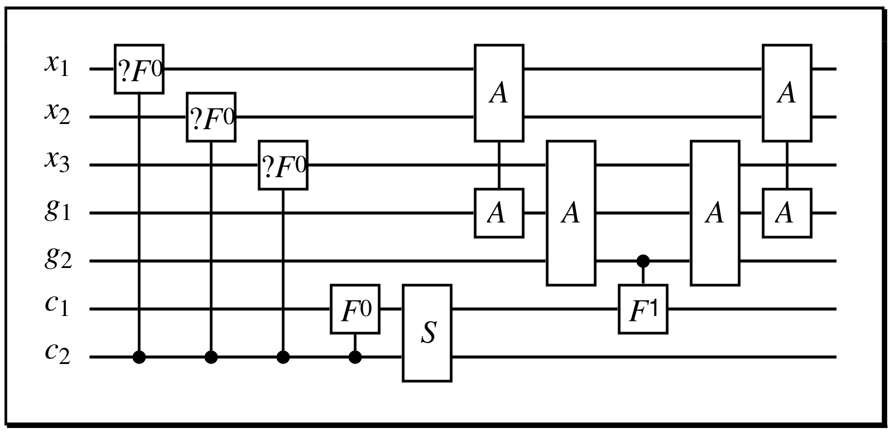

Figure 5. Quantum network for storing a pattern

3.3. Grover's algorithm revisited

Grover's original algorithm only applies to the case where all basis states are represented in the superposition equally to start with and one and only one basis state is to be recovered. In other words, strictly speaking, the original algorithm would only apply to the case when the set $\wp$ of patterns to be memorized includes all possible patterns of length n and when we know all n bits of the pattern to be recalled -- not a very useful associative memory. However, several other papers have since generalized Grover's original algorithm and improved on his analysis to include cases where not all possible patterns are represented and where more than one target state is to be found [Boy96] [Bir98] [Gro98]. Strictly speaking it is these more general results which allow us to create a useful QuAM that will associatively recall patterns.

In particular, [Bir98] is useful as it provides bounds for the case of using Grover's algorithm for the case of arbitrary initial amplitude distributions (whereas Grover originally assumed a uniform distribution). It turns out that a high probability for success using Grover's original algorithm depends upon this assumption of initial uniformity as the following modified version of the example of Section 3.1.1 will show.

3.3.1. Grover example revisited

Recall that we are looking for the state $|0110\rangle$, and assume that we do not perform the first two steps of the algorithm shown in Figure 2 (which initialize the system to the uniform distribution) but that instead we have the initial state described by

$$|\psi\rangle = \frac{1}{\sqrt{6}}(1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1),$$

that is a superposition of only 6 of the possible 16 basis states. The loop of step 3 is now executed. The first time through the loop, step 4 inverts the phase of the state $|\tau\rangle = |0110\rangle$ resulting in

$$|\psi\rangle \xrightarrow{\hat{I}_\tau} |\psi\rangle = \frac{1}{\sqrt{6}}(1,0,0,1,0,0,-1,0,0,1,0,0,1,0,0,1),$$

and step 5 then rotates all the basis states about the average, which is $\frac{1}{4\sqrt{6}}$, so

$$|\psi\rangle \xrightarrow{\hat{G}} |\psi\rangle = \frac{1}{2\sqrt{6}}(-1,1,1,-1,1,1,3,1,1,-1,1,1,-1,1,1,-1).$$

The second time through the loop step 4 again rotates the phase of the desired state giving

$$|\psi\rangle \xrightarrow{\hat{I}_\tau} |\psi\rangle = \frac{1}{2\sqrt{6}}(-1,1,1,-1,1,1,-3,1,1,-1,1,1,-1,1,1,-1),$$

and then step 5 again rotates all the basis state about the average, which now is $\frac{1}{16\sqrt{6}}$ so that

$$|\psi\rangle \xrightarrow{\hat{G}} |\psi\rangle = \frac{1}{8\sqrt{6}}(5,-3,-3,5,-3,-3,13,-3,-3,5,-3,-3,5,-3,-3,5).$$

Now squaring the coefficients gives the probability of collapsing into the corresponding state. In this case, the chance of collapsing into the $|\tau\rangle = |0110\rangle$ basis state is $0.66^2 \approx$ 44%. The chance of collapsing into one of the 15 basis states that is not the desired state is approximately 56%. This chance of success is much worse than that seen in Section 3.1.1, and the reason for this is that there are now two types of undesirable states: those that existed in the superposition to start with but that are not the state we are looking for and those that were not in the original superposition but were introduced into the superposition by the $\hat{G}$ operator. The problem comes from the fact that these two types of undesirable states acquire opposite phases and thus to some extent cancel each other out. Therefore, during the rotation about average performed by the $\hat{G}$ operator the average is smaller than it should be if it were to just represent the states in the original superposition. As a result, the desired state is rotated about a suboptimal average and never gets as large a probability associated with it as it should. In [Bir98], Biron, *et al.* give an analytic expression for the maximum possible probability using Grover's algorithm on an arbitrary starting distribution:

$$P_{\max} = 1 - \sum_{j=r+1}^{N} \left| l_j - \bar{l} \right|^2, \tag{24}$$

where N is the total number of basis states, r is the number of desired states (looking for more than one state is another extension to the original algorithm), l_j is the initial amplitude of state j, and they assume without loss of generality that the desired states are numbered 1 to r and the other states are numbered r+1 to N. $\bar{l}$ is the average amplitude of all the undesired states, and therefore the second term of Equation (24) is proportional to the variance in the amplitudes. Obviously, in the uniform case that the original algorithm assumed, the variance will be 0 and therefore P_{max} = 1; and in the example of Section 3.1.1 we do get 96% probability of success. The reason we do not reach the theoretical maximum is that Equation (24) is a tight bound only in the case of non-integer time steps. Since this is not realistic, it becomes in practice an upper bound. Now consider the case of the initial distribution of the example in Section 3.3.1. The variance is proportional to $10*0.13^2+5*0.28^2 = 0.56$ and thus $P_{max} = 0.44$.

In order to rectify this problem, we modify Grover's algorithm as in Figure 6. The difference between this and Grover's original algorithm is first, we do not begin with the state $|\bar{0}\rangle$ and transform it into the uniform distribution. Instead we assume some other initial distribution (such as would be the result of the pattern storage algorithm described in Section 3.2). This modification is actually suggested in [Bir98]. The second modification, which has not been suggested before, is that of step 3 in Figure 6. That is, the second state rotation operator not only rotates the phase of the desired states, but also rotates the phases of all the stored pattern states as well. The reason for this is to force the two different kinds of nondesired states to have the same phase, rather than opposite phases as in the original algorithm. After step 4 in Figure 6, then, we can consider the state of the system as the input into the normal loop of Grover's algorithm.

1. $|\psi\rangle = \hat{I}_\tau|\psi\rangle$
2. $|\psi\rangle = \hat{G}|\psi\rangle$
3. $|\psi\rangle = \hat{I}_\wp|\psi\rangle$
4. $|\psi\rangle = \hat{G}|\psi\rangle$
5. Repeat $\frac{\pi}{4}\sqrt{N}$ -2 times
6. $\quad|\psi\rangle = \hat{I}_\tau|\psi\rangle$
7. $\quad|\psi\rangle = \hat{G}|\psi\rangle$
8. Observe the system

Figure 6. Modified Grover's algorithm

With this modification of the algorithm, we can once again rework the example of Section 3.1.1, again starting with the state

$$|\psi\rangle = \tfrac{1}{\sqrt{6}}(1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1).$$

The first two steps are identical to those in the previous example:

$$|\psi\rangle \xrightarrow{\hat{I}_\tau} |\psi\rangle = \tfrac{1}{\sqrt{6}}(1,0,0,1,0,0,-1,0,0,1,0,0,1,0,0,1)$$

and

$$|\psi\rangle \xrightarrow{\hat{G}} |\psi\rangle = \tfrac{1}{2\sqrt{6}}(-1,1,1,-1,1,1,3,1,1,-1,1,1,-1,1,1,-1).$$

Now, all the states present in the original superposition are phase rotated and then all states are again rotated about average:

$$|\psi\rangle \xrightarrow{\hat{I}_\wp} |\psi\rangle = \tfrac{1}{2\sqrt{6}}(1,1,1,1,1,1,-3,1,1,1,1,1,1,1,1,1)$$

and

$$|\psi\rangle \xrightarrow{\hat{G}} |\psi\rangle = \tfrac{1}{4\sqrt{6}}(1,1,1,1,1,1,9,1,1,1,1,1,1,1,1,1).$$

Finally, we enter the loop of line 5 and have

$$|\psi\rangle \xrightarrow{\hat{I}_\tau} |\psi\rangle = \tfrac{1}{4\sqrt{6}}(1,1,1,1,1,1,\text{-}9,1,1,1,1,1,1,1,1,1)$$

for step 6 and

$$|\psi\rangle \xrightarrow{\hat{G}} |\psi\rangle = \tfrac{1}{16\sqrt{6}}(\text{-}1,\text{-}1,\text{-}1,\text{-}1,\text{-}1,\text{-}1,39,\text{-}1,\text{-}1,\text{-}1,\text{-}1,\text{-}1,\text{-}1,\text{-}1,\text{-}1,\text{-}1)$$

for step 7. Squaring the coefficients gives the probability of collapsing into the desired $|\tau\rangle = |0110\rangle$ basis state as 99% -- a significant improvement that is critical for the QuAM proposed in the next section.

4. Quantum Associative Memory

A quantum associative memory (QuAM) can now be constructed from the two algorithms of Section 3. Define $\hat{P}$ as an operator that implements the algorithm of Figure 4 for memorizing patterns described in Section 3.2. Then the operation of the QuAM can be described as follows. Memorizing a set of patterns is simply

$$|\psi\rangle = \hat{P}|\overline{0}\rangle, \tag{25}$$

with $|\psi\rangle$ being a quantum superposition of basis states, one for each pattern. Now, suppose we know n-1 bits of a pattern and wish to recall the entire pattern. We can use the modified Grover's algorithm to recall the pattern as

$$|\psi\rangle = \hat{G}\hat{I}_{\wp}\hat{G}\hat{I}_{\tau}|\psi\rangle \tag{26}$$

followed by

$$|\psi\rangle = \hat{G}\hat{I}_{\tau}|\psi\rangle \tag{27}$$

repeated T times (how to calculate T is covered in Section 4.2), where $\tau = b_1b_2b_3?$ with b_i being the value of the ith known bit. Since there are 2 states whose first three bits would match those of τ, there will be 2 states that have their phases rotated, or marked, by the $\hat{I}_\tau$ operator. Thus, with $2n$+1 neurons (qubits) the QuAM can store up to N=2^n patterns in $O(mn)$ steps and requires $O(\sqrt{N})$ time to recall a pattern (see Figure 7). This last bound is somewhat slower than desirable and may perhaps be improved with an alternative pattern recall mechanism.

Generate the initial state $|\psi\rangle = |\bar{0}\rangle$
For $m \geq p \geq 1$
    $FLIP|\psi\rangle$
    $\hat{S}^{p}|\psi\rangle$
    $SAVE|\psi\rangle$
$|\psi\rangle = \hat{I}_{\tau}|\psi\rangle$
$|\psi\rangle = \hat{G}|\psi\rangle$
$|\psi\rangle = \hat{I}_{\wp}|\psi\rangle$
$|\psi\rangle = \hat{G}|\psi\rangle$
    Repeat T times
        $|\psi\rangle = \hat{I}_{\tau}|\psi\rangle$
        $|\psi\rangle = \hat{G}|\psi\rangle$
    Observe the system

Figure 7. Algorithm for QuAM

4.1. A QuAM example

Suppose that we have a set of patterns $\wp$ = {0000, 0011, 0110, 1001, 1100, 1111}. Then using the notation of Section 3.1.1 and Equation (25) a quantum state that stores the pattern set is created as

$$|\bar{0}\rangle \xrightarrow{\hat{P}} |\psi\rangle = \tfrac{1}{\sqrt{6}}(1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1).$$

Now suppose that we want to recall the pattern whose first three bits are 011. Then $\tau = 011?$, and applying Equation (26) gives

$$|\psi\rangle \xrightarrow{\hat{I}_{\tau}} |\psi\rangle = \tfrac{1}{\sqrt{6}}(1,0,0,1,0,0,-1,0,0,1,0,0,1,0,0,1),$$

$$|\psi\rangle \xrightarrow{\hat{G}} |\psi\rangle = \tfrac{1}{2\sqrt{6}}(-1,1,1,-1,1,1,3,1,1,-1,1,1,-1,1,1,-1),$$

$$|\psi\rangle \xrightarrow{\hat{I}_{\wp}} |\psi\rangle = \tfrac{1}{2\sqrt{6}}(1,1,1,1,1,1,-3,-1,1,1,1,1,1,1,1,1),$$

and

$$|\psi\rangle \xrightarrow{\hat{G}} |\psi\rangle = \tfrac{1}{8\sqrt{6}}(1,1,1,1,1,1,17,9,1,1,1,1,1,1,1,1).$$

At this point, there is a 96.3% probability of observing the system and finding the state $|011?\rangle$. Of course there are two states that match and state $|0110\rangle$ has a 78% chance while state $|0111\rangle$ has a 22% chance. This may be resolved by a standard voting scheme and thus we have achieved our goal -- we can observe the system to see that the completion of the pattern 011 is 0110. Notice that the loop of line 10 in Figure 7 is repeated T times but that in this case it was never entered because T happens to be zero for this example.

Figure 8 shows a high-level quantum network for the QuAM. To the left of the longer wavy line, a high-level version of the storage algorithm described in Section 3.2 is represented. The ellipses indicate that this sequence is repeated for each pattern to be stored. Notice that most of the ancillary qubit lines used in the storage algorithm do not extend to the second part because they are not needed. In other words the QuAM exists only in the x register. The operators to the right of the longer wavy line show a high-level implementation of the modified Grover's algorithm described in Section 3.3. The second pair of ellipses represent the fact that the final two operators will be repeated a number of times (as determined by Equation (38) below).

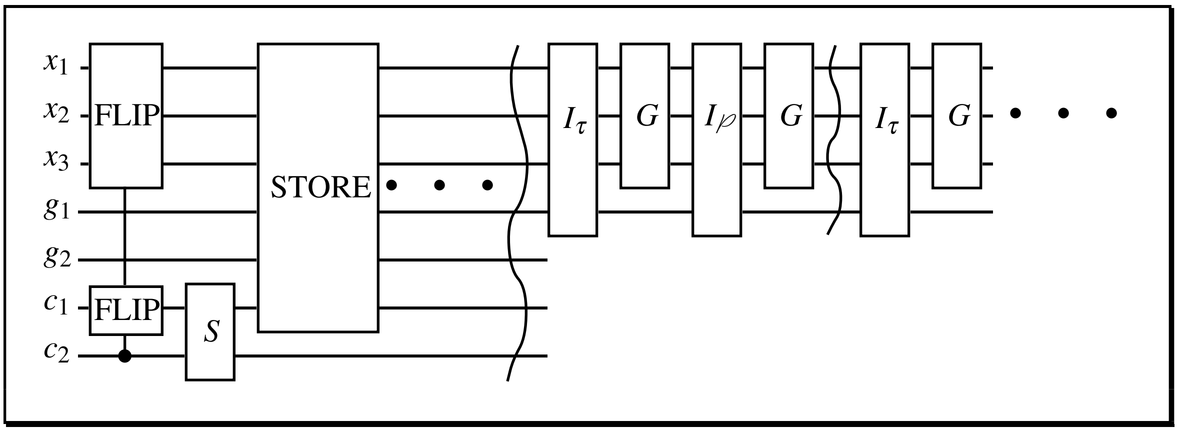

Figure 8. High-level quantum network for the QuAM

Using some concrete numbers, assume that $n = 2^4$ and $m = 2^{14}$ (we let m be less than the maximum possible 2^{16} to allow for some generalization and to avoid the contradictory patterns that would otherwise result). Then the QuAM requires $O(mn) = O(2^{18}) < 10^6$ operations to memorize the patterns and $O(\sqrt{N}) = O(\sqrt{2^{16}}) < 10^3$ operators to recall a pattern. For comparison, in [Bar96] Barenco gives estimates of how many operations might be performed before decoherence for various possible physical implementation technologies for the qubit. These estimates range from as low as 10^3 (electrons in GaAs and electron quantum dots) to as high as 10^{13} (trapped ions), so our estimates fall comfortably into this range, even near the low end of it. Further, the algorithm would require only $2n + 1 = 2*16+1 = 33$ qubits! For comparison, a classical Hopfield type network used as an associative memory has a saturation point around $0.15n$. In other words, about $0.15n$ patterns can be stored and recalled with n neurons. Therefore, with n=16 neurons, a Hopfield network can store only $0.15*16 \approx 2$ patterns. Conversely, to store 2^{14} patterns would require that the patterns be close to 110,000 bits long and that the network have that same number of neurons. So the QuAM provides significant advantages over a classical associative memory.

The QuAM also compares favorably with other quantum computational algorithms because it requires far fewer qubits to perform significant computation that appears to be impossible classically. For example, Shor's algorithm requires hundreds or thousands of qubits to perform a factorization that can not be done classically. Vedral, *et al.* give estimates for number of qubits needed for modular exponentiation, which dominates Shor's algorithm, anywhere from $7n+1$ down to $4n+3$ [Ved96]. For a 512-bit number (which RSA actually claims may not be large enough to be safe anymore, even classically), this translates into anywhere from 3585 down to 2051 qubits. As for elementary operations, they claim $O(n^3)$, which in this case would on the order of 10^8. Therefore, the algorithm presented here requires orders of magnitude fewer operations and qubits than Shor's in order to perform significant computational tasks. This is an important result since quantum computational technology is still immature -- and maintaining and manipulating the coherent superposition of a quantum system of 30 or so qubits should be attainable sooner than doing so for a system of 2000 qubits.

As mentioned in the introduction, very recently Jones and Mosca have succeeded in physically implementing Deutsch's algorithm on a nuclear magnetic resonance (NMR) quantum computer based on the pyrimidine base cytosine [Jon98]. Even more pertinent to this work, Chuang *et al*. have succeeded in physically implementing Grover's algorithm for the case $n=2$ using NMR technology on a solution of chloroform molecules [Chu98]. It is therefore not unreasonable to assume that a quantum associative memory may be implemented in the not too distant future.

4.2. Probability of success

Let N be the total number of basis states, r_1 be the number of marked states that correspond to stored patterns, r_0 be the number of marked states that do not correspond to stored patterns, and p be the number of patterns stored in the QuAM. We would like to find the average amplitude $\bar{k}$ of the marked states and the average amplitude $\bar{l}$ of the unmarked states after applying Equation (26). It can be shown that

$$k_0 = 4a - ab, \tag{28}$$

$$k_1 = 4a - ab + 1, \tag{29}$$

$$l_0 = 2a - ab, \tag{30}$$

and

$$l_1 = 4a - ab - 1. \tag{31}$$

Here k_0 is the amplitude of the spurious marked states, k_1 is the amplitude of the marked states that correspond to stored patterns, l_0 is the amplitude of the spurious unmarked states, l_1 is the amplitude of the unmarked states that correspond to stored patterns after applying Equation (26), and

$$a = \frac{2(p - 2r_1)}{N}, \tag{32}$$

and

$$b = \frac{4(p + r_0)}{N}. \tag{33}$$

A little more algebra gives the averages as

$$\bar{k} = 4a - ab + \frac{r_1}{r_0 + r_1}, \tag{34}$$

and

$$\bar{l} = -ab + \frac{2a\left(N + p - r_0 - 2r_1\right)}{N - r_0 - r_1} - \frac{\left(p - r_1\right)}{N - r_0 - r_1}. \tag{35}$$

Now we can consider this new state described by Equations (28-31) as the arbitrary initial distribution to which the results of [Bir98] can be applied. These can be used to calculate the upper bound on the accuracy of the QuAM as well as the appropriate number of times to apply Equation (27) in order to be as close to that upper bound as possible. The upper bound on accuracy is given by

$$P_{\max} = 1 - \left(N - p - r_0\right)\left|l_0 - \bar{l}\right|^2 - \left(p - r_1\right)\left|l_1 - \bar{l}\right|^2, \tag{36}$$

whereas the actual probability at a given time t is

$$P(t) = P_{\max} - \left(N - r_0 - r_1\right)\left|\bar{l}(t)\right|^2. \tag{37}$$

The first integer time step T for which the actual probability will be closest to this upper bound is given by rounding the function

$$T = \frac{\frac{\pi}{2} - \arctan\left(\frac{\bar{k}}{\bar{l}}\sqrt{\frac{r_0 + r_1}{N - r_0 - r_1}}\right)}{\arccos\left(1 - 2\frac{r_0 + r_1}{N}\right)} \tag{38}$$

to the nearest integer.

4.3. Extension to non-binary patterns

The algorithm described in Section 4.1 can handle only binary patterns. Nominal data with more than two values can be handled one of two ways: convert the multiple values into a binary representation or extend the algorithm to handle data of more than two values. In [Ven98b] a generalization of the pattern storage algorithm to more that two values is presented; if Grover's algorithm can likewise be generalized then a straightforward generalization of the QuAM to the case of nonbinary patterns can easily be conceived. This may be preferable to converting to a binary format since doing so would introduce a number of new qubits, requiring larger quantum registers which could result in degradation of pattern completion accuracy.

5. Concluding Comments

A unique view of the associative pattern completion problem is presented that allows the proposal of a quantum associative memory with exponential storage capacity. It employs simple spin-1/2 (two-state) quantum systems and represents patterns as quantum operators. This approach introduces a promising new field to which quantum computation may be applied to advantage -- that of neural networks. In fact, it is the authors' opinion that this application of quantum computation will, in general, demonstrate greater returns than its application to more traditional computational tasks (though Shor's algorithm is an obvious exception). We make this conjecture because results in both quantum computation and neural networks are by nature probabilistic and inexact, whereas most traditional computational tasks require precise and deterministic outcomes.

This paper presents a quantum computational learning algorithm that takes advantage of the unique capabilities of quantum computation to produce an important advance in the field of neural networks. In other words, the paper makes an important contribution to both the field of neural computation and to the field of quantum computation -- producing both a new neural network result and a new quantum algorithm that accomplishes something that no classical algorithm has been able to do -- creating a reliable associative memory with a capacity exponential in the length of the patterns to be stored.

The most urgently appealing future work suggested by the result of this paper is, of course, the physical implementation of the algorithm in a real quantum system. As mentioned in Sections 1 and 4, the fact that very few qubits are required for non-trivial problems together with the recent physical realization of Grover's algorithm helps expedite the realization of quantum computers performing useful computation. In the mean time, as discussed in Section 4, the time bound for recall of patterns is slower than desirable, and alternatives to Grover's algorithm for recalling the patterns are being investigated. Also, a simulation of the quantum associative memory may be developed to run on a classical computer at the cost of an exponential slowdown in the length of the patterns. Thus, association problems that are non-trivial and yet small in size will provide interesting study in simulation. We are also investigating associative recall of nonbinary patterns using spin systems higher than 1/2 (systems with more than two states). Another important area for future research is investigating further the application of quantum computational ideas to the field of neural networks -- the discovery of other quantum computational learning algorithms. Further, techniques and ideas that result from developing quantum algorithms may be useful in the development of new classical algorithms. Finally, the process of understanding and developing a theory of quantum computation provides insight and contributes to a furthering of our understanding and development of a general theory of computation.

References

[Bar96] Barenco, Adriano, "Quantum Physics and Computers", *Contemporary Physics*, vol. 37 no. 5, pp. 375-89, 1996.

[Beh96] Behrman, E.C., J. Niemel, J. E. Steck and S. R. Skinner, "A Quantum Dot Neural Network", *Proceedings of the Workshop on Physics of Computation*, Boston, pp. 22-4, November 1996.

[Ben82] Benioff, Paul, "Quantum Mechanical Hamiltonian Models of Turing Machines", *Journal of Statistical Physics*, vol. 29 no. 3, pp. 515-546, 1982.

[Bir98] Biron, David, Ofer Biham, Eli Biham, Markus Grassl and Daniel A. Lidar, "Generalized Grover Search Algorithm for Arbitrary Initial Amplitude Distribution", to appear in the *Proceedings of the 1st NASA International Conference on Quantum Computation and Quantum Communications*, February 1998.

[Boy96] Boyer, Michel, Gilles Brassard, Peter Høyer and Alain Tapp, "Tight Bounds on Quantum Searching", *Workshop on Physics and Computation*, pp. 36-43, November 1996.

[Chu98] Chuang, Isaac, Neil Gershenfeld and Mark Kubinec, "Experimental Implementation of Fast Quantum Searching", *Physical Review Letters*, vol. 80 no. 15, pp. 3408-11, April 13, 1998.

[Deu92] Deutsch, David and Richard Jozsa, "Rapid Solution of Problems by Quantum Computation", *Proceedings of the Royal Society, London A*, vol. 439, pp. 553-8, 1992.

[Deu89] Deutsch, David, “Quantum Computational Networks”, *Proceedings of the Royal Society, London A*, vol. 425, pp. 73-90, 1989.

[Deu85] Deutsch, David, “Quantum Theory, The Church-Turing Principle and the Universal Quantum Computer”, *Proceedings of the Royal Society*, London A, vol. 400, pp. 97-117, 1985.

[Fey86] Feynman, Richard P., “Quantum Mechanical Computers”, *Foundations of Physics*, vol. 16 no. 6, pp. 507-531, 1986.

[Fey65] Feynman, Richard P., R. B. Leighton and Mark Sands, *The Feynman Lectures on Physics*, vol. 3, Addison-Wesley Publishing Company, Reading Massachusetts, 1965.

[Fre82] Fredkin, Edward and Tommaso Toffoli, “Conservative Logic”, *International Journal of Theoretical Physics*, vol. 21, nos. 3/4, pp. 219-253, 1982.

[Grö88] Grössing, G. and A. Zeilinger, “Quantum Cellular Automata”, *Complex Systems*, vol. 2, pp. 197-208, 1988.

[Gro98] Grover, Lov K., “Quantum Search on Structured Problems”, to appear in the *Proceedings of the 1st NASA International Conference on Quantum Computation and Quantum Communications*, February 1998.

[Gro96] Grover, L., “A Fast Quantum Mechanical Algorithm for Database Search”, *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, ACM, New York, pp. 212-19, 1996.

[Hop82] Hopfield, J. J., “Neural Networks and Physical Systems with Emergent Collective Computational Abilities”, *Proceedings of the National Academy of Scientists*, vol. 79, pp. 2554-2558, 1982.

[Jon98] Jones, J. A. and M. Mosca, “Implementation of a Quantum Algorithm to Solve Deutsch’s Problem on a Nuclear Magnetic Resonance Quantum Computer”, *Journal of Chemical Physics*, to appear, 1998.

[Joz97] Jozsa, Richard, “Entanglement and Quantum Computation”, *Geometric Issues in the Foundations of Science*, eds. S. Hugget, L. Mason, K. P. Tod, S. T. Tsou and N. M. J. Woodhouse, Oxford University Press, 1997.

[Kos88] Kosko, Bart, “Bidirectional Associative Memories”, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, pp. 49-60, 1988.

[Per96] Perus, Mitja, “Neuro-Quantum Parallelism in Brain-Mind and Computers”, *Informatica*, vol. 20, pp. 173-83, 1996.

[Sho97] Shor, Peter, “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer”, *SIAM Journal of Computing*, vol. 26 no. 5, pp. 1484-1509, 1997.

[Sim97] Simon, D., “On the Power of Quantum Computation”, *SIAM Journal of Computation*, vol. 26 no. 5, pp. 1474-83, 1997.

[Ved97] Vedral, V., M. B. Plenio, M. A. Rippin and P. L. Knight, “Quantifying Entanglement”, *Physical Review Letters*, vol. 78 no. 12, pp. 2275-9, 1997.

[Ved96] Vedral, Vlatko, Adriano Barenco, and Artur Ekert, “Quantum Networks for Elementary Arithmetic Operations”, *Physical Review A*, vol. 54 no. 1, pp. 147-53, 1996.

[Ven98a] Ventura, Dan and Tony Martinez, “Quantum Associative Memory with Exponential Capacity”, *Proceedings of the International Joint Conference on Neural Networks*, pp. 509-13, May 1998.

[Ven98b] Ventura, Dan and Tony Martinez, “Initializing the Amplitude Distribution of a Quantum State”, submitte to *Physical Review Letters*, June 16, 1998.

[Ven98c] Ventura, Dan and Tony Martinez, "A Quantum Associative Memory Based on Grover's Algorithm", submitted to the *International Conference on Artificial Neural Networks and Genetic Algorithms*, April 1999.

[You88] Young, Nicholas, *An Introduction to Hilbert Space*, Cambridge University Press, Cambridge, England, 1988.

Chapter 12

A Quantum Computational Learning Algorithm

A wise man will hear and will increase in learning.
-- Proverbs 1:5

submitted to the
Journal of Artificial Intelligence Research, July 1998.

Abstract

An interesting classical result due to Jackson allows polynomial-time learning of the function class DNF using membership queries. Since in most practical learning situations access to a membership oracle is unrealistic, this paper explores the possibility that quantum computation might allow a learning algorithm for DNF that relies only on example queries. A natural extension of Fourier-based learning into the quantum domain is presented. The algorithm requires only an example oracle, and it runs in $O(\sqrt{2^n})$ time, a result that appears to be classically impossible. The algorithm is unique among quantum algorithms in that it does not assume *a priori* knowledge of a function and does not operate on a superposition that includes all possible basis states.

1. Introduction

The field of computational learning theory (COLT) attempts to formalize the process of inductive learning and to develop bounds for classes of functions that are learnable in some formal sense. One particular function class that has received a great deal of attention is the class of binary functions that are representable in disjunctive normal form (DNF). This function class is very expressive and has resisted attempts to produce an algorithm for learning it under the rigorous PAC-learning model often used in computational learning theory. Recently, however, Jackson has produced an algorithm that learns DNF under the uniform distribution in polynomial time with access to a membership oracle (these terms will be defined shortly). This is an impressive theoretical result that relies on two key assumptions: uniform distribution of the function examples and access to a membership oracle. While the assumption of a uniform distribution is often reasonable in practice, access to a membership oracle usually is not.

The field of quantum computation (QC) investigates the power of the unique characteristics of quantum systems used as computational machines. Early quantum computational successes have been impressive, yielding results that are classically impossible [Sho97] [Gro96] [Hog96] [Ter98]. Thus, in an attempt to extend Jackson's result so that it does not rely on a membership oracle, this paper presents a Fourier-based quantum learning algorithm. Specifically, the algorithm is designed to find the large Fourier coefficients of Boolean functions in polynomial time with access to only a classical example oracle. The algorithm is not completely successful because it fails to guarantee learning in polynomial time. However, it can be shown to learn in time $O(\sqrt{2^n})$. In contrast, classically estimating the Fourier coefficients requires $O(2^n)$ time. Therefore, although we have failed to accomplish the ultimate goal of polynomial-time learning using only an example oracle, we have nevertheless discovered a result that is both theoretically interesting (since it accomplishes something that seems classically impossible) and to some extent useful in a practical sense (since it extends the size of problems to which we can practically apply a Fourier-based learning scheme. We have, in essence, traded Jackson's impressive theoretical polynomial time bound for a practical assumption and a more modest time bound. It is also interesting to note that Grover's well-known quantum searching algorithm also provides the same $O(\sqrt{2^n})$ improvement over its classical equivalent [Gro96].

Section 2 provides a simplified overview of computational learning theory and briefly discusses work in COLT related to our results. Section 3 introduces quantum computation and some of its basic ideas and early successes. Since a familiarity with basic ideas from computational learning theory is assumed, only a few necessary remarks on the subject are provided here; quantum computation, on the other hand, is treated more carefully as it is likely that readers will be less familiar with it. Since neither of these subjects can be properly covered here, references for further study are also provided. Section 4 discusses Fourier-based learning methods in some detail. A Fourier-based, inductive quantum learning algorithm that only requires access to an example oracle is presented in Section 5. Section 5 also discusses why the algorithm will not learn the function class DNF in polynomial time and then shows that it will learn in $O(\sqrt{2^n})$ time. The paper concludes with final remarks and directions for further research in Section 6.

2. Computational Learning Theory

A rigorous approach to machine learning is traditionally traced back to Valiant [Val84], and the resulting computational learning theory has provided a formal basis for machine learning. In particular, the PAC (Probably Approximately Correct) model has yielded some nice theoretical results proving learnability of various function classes. Under this model an example oracle E for a function f may be queried for a random example of the form $x \rightarrow f(x)$ from f according to a distribution D that governs the frequency of the examples (basically this is equivalent to having access to a "large enough" training set). A function class F is termed *strongly* PAC-learnable if there exists an algorithm A such that for any $f \in F$ and for any D, A produces a hypothesis h such that $P(|h(x)-f(x)| > \varepsilon) < \delta$, and A requires at most m (where m is polynomial in the size of the input) queries of E and runs in time polynomial in m, $1/\delta$ and $1/\varepsilon$. Here ε is called the error and δ the confidence. In other words, a function class is *strongly* PAC-learnable if there exists a learning algorithm that with high probability will produce a relatively accurate hypothesis in a polynomial amount of time. Another type of learning that may be defined is *weak* learning. A function class F is *weakly* PAC-learnable if F is PAC-learnable for $\varepsilon = 1/2 - 1/p(n,r)$, where p is a fixed polynomial and r is the size of f. That is, a function class is *weakly* PAC-learnable if there exists a learning algorithm that with high probability will produce a hypothesis that is at least slightly more accurate than random guessing.

However, the class DNF (disjunctive normal form) in which functions are expressed as a disjunction of clauses, each of which consists of a conjunction of literals (binary variables or their negation), has resisted efforts to develop provably efficient algorithms for its learnability. This is unfortunate because DNF is an extremely expressive class of functions, and it would therefore be significant to prove its learnability. Until recently, DNF as a general function class has resisted all attempts to find an algorithm that will learn it in the PAC sense; however, much work has been done regarding its learnability, and some encouraging results have been obtained for restricted subclasses of DNF [Aiz91] [Blu92] [Bsh93] [Kea87]. Currently, a particularly promising approach to the unrestricted class DNF is the use of discrete Fourier analysis in the development of machine learning algorithms, including work by Goldreich and Levin [Gol89], Kushilevitz and Mansour [Kus93], and Jackson [Jac94]. Jackson's work has yielded the most encouraging results so far concerning DNF, as he is able to produce the first positive learning results for the class of unrestricted DNF. Jackson's basic approach is to first note that binary functions may be represented as a discrete Fourier expansion and that all DNF functions may be at least weakly approximated using a single Fourier basis function whose coefficient is "large." His idea is to find those large coefficients by approximating the function using examples and to combine them in such a way as to produce a good hypothesis for the function. Jackson's learning algorithm, which he calls the Harmonic Sieve, combines techniques from discrete Fourier analysis due to Goldreich and Levin [Gol89] and Kushilevitz and Mansour [Kus93] with the boosting ideas of Shapire [Sha90] and Freund [Fre90] [Fre92]. The Harmonic Sieve guarantees strong learnability of DNF with two caveats: first, the function distribution D must be uniform (actually this is relaxed somewhat, but it is still restrictive) and second, access to a membership oracle M rather than to an example oracle E is required. A membership oracle is like a black box version of the function to be learned, so that while an example oracle simply returns a random example upon being queried, a membership oracle can be queried for specific examples. In other words, when queried with x, the membership oracle must return $f(x)$. This characteristic makes a membership oracle strictly more powerful than an example oracle, and the Harmonic Sieve's dependence upon such an oracle, along with its restrictions on the distribution D, eliminates it from consideration as an algorithm for learning DNF in the PAC sense. Further, although Jackson's results are extremely impressive from a theoretical standpoint, they will likely yield very few practical results because, in general, access to a membership oracle is not realistic. On the other hand, access to an example oracle is much more realistic as this is basically equivalent to having access to a large training set.

Bshouty and Jackson [Bsh95] realized this and investigated using quantum computation to improve Jackson's work, just as is proposed here. The result was an extension to the Harmonic Sieve that depended upon a *quantum example* oracle rather than upon a *classical membership* oracle. Since the quantum example oracle is strictly less powerful than the classical membership oracle, this is a positive theoretical result. However, as the authors point out, it is unclear how to construct a quantum example oracle (other than perhaps by utilizing a classical membership oracle), and therefore their approach is again not useful in a practical sense. In contrast, the algorithm presented here goes one step further by depending only upon a *classical example* oracle, thus providing the possibility of practical algorithms for learning the class DNF.

3. Quantum Computation

Quantum computation is based upon physical principles from the theory of quantum mechanics (QM), which is in many ways counterintuitive. Yet it has provided us with perhaps the most accurate physical theory (in terms of predicting experimental results) ever devised by science. The theory is well-established and is covered in its basic form by many textbooks (see for example [Fey65]). Several necessary ideas that form the basis for the study of quantum computation are briefly reviewed here.

3.1 Linear superposition

Linear superposition is closely related to the familiar mathematical principle of linear combination of vectors. Quantum systems are described by a wave function ψ that exists in a Hilbert space [You88]. The Hilbert space has a set of states, $|\phi_i\rangle$, that form a basis, and the system is described by a quantum state $|\psi\rangle$,

$$|\psi\rangle = \sum_i c_i |\phi_i\rangle. \tag{1}$$

$|\psi\rangle$ is said to be in a linear superposition of the basis states $|\phi_i\rangle$, and in the general case, the coefficients c_i may be complex. Use is made here of the Dirac bracket notation, where the ket $|\cdot\rangle$ is analogous to a column vector, and the bra $\langle\cdot|$ is analogous to the complex conjugate transpose of the ket. In quantum mechanics the Hilbert space and its basis have a physical interpretation, and this leads directly to perhaps the most counterintuitive aspect of the theory. The counter intuition is this -- at the microscopic or quantum level, the state of the system is described by the wave function ψ, that is, as a linear superposition of all basis states (i.e. in some sense the system is in all basis states at once). However, at the macroscopic or classical level the system can be in only a single basis state. For example, at the quantum level an electron can be in a superposition of many different energies; however, in the classical realm this obviously cannot be.

3.2 Coherence and decoherence

Coherence and *decoherence* are closely related to the idea of linear superposition. A quantum system is said to be coherent if it is in a linear superposition of its basis states. A result of quantum mechanics is that if a system that is in a linear superposition of states interacts in any way with its environment, the superposition is destroyed. This loss of coherence is called decoherence and is governed by the wave function ψ. The coefficients c_i are called probability amplitudes, and $|c_i|^2$ gives the probability of $|\psi\rangle$ collapsing into state $|\phi_i\rangle$ if it decoheres. Note that the wave function ψ describes a real physical system that must collapse to exactly one basis state. Therefore, the probabilities governed by the amplitudes c_i must sum to unity. This necessary constraint is expressed as the unitarity condition

$$\sum_i |c_i|^2 = 1. \tag{2}$$

In the Dirac notation, the probability that a quantum state $|\psi\rangle$ will collapse into an eigenstate $|\phi_i\rangle$ is written $|\langle\phi_i|\psi\rangle|^2$ and is analogous to the dot product (projection) of two vectors. Consider, for example, a discrete physical variable called spin. The simplest spin system is a two-state system, called a spin-1/2 system, whose basis states are usually represented as $|\uparrow\rangle$ (spin up) and $|\downarrow\rangle$ (spin down). In this simple system the wave function ψ is a distribution over two values (up and down) and a coherent state $|\psi\rangle$ is a linear superposition of $|\uparrow\rangle$ and $|\downarrow\rangle$. One such state might be

$$|\psi\rangle = \frac{2}{\sqrt{5}}|\uparrow\rangle + \frac{1}{\sqrt{5}}|\downarrow\rangle. \tag{3}$$

As long as the system maintains its quantum coherence it cannot be said to be either spin up or spin down. It is in some sense both at once. Under the classical interpretation, of course, it must be one or the other, and when this system decoheres the result is, for example, the $|\uparrow\rangle$ state with probability

$$\left|\langle\uparrow|\psi\rangle\right|^2 = \left(\frac{2}{\sqrt{5}}\right)^2 = 0.8. \tag{4}$$

A simple two-state quantum system, such as the spin-1/2 system just introduced, is used as the basic unit of quantum computation. Such a system is referred to as a quantum bit or *qubit*, and renaming the two states $|0\rangle$ and $|1\rangle$ it is easy to see why this is so.

3.3 Operators

Operators on a Hilbert space describe how one wave function is changed into another. Here they will be denoted by a capital letter with a hat, such as $\hat{A}$, and they may be represented as matrices acting on vectors. Using operators, an eigenvalue equation can be written $\hat{A}|\phi_i\rangle = a_i|\phi_i\rangle$, where a_i is the eigenvalue. The solutions $|\phi_i\rangle$ to such an equation are called eigenstates and can be used to construct the basis of a Hilbert space as discussed in Section 3.1. In the quantum formalism, all properties are represented as operators whose eigenstates are the basis for the Hilbert space associated with that property and whose eigenvalues are the quantum allowed values for that property. It is important to note that operators in quantum mechanics must be linear operators and further that they must be unitary so that $\hat{A}^\dagger\hat{A} = \hat{A}\hat{A}^\dagger = \hat{I}$, where $\hat{I}$ is the identity operator, and $\hat{A}^\dagger$ is the complex conjugate transpose, or adjoint, of $\hat{A}$.

3.4 Interference

Interference is a familiar wave phenomenon. Wave peaks that are in phase interfere constructively (magnify each other's amplitude) while those that are out of phase interfere destructively (decrease or eliminate each other's amplitude). This is a phenomenon common to all kinds of wave mechanics from water waves to optics. The well-known double slit experiment demonstrates empirically that at the quantum level interference also applies to the probability waves of quantum mechanics. As a simple example, suppose that the wave function described in (3) is represented in vector form as

$$|\psi\rangle = \frac{1}{\sqrt{5}}\begin{pmatrix}2\\1\end{pmatrix} \tag{5}$$

and suppose that it is operated upon by an operator $\hat{O}$ described by the following matrix,

$$\hat{O} = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}. \tag{6}$$

The result is

$$\hat{O}|\psi\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\frac{1}{\sqrt{5}}\begin{pmatrix}2\\ 1\end{pmatrix} = \frac{1}{\sqrt{10}}\begin{pmatrix}3\\ 1\end{pmatrix} \tag{7}$$

and therefore now

$$|\psi\rangle = \frac{3}{\sqrt{10}}|\uparrow\rangle + \frac{1}{\sqrt{10}}|\downarrow\rangle. \tag{8}$$

Notice that the amplitude of the $|\uparrow\rangle$ state has increased while the amplitude of the $|\downarrow\rangle$ state has decreased. This is due to the wave function interfering with itself through the action of the operator -- the different parts of the wave function interfere constructively or destructively according to their relative phases just like any other kind of wave.

3.5 Entanglement

Entanglement is the potential for quantum states to exhibit correlations that cannot be accounted for classically. From a computational standpoint, entanglement seems intuitive enough -- it is simply the fact that correlations can exist between different qubits -- for example if one qubit is in the $|1\rangle$ state, another will be in the $|1\rangle$ state. However, from a physical standpoint, entanglement is little understood. The questions of what exactly it is and how it works are still not resolved. What makes it so powerful (and so little understood) is the fact that since quantum states exist as superpositions, these correlations somehow exist in superposition as well. When the superposition is destroyed, the proper correlation is somehow communicated between the qubits, and it is this "communication" that is the crux of entanglement. There are different degrees of entanglement and much work has been done on better understanding and quantifying it [Joz97] [Ved97]. It is interesting to note from a computational standpoint that quantum states that are superpositions of *only* basis states that are maximally far apart in terms of Hamming distance are those states with the greatest entanglement. For example, a superposition of only the states $|00\rangle$ and $|11\rangle$, which have a maximum Hamming spread, is maximally entangled. Finally, it should be mentioned that while interference is a quantum property that has a classical cousin, entanglement is a completely quantum phenomenon for which there is no classical analog.

3.6 Quantum algorithms

The field of quantum computation, which applies ideas from quantum mechanics to the study of computation, was introduced in the mid-1980s [Fey86] [Ben82]. For a readable introduction to quantum computation see [Bar96]; for a more rigorous treatment see for example [Deu85]. The field is still in its infancy and very theoretical but offers exciting possibilities for the field of computer science -- the most important quantum algorithms discovered to date all perform tasks for which there are no classical equivalents. For example, Deutsch's algorithm [Deu92] is designed to solve the problem of identifying whether a binary function is constant (function values are either all 1 or all 0) or balanced (the function takes an equal number of 0 and 1 values). Deutsch's algorithm accomplishes the task in order $O(1)$ time, while classical methods require $O(2^n)$ time, where n is the number of binary inputs into the function. Simon's algorithm [Sim97] is constructed to solve the following promise problem. A function $f:\{0,1\}^n \rightarrow \{0,1\}^n$ is guaranteed to be either a random 1-1 function or a periodic 2-1 function such that $f(x) = f(x \oplus s)$ for all x for some n-bit string s (where $\oplus$ denotes the bitwise exclusive OR). Here again an exponential speedup is achieved; however, admittedly, both these algorithms have been designed for artificial, somewhat contrived problems. Grover's algorithm [Gro96], on the other hand, provides a method for searching an unordered quantum database in time $O(\sqrt{2^n})$, compared to the classical lower bound of $O(2^n)$. Here is a real-world problem for which quantum computation provides performance that is classically impossible (though the speedup is less dramatic than exponential). Finally, the most well-known and perhaps the most important quantum algorithm discovered so far is Shor's algorithm for prime factorization [Sho97]. This algorithm finds the prime factors of very large numbers in polynomial time, whereas the best known classical algorithms require exponential time. The implications for the field of cryptography are profound because many cryptographic systems, including the well-known RSA system [Riv78], depend upon the problem of prime factorization requiring exponential time. These quantum algorithms take advantage of the unique features of quantum systems to provide significant speedup over classical approaches.

It is worth mentioning that very recently several different groups have succeeded in physically realizing small-scale quantum computers and implementing some of the above mentioned algorithms with them [Jon98] [Chu98]. Also, work on quantum error correction has made impressive advances [Sho95] [Cor98] crucial to the construction of larger scale quantum computers. Therefore, although some formidable technological hurdles still exist, it is not unreasonable to suggest that quantum computational systems that perform nontrivial computation are much closer to realization than was thought possible even as recently as two or three years ago. In the mean time, it is important to develop a theory of quantum computation so that when the technology does become available, it may be exploited. Further, techniques and ideas that result from developing quantum algorithms may be useful in the development of new classical algorithms. Finally, the process of understanding and developing a theory of quantum computation provides insight and contributes to a furthering of our understanding and development of a general theory of computation.

4. Fourier-Based Learning

Here is given a brief description of Fourier-based learning as it applies to our approach. In what follows, the general Fourier-based learning approach used by Kushilevitz, Mansour, Jackson and others will for simplicity be referred to as KMJ. The subject of discrete Fourier analysis is well developed and is treated only very briefly here. For a more in depth presentation see any book on Fourier analysis, for example [Mor94].

4.1 Some ideas from Fourier analysis

A bipolar-valued binary function f: $\{0,1\}^n \rightarrow \{-1,1\}$ can be represented as a Fourier expansion (the expansion used here is actually a simplified Fourier expansion called a Walsh transform)

$$f(\vec{x}) = \sum_{\vec{a}\in\{0,1\}^n} \hat{f}(\vec{a})\chi_{\vec{a}}(\vec{x}), \tag{9}$$

where the Fourier basis functions $\chi_{\vec{a}}(\vec{x})$ are defined as

$$\chi_{\vec{a}}(\vec{x}) = (-1)^{\vec{a}^{\mathrm{T}}\vec{x}} \tag{10}$$

and the Fourier coefficients being given by

$$\hat{f}(\vec{a}) = \frac{1}{2^n} \sum_{\vec{b}\in\{0,1\}^n} f(\vec{b})\chi_{\vec{b}}(\vec{a}). \tag{11}$$

Actually, in the general case (11) should use χ^* (complex conjugate of χ); however, in the simplified case considered here (bipolar rather than complex output), $\chi^* = \chi$. The KMJ method learns f in polynomial time by approximating a polynomial number of large Fourier coefficients in the context of a boosting algorithm due to Freund [Fre90] [Fre92]. In order to determine the set $\mathcal{A}$ of large coefficients, the method requires access to a membership oracle. The large coefficients, $\hat{f}(\vec{a})$ (for $\vec{a} \in \mathcal{A}$) are then approximated by

$$\tilde{\hat{f}}(\vec{a}) = \frac{1}{m} \sum_{\vec{x}\in\mathcal{T}} f(\vec{x})\chi_{\vec{x}}(\vec{a}), \tag{12}$$

with $\mathcal{T}$ being a set of m carefully chosen examples of the form $\vec{x} \rightarrow f(\vec{x})$ with $\vec{x} \in \{0,1\}^n$ and $f(\vec{x}) \in \{-1,1\}$. Finally, using the fact that the function can be at least weakly approximated with a polynomial number of large coefficients (the set $\mathcal{A}$) and using (12) to approximate those coefficients, the function f may be approximated by

$$\tilde{f}(\vec{x}) = \sum_{\vec{a}\in\mathcal{A}} \tilde{\hat{f}}(\vec{a})\chi_{\vec{a}}(\vec{x}). \tag{13}$$

We propose here a quantum algorithm that determines the set $\mathcal{A}$ using only an example oracle (i.e. a training set) rather than the membership oracle required by KMJ. In other words, the KMJ approach requires the ability to choose which examples will be used to learn the function (which in typical learning problems we can not do); on the other hand, the method that is proposed here makes no such requirement -- a standard training set suffices.

4.1.1 A FOURIER EXAMPLE

A simple example will help illustrate the concept of a Fourier expansion. Let $n = 2$ and

$$f = \begin{cases} 00 \rightarrow \ \ 1 \\ 01 \rightarrow \ \ 1 \\ 10 \rightarrow -1 \\ 11 \rightarrow \ \ 1 \end{cases}.$$

To calculate the Fourier basis functions use (10) and for example,

$$\chi_{00}(00) = -1^{(00)\binom{0}{0}} = -1^{0} = 1$$

and

$$\chi_{01}(11) = -1^{(01)\binom{1}{1}} = -1^{1} = -1.$$

The other 14 values for the 4 Fourier functions are calculated similarly. Next, calculate the Fourier coefficients using (11). For example,

$$\hat{f}(11) = \frac{1}{4}\big(f(00)\chi_{00}(11) + f(01)\chi_{01}(11) + f(10)\chi_{10}(11) + f(11)\chi_{11}(11)\big).$$

That is,

$$\hat{f}(11) = \frac{1}{4}\big((1)(1) + (1)(-1) + (-1)(-1) + (1)(1)\big) = \frac{1}{2},$$

and the other coefficients are found in the same manner. Finally, the Fourier expansion of f can be written using (9),

$$f(\vec{x}) = \hat{f}(00)\chi_{00}(\vec{x}) + \hat{f}(01)\chi_{01}(\vec{x}) + \hat{f}(10)\chi_{10}(\vec{x}) + \hat{f}(11)\chi_{11}(\vec{x}).$$

Which in this example evaluates to

$$f(\vec{x}) = \frac{1}{2}\chi_{00}(\vec{x}) - \frac{1}{2}\chi_{01}(\vec{x}) + \frac{1}{2}\chi_{10}(\vec{x}) + \frac{1}{2}\chi_{11}(\vec{x}),$$

and solving for any of the values 00, 01, 10, or 11 will result in the appropriate output for f. Now if instead of knowing f, we only have a training set such as

$$\mathcal{T} = \begin{cases} 00 \rightarrow \ \ 1 \\ 01 \rightarrow \ \ 1, \\ 10 \rightarrow -1 \end{cases}$$

then the coefficients are approximated using (12) instead of (11). For example now

$$\tilde{\hat{f}}(11) = \frac{1}{3}\big(f(00)\chi_{00}(11) + f(01)\chi_{01}(11) + f(10)\chi_{10}(11)\big),$$

which simplifies to

$$\tilde{\hat{f}}(11) = \frac{1}{3}\big((1)(1) + (1)(-1) + (-1)(-1)\big) = \frac{1}{3}.$$

The Fourier expansion is now approximated using (13) instead of (9).

$$\tilde{f}(\vec{x}) = \tilde{\hat{f}}(00)\chi_{00}(\vec{x}) + \tilde{\hat{f}}(01)\chi_{01}(\vec{x}) + \tilde{\hat{f}}(10)\chi_{10}(\vec{x}) + \tilde{\hat{f}}(11)\chi_{11}(\vec{x}),$$

which simplifies to

$$\tilde{f}(\vec{x}) = \frac{1}{3}\chi_{00}(\vec{x}) - \frac{1}{3}\chi_{01}(\vec{x}) + \frac{3}{3}\chi_{10}(\vec{x}) + \frac{1}{3}\chi_{11}(\vec{x}).$$

Now solving for 00, 01, or 10 will give 4/3, 4/3, and -4/3 respectively. While these are not the correct values, they are definitely the correct sign. On the other hand, solving for 11 results in 0, which is equivalent to "don't know".

4.2 Matrix formulation

The first step is to reformulate the Fourier equations in matrix form. Note that the $\chi_{\vec{a}}$ can be considered vectors in a 2^n-dimensional space indexed by $\vec{x} \in \{0,1\}^n$. These $\chi_{\vec{a}}$ form an orthonormal basis for the function space with inner product

$$\langle \chi_{\vec{a}}, \chi_{\vec{b}} \rangle = E_x\big[\chi_{\vec{a}}(\vec{x})\chi_{\vec{b}}(\vec{x})\big] = \begin{cases} 1 & \text{if } \vec{a} = \vec{b} \\ 0 & \text{otherwise} \end{cases}. \tag{14}$$

Again, in the general case one of the χ in (14) should be χ^*, but for the special case of bipolar outputs $\chi^* = \chi$. Now let $\boldsymbol{B}$ be the matrix formed by taking the $\chi_{\vec{a}}$ as the rows. Because of the orthonormality of the basis, the columns are also formed by the $\chi_{\vec{a}}$. Also, define f as a vector in an 2^n-dimensional space indexed by $\vec{x} \in \{0,1\}^n$ such that

$$f_{\vec{x}} = f(\vec{x}). \tag{15}$$

Then

$$\frac{1}{2^n}\boldsymbol{B}f = \hat{f} \tag{16}$$

gives the Fourier coefficients as a vector in a 2^n-dimensional space indexed by $\vec{a} \in \{0,1\}^n$ so that

$$\hat{f}_{\vec{a}} = \hat{f}(\vec{a}). \tag{17}$$

To evaluate $\chi_{\vec{a}}(\vec{x})$, define y as the 2^n-vector indexed by $\vec{b} \in \{0,1\}^n$ such that

$$y_{\vec{b}} = \begin{cases} 1 & \text{if } \vec{b} = \vec{x} \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

and calculate

$$\boldsymbol{B}y = \chi. \tag{19}$$

Now χ is a vector in a 2^n-dimensional space indexed by $\vec{b} \in \{0,1\}^n$, the $\vec{a}$th element of which is equal to $\chi_{\vec{a}}(\vec{x})$. Finally, the Fourier representation of f is given as

$$f(\vec{x}) = (\boldsymbol{B}y)^{\dagger}(\frac{1}{2^n}\boldsymbol{B}f). \tag{20}$$

Following the thinking of KMJ, f may be approximated by approximating the Fourier coefficients using a set of m examples, $\mathcal{T}$. Define $\tilde{f}$ as a 2^n-vector indexed by $\vec{x} \in \{0,1\}^n$ such that

$$\tilde{f}_{\vec{x}} = \begin{cases} f(\vec{x}) & \text{if } \vec{x} \in \mathcal{T} \\ 0 & \text{otherwise} \end{cases}. \tag{21}$$

Then the Fourier representation of the approximate function is

$$\tilde{f}(\vec{x}) = (\boldsymbol{B}y)^{\dagger}(\frac{1}{m}\boldsymbol{B}\tilde{f}). \tag{22}$$

Now, using linear algebra gives

$$\tilde{f}(\vec{x}) = (\boldsymbol{B}y)^{\dagger}(\frac{1}{m}\boldsymbol{B}\tilde{f}) = \frac{1}{m}y^{\dagger}\boldsymbol{B}^{\dagger}\boldsymbol{B}\tilde{f} = \frac{2^n}{m}y^{\dagger}\tilde{f} = \begin{cases} \frac{2^n}{m}f(\vec{x}) & \text{if } \vec{x} \in \mathcal{T} \\ 0 & \text{otherwise} \end{cases}. \tag{23}$$

Recall that $f(\vec{x})$ = {-1,1} and can never equal 0. Therefore, approximating f by approximating all 2^n Fourier coefficients results in memorization of the training set (up to the sign of the output) and no generalization whatsoever on new inputs. As a consequence of this, it is clear that the generalization inherent in Fourier-based learning methods is due at least in part to the fact that not all the Fourier coefficients are used in the approximation.

4.2.1 A MATRIX VERSION OF THE EXAMPLE

For clarity, the example of Section 4.1.1 is repeated here using the matrix formulation. According to (15) the function f is now written as a vector,

$$f_{\vec{x}} = \begin{pmatrix} 1 \\ 1 \\ -1 \\ 1 \end{pmatrix}.$$

The matrix $\boldsymbol{B}$ is

$$\boldsymbol{B}=\begin{pmatrix}1&1&1&1\\1&-1&1&-1\\1&1&-1&-1\\1&-1&-1&1\end{pmatrix}$$

and using (16) the Fourier expansion of *f* is now written

$$\frac{1}{4}\boldsymbol{B}f=\frac{1}{4}\begin{pmatrix}1&1&1&1\\1&-1&1&-1\\1&1&-1&-1\\1&-1&-1&1\end{pmatrix}\begin{pmatrix}1\\1\\-1\\1\end{pmatrix}=\frac{1}{2}\begin{pmatrix}1\\-1\\1\\1\end{pmatrix}.$$

Now considering the case where the function is unknown but a training set is available, the approximate function represented by the training set is written as a vector using (21),

$$\tilde{f}_{\vec{x}}=\begin{pmatrix}1\\1\\-1\\0\end{pmatrix},$$

and the approximate Fourier transform is obtained from (22),

$$\frac{1}{3}\boldsymbol{B}\tilde{f}=\frac{1}{3}\begin{pmatrix}1&1&1&1\\1&-1&1&-1\\1&1&-1&-1\\1&-1&-1&1\end{pmatrix}\begin{pmatrix}1\\1\\-1\\0\end{pmatrix}=\frac{1}{3}\begin{pmatrix}1\\-1\\3\\1\end{pmatrix}.$$

Actually, (22) gives not just the approximate Fourier transform for *f*, but the method for calculating the individual values for the approximate expansion. For example, to calculate the approximate value for 11, use (18) to get

$$y=\begin{pmatrix}0\\0\\0\\1\end{pmatrix}.$$

Then by (22)

$$f(\vec{x})=\left(\begin{pmatrix}1&1&1&1\\1&-1&1&-1\\1&1&-1&-1\\1&-1&-1&1\end{pmatrix}\begin{pmatrix}0\\0\\0\\1\end{pmatrix}\right)^{\dagger}\frac{1}{3}\begin{pmatrix}1&1&1&1\\1&-1&1&-1\\1&1&-1&-1\\1&-1&-1&1\end{pmatrix}\begin{pmatrix}1\\1\\-1\\0\end{pmatrix}=\begin{pmatrix}1&-1&-1&1\end{pmatrix}\frac{1}{3}\begin{pmatrix}1\\-1\\3\\1\end{pmatrix}=0,$$

just as in the example of Section 4.1.1 and as proved in (23).

4.3 Quantum formulation

The next step is to extend the vector representation into the quantum domain. Given a set of n (where n is the length of the binary input $\vec{x}$ to f) qubits whose basis states, $|\vec{x}\rangle$, correspond to all the different values for $\vec{x}$, define

$$|f\rangle = \sum_{\vec{x}\in\{0,1\}^n} c_{\vec{x}}|\vec{x}\rangle, \tag{24}$$

where the amplitudes

$$c_{\vec{x}} = f(\vec{x}) = \{-1,1\} \tag{25}$$

are properly normalized according to (2), as a quantum state of the n qubits that describes the function f. The domain of f is encoded in the state labels and the range of the function in the phases of the amplitudes. Similarly, given a set of examples $\mathcal{T}$, define

$$\left|\tilde{f}\right\rangle = \sum_{\vec{x}\in\mathcal{T}} c_{\vec{x}}|\vec{x}\rangle, \tag{26}$$

where again the amplitudes are defined as in (25). This quantum state of the n qubits describes the partial function $\tilde{f}$. Next, define the operator $\hat{B}$ as the matrix $\boldsymbol{B}$ and note that it acts on a quantum state, transforming it from the $|\vec{x}\rangle$ basis to the Fourier basis, $\chi_{\vec{a}}$. Now that transformation takes the form

$$\hat{B}|f\rangle = \left|\hat{f}\right\rangle, \tag{27}$$

or

$$\hat{B}\left|\tilde{f}\right\rangle = \left|\hat{\tilde{f}}\right\rangle, \tag{28}$$

slightly abusing the hat notation by using it to symbolize both an operator and the quantum state vector associated with the Fourier expansion of f. Similar to (19)

$$\hat{B}|\vec{x}\rangle = |\chi\rangle \tag{29}$$

results in a quantum state that contains in its amplitudes the values of all Fourier basis functions for the input $\vec{x}$. Finally,

$$f(\vec{x}) = \left\langle \hat{B}\vec{x} \middle| \hat{B}f \right\rangle \tag{30}$$

represents the full function and

$$\tilde{f}(\vec{x}) = \left\langle \hat{B}\vec{x} \middle| \hat{B}\tilde{f} \right\rangle \tag{31}$$

represents an approximation of the function from $\mathcal{T}$. Notice that we are still dealing with quantum amplitudes, and therefore the result of the calculation is not directly available to us. As it turns out, the quantum system will not be used to calculate $f(\vec{x})$ but only to determine the relative amplitudes of the Fourier coefficients -- those coefficients that are large will have "large" amplitudes.

4.3.1 A QUANTUM EXAMPLE

Finally, the example of Section 4.1.1 is repeated using the quantum formulation. This example is particularly interesting in that it emphasizes the difference of this quantum formulation from the previous two, indicating that we can carry the analogy only so far. According to (24) the function f is now written as a quantum superposition,

$$|f\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle - \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle.$$

The operator $\hat{B}$ is

$$\hat{B} = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

and using (27) the Fourier expansion of f is now written

$$\hat{B}|f\rangle = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}\frac{1}{2}\begin{pmatrix} 1 \\ 1 \\ -1 \\ 1 \end{pmatrix} = \frac{1}{2}|00\rangle - \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle.$$

Now considering the case where the function is unknown but a training set is available, the approximate function represented by the training set is written as a quantum superposition using (26),

$$|f\rangle = \frac{1}{\sqrt{3}}|00\rangle + \frac{1}{\sqrt{3}}|01\rangle - \frac{1}{\sqrt{3}}|10\rangle,$$

and the approximate Fourier transform is obtained from (28),

$$\hat{B}\left|\tilde{f}\right\rangle = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}\frac{1}{\sqrt{3}}\begin{pmatrix} 1 \\ 1 \\ -1 \\ 0 \end{pmatrix} = \frac{1}{2\sqrt{3}}|00\rangle - \frac{1}{2\sqrt{3}}|01\rangle + \frac{3}{2\sqrt{3}}|10\rangle + \frac{1}{2\sqrt{3}}|11\rangle.$$

Note that the amplitudes are not the exact Fourier coefficients now, but are instead proportional to them. This is because of the unique statistical nature of quantum systems. It is also important to mention a subtle point here about using quantum computation to approximate the Fourier coefficients. Because of the normalization condition (2) we can not use the appropriate 1/m normalization of Equations (12) and (22). The result of this is that Equation (31) is not really *approximating* the Fourier coefficients of the original 2-valued function f:$\{0,1\}^n \rightarrow \{-1,1\}$ but rather is *computing* the Fourier coefficients of a 3-valued function $\tilde{f}$:$\{0,1\}^n \rightarrow \{-1,0,1\}$, which has the same "large" coefficients as f, except that they are normalized by an exponential factor. Fortunately, we will not need to use the quantum system to approximate the large coefficients but only to indicate which *are* the large coefficients. Still, this important difference will affect the efficiency of the algorithm, and we will address the problem it creates in Section 5.1. Finally, using (30) and (31) gives us the ability to calculate function values and approximate function values, respectively. For example, to calculate the value for the binary string 11, use (30) to get

$$f(\vec{x}) = \left\langle \hat{B}\vec{x} \middle| \hat{B}f \right\rangle = \frac{1}{2}\begin{pmatrix} 1 & -1 & -1 & 1 \end{pmatrix}\frac{1}{2}\begin{pmatrix} 1 \\ -1 \\ 1 \\ 1 \end{pmatrix} = 1,$$

and use (31) to get

$$\tilde{f}(\vec{x}) = \left\langle \hat{B}\vec{x} \middle| \hat{B}\tilde{f} \right\rangle = \frac{1}{2}\begin{pmatrix} 1 & -1 & -1 & 1 \end{pmatrix} \frac{1}{2\sqrt{3}} \begin{pmatrix} 1 \\ -1 \\ 3 \\ 1 \end{pmatrix} = 0$$

just as in the previous examples and as proved in (23). We might also mention here that although mathematically these equations yield the correct values, from a physical standpoint, they are impossible to implement. Fortunately again, we do not need to be able to implement (30) or (31), but instead just need to be able to implement (28), which is physically realizable.

In the KMJ algorithm, the large coefficients are approximated using the instances in $\mathcal{T}$ as in (12) but recall that deciding which *are* the large coefficients first requires a membership oracle. The goal of the quantum formulation of the approximate Fourier expansion in (31) is to determine which are the large coefficients using only the training set $\mathcal{T}$ instead of using a membership oracle. The approach is to construct and observe the quantum system of Equation (28). The intuitive idea is that since the large Fourier coefficients will be represented in this quantum state as amplitudes with "large" magnitude, they will have a "large" probability of being observed. Since observing the system will cause it to collapse to a single basis state corresponding to a single Fourier basis function, the process must be repeated and statistics kept for each basis state until a statistically significant measure of the relative amplitudes of the coefficients is achieved. How many times this observation process must be repeated to find a large coefficient will depend on how many large coefficients there are and on their relative magnitudes. This turns out to be the catch. As mentioned in 4.3.1, because of the nature of quantum systems, it is impossible to directly calculate Equation (12). Instead, when using a quantum system the results of (12) are in essence multiplied by a normalization constant.

5. Finding a Large Coefficient by Quantum Fourier Sampling

Blum *et al*. have shown that under the uniform distribution, there exists a Fourier basis function that weakly approximates any DNF function *f* [Blu94]. In other words, there exists a Fourier basis function, $\hat{f}_W$, that agrees with *f* on at least 1-ε inputs, where

$$\varepsilon = \frac{1}{2} - \frac{1}{p(n,s)} \tag{32}$$

with *p* being some fixed polynomial, *n* being the number of inputs to *f*, and *s* being the size of the representation of *f*. This implies that the magnitude of the Fourier coefficient for the weak approximator is

$$\left|\hat{f}_W\right| = O\left(\frac{1}{p(n,s)}\right), \tag{33}$$

and using Chernoff bounds [Hag89] this can be closely approximated using a polynomial number *m* of examples drawn randomly from an example oracle, if we know which of the 2^n basis functions it is. In other words, since this coefficient is relatively large, if we had some way of sampling the Fourier coefficient distribution, it should be readily identifiable with a polynomial number of samples. Following this line of thinking, the results developed in Section 4 suggest a simple algorithm for finding this large Fourier coefficient which is presented in Figure 1.

Repeat until confidence > δ
    form $\left|\tilde{f}\right\rangle$, a quantum superposition representing the training set
    compute the Fourier transform on the superposition using the operator $\hat{B}$
    observe the system to produce a random sample of the coefficient distribution
Approximate the most commonly observed coefficient classically

Figure 1. Algorithm for finding large Fourier coefficients

There are four ingredients for the algorithm, the first three of which are performed on a quantum computer while the fourth is performed on a classical computer . In order for the algorithm to run in polynomial time, all of them must be shown to be polynomial. Specifically, we must address: 1) The construction of the state $\left|\tilde{f}\right\rangle$; 2) The implementation of the operator $\hat{B}$; 3) The number of times this process must be repeated in order to identify a large coefficient; and 4) The classical approximation of that coefficient. Constructing the state $\left|\tilde{f}\right\rangle$ is nontrivial and a method for doing so in $O(mn)$ time is detailed in [Ven98]. Implementing $\hat{B}$ turns out to be extremely easy on a quantum computer, and it is in fact the basis of most quantum algorithms discovered to date. Computing the Walsh transform of a quantum state is accomplished simply by applying the elementary quantum operator

$$\hat{H} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{34}$$

to each qubit in parallel (see for example [Gro96]). In other words, $\hat{B} = \hat{H} \otimes \cdots \otimes \hat{H}$, where $\otimes$ is the tensor product (direct matrix product) and $\hat{H}$ appears in the product n times, one for each input. Approximating a coefficient is easy on a classical computer and can be accomplished using Equation (12) and a polynomial number of function examples.

5.1 A quantum quirk in estimating coefficients

This leaves the question of how many times we must repeat the Fourier sampling process in order to find, with high probability, a large coefficient, and in fact, this is a difficult point. To see why, consider the following. The unitarity condition of Equation (2) requires that the squares of the coefficients in a quantum state sum to 1. For convenience we reproduce the equation here, recalling that the *quantum* coefficients c_i are within a multiplicative constant of being equal to the *Fourier* coefficients $\hat{f}(\vec{a})$.

$$\sum_i \left|c_i\right|^2 = 1. \tag{35}$$

As mentioned in 4.3.1, what the Fourier sampling algorithm is really doing is not approximating the Fourier coefficients of $f:\{0,1\}^n \to \{-1,1\}$ but rather is calculating the Fourier coefficients of $\tilde{f}:\{0,1\}^n \to \{-1,0,1\}$. Another way to look at this is to say that the Fourier sampling algorithm *is* approximating the coefficients of $f:\{0,1\}^n \to \{-1,1\}$ but with a normalization constant of $1/\sqrt{2^n}$ rather than the $1/m$ required by Equation (12). (Either viewpoint will produce the same result.) Because of this, in contrast to the nice polynomial sized Fourier coefficient given in (33), the size of the corresponding quantum coefficient c_W, is

$$c_W = O\left(\frac{\sqrt{m/2^n}}{p(n,s)}\right) = O\left(\sqrt{\frac{m}{2^n}}\right), \tag{36}$$

and its square is thus

$$(c_W)^2 = O\left(\frac{m/2^n}{p^2(n,s)}\right) = O\left(\frac{m}{2^n}\right). \tag{37}$$

Therefore Equations (35-37) require that the number C of nonzero coefficients be at least

$$1 = C\left[O\left(\sqrt{\frac{m}{2^n}}\right)\right]^2 \Rightarrow C = O\left(\frac{2^n}{m}\right). \tag{38}$$

Also, the smallest possible quantum coefficient, c_S is

$$c_S = \frac{1}{\sqrt{2^n}}. \tag{39}$$

Therefore, if the number of examples m is significantly less than 2^n, which will be the case if we require a polynomial-time algorithm, then there are an exponential number of nonzero coefficients in the quantum state with the largest only a factor of $\sqrt{m}$ larger than the smallest. (When the system is observed, this difference is magnified to a factor of m; however this is not significant in the end result.) This means that $O(2^n)$ samples of the amplitude distribution (obtained by observing the quantum system) will be required to discriminate the "large" coefficients from the "small" ones.

Another way to look at this is to say that the vector representing the function is very sparse. As a result, it correlates well with almost every Fourier basis function and therefore, the "large" coefficients are not very much larger than the "small" ones. In fact, as mentioned above, this is unavoidable in a quantum system because of the requirement that all operators be unitary. Therefore, the quantum system is attempting to calculate information about an exponential number of Fourier coefficients using only a polynomial number of function examples. Because of the unitarity requirement this polynomial amount of information gets normalized by an exponential instead of the polynomial necessary for closely approximating the coefficients as in (12). Because of the normalization inherent in unitary quantum processes, it is unlikely that a quantum algorithm can be developed to overcome this problem. However, this is certainly not a proof that it is impossible.

There is an equally valid classical analysis based upon Parseval's identity that will yield the same result, though some of the intermediate results are different by $O(\sqrt{\cdot})$.

5.2 A non-classical result -- finding large coefficients in $O(\sqrt{2^n})$ time

If the quantum algorithm of Figure 1 requires exponential time to identify the large quantum coefficients, it is certainly no better than a classical algorithm that simply approximates each Fourier coefficient using Equation (12) and thus also requires $O(2^n)$ time. However, the algorithm actually requires only $O(\sqrt{2^n})$ steps in order to identify the large quantum coefficients and thus still represents a significant improvement over the classical approach. To see why, choose the number of examples m drawn from an example oracle to be

$$m = \sqrt{2^n}. \tag{40}$$

Then by Equation (36) the quantum coefficient c_W which corresponds to the Fourier function that weakly approximates f is

$$c_W = O\left(\frac{\sqrt{m/2^n}}{p(n,s)}\right) = O\left(\frac{\sqrt{\sqrt{2^n}/2^n}}{p(n,s)}\right) = O\left(\frac{1}{2^{n/4}}\right), \tag{41}$$

and using Equation (37) its square is

$$(c_W)^2 = O\left(\frac{m/2^n}{p^2(n,s)}\right) = O\left(\frac{\sqrt{2^n}/2^n}{p^2(n,s)}\right) = O\left(\frac{1}{\sqrt{2^n}}\right). \tag{42}$$

Using Equation (38) the number C of nonzero coefficients is now at least

$$1 = C\left[O\left(\frac{1}{2^{n/4}}\right)\right]^2 \Rightarrow C = O\left(\frac{1}{\sqrt{2^n}}\right). \tag{43}$$

Since the smallest possible quantum coefficient c_S is still give by equation (39), we now have a situation in which there are still an exponential number of nonzero coefficients in the quantum state, but now the largest is a factor of $2^{n/4}$ larger than the smallest. When the system is observed, this difference is magnified to a factor of $\sqrt{2^n}$, and therefore $O(\sqrt{2^n})$ samples of the amplitude distribution will suffice to identify the large coefficient.

5.3 Analysis and discussion

Using some concrete numbers, assume that $n = 30$ and that therefore $m = \sqrt{2^{30}} = 2^{15} = 32768$, numbers which can represent a very non-trivial learning task. Then the algorithm requires $O(2^{15}) < 10^5$ operations. For comparison, [Bar96] gives estimates of how many operations might be performed before decoherence for various possible physical implementation technologies for the qubit. These estimates range from as low as 10^3 (electron GaAs and electron quantum dots) to as high as 10^{13} (trapped ions), so our example falls comfortably into this range, even near the low end of it. Further, the algorithm would require only 61 qubits. (Although it appears that the algorithm presented here requires only n qubits, the algorithm depends on a method for representing the training set as a quantum state. As mentioned before, an explicit algorithm for constructing such a quantum state exists [Ven98]; however it requires $2n+1$ qubits.)

In contrast, Shor's algorithm requires hundreds or thousands of qubits to perform an interesting factorization. For example, [Ved96] gives estimates for the number of qubits needed for modular exponentiation, which dominates Shor's algorithm -- anywhere from $7n+1$ down to $4n+3$. For a 512 bit number (which RSA actually claims may not be large enough to be safe anymore, even classically), this translates into anywhere from 3585 down to 2051 qubits. As for elementary operations, they claim $O(n^3)$, which would be in this case $O(10^8)$. Therefore, the algorithm presented here requires orders of magnitude fewer operations and qubits than Shor's in order to perform significant computational tasks. This is an important result since quantum computational technology is still immature, and maintaining and manipulating the coherent superposition of a quantum system of 60 qubits should be attainable sooner than doing so for a system of 2000 qubits. In this sense, the algorithm compares nicely with other quantum algorithms.

As mentioned earlier, the algorithm also compares favorably with classical approaches to finding large coefficients. Jackson's Harmonic Sieve runs in polynomial time; however, as discussed in Section 2 this algorithm requires a membership oracle, or black box access to the function f that we are trying to learn, in order to find the large coefficients. In cases where this is not realistic, the algorithm presented here provides an alternative that requires only an example oracle. There are no known classical algorithms for finding large coefficients that use only an example oracle and run any faster than $O(2^n)$, so again the algorithm presented here represents a significant improvement.

6. Concluding Comments

This paper presents a quantum computational learning algorithm that takes advantage of the unique capabilities of quantum computation to produce a significant result in both the field of computational learning and the field of quantum computation. The main result of the paper is a quantum computational learning algorithm for learning the function class DNF over the uniform distribution using only an example oracle in time significantly faster than any known classical algorithm operating under the same constraints. In other words, the paper makes an important contribution to both the field of computational learning theory and to the field of quantum computation -- producing both a new learning theoretic result and a new quantum algorithm that accomplishes something that no classical algorithm has been able to do. Further, the algorithm is unique among quantum computational algorithms as it does not assume *a priori* knowledge of a function f and does not operate on a quantum superposition that includes all possible basis states.

Further, the paper introduces a promising new field to which quantum computation may be applied to advantage -- that of computational learning. In fact, it is the authors' opinion that this application of quantum computation will, in general, demonstrate much greater returns than its application to more traditional computational tasks (though Shor's algorithm is an obvious exception). We make this conjecture because results in both quantum computation and computational learning are by nature probabilistic and inexact, whereas most traditional computational tasks require precise and deterministic outcomes.

One obvious area for future work is, of course, the physical implementation of the algorithm in a real quantum system. As mentioned in Sections 3.6 and 5.3, the fact that this algorithm requires very few qubits for non-trivial problems combined with the recent advances in quantum technology suggests that the realization of quantum computers performing useful computation may be possible in the near future. A second important topic for future work is to determine whether or not the improvement obtained here over classical methods is optimal. If it is not, future work may produce further gains. In the mean time, a simulation of the quantum algorithm is being developed to run on a classical computer at the cost of an exponential slowdown in the size of the learning problem. Thus, learning problems that are non-trivial and yet small in size will provide interesting study in simulation. Various methods of using and combining the large Fourier coefficients are currently being investigated. One method, of course, is to use the boosting method that Jackson uses in his Harmonic Sieve, and we are considering other methods of combination as well. We are also investigating using higher spin systems for learning nonbinary problems. Another obvious and important area for future research is investigating further the application of quantum computational ideas to the fields of computational learning theory and machine learning -- the discovery of other quantum computational learning algorithms.

As a final comment, Grover's well-known algorithm for quantum search also provides a $O(\sqrt{2^n})$ improvement over classical approaches to the same problem. Grover's result is particularly important because in the case of searching an unordered database, the $O(2^n)$ bound for classical approaches is tight -- there is no classical way to do better. In the case of learning DNF using only an example oracle, it is still an open problem whether or not a classical algorithm exists that is better than $O(2^n)$; however, it is generally suspected that no such algorithm exists. This provides fuel for speculation that perhaps there exists a quantum information theoretic law that bounds the improvement of quantum algorithms over classical algorithms by $O(\sqrt{2^n})$. According to Shor's paper, the best classical algorithm for prime factorization of an integer that can be represented in n bits runs in time $O(g)$, where $g = e^{n^{1/3}(\log n)^{2/3}}$ [Len93]. Since $g = O(\sqrt{2^n})$ but $g \neq \Theta(\sqrt{2^n})$, Shor's algorithm in some sense does not achieve as much of a speedup as do Grover's algorithm and the algorithm presented here.

References

[Aiz91] Aizenstein, H. and L. Pitt, "Exact Learning of Read-Twice DNF formulas", *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pp. 170-9, 1991.

[Bar96] Barenco, Adriano, "Quantum Physics and Computers", *Contemporary Physics*, vol. 37 no. 5, pp. 375-89, 1996.

[Ben82] Benioff, Paul, "Quantum Mechanical Hamiltonian Models of Turing Machines", *Journal of Statistical Physics*, vol. 29 no. 3, pp. 515-546, 1982.

[Blu94] Blum, A., M. Furst, J. Jackson, M. Kearns, Y. Mansour and S. Rudich, "Weakly Learning DNF and Characterizing Statistical Query Learning Using Fourier Analysis", *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pp. 253-62, 1994.

[Blu92] Blum, A. and S. Rudich, "Fast Learning of *k*-term DNF formulas with Queries", *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pp. 382-9, 1992.

[Bsh95] Bshouty, Nader H. and Jeffrey Jackson, "Learning DNF Over the Uniform Distribution Using A Quantum Example Oracle", *Proceedings of the 8th Annual Conference on Computational Learning Theory (COLT95)*, Santa Cruz, California, pp. 118-127, 1995.

[Bsh93] Bshouty, Nader H., "Exact Learning via the Monotone Theory", *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pp. 302-11, 1993.

[Chu98] Chuang, Isaac, Neil Gershenfeld and Mark Kubinec, "Experimental Implementation of Fast Quantum Searching", *Physical Review Letters*, vol. 80 no. 15, pp. 3408-11, April 13, 1998.

[Cor98] Cory, David G. *et al.*, "Experimental Quantum Error Correction", *LANL e-print archive*, quant-ph/9802018, http://xxx.lanl.gov, 1998.

[Deu92] Deutsch, David and Richard Jozsa, "Rapid Solution of Problems by Quantum Computation", *Proceedings of the Royal Society, London A*, vol. 439, pp. 553-8, 1992.

[Deu85] Deutsch, David, "Quantum Theory, The Church-Turing Principle and the Universal Quantum Computer", *Proceedings of the Royal Society*, London A, vol. 400, pp. 97-117, 1985.

[Fey86] Feynman, Richard P., "Quantum Mechanical Computers", *Foundations of Physics*, vol. 16 no. 6, pp. 507-531, 1986.

[Fey65] Feynman, Richard P., R. B. Leighton and Mark Sands, *The Feynman Lectures on Physics*, vol. 3, Addison-Wesley Publishing Company, Reading Massachusetts, 1965.

[Fre92] Freund, Y., "An Improved Boosting Algorithm and its Implications on Learning Complexity", *Proceedings of the 5rd Annual Workshop on Computational Learning Theory*, pp. 391-98, 1992.

[Fre90] Freund, Y., "Boosting a Weak Learning Algorithm by Majority Voting", *Proceedings of the 3rd Annual Workshop on Computational Learning Theory*, pp. 202-216, 1990.

[Gol89] Goldreich, O. and L. A. Levin, "A Hard-Core Predicate for All One-Way Functions", *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pp. 25-32, 1989.

[Gro96] Grover, L., "A Fast Quantum Mechanical Algorithm for Database Search", *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, ACM, New York, pp. 212-19, 1996.

[Hag89] Hagerup, Torben and Christine Rub, "A Guided Tour to Chernoff Bounds", *Information Processing Letters*, vol. 33, pp. 305-8, 1989.

[Hog96] Hogg, Tad, "Quantum Computing and Phase Transitions in Combinatorial Search" *Journal of Artificial Intelligence Research*, vol. 4, pp. 91-128, 1996.

[Jac94] Jackson, Jeff, "An Efficient Membership-Query Algorithm for Learning DNF with Respect to the Uniform Distribution", *Proceedings of the 35th Symposium on the Foundations of Computer Science*, pp. 42-53, 1994. An extended version will appear in the *Journal of Computer and System Sciences*.

[Jon98] Jones, J. A. and M. Mosca, "Implementation of a Quantum Algorithm to Solve Deutsch's Problem on a Nuclear Magnetic Resonance Quantum Computer", *Journal of Chemical Physics*, to appear, 1998.

[Joz97] Jozsa, Richard, "Entanglement and Quantum Computation", *Geometric Issues in the Foundations of Science*, eds. S. Hugget, L. Mason, K. P. Tod, S. T. Tsou and N. M. J. Woodhouse, Oxford University Press, 1997.

[Kea87] Kearns, M., M. Li, L. Pitt and L. Valiant, "On the Learnability of Boolean Formula", *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pp. 285-95, 1987.

[Kus93] Kushilevitz, Eyal and Yishay Mansour, "Learning Decision Trees Using the Fourier Spectrum", *SIAM Journal of Computing*, vol. 22 no. 6, pp. 1331-48, 1993.

[Len93] Lenstra, A. K. and H. W. Lenstra, Jr., eds., "The Development of the Number Field Sieve", *Lecture Notes in Mathematics*, vol. 1554, 1993.

[Mor94] Morrison, Norman, *Introduction to Fourier Analysis*, Wiley, New York, 1994.

[Riv78] Rivest, R. L., A. Shamir and L. Adleman, "A Method of Obtaining Digital Signatures and Public-key Cryptosystems", *Communications of the ACM*, vol. 21, pp. 120-6, 1978.

[Sha90] Shapire, R. E., "The Strength of Weak Learnability", *Machine Learning*, vol. 5, pp. 197-227, 1990.

[Sho97] Shor, Peter, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", *SIAM Journal of Computing*, vol. 26 no. 5, pp. 1484-1509, 1997.

[Sho95] Shor, Peter, “Scheme for Reducing Decoherence in Quantum Computer Memory”, *Physical Review A*, vol. 52 no. 4, pp. 2493-6, pp. 1995.

[Sim97] Simon, D., “On the Power of Quantum Computation”, *SIAM Journal of Computation*, vol. 26 no. 5, pp. 1474-83, 1997.

[Ter98] Terhal, Barbara. M. and John. A. Smolin, “Single Quantum Querying of a Database”, *Physical Review A* , to appear, September 1998.

[Val84] Valiant, L. G., “A Theory of the Learnable”, *Communications of the ACM*, vol. 27 no. 11, pp. 1134-42, 1984.

[Ved97] Vedral, V., M. B. Plenio, M. A. Rippin and P. L. Knight, “Quantifying Entanglement”, *Physical Review Letters*, vol. 78 no. 12, pp. 2275-9, 1997.

[Ved96] Vedral, Vlatko, Adriano Barenco, and Artur Ekert, “Quantum Networks for Elementary Arithmetic Operations”, *Physical Review A*, vol. 54 no. 1, pp. 147-53, 1996.

[Ven98] Ventura, Dan and Tony Martinez, “Initializing the Amplitude Distribution of a Quantum State”, submitted to *Physical Review Letters*, June 1998.

[You88] Young, Nicholas, *An Introduction to Hilbert Space*, Cambridge University Press, Cambridge, England, 1988.

Part IV

Conclusion

How did it get so late so soon?
It's night before it's afternoon.
December is here before it's June.
My goodness how the time has flewn.
How did it get so late so soon?
-- Dr. Seuss

This dissertation has examined two approaches to dealing with the high-dimensional spaces inherent in computational learning problems. The first approach, presented in Part II, makes the classical assumption that considering the entire state space of large problems is an intractable endeavor. Therefore, a method must be developed for considering only those parts of the space that are in some sense "important". The approach taken here is a hybrid system combining the desirable features of evolutionary computation and neural computation.

The second half of the dissertation, detailed in Part III, considers an idea that classically makes little sense -- what if we could process the entire state space in a reasonable amount of time? Quantum computation offers this possibility at a price. The tradeoff is that although an exponential amount of information can be processed in polynomial time, most of that information is not available to us. The trick is to figure out how to get enough information to justify the use of this alternative computational paradigm. The questions of when this may be done and how to do it are beginning to be answered.

Part IV consists of a single chapter that sums up the contributions made in this dissertation, makes some concluding comments and discusses some possibilities for future research directions.

Chapter 13

Concluding Comments

This is like deja vu all over again.
-- Yogi Berra

As mentioned previously, the field of computational learning suffers from the "curse of dimensionality". As problem size increases linearly, the amount of information grows exponentially. Most of the work in the field of computational learning deals with how to get around this conundrum and still produce useful computation. In some cases the answer has been to live with processing all or most of the exponential amount of information because it is critical to consider it all or because it is believed that the problem in question will never get "too large". In most cases, however, scaling to large problems is a critical factor, and some method for considering only a very small portion of the available information must be devised. And yet, even though the system has considered only a small fraction of the available information, it must produce reasonable results.

The consequence of these requirements is that a large portion of the research done in computational learning is focused on discovering heuristics or biases that can be said to be intelligent in some sense about which information is used and which is ignored. The first half of this dissertation follows this trend by producing a novel hybrid system that combines the strength of evolutionary computation, which is exploration of huge state spaces, with the strengths of neural computation, which are rapid execution and generalization when faced with novel information.

The burgeoning field of quantum computation suggests a novel approach to the computational learning problem. The unique capabilities of quantum systems provide the possibility of confronting the "curse of dimensionality" in a manner that is classically impossible, by considering the exponential amount of information in polynomial time using the massive state space of the quantum system. This sounds too good to be true, and there is indeed a catch. Even though quantum systems can and do process an exponential amount of information in linear space and time, most of that information is unavailable outside of the quantum system. In fact, by the nature of quantum systems, the very act of extracting information from them destroys most of that information. In the general case it destroys all of it except for a small amount that could have been obtained by classical means. However, the amazing aspect of quantum computation is that this is not always quite true. Even though an exponential amount of information is never directly available, it is possible to extract extra global information about the system that is not available classically. Therefore, the trick is to recast classical computational problems in a form amenable to this quantum approach such that the global information can be used to advantage. The second half of this dissertation considers how best to do this and produces several interesting results, suggesting that a theory of quantum computational learning is a promising new avenue to explore in the quest for computational intelligence.

1. Summary and Contributions

The first three chapters of the dissertation provide background information that motivates the work in the two main parts. Chapter 3 makes the first original contribution to computational learning by attempting to distill a necessary set of characteristics for computationally intelligent systems. These include formal primitives, knowledge (acquisition, representation, explanation), complex symbols, symbol grounding, learning, robustness (and fault tolerance and adaptability), duality, multiple levels and scaling.

The two main parts of the dissertation address several of these issues including knowledge acquisition, learning and robustness in Part II and knowledge representation, learning and scaling in Part III (as noted in Chapter 2, there is work, more of a biological bent, that suggests that a quantum mechanical approach to intelligence also embodies issues such as symbol grounding, duality and multiple levels of intelligence).

Chapter 4 explicitly considers a problem of data acquisition, when it is desirable to have a computational learning system but when no data exists from which it can learn. The solution is a hybrid system that makes use of evolutionary computation and black box access to the problem to be learned to acquire useful data for training a neural network or other computational learning mechanism.

Chapters 5 and 6 consider the evolutionary and neural component of Chapter 4 as a single hybrid entity for the purpose of learning optimization. Given a problem for which there exists a large range of possible responses, what is the best response in a given situation? Empirical results over hundreds of experiments suggests that the hybrid does a good job of discovering near optimal solutions with a modest amount of computational expense.

Chapter 7 naturally extends the idea of optimization to continuous control of a system with feedback -- the application of neurocontrol for closed loop systems. Initial empirical results suggest that the neural/evolutionary hybrid has potential for developing viable neurocontrol.

Chapter 8 begins the consideration of a quantum computational approach to computational learning by attempting to construct the quantum analog of a classical artificial neuron model. Although the approach is somewhat naive, it is nevertheless important as a foundation for the ideas developed in the rest of the dissertation.

Chapter 9 takes another step towards a viable quantum computational learning system by introducing the idea of a quantum associative memory that possesses an exponential capacity. In other words, an exponential number of memory patterns can be stored and recalled with a linear number of neurons. This is a significant advance over classical approaches to associative memories. This chapter lays the basic foundation for the idea of quantum associative memory but does not explicitly worry about the problem of physical realizability.

Chapter 10 deviates from quantum computational learning to make a fundamental contribution to the field of quantum computation itself. A quantum algorithm for representing training examples in a quantum system is developed. This contrasts starkly with existing quantum algorithms that assume an entire function is known *a priori*, which is almost never the case in computational learning problems. The focus here is developing an algorithm that obeys the laws of quantum mechanics so that it will be realizable in actual quantum systems. The result opens the door for the development of useful quantum algorithms for computational learning.

Chapter 11 returns to the idea of a quantum associative memory, building on the results of Chapter 10 and of other quantum computational algorithms to demonstrate that such a memory is physically realizable and potentially very advantageous over classical associative memories.

Finally Chapter 12 revisits supervised quantum learning, introduced in Chapter 9, though the approach is now based on ideas from machine learning rather than from neural networks. The problem of learning a powerful class of binary functions known DNF (disjunctive normal form) is considered. A Fourier-based quantum learning algorithm is presented that learns from a set of random training examples. The algorithm achieves significant speedup over classical methods that also use random examples. It is an important point that the examples can be random and still produce good results. The best classical algorithm requires the ability to request specific examples, which in practice is almost never realistic. Therefore, the fact that the quantum algorithm can learn using random examples (which is the usual real world situation) is a significant result.

The original goal of this dissertation was to make progress in confronting the exponential amount of information involved in computational learning problems. The results, in particular, the quantum computational learning results, constitute significant advances in this aspect of computational learning. Also, from a quantum computational standpoint, it is likely that the synergy of quantum computation and computational learning, of which this dissertation offers several examples, will result in the most impressive achievements for the field. This is because by their very natures quantum computation and computational learning both are probabilistic and inexact. Thus it appears that computational learning problems are natural candidates to take advantage of the unique capabilities of quantum computation. On the other hand, many traditional computational problems require determinism and precision, which will make them less amenable, in general, to quantum solutions. Except for the first two chapters and this one, each chapter has either already been published in or is currently under review for publication in a well-known conference proceedings or leading research journal.

2. Future Research Directions

Several important avenues for future research have been briefly suggested, either implicitly or explicitly in the various chapters. Here, some of the most significant are briefly discussed.

2.1. The Neural/Evolutionary Hybrid

The most obvious direction for future work involving the neural/evolutionary hybrid involves application of this method to a large-scale, real-world problem such as network optimization (which is actually the problem that inspired this work). The major difficulty with applying the hybrid model to real-world problems is ascertaining what percentage of optimum the method achieves. In practice this will usually be impossible; therefore a measure of performance will have to be obtained by comparing this approach with current methods for solving these problems. Of course, this involves access to the system for which we wish to develop the hybrid model (such as a large computer network) and this is often not possible in an academic setting.

Obtaining access to such a system will require better arguments for the utility of the approach and this in turn may be accomplished by further theoretical analysis of the model's performance. For example, it would be nice to quantify stability bounds for the system. Perhaps the most interesting, and not coincidentally the most difficult, is whether or not the generalization capabilities of the neural network can be analytically correlated with the performance of the evolutionary discovery of training examples. This problem seems inherently difficult for at least two reasons: generalization in even modestly complex neural networks is not well understood and the behavior of evolutionary computation mechanisms is difficult to quantify. The ultimate goal is to be able to say that for a specific problem that if m training examples with a minimum fitness score of y can be found, then the system will have a probability P of finding a solution that is within ε of being optimum. In other words, it would be extremely desirable to discover some function $f{:}\Re\times\aleph\rightarrow\Re^2$, where $\Re$ is the reals and $\aleph$ is the natural numbers, such that $f(m,y) = (p,\varepsilon)$. Equally useful would be an inverse function $g{:}\Re^2\rightarrow\Re\times\aleph$ such that $g(p,\varepsilon) = (m,y)$.

Also interesting as future work and perhaps more immediately realizable are experiments with variations on and enhancements to the basic theme. Different search mechanisms (other evolutionary approaches, simulated annealing, etc.) may be employed as the front end and different learning algorithms (other neural networks, machine learning, fuzzy logic, etc.) as the back end of the hybrid. In fact, a hierarchical hybrid approach can be envisioned in which a hybrid search algorithm is used to find a training set which is then used to train a hybrid learning system. Improvements to the basic model which are readily implementable include active learning of the training set in which the learner decides what sorts of training examples will be most helpful and queries the search mechanism for such examples. In essence, the evolutionary search primes the neural network which then provides feedback to focus the search in areas of the state space for which it currently has inadequate information. Another obvious idea is to allow the learner to continue to learn even as it is operating as an optimization/control system. This can be accomplished by allowing the evolutionary mechanism to continue to produce training examples after its initial creation of a training set. The learner must then be capable of naturally incorporating the new knowledge into its existing system model (incremental learning). This will be beneficial in several ways. First, the evolutionary discovery of good training examples is by far the most time consuming part of the process. If incremental learning is a possibility and if initially lower performance is acceptable, the training and execution of the neural network can be done far before the discovery of examples is complete with the knowledge that system performance will continue to increase as more good training examples are discovered. This idea can then be combined with the idea of active learning so that the system can monitor its performance on-line, and if it notices certain situations in which it is performing poorly when compared to its average, it can query the evolutionary search mechanism for training examples to help rectify the situation while still executing on-line. Finally, this ability will translate into the capability to track problems that change over time. For example, consider the problem of traffic control. The proper settings and timings of traffic lights is dependent on time of day, time of the week, and perhaps even time of the year. A neural network controlling this system should be able to adapt to these temporal patterns.

2.2. Quantum Computational Learning

Obviously, physical implementation of a quantum learning algorithm on a quantum computer is an important direction for future research. As mentioned in the body of the dissertation, impressive advances in quantum computational technology are being made. In fact, the technology is developing much faster than any but the most optimistic physicists thought possible even two or three years ago. Of course, this particular avenue of future work must be left to the experimental physics community.

The work presented here on integrating quantum computation and computational learning has introduced the promising field of quantum computational learning. The ideas presented in this dissertation consider applying quantum ideas to only a small handful of learning algorithms. There are many more possibilities to be considered. Also, it is conceivable that as quantum computational learning ideas become more mature, learning primitives that have no classical analog may be discovered, just as has happened with quantum computation vs. standard classical computation.

The idea of the quantum neuron introduced in Chapter 8 is not well-developed, having been set aside as the research progressed along other lines. The basic idea needs to be subjected to better theoretical analysis. Perhaps a more manageable starting point would be to consider first the case of a small set of discrete weights. The next step is to develop a training algorithm for the neuron, probably first just a quantum analog of basic Hebbian learning. Unitary operators for classification must be discovered. Finally, the idea of linear superposition has so far been applied only to the weights of the neuron. It is certainly conceivable that this may be extended to include the neuron's inputs, its output, and its activation function, among other things.

As mentioned in Chapter 11, it would be desirable to extend the quantum associative memory's capability to non-binary patterns. In fact, the ability to represent non-binary patterns with quantum systems of more than two states is briefly laid out in Chapter 10. Therefore, very basic recall of non-binary patterns should already be possible. However, making intelligent associations will probably require extending the recall mechanism which in turn depends upon extending Grover's basic algorithm. Another concern that must be considered is that this extension to non-binary patterns involves complex phases. The different values of a single input are represented as points on the unit circle in the complex plane. This implicitly imposes an ordering on these values in the sense that one value will be close to a second in terms of phase (position of the circle) while a third will be further away (more out of phase). If the data is linear and therefore also has an explicit ordering, then this may be used to advantage. However, if the data is strictly nominal, it is unclear what effect this unavoidable ordering may have on system performance. Also as discussed in Chapter 11, the time bound for recall of patterns in the QuAM is slower than desirable. This is an artifact of the recall mechanism being based on Grover's algorithm (which by the way has been shown to be optimal for quantum search). Therefore, perhaps an alternative to Grover's algorithm for recalling the patterns can be discovered.

Also, classical simulation of the quantum associative memory is possible at the cost of an exponential slowdown in the length of the patterns. Thus, association problems that are non-trivial and yet small in size may provide further insight in simulation. However, interesting results will be very modest on a classical computer since only the very smallest problems can feasibly be simulated and these problems are already easily solved classically. The fact is, for most interesting empirical validations of quantum algorithms, it is imperative that we have a quantum computer.

In the meantime, another interesting possibility that has not been explored here at all is that of developing classical algorithms that are somehow inspired by quantum ones. Of course, the idea has inherent limitations because of the fact that classical and quantum systems are fundamentally different. However, just the practice of looking at a problem from an entirely new perspective, as quantum computation does, can often provide insights into the original ideas that are not available any other way. A variation on this idea is the development of classical approximations to quantum algorithms that perform as well in limited circumstances. This would be useful in allowing somewhat more empirical analysis (of limited usefulness) of the quantum algorithm by avoiding the exponential slowdown necessitated by faithfully simulating the quantum system. Further, the classical approximation may be useful in its own right as a learning algorithm.

A classical simulation of the quantum learning algorithm of Chapter 12 may also be developed at the cost of an exponential slowdown in the size of the learning problem. Also, various methods of using and combining the Fourier basis functions discovered by the algorithm may be explored. One method, as suggested in Chapter 12, is to use the boosting method that Jackson uses in his Harmonic Sieve, and other methods of combination may be conceived as well. Further, generalizing the algorithm to handle non-binary data using higher spin systems may also be investigated. Finally, another obvious and important area for future research is investigating further the application of quantum computational ideas to the field of computational learning -- the discovery of other quantum computational learning algorithms.

Finally, the process of understanding and developing a theory of quantum computational learning will contribute to a furthering of our understanding and development of a general theory of computation.

Quantum and Evolutionary Approaches to Computational Learning

Dan Ventura

Department of Computer Science

Degree of Doctor of Philosophy, August 1998

ABSTRACT

This dissertation presents two methods for attacking the problem of high-dimensional spaces inherent in most computational learning problems. The first approach is a hybrid system for combining the thorough search capabilities of evolutionary computation with the speed and generalization of neural computation. This neural/evolutionary hybrid is utilized in three different settings: to address the problem of data acquisition for training a supervised learning system; as a learning optimization system; and as a system for developing neurocontrol. The second approach is the idea of quantum computational learning that overcomes the "curse of dimensionality" by taking advantage of the massive state space of quantum systems in order to process information in a way that is classically impossible. The quantum computational learning approach results in the development of a neuron with quantum mechanical properties, a quantum associative memory and a quantum computational learning system for inductive learning.

COMMITTEE APPROVAL:

Tony R. Martinez, Committee Chairman

Jean-Francois Van Huele, Committee Member

Parris K. Egbert, Committee Member

Bryan S. Morse, Committee Member

John C. Higgins, Committee Member

Scott N. Woodfield, Graduate Coordinator

Date