Hybrid AI and Machine Learning Systems

Logan R. Kearsley
March 28, 2006
Computer Systems Lab, 2006-2007

March 28, 2007

Abstract

Many different Artificial Intelligence and Machine Learning architectures and algorithms have been developed, each with their own strengths and weaknesses that make them particularly suited to certain classes of problems. Two approaches to artificial intelligence are dealt with here: subsumption architecture, and neural networks. S.A.'s are good at building up complex behaviors from sets of simpler ones, when a problem can be broken down into independent peices. Neural networks are good at memorizing associations and making inferences about new data based on stored memories, but can take impractically long to train when the datasets are large. Performance on various tasks can be improved by combining the two approaches to take advantage of the strengths of both.

Keywords: AI, machine learning. neural networks, subsumption architecture

1 Introduction

The purpose of this project is to design a system that combines the capabilities of neural networks and subsumption architectures to produce a more flexible and versatile hybrid system. In this paper I describe a set of basic library functions for the manipulation of subsumption architectures and neural networks, and their use in building a hybrid system, which is then evaluated on the test problem of simple character recognition, and compared to the performance of a simple single-network model.

This project serves as a starting point for further investigations of the improvements that can be made by in AI models by using multiple approaches that cover each other's weak points. The results could be applied to almost any problem that requires high adaptability and dealing with 'fuzzy' input, such as character or speech recognition with multiple users or real-world robot navigation.

2 Background

Most AI and Machine Learning research to this point has consisted of pursuing separate single methods, either to maximize utility for a single problem type, or to duplicate biological models. To my knowledge, little research has been done in hybrid systems that combine the best aspects of multiple other methods to produce a highly versatile AI/ML system without necessarily trying to model biological nervous system functions. A similar research project was carried out by Julian Togelius at the University of Sussex, focusing on the control of a mobile robot, in 2003. Hybrid neural networks have been studied which incorporate symbolic representations into neural network programs for, among other things, speed and ease of control while retaining the robustness and generalization capabilities of pure neural nets since the late 80s.

The subsumption architecture model was first used in 1984 and invented by Alexandre Parodi. Subsumption architectures make use of multiple behavior layers which process inputs and produce outputs independently, with some layers capable of temporarily overiding or subsuming the actions of others. This allows for low-level, high-priority reflex layers to deal with immediate problems, while higher layers control the mid-term and global goals of the agent. Brooks modified the original hierarchical layer model to produce a competitive architecture in which each layer competes for priority with a master scheduler, rather than having its priority fixed.

3 Development Part 1

Requirements: The neural network should be able to memorize any arbitrary set of input/output associations, where inputs and outputs are in the form of lists of numbers between 0 and 1.

Overview: The neural network code is written in C, using the gcc compiler. Functions are available to generate, save, and load from a file networks of arbitrary dimensions, to query a given network with an input vector for the associated output vector, and to teach new associations to the net in cycles.

Limitations: The primary limitations on this project have been time to do the necessary debugging and testing.

Iterative Evaluation Plan: I have used the Spiral Lifecycle Model to develop this project. Each new component was exhaustively tested before adding on the next one, which often revealed previously inconsequential flaws in that required re-visiting the development of older components.

Research Theory and Design criteria: The central algorithm for this project is the back-propagation neural network learning algorithm. Back-propagation is a method of sequentially updating the weights in each layer of a multi-layer, simply connected network based on the errors in the following layer. At the output layer, errors are calculated for each neuron based on the difference between the actual output and the target, and weights are updated accordingly. These errors are then distributed among each of the neurons in the next layer up, weighted by the output of each of those neurons, and the process is repeated to propagate the errors and corrections backwards through the entire network.

Testing and analysis: The primary test for this program consists of providing the network with a list of input/output associations and run training cycles until the network either converges on the correct solution, or it becomes clear that it will not $(forexample, byconvergingonanincorrectsolutionorbeginningtodiverge)$. We can thus test the efficiency of the program by examining how many epochs are required to fully train the network. In the event that the network did not converge, small numbers of training epochs were run, with verbose output of the input and output values of all layers of the network in the process in order to identify where and why incorrect adjustments are being made.

4 Development Part 2

Phase 1 Phase 1 consisted on creating a simple bitmap font format with which to train a neural network for character recognition, and testing individual networks on learning a small character set. A program was written to take a text file and convert it into an image using the font data, and a second program was written to train a number of networks to recognize each character in the set to a minimal initial confidence level.

Phase 2 Phase 2 consisted of creating a program to read an image and run one character at a time through all of the neural networks in order to identify the closest match and transcribe an ASCII character. This program has the ability to continuously re-train the networks to improve accuracy if the correct transcription is given so that it can identify mistakes. Once this was tested and shown to work, the character set was expanded to include all 95 ASCII printable characters in one font.

Future Work The next stage will involve adding the capacity to generate new networks and add characters to the system's knowledge base on-demand.

5 Results

The purpose of this project has been to show that a combination of different AI strategies, specifically subsumption architecture and neural networks, could outperform neural networks alone in some circumstances. It seems so far to have been successful in demonstrating this point.

A complete analysis isn't possible yet, but while the character recognition program employing a multiple-network subsumption model has been able to very rapidly learn to properly recognize a complete 95 character ASCII character set, the single-network test has been unable even to complete low-confidence initial training.

The most substantive and generalizable conclusion that can be drawn from this project is that multilayer neural networks are capable of learning more rapidly and with higher confidence given smaller association sets; ergo, wherever it is possible to break a problem down into smaller parts, it's more efficient to use many smaller specialized networks than to try and train a single large network to accomplish everything.

6 Appendix: API

Debugging

mprntnet($neuro_t * w$)

printout($float * o, intl$)

I/O

loadInput($neuro_t * w, float * I$)
loadTarget($neuro_t * w, float * T$)
copyOutput($neuro_t * w, float * O$)
getInput(w)
getOutput(w)
getInputSize(w)
getOutputSize(w)
getLayerDepth(w)
mquery($neuro_t * w$);
Internal States
setLearningSpeed(w, s)
setMomentum(w, s)
momentumOn($neuro_t * w$)
momentumOff($neuro_t * w$)
swapOn($neuro_t * w$)
swapOff($neuro_t * w$)
swapInput(w, i)
addInput($neuro_t * w$)
Training
backprop($neuro_t * w$)
Loading, Saving, and Generating Nets
mknet(w)
mldnet($FILE * f, neuro_t * w$)
mrldnet($FILE * f, neuro_t * w$)
mrndnet($neuro_t * w, int * dim, intdep$)
msvnet($FILE * f, neuro_t * w$)
mdstrynet($neuro_t * w$)
Binary Loading and Saving
bldnet($FILE * f, neuro_t * w$)
brldnet($FILE * f, neuro_t * w$)
bsvnet($FILE * f, neuro_t * w$)

References

[1] Joanna Bryson, Alan Smaill, and Geraint Wiggins, “The Reactive Accompanist: Applying Subsumption Architecture To Software Design”, March 2005.

Describes the principles of subsumption architecture and sets out a list of criteria that identify a problem as being suited to its use. Also describes a test project using subsumption architecture and suggests several other potential uses.

[2] Julian Togelius, "Evolution of the layers in a subsumption architecture" robot controller, September 2003.

Describes the use of subsumption architecture to break up and simplify the problems to be solved by learning systems, such as genetic algorithms or neural networks, for controlling robots. This is essentially the same thing I'm doing, except that I'm looking at more general software design, not just robot controllers.

[3] John F. Kolen and Jordan B. Pollack, "Back Propagation is Sensitive to Initial Conditions".

The learning process in back-propagation networks is chaotic, as there may be many solutions to any given problem. Thus, learning rates and final configuration are heavily dependent on initial conditions that will predispose the net to converging on a particular set of solutions.

[4] "The Back-Propagation Algorithm", http://www.cs.ucc.ie/ dgb/courses/tai/notes/handout9.pdf $(November 28, 2006)$.

Describes the theory and implementation of the back-propagation algorithm.