Evolving Neural Network Controllers for Unstable Systems

Alexis P. Wieland*
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024-1596
alexis@CS.UCLA.EDU

Abstract

This paper describes how genetic algorithms (GAs) were used to create recurrent neural networks to control a series of unstable systems. The systems considered are variations of the pole balancing problem: network controllers with two, one, and zero inputs, variable length pole, multiple poles on one cart, and a jointed pole. Also, work in progress on a two-legged walker is briefly discribed.

Introduction

Neural networks, and in particular recurrent neural networks, are well suited to task of controlling complex systems. Unfortunately, creating recurrent networks capable of performing complex control tasks is generally difficult, particularly since the desired control signal for a given system is often known only in general terms. This paper describes how genetic algorithms (GAs) were successfully used to "evolve" networks for a series of difficult control tasks.

The prototypical control problem that has been used with neural networks is pole balancing (also known as the cart-pole, broom balancer, or inverted pendulum problem). The problem involves balancing a pole hinged to a cart which is on a finite length track by exerting forces either left or right on the cart, Figure 1a. This problem is of interest because it describes an inherently unstable system and is representative of a wide class of problems.

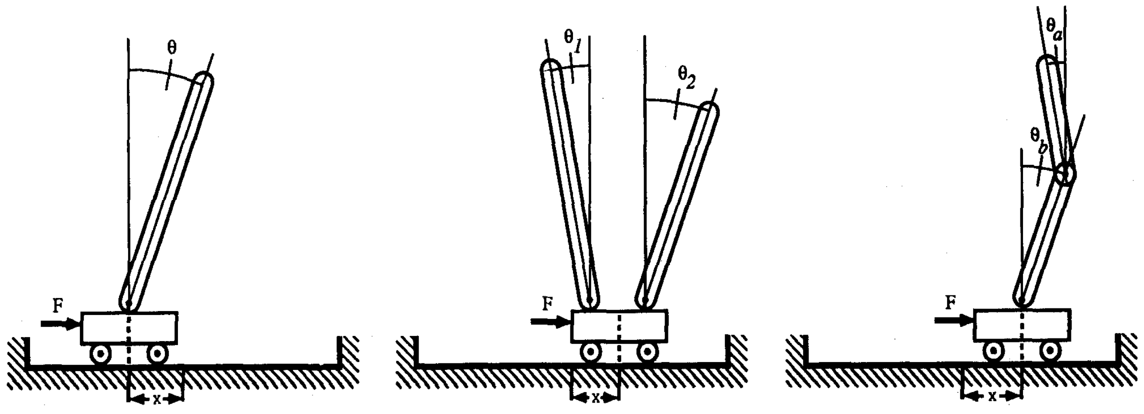

Figure 1: Pole Balancing Problems

*The author is also a part-time employee of The MITRE Corporation, McLean, VA.

The basic pole balancing problem is simple to solve. When provided with four inputs, the position and velocity of the cart and the angle and angular velocity of the pole, the task can and has been solved using a single computing "neuron" [21,22]. In order to make the problem more difficult, the controlling network is often provided with only two inputs, typically the position of the cart and the angle of the pole. This in effect requires the network to "learn" to compute a derivative, the difference between the previous inputs and the current inputs, in order to solve the task.

This paper considers the two input pole balancing problem as well as the one and zero input pole balancing problems. Also, the related problems of a balancing many poles of different lengths on a single cart and balancing a single jointed pole are considered. This paper concludes with a discussion of work in progress on a considerably more difficult task, control of a two-legged walker.

Additional details of the pole balancing experiments, including details of the simulations, all of the equations of motion, and a comparison to traditional control theory, can be found in [23].

Past Work

This section surveys the large amount of related neural network, GA, and the pole balancing literature.

The pole balancing problem is a standard control problem that has been examined exhaustively in control theory texts (e.g., [3,6]). Control theoretic solutions to the multiple pole and the jointed pole problems have been addressed in a pair of dissertations, [11] and [19], respectively.

Neural networks and their predecessors have a long tradition of addressing the (single) pole balancing problem. The ADALINE model was trained to balance poles using the Widrow-Hoff LMS algorithm nearly thirty years ago [21,22]. In addition, the now classic papers on reinforcement learning addressed the pole balancing problem, [1,17]. Pole balancing continues to be a standard research problem for neural networks (e.g., [20,13,18]). All of these works focused solely on the single unjointed pole balancing problem.

GA's have been used extensively for creating neural networks, e.g. [10]. An overview of evolving neural networks with an emphasis on combining GA and back-propagation learning is in [2]. In particular, GA's have been used to develop single pole balancers [14,8].

The pole balancing problem has also been used to demonstrate the power and versatility of other computing paradigms. Of particular relevance is the work on cellular automata with a steepest descent learning procedure used to balance both a single pole and a jointed pole, [15,16].

The Networks Used

Neural nets are by their nature well suited to performing control tasks. Their general paradigm of weighing and combining information from many sources makes them good at integrating and filtering the many often redundant control signals that are available. Many controllers require memory to compute derivatives or integrals. In neural network terms, this requires recurrent networks, i.e., networks that contain feedback. All the networks discussed in this paper are recurrent.

The bulk of this research was carried out on fully connected recurrent networks, shown in Figure 2. Every neuron in these networks receives an input that is the weighted sum of all the external inputs to the network plus the previous state of all the neurons, including itself. A subset of the computing neurons are designated as output neurons and their outputs are used as control signals, but aside from this designation the output neurons are identical to the other neurons in the network.

Arbitrarily connected networks were also considered. Strings of structures containing *from*, *to*, and *weight* slots were used to evolve networks with arbitrarily complex interconnections. This approach has been exceptionally successful for other hard problems, [4].

Unlike most other pole balancing work, the nodes were able to produce a range of outputs between -1 and 1.[1] This gave the network both greater internal computational power and the finer control signals needed by the more difficult tasks addressed.

[1]Weights, thresholds, and node values were stored in eight bits, representing the values $-1, -\frac{253}{255}, -\frac{251}{255}, \ldots, \frac{253}{255}, 1$. Note that this representation does *not* allow for any representation of zero. Thus networks are not able exploit unstable equilibria since they were always producing "noise."

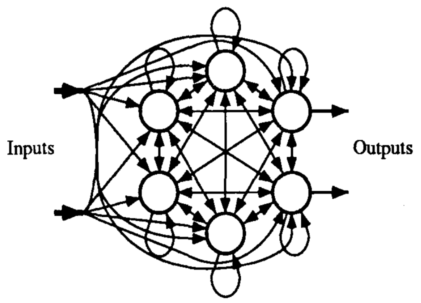

Figure 2: Fully Recurrent Network Topology

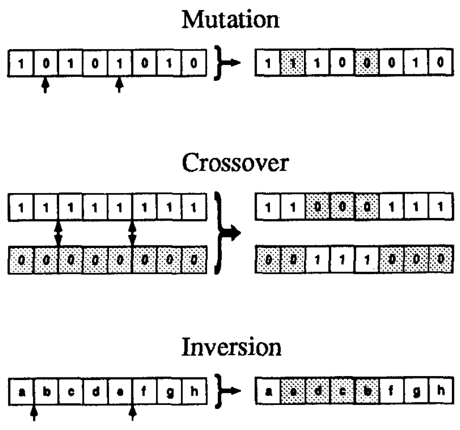

Figure 3: Genetic Operators

Genetic Algorithms

GA's, first introduced by Holland [12], are a stochastic search method based loosely on the process of evolution and natural selection. With GA, the system being evolved is encoded in a "gene," a string that can represent any of the class of systems. An initial set or "population" of these strings is created randomly and then a process of "natural selection" and "reproduction" continues until a goal is achieved.

"Parents" are selected stochastically based on their relative "fitness" in the population. Reproduction is based on a series of "genetic operators." The three genetic operators used in this work are mutation, crossover, and inversion, shown pictorially in Figure 3. Mutation refers to randomly changing parts of a gene. Crossover takes part of the child's gene from one parent and the remainder of the gene from the other parent. Inversion involves reversing the order of a fraction of the gene. The text by David Goldberg, [9], is recommended to readers interested in pursuing GAs further.

The basic fitness of a controller was simply the time that it was able to keep its pole(s) from falling down or the cart from hitting an end of the track. While the networks quickly evolved to keep the pole from passing the cutoff angle, it often took considerably longer to discover the relevance of the ends of the track. Therefore fitness functions that penalized the network for leaving the center of the track were sometimes used. These functions were of the form

$$f(t+1) = f(t) + 1 - \lambda\left(\frac{x}{L}\right)^2 \tag{1}$$

where $f(t)$ is the fitness at time t, x is the distance of the cart from the center of the track, L is half the length of the track, and $0 \leq \lambda \leq 1$ is a constant factor determining the degree of penalty. Also, when one or no inputs were provided, the evolution process was sped by requiring that the network generate the missing state variables as additional outputs.

Pole Balancing

The traditional single unjointed pole balancing problem can be solved optimally by exerting a force F on the pole given by:

$$F = F_{max}\ \mathrm{sgn}(k_1 x + k_2 \dot{x} + k_3 \theta + k_4 \dot{\theta}), \tag{2}$$

where F_{max} is the maximum force, x and $\dot{x}$ are the position and velocity of the cart[2], θ and $\dot{\theta}$ are the angle and angular velocity of the pole, and k_1, k_2, k_3, and k_4 are coefficients that depend on the masses and frictions of the system, [21,22]. Equation 2 also describes a single four input linear thresholding neuron. Thus, it is possible to solve the standard pole balancing problem with a neural network composed of a single neuron.

[2]The "dot" notation for time derivatives is used throughout the remainder of this paper: $\dot{x} = dx/dt$ is the velocity of the cart, $\ddot{x} = d^2x/dt^2$ is the acceleration of the cart.

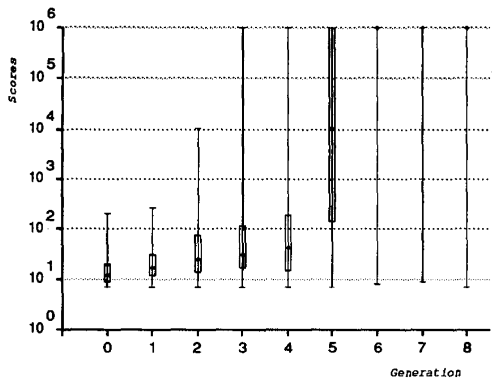

Figure 4: Evolution of Two Input Pole Balancer.

A slightly more difficult variation of this problem uses a two component state vector containing x and θ, but neither $\dot{x}$ nor $\dot{\theta}$. Stable control of this system requires the controller to compute an estimate of the velocities. Using GA and fully recurrent networks the two input single pole balancing problem required six generations to evolve to a point where the majority of the controllers could balance a pole for at least 10^6 time steps, approximately 5.5 hours of simulated time. Figures 4 shows the maximum, median, minimum, and a rectangle from the median of the larger half of the values to the median of the smaller half values of the values of the population's fitness at successive generations. The fitness function used was the length of time that the pole remained balanced.

Further variations of the single pole balancing problem were considered. Controllers that required only one input, the angle θ, were evolved. These tasks required slightly larger networks (ten neurons) and a consistent starting location (centered on the track). Using a fitness function that required two outputs, one for the force and the other for an estimate of the current x location, sped the rate of evolution considerably. These networks evolved in about thirty generations. Without such a fitness function the task was more difficult and required approximately 100 generations to evolve.

A further extension was to use a network with *no* inputs. For this problem the cart was consistently started in the center of the track with the pole vertical.[3] The most successful networks that were evolved for this problem were only able to balance a pole for a few hundred time steps, a few seconds of simulated time. Even when a fitness function that required the network to produce an estimate of x and θ was used, these estimates quickly deteriorated (due to "round-off error"), causing the controller to fail.

A variation of the single pole balancing problem that surprisingly was *not* difficult considered variable length poles. Two related problems were investigated. In the first problem the length of a pole was randomly drawn from between 0.1 and 1 meters but then fixed for the duration of the simulation. In the second, the length of the pole varied between 0.1 and 1 meter in a slow random fashion throughout the simulation. In both cases the mass of the pole varied in proportion to the length, that is the pole did not compress but rather the end "evaporated." While this task was slightly more difficult than the standard fixed length pole problem (a 10 neuron fully recurrent network requiring 25 to 30 generations to evolve), there was no evidence that the network ever computed an estimate of the pole's length as initially anticipated. Further, most good standard pole balancers were able to balance the variable length pole systems for at least 10^4 time steps.

Multiple Poles

An interesting and considerably more challenging version of the pole balancing problem involves balancing more than one pole on the same cart, a two pole example is shown in Figure 1b. As long as the poles are of different lengths they will react differently to a force applied to the cart and can therefore be balanced simultaneously. The equations of motion for the multiple pole problem are:[4]

$$\ddot{x} = \frac{F - \mu_c \operatorname{sgn}(\dot{x}) + \sum_{i=1}^{N} \tilde{F}_i}{M + \sum_{i=1}^{N} \tilde{m}_i} \tag{3}$$

$$\ddot{\theta}_i = -\frac{3}{4 l_i}\left(\ddot{x}\, cos\theta_i + g \sin\theta_i + \frac{\mu_{pi}\dot{\theta}_i}{m_i l_i}\right) \tag{4}$$

[3]Note that while this is an (unstable) equilibrium point, because the neurons used were unable to produce an output of zero (see Footnote 1) the network was not able to make use of this degenerate solution.

[4]For $N = 1$ these equations are equivalent to those used in [1] and elsewhere.

where x is the distance of the cart from the center of the track, θ_i is the angle of the ith pole from the vertical, N is the number of poles on the cart, g is the acceleration due to gravity, m_i and l_i are the mass and the half length of the ith pole, M is the mass of the cart, μ_c is the coefficient of friction of the cart on the track, μ_{pi} is the coefficient of friction for the ith hinge, F is the force applied to the cart, $\tilde{F}_i$ is the effective force from the ith pole on the cart:

$$\tilde{F}_i = m_i l_i \dot{\theta}_i^2 \sin\theta_i + \frac{3}{4} m_i \cos\theta_i \left(\frac{\mu_{pi}\dot{\theta}_i}{m_i l_i} + g\sin\theta_i\right) \tag{5}$$

and $\tilde{m}_i$ is the effective mass of the ith pole:

$$\tilde{m}_i = m_i \left(1 - \frac{3}{4}\cos^2\theta_i\right) \tag{6}$$

Equation 4 shows that the angular acceleration of a pole for a given $\ddot{x}$ is greater the more vertical the pole and is inversely proportional to the length of the pole. Therefore, if the shorter pole is vertical and the longer pole is tilted, bringing both poles vertical requires that a force be applied that causes the longer pole to lean even further until the faster rotating shorter pole passes it. Then the opposite force is applied and both poles are brought upright together. Figure 5 shows the output of a neural network controller executing this maneuver.

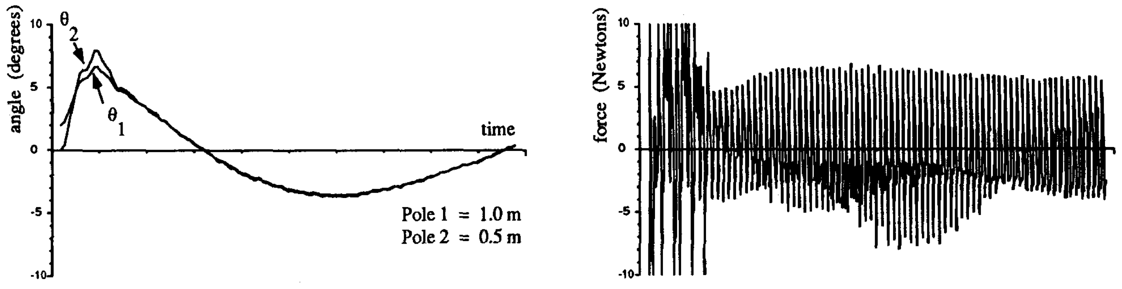

Figure 5: Control of Two Poles on One Cart

Network controllers for the two pole problem were evolved for the case where the poles had lengths of 1.0 and 0.1 meters, requiring approximately 150 generations of a population of 2048 ten node fully recurrent neural networks. Evolution was then continued while the shorter pole was lengthened by 1% increments until it was 0.9 meters long. It can be shown that at in this final configuration, stationary poles that are essentially vertical can only differ by at most 0.1° and still be successfully balanced, [23].

The process of slowly adjusting the pole length was similar to the "shaping" process that has been used with gradient descent learning paradigms, [24]. It was most remarkable how *slowly* this shaping process proceeded. The controllers often required as much as a dozen generations to recover from the 1% change.

Jointed Pole

The final variation of the pole balancing problem considered a jointed pole, shown in Figure 1c. As with the multiple pole problem, as long as the lengths and therefore the natural frequencies of the poles are sufficiently different it was possible to balance the system. The equations of motion for a pole with a single joint are quite complex and can be found in [19,23].

The jointed pole problem proved to be simpler to evolve than the multiple pole problem. Unlike the multiple pole problem, the control signal that brought the poles closer to vertical was always the correct signal to use. For this reason it was possible to strongly favor selecting parents from a small set of best performers. 20 to 30 generations were required to evolve controllers for the jointed pole problem when the bottom pole was 1 meter and the top pole was 0.1 meter.

Two-Legged Walker

Currently work is underway to evolve controllers for a two-legged walker, Figure 6. As with the pole and cart system, the equations that define the motion of the walker include terms for gravity, friction, momentum, etc. This realistic model and its general lack of staticly stable states distinguishes this system from much of the past work in two-legged walking (e.g., [7]).

At the time of this writing the results from the walker are promising but not complete. Currently, after evolving to the point where the walkers take one or two faltering steps, the entire population will settle on some action that provides forward motion in the short term, such as lunging to the floor or kicking high into the air and falling into a splits position. Methods for combatting this "premature convergence," such as [5], are being used.

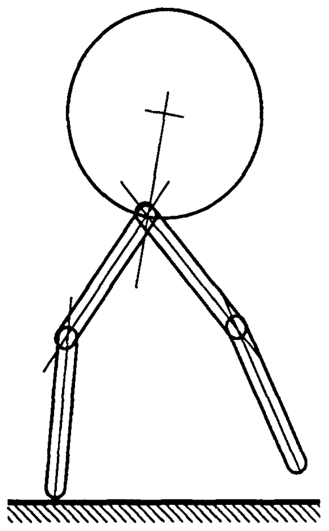

Figure 6: Two-Legger Walker Problem.

Discussion

This paper discussed a series of control problems and the ways that GA's have been used to create recurrent neural networks to address those problems.

GAs were able to quickly evolve networks for the one and two-input pole balancing problems. Networks with *zero* inputs were only able to balance poles for a few seconds of simulated time due to network's inability to maintain accurate estimates of their position and pole angle. The multiple pole problem was the author's favorite. The equations of motion are simple and the problem can be made as challenging as desirable by adjusting the ratio of the pole lengths. The jointed pole problem was found to be comparatively simple to evolve.

The difficulty that was encountered while "shaping" the multiple pole system points out a major difference between gradient descent learning and GAs. Shaping creates a situation where only a small number of weights need to be finely tuned, an ideal task for gradient descent learning but which is difficult for GAs.

This work has been successful at creating recurrent neural nets to control unstable systems. The ongoing work into controlling a two-legged walker represents a considerably more difficult but more realistic task.

Acknowledgements

Special thanks to D. Jefferson, D. Touretzky, R. Collins, R. Leighton, and my wife A. Mercer.

The figures and plots in this report were created using the publicly available *GoodNeWS* software developed by the Turing Institute, Glasgow, Scotland.

References

[1] A. Barto, R. Sutton & C. Anderson. "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems." *IEEE SMC-13:834-846.* (1983)

[2] R. Belew, J. McInerney & N. Schraudolph. "Evolving Networks: Using the Genetic Algorithm with Connectionist Learning." UCSD Tech. Report: #CS90-174. (1990)

[3] R. H. Cannon, Jr. *Dynamics of Physical Systems.* New York: McGraw-Hill Book Company. (1967)

[4] R. Collins & D. Jefferson. "An Artificial Neural Representation for Artificial Organisms." In R. Männer & D. Goldberg (eds.), *Proc. Parallel Problem Solving from Nature.* Berlin: Springer-Verlag (in press).

[5] R. Collins & D. Jefferson. "Selection in Massively Parallel Genetic Algorithms." Submitted to *4th Inter. Conf. on GA.* (1991)

[6] B. Friedland. *Control System Design, An Introduction to State-Space Methods.* New York: McGraw-Hill Book Company. (1986)

[7] H. de Garis. "Genetic Programming: Modular Evolution for Darwin Machines." *IJCNN-90 Wash. D.C.* (1990).

[8] D. Goldberg. "Computer-Aided Gas Pipeline Operation Using Genetic Algorithms and Rule Learning." Ph.D. Dissertation. Ann Arbor: U. Michigan. (1983)

[9] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley. (1989)

[10] S. A. Harp, T. Samad & A. Guha. "Towards the Genetic Synthesis of Neural Networks." In J. D. Schaffer (ed.) *Proc. 3rd Int. Conf. on GA.* San Mateo, CA: Morgan Kaufmann Publishers. (1989)

[11] D. T. Higdon. "Automatic Control of Inherently Unstable Systems with Bounded Control Inputs." Ph.D. Dissertation, Dept. Aeronautics & Astronautics, Stanford Univ. (1963)

[12] J. Holland. *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: U. Michigan Press. (1975)

[13] M. I. Jordan & R. A. Jacobs. "Learning to Control an Unstable System with Forward Modeling." In D. Touretzky (ed.), *NIPS 2.* San Mateo, CA: Morgan Kaufmann. (1990)

[14] J. R. Koza. "Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems." Stanford Tech. Report: STAN-CS-90-1314. (1990)

[15] Y. Lee, S. Qian, R. Jones, C. Barnes, G. Flake, M. O'Rourke, K. Lee, H. Chen, G. Sun, Y. Zhang, D. Chen, & G. Giles. "Adaptive Stochastic Cellular Automata: Theory." Los Alamos Natl Lab Tech. Report: LA-UR 90-1229. (1990)

[16] Y. Lee, S. Qian, R. Jones, C. Barnes, G. Flake, M. O'Rourke, K. Lee, H. Chen, G. Sun, Y. Zhang, D. Chen, & G. Giles. "Adaptive Stochastic Cellular Automata: Applications." Los Alamos Natl Lab Tech. Report: LA-UR 90-1227. (1990)

[17] D. Michie & R. A. Chambers. "BOXES: An Experiment in Adaptive Control." In E. Dale & D. Michie (eds.) *Machine Intelligence 2*, Edinburgh: Oliver and Boyd, pp. 137-152. (1968)

[18] B. E. Rosen, J. M. Goodwin & J. J. Vidal. "Adaptive Range Coding." *NIPS 3.* (1990)

[19] J. F. Schaefer. "On the Bounded Control of Some Unstable Mechanical Systems." Ph.D. Dissertation, Dept. of Elect. Eng., Stanford Univ. (1965)

[20] J. H. Schmidhuber. "Making the World Differentiable: On Using Supervised Learning Fully Recurrent Neural Networks for Dynamic Reinforcement Learning and Planning in Non-Stationary Environments." Technische Universität München Tech. Report: FKI-126-90. (1990)

[21] B. Widrow & F. W. Smith. "Pattern Recognizing Control Systems." *Comp. Info. Sci. (COINS) Symposium,* 1963. (1963)

[22] B. Widrow. "The Original Adaptive Neural Net Broom-Balancer." *Proc. IEEE Inter. Symp. Circuits & Systems,* pp. 351-357. (1987)

[23] A. P. Wieland. "Evolving Controls for Unstable Systems." In D. Touretzky (ed.), *Connectionist Models: Proc. 1990 Summer School.* San Mateo, CA: Morgan Kaufmann. (1991)

[24] A. P. Wieland & R. R. Leighton. "Shaping Schedules as a Method for Accelerating Learning." *INNS Mtg.* (1988)