Proceedings of the International Workshop on Neural Networks and Neurocontrol, 1997

Using Evolutionary Computation to Facilitate Development of Neurocontrol

Dan Ventura *and* ***Tony Martinez***

The field of neurocontrol, in which neural networks are used for control of complex systems, has many potential applications. 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). The survivors of this evolutionary process are used to construct a training set for the NN. The NN learns the training set, is able to generalize to new system 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 for the pole balancing problem.

1. Introduction

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 proposes a method of using evolutionary computation (EC) to develop a training set for training an 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.

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 [Cau91], [Har90], [Sau94], and [Wie90]. 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 [Mia96], [Mon89], and [Rom93].

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 four discusses applying this approach to the well-known pole balancing problem, the prototypical example for system control. Finally, section 5 provides conclusions and directions for future work.

Neural Network and Machine Learning Laboratory
http://axon.cs.byu.edu
Department of Computer Science
Brigham Young University
Provo, UT 84602 USA
email: dan@axon.cs.byu.edu
martinez@cs.byu.edu

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 [Ven95] [Ven96a] [Ven96b], single iteration, open-loop optimization type problems have been considered.

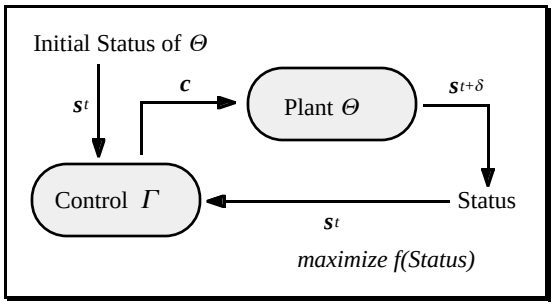

Figure 1. Control system with feedback

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 generelizeable, 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 (hopefully) 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 (stable vs. unstable for instance) 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 (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 $\boldsymbol{s}_i^{t=0} \rightarrow \boldsymbol{c}_{max_i}$. This approach is summarized in figure 2.

```
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

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

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 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.

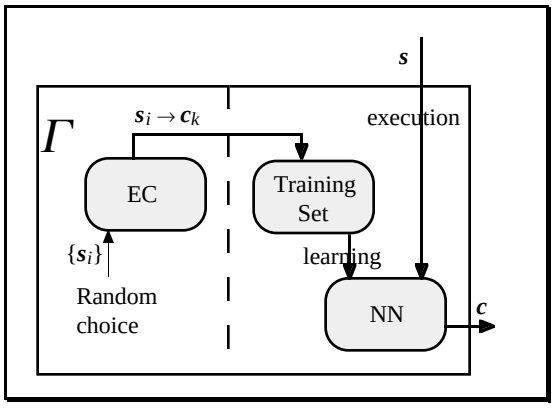

Figure 3. The control Γ

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 a reasonable simulation thereof) 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.

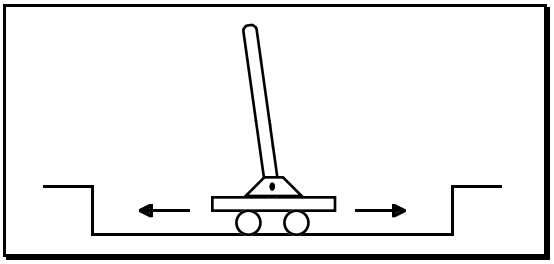

Figure 4. The pole balancing problem

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, but the simple system shown in figure 4 will suffice for our purposes.

This system was simulated using a discrete time, state-space representation (see Appendix)

with a fixed 1 millisecond time increment. A training set of 50 instances of the form $\left(x, \dot{x}, \theta, \dot{\theta}\right) \rightarrow F$ was generated using the evolutionary method described in section 3. The length of time allowed for system stabilization during the evolutionary evaluation, δ, was set to 10 time steps (10 milliseconds). 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 part of the state-space that will be most helpful for neural network generalization. In this case of the pole balancer, we are most interested in the those states in which the pole angle and its derivative are small (since once either is too large the pole will fall for sure due to the system limitations on track width and available force). Likewise, only states with small values for horizontal position and cart velocity will be helpful in training the neural network. Once the training set had been generated, a simple backpropagation network with 5 hidden nodes was trained for 1000 epochs using the training set. The learning rate was set at .5, and a momentum term with a coefficient of .95 was also employed.

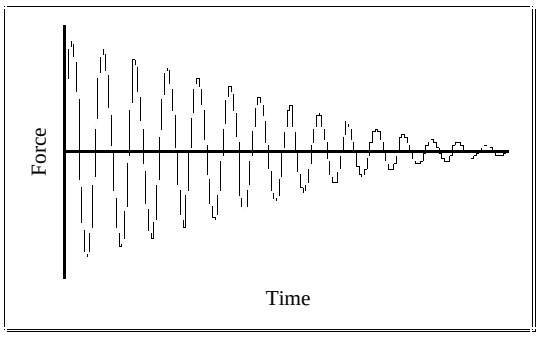

Figure 5. Force applied by neurocontroller to cart over time

After training for 1000 epochs, the backprop was tested for 30 seconds (30000 time steps) as a neurocontroller for the pole balancer. If the pole fell or the cart hit the wall within 30 seconds, the neurocontroller failed (either the training set was bad or the backprop got stuck in a local minima) and was retrained using the same training set. If the backprop failed to converge after several trials the training set could be regenerated and the entire process started over. However, very rarely did the backprop fail to find a simple oscillatory solution within a few training attempts. An example of one such solution is shown in figure 5, which graphs force applied to the cart by the neurocontroller verses time. The graph represents a time of 30 seconds.

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. 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, better avoidance of local minima during NN training, and combining these techniques with those that evolve the network architecture.

Appendix: Dynamics of the Pole Balancing Problem

The differential equations describing the pole balancing system given here are taken from [Wie90] and are as follows.

$$\ddot{x} = \frac{F - \mu_c sgn(\dot{x}) + \tilde{F}}{M + m\left(1 - \frac{3}{4}\cos^2\theta\right)} \tag{1}$$

$$\ddot{\theta} = -\frac{3}{4l}\left(\ddot{x}\cos\theta + g\sin\theta + \frac{\mu_p \dot{\theta}}{ml}\right) \tag{2}$$

In equation (1),

$$\tilde{F} = ml\dot{\theta}^2 \sin\theta + \frac{3}{4} m \cos\theta \left(\frac{\mu_p \dot{\theta}}{ml} + g\sin\theta\right). \tag{3}$$

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* 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. Assuming no friction and relatively small angles and velocities, a simple version of the pole and cart system can be approximated by the following two differential equations.

$$(M+m)\ddot{x} + ml\ddot{\theta} = F \tag{4}$$

$$ml\ddot{x} + \frac{4}{3}ml^2\ddot{\theta} + mgl\theta = 0 \tag{5}$$

(Recall that the assumption of small velocities allows us to ignore the $\dot{x}$ and $\dot{\theta}$ terms and the assumption of small angles allows the approximations $\cos\theta \approx 1$ and $\sin\theta \approx \theta$.) Table I gives the values used for the variables during the simulation.

Table I. Values for the pole balancing simulation

Variable	Meaning	Value
x	horizontal cart position	[-2, 2] m
q	angle of pole off vertical	[-.15, .15] rad
F	force applied to cart	[-10, 10] N
M	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

Equations (4) and (5) above can be converted into a state-space representation so that the system is in the linear form:

$$\dot{\vec{y}} = A\vec{y} + B\vec{u}, \tag{6}$$

$$\vec{z} = C\vec{y}. \tag{7}$$

with the state vector $\vec{y} = \left[x, \dot{x}, \theta, \dot{\theta}\right]^T$, the control vector $\vec{u} = F$, and the output vector $\vec{z} = [x, \theta]^T$. Finally, equations (6) and (7) can be discretized for simulation using an appropriately small time increment:

$$\vec{y}(t+1) = A\vec{y}(t) + B\vec{u}(t), \tag{8}$$

$$\vec{z}(t) = C\vec{y}(t) + D\vec{u}(t). \tag{9}$$

The matrices shown below were generated using Matlab and are discretized with a time step size of 1 millisecond.

$$A = \begin{bmatrix} 1.0000 & 0.0010 & 0.0000 & 0.0000 \\ 0.0000 & 1.0000 & -0.0007 & 0.0000 \\ 0.0000 & 0.0000 & 1.0000 & 0.0010 \\ 0.0000 & 0.0000 & 0.0159 & 1.0000 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0005 \\ 0.9670 \\ -0.0007 \\ -1.4790 \end{bmatrix}$$

$$C = \begin{bmatrix} 0.0010 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0010 & 0.0000 \end{bmatrix}$$

$$D = \begin{bmatrix} 0.0005 \\ -0.0007 \end{bmatrix}$$

References

[Bar83] Barto, Andrew G., Sutton, Richard S. and Anderson, Charles W., "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.

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

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

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

[Mia96] Miagkikh, V. V., Topchy, A. P. and Lebedko, O. A., "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.

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

[Rum86] Rumelhart, David E., McClelland, James L, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, 1986.

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

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

[Ven95] Ventura, Dan, Andersen, Tim and Martinez, Tony R., "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.

[Ven96a] Ventura, Dan and Martinez, Tony R., "Robust Optimization Using Training Set Evolution", *Proceedings of the International Conference on Neural Networks*, pp. 524-8, 1996.

[Ven96b] Ventura, Dan and Martinez, Tony R., "A General Evolutionary/Neural Hybrid Approach to Learning Optimization Problems", *Proceedings of the World Congress on Neural Networks*, pp. 1091-5, 1996.

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