A Neural Network Pole Balancer
that Learns and Operates on a Real Robot in Real Time

Dean Hougen hougen@cs.umn.edu
John Fischer jfischer@cs.umn.edu
Deva Johnam johnam@cs.umn.edu

Artificial Intelligence, Robotics, and Vision Laboratory
Department of Computer and Information Sciences
University of Minnesota
4-192 Electrical Engineering and Computer Science Building
200 Union St. SE, Minneapolis, MN 55455

Abstract

A neural network approach to the classic inverted pendulum task is presented. This task is the task of keeping a rigid pole, hinged to a cart and free to fall in a plane, in a roughly vertical orientation by moving the cart horizontally in the plane while keeping the cart within some maximum distance of its starting position. This task constitutes a difficult control problem if the parameters of the cart-pole system are not known precisely or are variable. It also forms the basis of an even more complex control-learning problem if the controller must learn the proper actions for successfully balancing the pole given only the current state of the system and a failure signal when the pole angle from the vertical becomes too great or the cart exceeds one of the boundaries placed on its position.

The approach presented is demonstrated to be effective for the real-time control of a small, self-contained mini-robot, specially outfitted for the task. Origins and details of the learning scheme, specifics of the mini-robot hardware, and results of actual learning trials are presented.

1 Introduction

Pole-balancing is the task of keeping a rigid pole, hinged to a cart and free to fall in a plane, in a roughly vertical orientation by moving the cart horizontally in the plane while keeping the cart within some maximum distance of its starting position (see Figure 1). Variously known as the inverted pendulum problem, pole-balancing, stick-balancing, or broom-balancing, this task is a classic object of study in both system dynamics (where the equations of its motion are of interest)[4], [16] and control theory (where control systems capable of balancing the pole are of interest)[20]. The dynamics are by now well understood, but they provide a difficult task nonetheless, as the system is inherently unstable. Further, if the system parameters are not known precisely, or may vary, then the task of constructing a suitable controller is, accordingly, more difficult.

The pole-balancing task is quickly becoming a classic object of study in the theory of control-learning, as well. Systems that can learn to balance poles were first developed over thirty years ago [6], [23] and many more have been developed since (e.g. [1], [3], [5], [7], [8], [9], [11], [15], and [19]). All of these systems have fallen

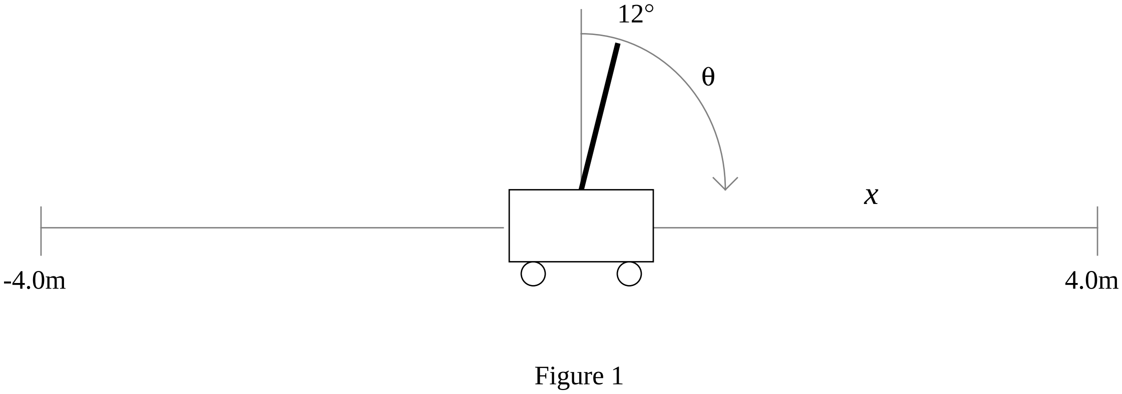

Figure 1

short of fully solving the inverted pendulum problem (see Subsection 5.3 "Discussion") due to the difficult nature of the problem. In the most difficult version of the problem (which we refer to in this paper as the full or entire problem), the learning system is provided with nothing but a vector encoding the current state of the cart-pole system and a single failure signal which indicates that any one of the four error conditions (pole fell left, pole fell right, cart went too far left, or cart went too far right) has occurred. Because a great many actions will have been specified by the controller when failure is finally signalled, determining which actions were "correct" and which were the cause of failure is an extremely difficult credit and blame assignment problem.

Perhaps due to the difficulty of this problem, or perhaps due to a failure to understand the shortcomings of simulation, most researchers have used their control-learning systems only on simulations of cart-pole systems. What is likely the first control system to learn to balance a pole in an actual physical system without the aid of an outside teacher was only recently developed (see [12]). Yet even this system does not solve the complete problem on its own; fully half the problem is solved for it before it even begins the training process (see Subsection 5.3 "Discussion"). We present, then, what we believe to be the first control system which learns to solve the entire pole-balancing problem for a real, physical cart-pole system.

2 Problem Definition

The pole balancing problem, as it pertains to control-learning theory, is really a class of problems of the general form described above. For simulation results, a standard (as presented in [2]) is generally, but not always (e.g [9]), followed. For actual cart-pole systems, the parameters are naturally quite variable. For our cart-pole system, many of the system parameters are known only very roughly and others are unknown. It is not necessary for us to known the particulars of our cart-pole system, as it is the task of the learning system to decide the correct control actions and the learning system is designed to work on the class of problems, not on any particular instantiation of it.

Approximations of the cart-pole system parameters are as follows (see Figure 1): The mass of the cart is roughly 1 kilogram, the pole has a roughly uniform mass of 50 grams and a length of approximately 1 meter, giving a center of mass approximately 0.5 meters from the pivot. The cart is restricted to stay within approximately 4.0 meters of its initial position and the pole within roughly 12° of vertical. Violating either of these conditions causes a failure signal to be generated and the trial to end.

The standard of allowing the cart only "bang-bang" control (allowing only for a preset force to be applied in either direction on the cart, rather than a variable force application) was approximated by the application of a single torque pulse in either direction by the motor on any time step. The magnitude of this torque pulse is not known.

The coefficient of friction in the "hinge" between the cart and the pole (actually a variable resistor, see Section 4 "The Robot") was not estimated, nor was the friction in the axles nor that produced by the motor itself, nor was the coefficient of sliding friction between the tires and the floor. Finally, changes due to a reduction in battery power during the run were not estimated.

To reduce the computational load on the mini-robot's processor (see Section 4.2 "Mini-Computer Board"), the state vector was restricted to approximations of the cart position and pole angle. (Standard versions of the pole-balancing problem provide the controller with cart velocity and pole angular velocity as well.)

3 SONNET

The learning system follows the Self-Organizing Neural Network with Eligibility Traces (SONNET) paradigm first described in [11]. The SONNET paradigm delineates a general class of control-learning systems which are appropriate for problems in which correct control actions are not known, but a feedback mechanism that allows for an overall evaluation of system performance (success and/or failure signals) is available and for which system performance is temporally based on network responses.

The SONNET paradigm combines the self-organizing behavior of Kohonen's Self-Organizing Topological Feature Maps (Kohonen Maps, see [14]) with the concept of an eligibility trace (see Section 3.2 "The Eligibility Trace") to create a class of novel and powerful control-learning systems. A SONNET system (known as PBMax after its intended task and particulars of its computations) was applied to a standard simulation of the pole-balancing problem [11]. Here, a (new) SONNET-style network is applied to an instantiation of the inverted pendulum problem using a real cart-pole system for the first time.

3.1 Self-Organizing Maps

Kohonen has proposed as set of connectionist systems based the recognition that in biological neural networks (i.e. brains) there are regions (especially in the cerebral cortex) for which topology preserving relations exist between patterns of input from sensory organs and the physical arrangement of the neurons themselves [14, pp.119-122]. These areas provide efficient representations for interrelated data. Kohonen Maps, then, are a class of conceptually similar artificial maps that use several simple techniques (such as lateral inhibition) in concert to achieve the same general effects as those found in biological systems. Kohonen Maps have been used extensively as pattern classifiers.

3.2 The Eligibility Trace

Biological neurons are highly complex and their functions are only very roughly approximated by the artificial neurons in today's connectionist systems. One function found in biological neural networks, as noted in [13], is what we refer to here as the 'eligibility trace'. This function is the result of neurons becoming more amenable to change immediately after they fire. This plasticity reduces with time, but provides an opportunity for learning based on feedback following the neuron's activity.

Building on this idea, we can construct an artificial neural network capable of learning based on temporal feedback. The network receives input, gives responses and, when feedback signals arrive, is updated based on the neurons' eligibilities for change. Notably, this feedback can come from some other system (such as a cart-pole system) that the network is causally connected to.

3.3 The PBMin Learning System

A particular learning system, the Pole Balancer for the Mini-robot (PBMin) was constructed under the SONNET paradigm. Unlike previous SONNET systems, PBMin is restricted to using integer values, due to restrictions imposed by the mini-robot's onboard processor. This fact should be kept in mind when viewing the equations given below.

3.3.1 Welcome to the neighborhood

One concept borrowed for SONNET from Kohonen Maps is that of 'neighboring.' This concept relates to the topological arrangement of the neurons which does not change as the network learns. In PBMin, the topology is rectangular; the nearest neighbors of any neuron are those neurons which have identification numbers differing by one in either or both digits from those of the neuron in question. For example, the nearest neighbors of neuron (3,3) are (2,2), (2,3), (2,4), (3,2), (3,4), (4,2), (4,3), and (4,4). In Figure 2, all nearest neighbors are directly connected by line segments.

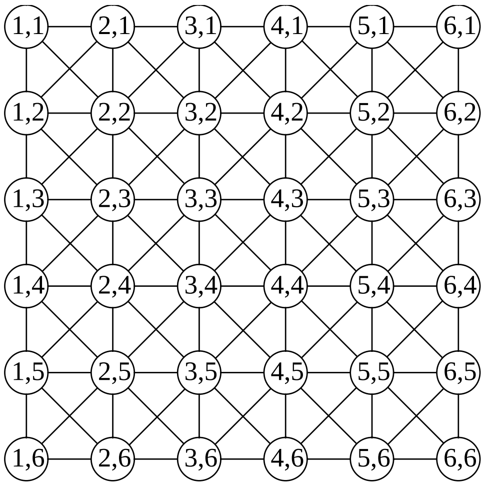

Figure 2

A 'neighborhood', then, is a set of neurons within some "distance" (defined according to network's topology) of the neuron in question. A neighborhood may have any width from zero (the neighborhood is restricted to the neuron itself) to the width of the network itself (such that the entire network is within the neighborhood).

More formally, for a rectangular network, if U is the set of units u in the network, (i, j) the numerical identifier of each neuron, and ω the width of the neighborhood, then the neighborhood N of neuron n is defined as

$$\{ u \in U \| \; |i_n - i_u| < \omega \wedge |j_n - j_u| < \omega \} . \qquad (1)$$

In PBMin, a constant neighborhood width of one was used; the neighborhood for any given neuron was therefor reduced to the set of nearest neighbors plus the neuron itself. Figure 2, therefor, shows the entire connectivity of the network. The rectangular topology was chosen to match the two-dimensional input (state) vector (see Section 2 "Problem Definition").

3.3.2 Cooperation and competition

Each neuron has one "weight" (an integer) associated with it for each dimension of the input space. Because PBMin receives a two-dimensional input vector, each neuron correspondingly has two input weights. These weights are initially set so that the neurons are laid out in a rectangular matrix which matches the shape of the network's topology and uniformly covers approximately the central 15% of the input space. Each time a new input X is given to the network, its values are compared with the values of the weights of all of the neurons. To reduce computation time, the similarity measure used in this comparison was simply the sum of the differences between the entries in the input vector and the corresponding weights. (More commonly, the Euclidean distance would be used.) The neuron *s* which has weights most closely matching the input vector (according to the similarity measure) is called the “selected” neuron.

Formally, the selected neuron *s* is defined as

$$\begin{aligned} \exists s \Big[\left|w_{s_1} - X_1\right| + \left|w_{s_2} - X_2\right| \le \\ \left|w_{u_1} - X_1\right| + \left|w_{u_2} - X_2\right|, \\ \forall (u \in U), s \in U \Big] \end{aligned} \qquad (2)$$

where w_n is the pair of weights for neuron *n.* If more than one neuron satisfies equation (2), then one of these neuron is selected by some arbitrary method. The weights of the selected neuron are updated to match the input vector more closely according to the equation

$$w^{new} = w^{old} + \alpha (x - w^{old}) , \qquad (3)$$

where *x* is the entry in the input vector to which the given weight corresponds, and α is a scaling factor which, for PBMin, was kept at 0.002.

The neurons in the neighborhood of the selected neuron also have their weights updated according to equation (3). In this way, all the neurons in the network compete for selection and each neuron, when selected, cooperates with the others in its neighborhood to arrive at new input weight values.

3.3.3 Self-organized output

These concepts from Kohonen Maps are extended for control by adding one or more output values (weights) for each neuron. These output weights may be updated in a manner similar to that used for the input weights. For example, if an output response is given by the selected neuron, and this value differs from the correct out response, the output weight of the selected neuron might be updated according to the equation

$$v^{new} = v^{old} + \beta (r - v^{old}) , \qquad (4)$$

where *v* is the output weight, *r* is the correct response, and β is a scaling factor. This idea has, in fact, been used to create networks which can learn to self-organize output as well as input (e.g. [10], [17], and [18]). The limiting factor to these approaches is that they require a teacher be available to give the network a “correct” answer. These supervised learning schemes are appropriate to some problem domains, but other domains require that the learning system solve the problem independently.

3.3.4 Eligibility for adaptation

The use of an eligibility trace makes a teacher unnecessary in domains in which system performance is temporally dependant on network responses. For PBMin, the eligibility for adaptation *e*, for each neuron *n*, was calculated according to the equation

$$e_n^{new} = \delta e_n^{old} + \iota \sigma (o_n) , \qquad (5)$$

where δ is the rate of eligibility decay, ι is the initial eligibility for change for a neuron which has just fired, and σ is a function of the neuron’s output *o*. Only the selected neuron gives an output response, so σ is defined to be zero for all neurons besides the one selected. For non-selected neurons, then, equation (5) reduces to a simple decay in the level of eligibility. All neurons have their eligibility level initially set to zero.

Conceptually, PBMin has a single output weight for each neuron which may take on both positive and negative values; a positive value indicates a push (torque pulse) in one direction and a negative value a push in the other direction. PBMin also has two separate eligibility traces; one for reward and the other for punishment. The function σ is such that an eligibility for reward will increase the magnitude of a neuron’s output value without changing its sign. An eligibility for punishment, on the other hand, might result in either a reduction in magnitude without a change in sign, or might cause a change in sign and a change (either reduction or increase) in magnitude. Which of these results is the case depends on the neuron’s initial value and its eligibility for punishment. Both reward and punishment signals are generated only upon failure, so output values stay constant within a trial, but change between trials. The central difference between the eligibility for reward and that for punishment is the decay rate. For reward the decay rate δ_r is 0.99 and for punishment the decay rate δ_p is 0.90.

Building on the eligibility function, the output weights of the neurons are updated according to

$$v_n^{new} = \sum_{N_n} \frac{v^{old} + \rho (S) e}{\xi} \qquad (6)$$

where ρ is the feedback response from the controlled system *S*, and ξ is the number of neurons in the neighborhood in question. In this way neurons cooperate to ensure that nearby units have similar output values. Note, however, that there is no competition to be the neuron selected for change. All neurons are updated on failure (when there is feedback from the controlled system).

4 The Robot

The robot, like the learning system, is named PBMin. PBMin is an inexpensive, small, self-contained robot. It has its own onboard mini-computer and operates independently once the pole-balancing program is loaded.

Figure 3

4.1 Chassis

The chassis is made from a modified radio controlled car (see Figure 3). Most of the original electronic components, except for the motors, are removed. A 7.2 volt battery is attached to the back end of the chassis. The plastic that originally covered the radio controlled car is removed. The mini-computer board, the motor board, and the pole angle sensor are stacked in the middle of the car. PBMin's position sensor is mounted on the front of the chasis.

4.2 Mini-Computer Board

The mini-computer board contains the MicroController Unit (MCU) and memory (see Figure 4). The mini-computer board is 2 1/4 inches wide and 4 inches long.

The MCU is a 8 megahertz 68HC11A1FN microcontroller made by Motorola. PBMin makes use of two of the MCU's internal devices -- they are: (1) the analog to digital converters, which are used to get the angle of the pole and obtain PBMin's position and (2) the serial port, which is used to receive the pole balancing program from the main computer.

The memory on the mini-computer board is divided into Random Access Memory (RAM) and Erasable Programmable Read Only Memory (EPROM). The mini-computer board has 32 kilobytes of RAM, which is used to store and run the pole balancing program, and 16 kilobytes of EPROM, which is used to start up the MCU.

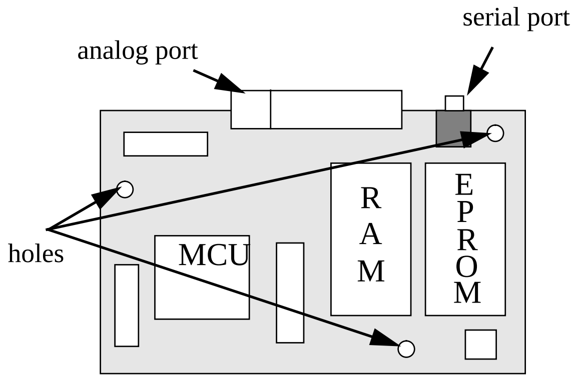

Figure 4

4.3 Motor Board

The motor board is the same size as the mini-computer board and is stacked immediately below it on the chassis. The motor board supplies power to PBMin's drive motor allowing the robot to go forwards or backwards, or stop.

4.4 Pole Angle Sensor

The pole angle sensor is a variable resistor attached to a pole (see Figure 5(a)). The pole that PBMin uses is a common yard stick. As the pole tips it turns the shaft of the variable resister. The variable resistor converts the angle of the pole into a voltage, which is read by the mini-computer board.

4.5 Cart Position Sensor

The position sensor is made up of two infrared transmitter and receiver pairs (IR modules, see Figure 5(b)). A transmitter sends out infrared light, which bounces off the paper disk, to its receiver. This paper disk has white and black regions on it (see Figure 5(c)) and is attached to PBMin's left front wheel. When PBMin moves forwards and backwards, the wheel and disk rotate, and the receiver senses different amounts of infrared light (see Figure 5(d) and (e)).

When an IR module senses a white region, it sends a signal to the mini-computer board. The mini-computer board looks at the two signals it receives from the two IR modules and interprets which way the wheel has turned and by how much.

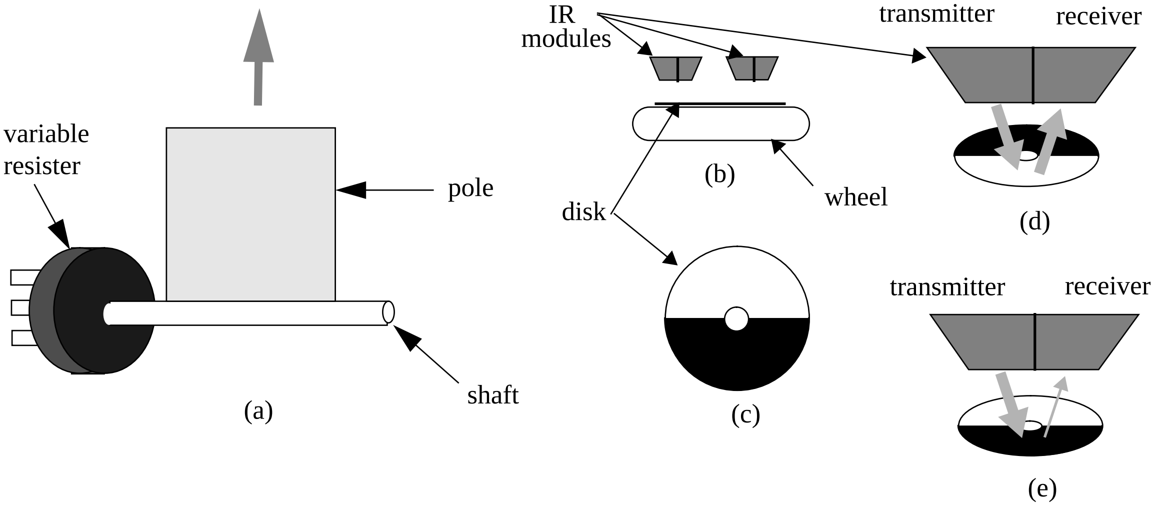

Figure 5

5 Results and Discussion

Experimental results have shown the PBMin learning system to be an effective real-time controller for pole-balancing and, by extension, that the SONNET paradigm (previously only used for the control of simulated systems) can be used for the construction of real-world controllers. The performance of the PBMin system is seen to approach optimal control for this robot on this task.

5.1 Experimental Set-up

The robot was run in a large, open, carpeted room. The size of the room dictated the size of the “track.” The robot allowed the robot to go 4 meters in either direction of start.

The robot was programmed with three different programs: (1) the PBMin learning system described above (see Subsection 3.3 “The PBMin Learning System”), (2) a preprogrammed solution that we believe produces optimal control given only the cart position and pole angle, and (3) a program that used a file of random numbers for completely random control of the cart-pole system. For each program, the robot performed runs of 50 trials and sent back results.

5.2 Results

As a baseline for comparison, the robot was controlled by random numbers. Three runs of fifty trials were performed. The average time for these trials was 1.6 seconds.

To get a measure of the potential balancing ability of the robot, the network was given the optimal solution. One run of fifty trials was performed. The average trial time of this run was 12.8 seconds.

The PBMin learning system was run with different sets of random initial output weights. Five runs of the first set of weights were performed and an average trial time of 3.3 seconds was obtained. One run each of three additional sets of weights were performed and an average trial time of 3.3 seconds was obtained. The fact that the average trial time stayed constant regardless of the values of the random initial weights demonstrates that the ability to of the system to learn is not dependent on the initial weight values.

5.3 Discussion

Because of the difficulty of this problem, many authors (e.g.[7], [8], [22], and [23]) have constructed systems which learn by watching the actions of a who person controls the cart-pole system and a few (e.g. [7] and [9]) have made systems which learn by observing the cart-pole system as it is controlled by another computer program. There systems are completely incapable of solving the pole-balancing problem without this outside aid. Others (e.g. [5]) have been forced to give their systems “hints” in order for them to solve the problem.

Most systems which are said to solve the pole-balancing problem without teachers or hints really do no such thing. In these authors’ systems (e.g. [3], [15], [19], and [21]) the input space is partitioned by the researcher, rather than by the learning system. This head start makes a significant difference. In one of the few instances where the entire pole-balancing problem is approached [1], a single layer neural network that can learn to balance a pole quite well when the input space partitioning is precomputed by the author, fails to learn much beyond the level of random control when applied to the entire problem and a two layer network takes, on the average, nearly 100 times as long to learn when the input space partitioning was not provided for it. (The only other case that we are aware of where learning pole-balancing is attempted without a pre-partitioning of the input space is in the first SONNET paper [11].)

PBMin is applied to the entire pole-balancing problem; it needs to dynamically partition the input space while learning the output space. To reduce the computational load on the processor, however, it was necessary to restrict the input vector to two dimensions. This means that our system could not take cart velocity or pole angular velocity into account. These elements of the state vector must be taken into account if long-term stability is to be achieved.

In the only previous paper in which researchers attempt to apply a control-learning scheme to a real cart-pole system [12], the input space is partitioned by the researches, rather than by the learning system. Because of this, and because of differences between their cart-pole system and the one presented here, direct comparisons between results obtained with the two systems may be misleading. For instance, one might note that their system never exceeded a run length of 10 seconds within the first 50 trials whereas ours never failed to achieve this milestone. However, in fairness to their system, it should be mentioned that such quick learning was probably never a goal for those researchers as their system was provided with an external power source, whereas ours had a very limited battery capacity. Or, one might note that, in some versions, their system achieved levels of success (as measured by time until failure) that ours never did. This success, however, came only after at least 150 trials whereas our runs were stopped at 50 trials due to limitations imposed by the battery.

Perhaps the best comparison for our learning system is with the performance of other learning systems on the same hardware. Unfortunately, other learning schemes are simply too computationally expensive to be implemented on our mini-robot in real-time. We can compare our results, however, with two ends of the automatic control spectrum. These are complete random control and what we believe to be optimal control given only the pole angle and cart position. The averages of each of these strategies are plotted along with the results of a single run of the learning scheme in Figure 6. As can be seen, the controller's learned behavior approached the average performance of the "optimal" controller.

It should be noted that the optimal controller varied widely in its performance. From a minimum of 90 time steps until failure to a maximum of 847, the performance of the optimal controller highlights the effects of initial conditions and noise on the results obtained by any control scheme for this cart-pole system.

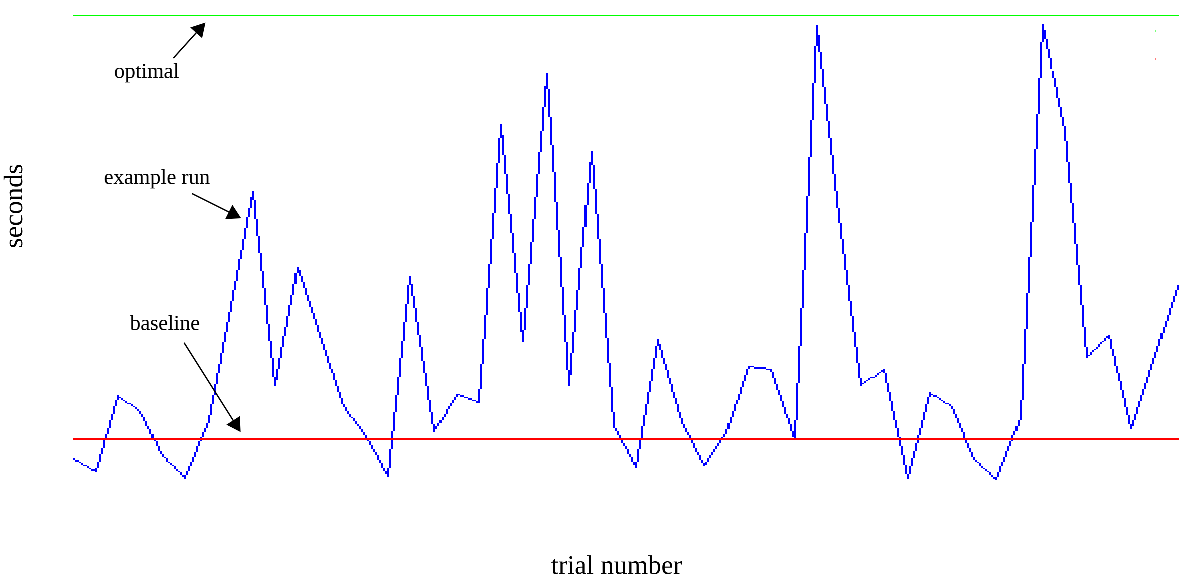

Figure 6

6 Acknowledgments

This research was carried out in its entirety at the Artificial Intelligence, Robotics, and Vision Laboratory of the University of Minnesota. We would like to thank the other members of the laboratory for their comments, questions, suggestions and support. Thanks in particular to Chris Smith for help on most aspects of this project, Mike McGrath for his work on the interface software, Maria Gini for her funding support, made possible by the AT&T Foundation and grant NSF/DUE-9351513, and her advice on dealing with real robots, and James Slagle for his help in developing the SONNET learning paradigm.

References

[1] C. Anderson. “Learning to control an inverted pendulum using neural networks.” *IEEE Control Systems Magazine*, Vol. 9, No. 3, 31-37, 1989.

[2] C. Anderson and W. Miller. “Challenging Control Problems,” in Neural Networks for Control, 475-510. W. Miller, R. Sutton, and P. Werbos, eds., MIT Press, Cambridge, MA, 1991.

[3] A. Barto, R. Sutton, and C. Anderson. “Neuronlike adaptive elements that can solve difficult learning control problems,” *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-13, 834-846, 1983.

[4] R. Cannon. Dynamics of Physical Systems, 703-710. McGraw-Hill, New York, 1967.

[5] M. Connell and P. Utgoff. “Learning to control a dynamic physical system,” in Proceedings AAAI-87, Vol. 2, 456-460. American Association for Artificial Intelligence, Seattle, 1987.

[6] P. Donaldson. “Error decorrelation: a technique for matching a class of functions,” in Proceedings: III International Conference on Medical Electronics, 173-178, 1960.

[7] A. Guez and J. Selinsky. “A trainable neuromorphic controller.” *Journal of Robotic Systems*, Vol. 5, No. 4, 363-388, 1988.

[8] A. Guez and J. Selinsky. “A neuromorphic contoller with a human teacher,” in IEEE International Conference on Neural Networks, Vol. 2, 595-602, 1988.

[9] D. Handelman and S. Lane. “Fast sensorimotor skill acquisition based on rule-based training of neural networks,” in Neural Networks in Robotics, G. Bekey and K. Goldberg, eds., Kluwer Academic Publishers, Boston, 1991.

[10] R. Hecht-Nielsen. “Counterpropagation networks.” *Applied Optics*, Vol. 26, No. 23, 4979-4984, 1987.

[11] D. Hougen. “Use of an eligibility trace to self-organize output,” in Science of Artificial Neural Networks II, D. Ruck, ed., SPIE 1966, 436-447, 1993.

[12] T. Jervis and F. Fallside. “Pole balancing on a real rig using a reinforcement learning controller.” Technical Report CUED/F-INFENG/TR115, Cambridge University Engineering Department, Cambridge, England.

[13] A. Klopf. “Brain function and adaptive systems -- a heterostatic theory,” in Proceedings of the International Conference on Systems, Man, and Cybernetics, 1974.

[14] T. Kohonen. Self-Organization and Associative Memory, Ch.5. 3rd. ed., Springer-Verlag, Berlin, 1989.

[15] D. Michie and R. Chambers. “Boxes: An experiment in adaptive control,” in Machine Intelligence, E. Dale and D. Michie, eds., Oliver and Boyd, Edinburgh, 1968.

[16] K. Ogata. System Dynamics, 531-536. Prentice-Hall, Englewood Cliffs, New Jersey, 1978

[17] H. Ritter and K. Schulten. “Extending Kohonen’s self-organizing mapping algorithm to learn balistic movements,” in Neural Computers, R. Eckmiller and C. von der Malsberg, eds., Vol. F41, 393-406. Springer, Heidelberg, 1987.

[18] H. Ritter, T. Marinetz, and K. Schulten. “Topology conserving maps for learning visio-motor-coordination.” *Neural Networks*, Vol. 2, No. 3, 1989.

[19] C. Sammut. “Experimental results from an evaluation of algorithms that learn to control dynamic systems,” in Proceedings of the Fifth International Conference on Machine Learning, 437-443. Morgan Kaufman, San Mateo, California, 1988.

[20] J. Schafer and R. Cannon. “On the control of unstable mechanical systems,” in Automatic and Remote Control III: Proceedings of the Third Congress of the International Federation of Automatic Control, Paper 6C, 1966.

[21] O. Selfridge, R. Sutton, and A. Barto. “Training and tracking in robotics,” in Proceedings IJCAI-85, 670-672. International Joint Conference on Artificial Intelligence, Los Angeles, 1985.

[22] V. Tolat and B. Widrow. “An adaptive ‘broom balancer’ with visual inputs,” in IEEE International Conference on Neural Networks, Vol. 2, 641-647, San Diego, 1988.

[23] B. Widrow. “The original adaptive neural net broom-balancer,” in International Symposium on Circuits and Systems, 351-357, 1987.