Real-Time Learning: a Ball on a Beam

H. Benbrahim, J. S. Doleac, J. A. Franklin, O. G. Selfridge

GTE Laboratories Incorporated
40 Sylvan Road
Waltham, Massachusetts 02254

Abstract

In the Real-Time Learning Laboratory at GTE Laboratories, we are implementing machine learning algorithms on actual hardware testbeds. We applied a modified connectionist actor-critic system to a real ball balancing task. The system learns to balance a ball on a beam in less than 5 minutes, and maintains the balance. The system has shown to be robust through sensor noise and mechanical changes; it has also generated many interesting questions for future research.

1 Introduction

This paper describes the work we are doing with a ball balancer in the Real-Time Learning Laboratory. A ball can roll along a few inches of a track on a flat metal beam, which an electric motor can rotate. A computer learning system running on a PC senses the position of the ball and the angular position of the beam. The system learns to prevent the ball from reaching either end of the beam.

Our purpose is to explore the powers and limitations of learning systems performing control in an actual mechanical domain.

1.1 Overview

Section 2 describes the hardware and software and their interaction, including the learning system. Section 3 presents the results of the learning system in balancing. Section 4 discusses those results, and what they mean in the light of parallel human manual balancing of the same hardware. The manual experiments are presented in Section 5. An appendix presents the actor-critic learning system in detail.

2 System Description

2.1 The Hardware-Software System

The balancer is a beam made of a 16 inch long by 4 inch wide section of aluminum attached at its center to a shaft, which a dc motor can turn in both directions. Two bumpers under the beam limit its movement to angles of about 20 degrees from the horizontal. The beam has a one inch high fence along one side. A metal ball rolls along the fence on the beam. Bumpers at the ends of the beam prevent the ball from falling off. See Figure 1.

A pressure sensor measures the ball's position; a potentiometer attached to the axle of the beam measures the beam's angle. A *Compaq* 386 PC using a data conversion interface reads those two positions.

This is a real-time task: at every clock interrupt of the PC (18.2 times a second), the computer reads the state of the system, issues the output command and updates the learning weights. The state of the balancer is represented by four variables: the position x of the ball, its velocity x', the angle θ of the beam, and its angular velocity θ'. Both velocities are calculated by evaluating the differences of the positions at successive clock interrupts.

Proceedings of the 1992 International Joint Conference on Neural Networks,
June, 1992, Baltimore, Maryland.

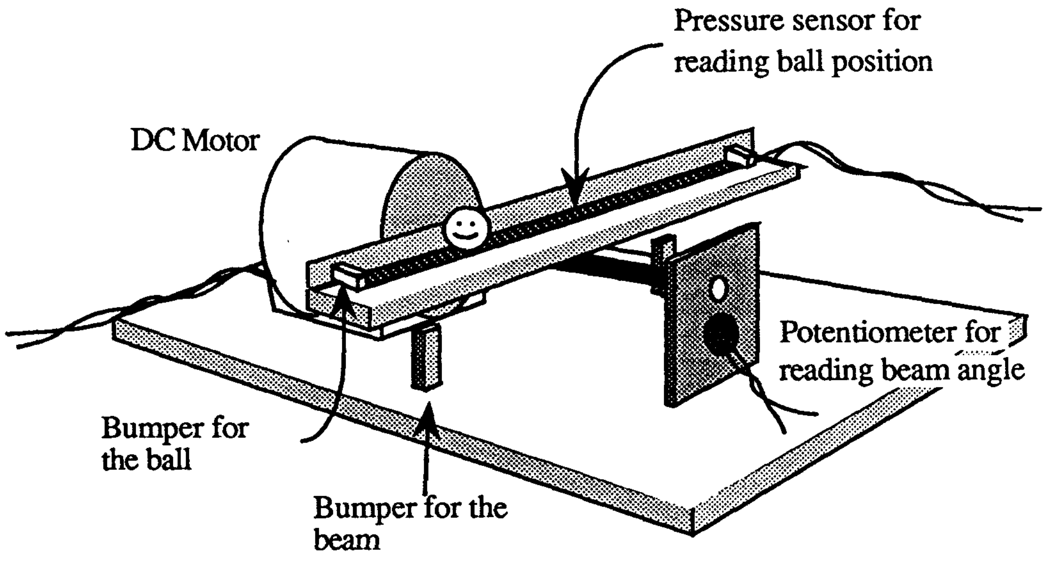
Pressure sensor for reading ball position
DC Motor
Potentiometer for reading beam angle
Bumper for the ball
Bumper for the beam

Figure 1 The ball balancer

The motor is controlled so that it is always trying to turn either clockwise or counter-clockwise. The PC issues commands to it through the data conversion interface, a power amplifier and a voltage-to-current converter.

2.2 Reinforcement Learning

The learning mechanism used in these experiments is a modified connectionist actor-critic reinforcement learning architecture [Barto-83]. The ideas behind it are roughly these:

The state of the system, as read and calculated from the sensor signals, is assigned to one of a number of quantized states, or *boxes* [Michie-68]. Each box can output a weight when it is occupied, and that weight is used to decide whether to turn the shaft clockwise or counterclockwise. This is the *action weight* In effect, it learns to predict the expected lifetimes of the ball after each of the two decisions it can make, and tries to make a decision that will maximize the overall expected lifetime.

When a box is occupied, the system elects that control action that leads to the longest estimated lifetime. When a failure occurs, then the system updates the weights of the boxes that contributed to that decision—more for the recent decisions, less for the more remote ones.

The *critic weight* of the box corresponds to the estimate that it makes about the outcomes of the two actions; those more or less match the expected lifetimes. The weight is changed after failure according to the *eligibility*, which is a decreasing function of time, as explained in the previous paragraph. The details are presented in the Appendix.

In our balancer, failure occurs when the ball gets too close to the end bumpers. Failure signals the end of the current "lifetime"; the learner then receives a *reinforcement* of $r = -1$, and the ball is automatically replaced outside of that failure zone.

The weights are updated at every time step in a way that is described mathematically in more detail in the Appendix.

Each of the four system variables is divided into regions: 5 regions for x, 6 for x', 3 for θ and 2 for θ'. There are also two failure zones. Each different combination of regions is a box. All this yields $5 \times 6 \times 3 \times 2 = 180$ boxes plus two failure zones.

The alignment and definition of these regions are clearly important. A future task is to test the system to ascertain its robustness with respect to those variables.

3 Results

The system learned to balance the ball faster and better than humans who tried.

3.1 The Experiments and Learning Curves

The software was written in C, and the experiments showed a high degree of replicability, but not a perfect one.

Figure 2 shows a typical learning series. The system performance is divided into *runs*; a run is an attempt to balance until a failure. The number of *steps* before failure is shown on the y-axis.

The data is plotted as single points, each a run; and as a running average, shown as a black line. The data is the number of steps before failure; in the figure, the data is topped at 300, but the actual performance far exceeded that at the right side of the figure. 300 steps represent about 15 seconds of real time.

It should be noted that the x-axis does *not* represent time, but successive failures, between which the time varies very widely.

The system learned to balance the ball. The results here show the learning curve; it is represented by the number of steps before each successive failure. We stop the experiment when the ball starts to balance for more than two minutes, because the better the system learns, the longer the experiments take.

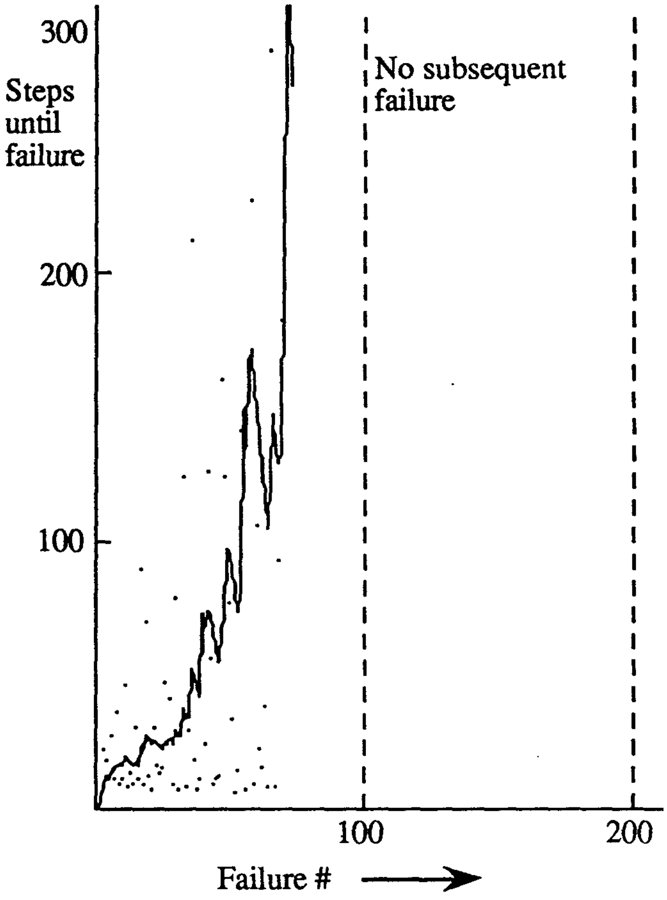

Figure 2 A learning series

3.2 Further experiments

As we mentioned above, the results typified by Figure 2 were not perfectly replicable; apparently the mechanical noise in the hardware caused enough differences to derail some series. Out of a total of 20 series, 16 learned (in the sense of Figure 2) in fewer than 200 runs, 3 in less than 300 runs, and one never did reach that success in over 1000 runs.

4 Discussion

It is of course important with a connectionist system not only that we show that it can perform useful learning in a real environment, but also that we explore the limitations of that performance and the assumptions and constraints underlying it. In doing so, we emphasize the similarities to and the differences from the pole-balancer.

4.1 The role of noise

In the simulations that were the basis for the pole-balancing theory [Barto-83], the action is chosen after adding Gaussian noise to the weight. This allows the balancer to try different actions for each box. In our experiments the balancer learns without the added noise. There is of course an inherent noise in the motor control, but it is not large enough to change the sense of rotation.The noise in the sensors, however, has a more important effect.

We have two kinds of noise in the sensors. One is the resolution of the sensors. It has effect only when the ball is near the boundary of a box. By choosing the boxes large enough this noise becomes irrelevant. The second noise which is however more important happens when the ball jumps; the position sensor which is a pressure sensor gives semi-random data.

In our experiments, noise in the sensors seemed to deteriorate performance; it should be noted that that noise is different in quality from the added Gaussian noise in the simulations. Furthermore, even in theory, it was not the "noisiness" of the noise that contributed to success, but only the fact of variations. Of course, a particular sequence of noise can lead to failure.

4.2 Boxes

Boxes are one way of dividing a difficult control problem into subproblems. Our division of the sensor ranges into regions was based merely on observations; it is an interesting subject for further exploration. There are clearly some tasks in which it may be important to avoid quantizing the state space in that way; it would be desirable to study the use of multi-layer nets, as was done for pole-balancing by [Anderson-88].

4.3 The parameters

We tested a range of parameters (those mentioned in the Appendix); by and large, the system performance was rather robust with respect to learning rates. In the experiments in the figures, all the parameters were set to 0.90.

5 Supervised learning: manual balancing

We gave the ball balancing task to a number of human subjects. A typical series is shown in Figure 3. Different subjects did very differently, but few approached the performance of the learning system. This was in spite of the fact the subjects had more *a priori* information available to them—for example, *to keep the ball in the center*, and *not to let the ball move very fast.*

The data of Figure 3 is identical in form to those of Figure 2.

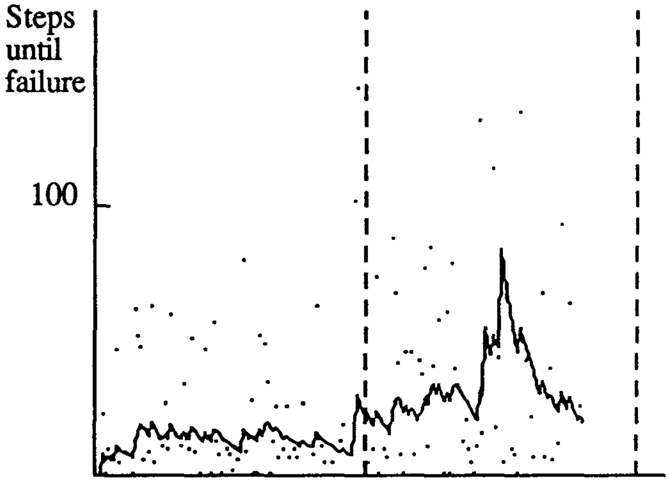

Figure 3 Human performance

It is reasonable in control tasks like this to be able to compare the system's performance with that of a person. In this case, however, a direct comparison is difficult, because the person and the system have to act with very different kinds of inputs. The system has instant knowledge of position—that is, its sensory perceptions have essentially no delays—but the person has neuronal processing delays that typically amount to a few tenths of a second. Nevertheless, it seemed useful to try to make the comparison.

We programmed the balancer to remember the actions when a person balanced the ball manually. For each box we took the average of the actions and updated the weights. When all the important boxes had been visited at least 10 times we ran the balancer with the weights it had learned without further updating them. With a perfect set of boxes the balancer should have the same performance as the teacher. On the average, the system performed at about 95% of the person's performance—5 minutes without a failure.

6 Conclusions

We have shown that learning systems can successfully handle control problems in real-time, and be robust to disturbances and mechanical changes.

7 Acknowledgments

We are grateful to Rich Sutton and John Vittal for helpful comments; and for specialized custom sensors to Interlink Electronics Incorporated.

*　　*　　*　　*　　*　　*

References

[Anderson-88] Anderson, C.W. "Learning to Control an Inverted Pendulum with Connectionist Networks," *Proc. Amer. Control Conf.*, 1988

[Barto-83] Barto, A. G., Sutton, R. S., & Anderson, C. W. "Neuronlike adaptive elements that can solve difficult control problems," *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-13: 834-846.

[Michie-68] Michie, D. & Chambers, R. "'Boxes' as a Model of Pattern-Formation" *Towards a Theoretical Biology; 1 Prolegomena*, C.H. Waddington, ed., 1968, Edinburgh University Press, Edinburgh

*　　*　　*　　*　　*　　*

Appendix

The actor-critic architecture is composed of two learning elements, the actor and the critic. The actor outputs the signal to the motor. The output, or action, is a function of the weights $\{w_i\}$ and the inputs $\{z\}$:

$$y(t) = f\left[\sum_{i=1}^{m} w_i(t)\; z_i(t)\right]$$

Here time t is a discrete variable, measured in 18.2 times a second by the hardware (55 ms steps).

If the actor receives a favorable reinforcement for its action, that action is more likely to be taken in the future when the actor is presented with the same state, according to the following weight update equation:

$$w_i(t + 1) = w_i(t) + \alpha \hat{r}(t) e_i(t)$$

where α is the learning rate for the actions weights;
$\hat{r}(t)$ is the effective reinforcement;
$e_i(t)$ is the eligibility, determined by the delay from the most recent entry into box i until failure.

The effective reinforcement used in this equation is determined by the critic. The critic takes the raw reinforcement r(t), and compares it to the predicted reinforcement:

$$\hat{r}(t) = r(t) + \gamma p(t) - p(t-1)$$

where γ is the discount factor for the critic, and
$p(t)$ is the prediction of the reinforcement.

When a box is entered, its eligibility is incremented:

$$e_i(t + 1) = \delta e_i(t) + (1 - \delta) y(t) z_i(t)$$

where δ is the eligibility decay rate.

At every step, all the eligibilities $\{e_i(t)\}$ are updated by multiplying by that decay rate.

The critic is updated in the same way, with its own parameters, and two more equations make this complete. These are the weight update equations for the critic:

$$v_i(t + 1) = v_i(t) + \beta \hat{r}_i(t) \bar{z}(t)$$

where β is the learning rate for the critic, and $\bar{z}$ is the eligibility for the critic; and

$$\bar{z}_i(t + 1) = \lambda \bar{z}_i(t) + (1 - \lambda) z_i(t)$$

where λ is the discount factor for the eligibility.

View publication stats