Self-Scaling Reinforcement: A new Algorithm for learning Continuous Control

Hamid Benbrahim
Judy A. Franklin

GTE Laboratories Incorporated
40 Sylvan Road
Waltham, MA 02254
e-mail: hbenbrahim@gte.com
jfranklin@gte.com

Abstract.

This article presents a new reinforcement learning algorithm. The algorithm generates a range of continuous real-valued actions, and the reinforcement signal is self-scaled. This alleviates the problem of choosing an appropriate learning rate and increases the learning speed. The algorithm is demonstrated within the actor-critic learning control architecture using a CMAC input representation. We apply the algorithm to a hardware ball balancer and show improvement in learning time and accuracy over similar methods.

1 Introduction

Much of the current research in applying reinforcement techniques to control/decision problems has suffered from two handicaps: the action outputs are restricted to a small finite set; and the operation is governed by learning rates for which it is hard to find appropriate values. Here we explore how to generate a continuous range of action outputs, and how to do so with parameters that the system adapts itself in order to improve its performance. We use a reinforcement algorithm that we term Self-Scaling Reinforcement (SSR); that is, the system automatically scales the reinforcement signal to enable more appropriate step sizes in updating the learning weights.

SSR is built upon Gullapalli's SRV (Stochastic Real-Valued) algorithm [Gullapalli-90]. In studying and comparing these methods for control, it is essential to test them on actual systems. Currently we apply reinforcement learning methods to the control of a ball balancer, a hardware implementation of a neural network control task— balancing a ball rolling on a tilt-able one dimensional beam so that it does not hit the bumpers at the ends. The ball balancer task is a conceptual successor to the pole balancer task [Barto-83].

2 The Ball Balancer Task

The ball balancer is a beam that a motor can turn clockwise and counter-clockwise, with different torques. A metal ball rolls freely along the beam. It is kept from falling off by a fence and two bumpers at either end. A pressure sensor measures the ball's position and a potentiometer measures the beam's angle. The velocities are obtained by subtracting consecutive readings.

Every time step (20 steps per second) a computer reads the ball's position and speed, and the beam's angle and angular velocity. The reinforcement learning algorithm generates an action to be sent to the motor. When the ball hits a bumper, the system gets a reinforcement r=-1, otherwise it gets r=0. Figure 1 shows the ball balancer.

We previously reported success in applying the SRV algorithm to the ball balancing task [Gullapalli-94].

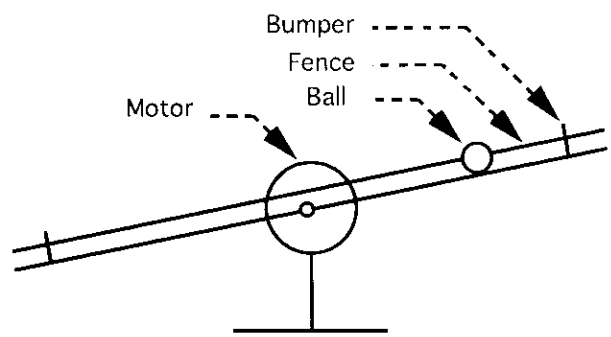

Figure 1: The Ball Balancer

in the Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems, held June 13-15, 1994 in New Haven, CT.

3 The Stochastic Real-valued (SRV) Algorithm

For completeness, we start by examining Williams' [Williams-92] statistical gradient following algorithm that uses a stochastic output unit. This unit is a Gaussian random number generator with mean μ and standard deviation σ. For every input vector x, the algorithm generates $\mu(x)$ and $\sigma(x)$. These are used to generate the action y(x) according to the probability distribution:

$$g(y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-\mu)^2}{2\sigma^2}}, \qquad (1)$$

where the input vector x is omitted for clarity.

The learning algorithm is governed by a weight vector w, that is updated by:

$$w(t+1) = w(t) + \alpha(r - \bar{r})e(w), \qquad (2)$$

where α is a learning rate, r is the reinforcement and $\bar{r}$ is a reinforcement base line, a prediction of r. $(r - \bar{r})$ is a factor that measures the difference between actual and predicted reinforcement. e(w) is a measure of how eligible each weight is for updating.

Williams defines the characteristic eligibility e(w) by the equation:

$$e(w) = \frac{\partial \ln g}{\partial w}. \qquad (3)$$

To better understand how this algorithm works let us develop these equations for the case where linear units produce μ and σ:

$$\mu(x) = w_\mu^T x, \text{ and } \sigma(x) = w_\sigma^T x.$$

$$e(w_\mu) = \frac{\partial \ln g}{\partial \mu}\frac{\partial \mu}{\partial w_\mu} = \frac{(y-\mu)}{\sigma^2}x, \text{ and}$$

$$e(w_\sigma) = \frac{\partial \ln g}{\partial \sigma}\frac{\partial \sigma}{\partial w_\sigma} = \frac{(y-\mu)^2}{\sigma^3}x.$$

The weight vectors are updated as follows:

$$w_\mu(t+1) = w_\mu(t) + \alpha_\mu(r - \bar{r})\frac{(y-\mu)}{\sigma^2}x, \qquad (4)$$

$$w_\sigma(t+1) = w_\sigma(t) + \alpha_\sigma(r - \bar{r})\frac{(y-\mu)^2}{\sigma^3}x. \qquad (5)$$

Note that if the output yields a reinforcement better than predicted, w_μ will be updated to move μ toward y. This will increase the probability that y is the action to be chosen next time the same input x occurs. And if r is less than predicted, μ will move away from y to decrease this probability. In the case of w_σ the standard deviation increases or decreases to determine the extent of action domain exploration. Williams proves the convergence of these types of algorithms.

Even though these methods converge, we found that in practical applications, the learning is very slow and it is very hard to find appropriate leaning rates. Notice also that when σ is very small, the term $1/\sigma^3$ overflows.

Gullapalli's SRV algorithm presents a significant improvement [Gullapalli-90]. It updates the unit's parameters as follows:

$$w_\mu(t+1) = w_\mu(t) + \alpha_\mu(r - \bar{r})\frac{(y-\mu)}{\sigma}\frac{\partial \mu}{\partial w_\mu}. \qquad (6)$$

$$\sigma = k(1 - \bar{r}), \qquad (7)$$

where k is a multiplying factor. It is assumed here that the maximum reinforcement is 1. In general this equation would be

$$\sigma = k(\max(r) - \bar{r}).$$

When the system has learned, the predicted reinforcement becomes close to the maximum reinforcement and σ becomes very small. This means that the action will be essentially equal to μ, and thus the search will be stopped.

In the example where $\mu(x) = w_\mu^T x$, the adaptation equation for $w_\mu(t)$ becomes

$$w_\mu(t+1) = w_\mu(t) + \alpha_\mu(r - \bar{r})\frac{(y-\mu)}{\sigma}x, \qquad (8)$$

$$r - (q(n-1) - \gamma q(n)) \quad (16)$$

where $0<\gamma<1$ is a constant gain. The idea is that the critic predicts discounted future reward. The critic solves the problem of "delayed" reinforcement via computing a difference in successive predictions in equation (16) and via another mechanism called eligibility traces. Eligibility traces are moving average filters of the inputs. They provide a decaying history that aids the algorithm in learning, when the reward signal is delayed. Successive predictions and eligibility traces are used here for ball balancing just as they are used by Barto et al. [Barto-83] in pole balancing.

- CMAC Neural Network

The sensor values from the ball balancer are inputs to a CMAC input representation. The CMAC was first proposed by Albus [Albus-75] and has been used extensively by Miller et al. [Miller-90] in robotic learning control. The CMAC combines table lookup with the generalization and nonlinearity of arrays of overlapping receptive fields, each array offset from the others. It is a more powerful representation than simple boxes and provides a faster avenue for learning in nonlinear networks than say a backpropagation network. We will not describe the CMAC here. The reader is referred to the above references for more details. The outputs of the CMAC representation become the inputs x to the SSR or SRV algorithm (the actor). The same inputs are used for the critic.

6 Results

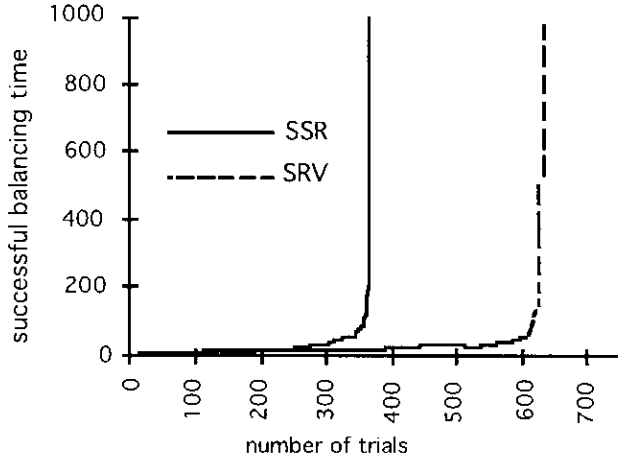

Figure 2: Learning curves

Figure 2 shows the learning curves for SRV and SSR. The curves represent the number of successful steps versus the number of trials. Each trial ends when the ball fails.

We found that SSR learns faster than SRV. After 350 trials, which corresponds to about 10 minutes, SSR learns to keep the ball balanced indefinitely, while SRV learns after 620 trials. We do not show an average of several runs, however, these results are consistent with all our experiments.

7 Discussion

We showed in this paper that SSR learns to balance the ball even faster than SRV and does not require a learning rate, while it was very difficult to find an appropriate learning rate for SRV. SSR continuously moves toward the actions that yield high reinforcement without overshooting. SRV follows a gradient to maximize the expected reinforcement. In other words SSR moves toward a specific goal while SRV follows a specific direction.

Even though SRV and SSR yield positive results, it has not yet been proven that they do, in fact, converge. With our experimentation with the ball balancer we found that SRV converges only when using small learning rates, and SSR converges constantly.

Our next step is to develop an algorithm that combines SSR with real-valued Q Learning. We anticipate better results because of the advantages that Q learning has over the actor-critic method [Watkins-89].

8 Acknowledgments

We gratefully acknowledge Oliver Selfridge and John Vittal for thoughtful suggestions and continual support.

9 References

[Albus-75] Albus, J. S. (1975). "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)" September 1975 ASME Journal of Dynamic Systems, Measurement, and Control, pp. 220-227

Gullapalli's algorithm shows great improvement in the learning speed, and is successful in different applications [Gullapalli-93, Gullapalli-94].

4 The Self Scaling Reinforcement algorithm (SSR)

The methods above suffer from a major handicap: If the system gets a very large reinforcement, let us say infinite, the weights will overshoot and therefore impede the system performance. If the reinforcement range of variation is known, the learning rate can then be adjusted to prevent overshooting. When this is not the case, however, a very small learning rate is used, and a slow learning process ensues.

SSR is based on the concept that if a certain action yields an infinite reinforcement then it should have its probability greatly increased. Moreover, since it is not guaranteed that the system will receive the same level of reinforcement for different input vectors, the notion of infinite reinforcement should be flexible. The best reinforcement the system gets is considered infinite and the worst is considered negatively infinite. We keep track of the maximum and minimum reinforcement, rmax and rmin respectively, and compute a scaled reinforcement $\hat{r}$ as follows:

$$\hat{r} = \exp(r - rmax) - \exp(rmin - r). \quad (9)$$

Notice that if r = rmax, then $\hat{r} = 1 - \exp(rmin - rmax) = 1 - \varepsilon$, and if r = rmin, then $\hat{r} = \exp(rmin - rmax) - 1 = \varepsilon - 1$. ε is very small when rmin-rmax << 0; consequently, we will temporarily ignore it for clarity purposes.

We compute rmax and rmin according to:

$$\text{if } r > rmax \text{ then } rmax = r, \quad (10)$$

$$\text{if } r < rmin \text{ then } rmin = r, \quad (11)$$

$$rmax(t+1) = \lambda rmax(t) + (1-\lambda)r, \quad (12)$$

$$rmin(t+1) = \lambda rmin(t) + (1-\lambda)r, \quad (13)$$

where λ is a positive number < 1. Equations (10)-(13) work together as follows. rmax increases as the system gets higher reinforcement values (eq 10). rmin decreases as the system gets lower reinforcement values (eq 11). Equations (12) and (13) allow the difference between rmax and rmin to converge, as the system learns, toward zero. rmin increases when low reinforcement values are infrequent (eq 13). We deliberately decrease rmax in order to filter out large spurious reinforcement values (eq 12). We have as yet no proof of convergence, except by empirical observation.

The algorithm's adaptation equations are

$$w_\mu(t+1) = w_\mu(t) + \hat{r}(y-\mu)\frac{\partial \mu}{\partial w_\mu}, \quad (14)$$

$$\sigma(t+1) = \gamma\sigma(t) + (1-\gamma)(rmax - rmin), \quad (15)$$

where γ is a positive number < 1. It is used as an averaging factor.

SSR is reminiscent of gain adaptation in supervised learning systems [Jacobs-87]. Equation (14) is of the form $w(t+1) = w(t) + \alpha ex$, where e is the error signal and α is a positive gain < 1. When r≥rmax, $\hat{r}$=1. This is equivalent to having an error e=(y-μ) and a gain α=1. When r≤rmin, $\hat{r}$=-1, e=-(y-μ) and α=1. If r is mid-distant from rmax and rmin, $\hat{r}$=0, equivalent to α=0. Thus, when $\hat{r}$ ranges continuously from -1 to 1, α ranges continuously from 1 to 0 then to 1. SSR adapts the magnitude of the gain and the sign of the error. The sign of the error depends on whether the reinforcement for the actual action is better or worse than average, and the magnitude of the gain depends on how close the reinforcement is to rmax and rmin. As the system learns (rmin - rmax) converges toward zero and thus ε (i.e. exp(rmin - rmax)) converges toward 1. Consequently, $\hat{r}$ converges toward zero. In terms of gain adaptation this behavior means that α decreases as the system approaches its target.

5 Learning Architecture

- Actor-Critic

The actor-critic architecture deals effectively with minimal reinforcement. The actor outputs the action or control, and the critic evaluates the action, given the "raw" performance evaluation r. The role of the critic is to assign an evaluation to each action. Without the critic, the system would receive only a 0 or -1. Since the system gets a nonzero reinforcement only when the ball fails, it needs the critic to build an evaluation function for the states that are far away from the failure states. As a result the system learns that having the ball in the center of the beam with a very small speed is the best situation. Without the critic, this situation would look to the system like any nonfailure situation. The critic network outputs a prediction q(n) of the reinforcement [Sutton-84]. Using this prediction, we obtain the following modified reinforcement signal:

[Barto-83] Barto, A.G., Sutton, R.S., Anderson, C.W. (1983). "Neuronlike elements that can solve difficult learning control problems." *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-13, 834-846, 1983.

[Gullapalli-90] Gullapalli, V. (1990). "A stochastic reinforcement learning algorithm for learning real-valued functions." *Neural Networks*, 3, 671-692

[Gullapalli-93] Gullapalli, V. (1993). "Learning control under extreme uncertainty." C.L. Giles, S.J. Hanson, and J.D. Cowan, editors, *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993. Morgan Kaufmann Publishers.

[Gullapalli-94] Gullapalli, V., Franklin, J. A., Benbrahim, H. (1993). "Acquiring Robot Skills via Reinforcement Learning." IEEE Control Systems Magazine, Vol. 14, February 1994.

[Jacobs-87] Jacobs, R. J. (1987). "Increased Rates of Convergence Through Learning Rate Adaptation," COINS Technical Report 87-117. Dept. of Comp. and Info. Science, U. Mass, Amherst, MA.

[Miller-90] Miller, W. T. III, Hewes, R. P., Glanz, F. H., and Kraft, L. G. III (1990). "Real-Time Dynamic Control of an Industrial Manipulator Using a Neural-Network-Based Learning Controller" IEEE Transactions on Robotics and Automation, Vol. 6, No. 1, Februrary 1990, pp. 1-9.

[Sutton-84] Sutton, R.S. (1984). "Temporal Credit Assignment in Reinforcement Learning," Doctoral Dissertation, Dept. of Comp. and Info. Science, U. Mass, Amherst, MA.

[Watkins-89] Watkins, C. J. C. H. (1989) . "Learning with Delayed Rewards" Doctoral Dissertation, Psychology Department, Cambridge University, 1989.

[Williams-92] Williams, R.J. (1992) . "Simple statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning." *Machine Learning*, 5, 8-229.