AHC for stochastic minimum cost path problems using Neural Net Function Approximators and some non-neural net results found after the workshop

Stuart E. Dreyfus
IEOR Dept.
University of California at Berkeley
Berkeley, CA. 94720
e-mail: dreyfus@euler.berkeley.edu

August 29, 1995

I plan to begin my presentation by suggesting that the adaptive heuristic critic (AHC) idea implemented using neural network value and action function approximators seems a plausible explanation of animal learned skillful coping behavior. I will note that the output of the function approximators should depend, not only on current stimulae (state), but on some internal state of the approximator that in turn depends on the prior trajectory of the process. (An Elman net is a particular realization of this requirement.) This automatically introduces sensitivity of value and action to whatever in the past is relevant to the dynamics and reinforcement. Having touched on this intriguing conceptual matter, I will quickly turn to an unrelated, mundane but practical, concern.

Suppose that, in an AHC algorithm, the value and action mappings are approximated via a parametrized functional form (e.g., a neural network). Parameters are adjusted to fit learned values and actions using TD(0) for the value adjustment. Let squared error=(value of current step+value of finishing given the next state-value of finishing given current state)**2. Changing the mapping parameters affects the second and third term of the squared error. On a small toy problem which is a special case of a markovian decision problem, if the dynamics are deterministic and parameters are adjusted in the negative gradient direction based on the above observation about parameter effects and the step-size slowly goes to zero, the right answer appears to result. Interestingly, if one neglects the effect of parameters on the second term in the squared error above when computing the "gradient", all still goes well, perhaps even more rapidly. Now, and this is the point of my presentation, if the dynamics are stochastic, using the wrong gradient (where the effect of parameter changes on the second term in the squared error is neglected) seems to yield almost, but not quite, correct expected values, while using the right gradient including the effect on the second term yields very wrong results. While I have some intuition about why the right gradient gives the wrong result for a stochastic problem, I have none concerning why the wrong gradient appears to give the right result for the deterministic case and almost, but not quite, correct results for the stochastic case. The listener is encouraged to use the wrong gradient in his/her AHC computations for stochastic problems, and hopefully to figure out why it almost works. More crucial, one wonders, since both gradients are probably provably wrong, if any procedure using a parametrized function approximator is provably correct or if "approximately optimal" is the best one can hope for concerning markovian decision models.

In particular, for concreteness and because they have been the subject of many of my computations, I shall talk about very short duration, triangular, shortest path problems. Described in words (since I don't know how to put figures into an e-mail), one starts at a point (call it (1,4) for concreteness), chooses either decision 1 or decision 2, observes the next point (which is either (2,5) or (2,3)) and a cost where CU(i,j) denotes the diagonally upward cost from (i,j), e.g. CU(1,4) is the cost of going from (1,4) to (2,5), and CD(i,j) denotes the diagonally downward cost, e.g., CD(1,4) is the cost of going from (1,4) to (2,3). Wherever one is at after stage 1, the process repeats, with a decision being chosen and a cost and a transition, either diagonally up or down, being observed. This continues for N stages. For deterministic problems, total cost is to be minimized; for stochastic problems where a decision does not always yield the same next point, expected total cost is to be minimized.

In my experiments, the transition cost is always deterministic and given by a table of CU's and CD's for possible points. For deterministic problems, decision 1 has probability 1 of going diagonally up and decision 2 has probability 1 of going diagonally down. For stochastic problems, decision 1 has probability PU1 of diagonally up (and 1-PU1 of diagonally down) and decision 2 has probability PU2 of diagonally up (and 1 -PU2 of diagonally down). The cost data and values of probabilities are, of course, unknown to the AHC algorithm and, through repeated passes through the network, optimal decisions are to be learned, as well as minimum (expected) costs, but not the data that is used by the "world" to generate costs and transitions.

My AHC algorithm uses a feedforward, 2-level, neural network to map the coordinates of a starting point into the (hopefully) minimum cost of starting there. It uses a second, similar, net to map coordinates into an output used to determine the probability of choosing decision 1 at that point. The formula is a sigmoid such that P(dec. 1)= 1/(1+exp(-output)). The nets have tanh activation functions for the hidden nodes and a linear top layer. If C(i,j) denotes the output of the cost net for coordinates (i,j), (i+1,k) denotes the next point presented by the "world" and "cost" is the cost of that step, the squared error function at a step is (cost+C(i+1,k)-C(i,j))**2. After a decision is made at (i,j) using the action net, the cost and next state are observed. The value net is run with (i,j) as input and also with (i+1,k) as input in order to compute the squared error. Then either the correct or wrong gradient described above is computed using backpropagation and the net's weights adjusted. Depending on the decision taken and the sign of the error in the squared error term, the action net's weights are adjusted via backpropagation to either reinforce or inhibit the decision taken. (Details of this action net weight adjustment seem irrelevant to the issue I'm addressing.) This process is repeated for each step through the network, and for many passes through the entire network. For an N-stage problem, C(N+1,k)=0 rather than what the net gives for that input.

For N=2, CU(1,4)=5, CD(1,4)=1, CU(2,5)=2, CD(2,5)=4, CU(2,3)=1, CD(2,3)=3, using dynamic programming to solve the deterministic version of the problem one gets C(2,5)=2, dec. 1, C(2,3)=1, dec. 1, and C(1,4)=2, dec. 2. For the stochastic problem with PU1=.75 and PU2=.25, D.P. gives C(2,5)=2.5, dec.1, C(2,3)=1.5, dec. 1, C(1,4)=3.75, dec. 2.

It might be noted that this sort of path problem is particularly difficult for AHC as described since values are in no sense smooth functions of input coordinates. The size of initial (random) neural network weights, the relative rate of adjustment of value network weights and of action network weights, and the representation of the points of the path network (one can use coordinates as above, or some other initial point rather than (1,4), or normalized versions of the coordinates etc.) all are crucial to performance, and suboptimal solutions are common. It took a great many tries before I could get a correct solution to a deterministic 3-stage version of the above problem. Hopefully some approximator more capable of dealing with value or action mappings that are not smooth functions of their arguments can be found. (The action, when an outfielder pursues a fly ball, for example, is not a continuous function of the ball's trajectory because, at some point, the fielder sometimes abruptly changes direction if the ball is going to hit high on the outfield wall.) A difficult problem such as this minimum cost path problem for which one can easily compute the correct answer using D.P. has its advantages for purposes of testing various mapping functions other than neural networks. On the other hand, the difficulties I encountered for such trivial problems when using neural networks could give AHC a bad name.

****

Following is a report on certain experiments performed after the workshop motivated by Leemon Baird's remarks after my talk and on his paper presented at the ML meeting. In them I used one of two simple function approximators rather than a neural net. One was linear in parameters and nonlinear in state and the other was nonlinear in both (as is a multilayer perceptron). They were performed on a two stage problem where at each stage a diagonally up move had prob. 3/4 and down had prob. 1/4. At stage 1, called vertex x=1 for purposes of input into the function approximator, up costs 4 and down costs 1. If the first transition is down the next vertex, called vertex 2, costs 1 if up and 5 if down. If the first transition is up to 3, up costs 4 and down costs 0. Hence the expected cost of finishing, starting at vertex 2, is 2; at vertex 3 is 3; and at vertex 1 is 6. In all experiments, an epoch consisted of cycling through the 3 vertices at which values are to be learned rather than following a trajectory (since the Baird method precludes using real trajectories). I would generate a random transition from vertex 1, compute E=cost +V(next)-V(Current) where V(x,W) is defined below as a function of state and parameters of the approximator. (No decision is considered in this work). At both vertex 2 and 3, E=cost-V(current) (i.e. V(next) is taken as 0 and is not to be learned). The gradient is the partial of E squared with respect to W, i.e. 2E times partial E partial W. After computing E based on the transition, at vertex 1 for the Baird method a second random transition is used to compute partial E partial W. For non-Baird, the original transition is used. W's are changed by .0001 times the negative of the gradient. Initially the W's are chosen to give close to the correct answer, 500,000 epochs are repeated with averaging of V's over the last 50,000; then 500,000 more with averaging over the last 50,000 to get the final V's (which always agreed with the first averaging so convergence seems to have occurred.) The two formulas for V were:

```
Linear in W:  V(x)=W1*x+W2*x**2+W3*x**3.
Nonlinear in W:  V(x)=W1*x+(W2**3+W2)*x**2 +W3*x**3
```

The results follow:

```
Baird, linear:  V(1)=5.89, V(2)=1.98, V(3)=2.95
Baird, nonlinear:  V(1)=5.09, V(2)=1.99, V(3)=2.99
Non-Baird, linear:  V(1)=5.63, V(2)=2.50, V(3)=2.41
Non-Baird, nonlinear:  V(1)=4.79, V(2)=2.25, V(3)=2.50
Non-Baird, linear, incorrect gradient:  V(1)=5.90, V(2)=1.96, V(3)=2.97
Non-Baird, nonlinear, incorrect gradient:  V(1)=5.47, V(2)=1.75, V(3)=3.42
```

Tentative conclusion: For nonlinear in parameters, no method gives correct result. For linear in parameters, both Baird and non-Baird with incorrect gradient seem more or less correct while non-Baird and nonlinear fails. No method can give exactly correct results because changing weights at one point affects the other vertices and E never goes to 0 for a stochastic process.