Course Notes
CECS 477 : Neural Networks
Lecture of February 16, 2000
By Jim Ries, JimR@acm.org

Reinforcement Learning

- AI machine learning dates back to late 1950's (most in late 198's, early 1990's)
- Operations Research (control)
- Markov decision processes
  - Make decisions based on current state. State of world changes. New state stochastically determined by previous state and current conditions (nothing in past).
- McCallum's paper asks how add history

Markov Decision Process

- Discrete time, stochastic system with discrete state space

$t=0,1,2,\ldots$
$s=(1,2,\ldots\}$
given state I, choose action $u \in U(I)$
i   U(I)
i   cost, our task is to minimize total cost
or
i   reward, our task is to maximize total reward
given i,u   j (new state) $P_{ij}(u)$
where $P_{ij}(u)$ is the probability of going from state i to j given action u
action policy:
μ_i i   u
$J^{\mu}(i)$ - estimate of plan μ starting form state i
infinite horizon: $J^{\mu}(i) = E_{\mu}[$ for t=0 to t=∞ $\gamma^t c_t \mid s_0=i]$

c_t = cost in time step t
s_0 = start state
γ^t = discount factor (if reward is in future, it matters less)
$0 < \gamma \leq 1$ (typically, γ would be decreasing with time)

finite horizon is same, but iterates to M (limit) only, and thus discount factor can be 1 without problem.

Sometimes, we want average cost, e.g.,

E_{μ} [for t=0 to t=∞ $\gamma^t c_t \mid s_0=i] / M$
(or the lim M ∞ of above in infinite case)

We use dynamic programming to find optimal policy. Reinforcement learning is a variant of dynamic programming.

value iteration

policy iteration

offline - traditional dynamic programming
online - reinforcement learning

<u>Bellman equation</u>

- idea: improve policy and estimates of value of policy through many iterations (successive approximations) and at the same time improve the policy.

$J_{k+1}(i)$ = min where $u \in U(i)$ $[C_\gamma(u) + \gamma * j \in s \quad P_{ij}(u)\, J_k(i)] = Q(i,u)$
above equation without "k's" is Bellman equation

greedy policy

"one-step backup" - uses next state to update estimate of current state.

<u>Policy Iteration</u>

- attempts to decouple learning of policy from learning of the value function

$J(i) = c_\gamma(u) + \gamma * j \in s \quad P_{ij}(u)\, J(j)$
where u is action chose $u = \mu(i)$
update policy, go back, …

<u>algorithm:</u>

```
init current policy
while current policy ≠ next policy
    learn value function of policy
    update policy (greedy actions based on current learned value function)
endwhile
```

Process will converge to optimal policy

- In value iteration, we use greedy policy at every step. Since value function is changing, so is the policy. Once we learn value function, the policy will stop changing.
- In policy iteration,…

Claim 1: Only policies that are greedy with regard to their own value function are optimal policies.
$Q^{J^*}(i,\mu^*(i))$ = min $u \in U(i)$ $Q^{J^*}(i,u)$
μ^* is optimal policy
J^* is value function of μ^*

Claim 2: A necessary and sufficient condition for value function to be value function of the optimal policy is:
$J(i)$ = min $u \in U(i)$ $[C_i(u) + \gamma * j \in s \quad P_{ij}(u)\, J(j)]$ (Bellman Equation)

Thus, a solution to Bellman's equation leads to an optimal policy.

Synchronous update: $J_{k+1}(i)$ = min Q(i,u) (update all states in parallel)
Asynchronous update: update each state iteratively.
Both converge.

Real-time Dynamic Programming (an instance of on-line)

- learn value function while interacting with the world
- update only states we "visit" (asynchronous updating)
- intermediate results will be used in control
- must update every state infinitely often to converge (can use random exploration scheme to assure all states are explored). Can also estimate confidence for each region, and intentionally visit states not often visited.

Random exploration

Prob(i,a) = $e^{Q(i,a)/t}$/ for all a $e^{Q(i,a)/t}$
t = "temperature"; t starts very high and settles ("cools") down after a while. So, we start random and make it more deterministic over time.

- this is essentially "simulated annealing"

Dynamic Programming with incomplete information

- e.g., we don't have probability of transition
- Baysian or Non-baysian estimate of transition probabilities

Non-baysian - indirect estimate. Direct estimate is Q-Learning (model-free RL)

Bartol, et. al. 1990

indirect estimate
$P_{ij}(u) = \eta_{ij}^{u} / \quad l \in s\ \eta_{il}^{u}$
where η_{ij}^{u} is # of observed transitions from i to j after action u
(maximum likelihood estimate)

Q-Learning
Off-line Q-learning
Adaptive Q-learning
Real-time Q-learning (also just called "Q-learning"; original Q-learning)

off-line Q-learning
$Q_{k+1}(i,u) = (1- \alpha_k(i,u))Q_k(i,u) + \alpha_k(i,u)[c_i(u) + \gamma$ * min over u' Q(succ(i,u), u')]
where α is the learning rate; rewrite as:
$\Delta Q_{k+1}(i,u) = \alpha_k(i,u)[c_i(u) + \gamma$ * min over u' Q(succ(i,u), u') - $Q_k(i,u)$]

So, since we update many times, and only learn a little (α) each time, we can try many successors and learn most probable ones.

In real-time, we have no model of world so we just get successor from the real world (prevailing conditions).

α should decrease over time. If each pair updated infinitely often and α decreases over time, algorithm will converge with probability 1.

Q-Learning	**Value Iteration**
more space	only need value of each state
update value many times	one-step updates
each step fast (good for real-time)	each step slow (due to)

Real-time and offline can be mixed. "Contemplate" after actions learned.

Reference:
Barto, A.G., Bradtke, S.J., and Singh, S.P. (1995) Learning to act using real-time dynamic programming, Artificial Intelligence 72: 81–138.
(see http://www.umass.edu/neuro/faculty/barto.html for more on Andrew Barto)

Storing Value functions (Q or J)
Exact Method
Lookup table representation, i.e.,

State	Action	Q-value
…	…	…

Approximate methods:
- linear, weighted sums
- back propagation net
- decision trees
- radial basis function
- statistical clustering to minimize space usage
- Baysian Network

Paper presented by Jonathon Marjamaa
"Self-Improving Reactive Agents Based On Reinforcement Learning, Planning, and Teaching"
By Long-Ji Lin, Carnegie Mellon University 1992

http://riesj.hmi.missouri.edu/CECS477/RL/