Q-factors and Q-learning

Stephen Pickett, 3.3.2008

Simulated Value iteration, one approach. Described by Watkins, 1989.

Simplest learning case.

Single Binary choice: eg fork in the road. Model for all brain patterns. What happens if I choose A or B, and can I use this to decide next time?

Update Q-factors by experience, ie iteration.

Q-factors and Q-learning

Q-factors can be used to express the cost-to-go of different choices at a given state in discrete systems, and Q-learning algorithm reduces to the same set of computations as those for shortest path networks.

However the method is much more powerful because it can be used to iterate without a clear system model that would allow from-the-end-backwards calculation of the Q-factors.

$$Q^*(0,u)=0$$

$$Q^*(i,u)=\sum_{0}^{n} p_{ij}(u)(g(i,u,j)+J^*(j)), \quad i=1,\ldots..,n$$

We can make this look like the Bellman Eq, and so the Q*'s are the solution to that.

$$J^*(i)=\min_{u\in U(i)} Q^*(i,u)$$

then:

$$Q^*(i,u)=\sum_{j=0}^{n} p_{ij}(u)(g(i,u,j)+\min_{v\in U(j)} Q^*(j,v))$$

Can be applied to stochastic Shortest Path problems as well as deterministic ones.

Value iteration using simulation, visit nodes according to p(u).

$$Q(i,u)=(1-\gamma)Q(i,u)+\gamma\left(\sum_{j=0}^{n} p_{ij}(u)(g(i,u,j)+\min_{v\in U(j)} Q(j,v))\right)$$

γ is the stepsize. As it diminishes, Q converges.

Q Learning Algorithm goes as follow (from Teknomo)

1. Set parameter , and environment (reward matrix R)
2. Initialize matrix Q as zero matrix
3. For each episode:
    * Select random initial state
    * Do while not reach goal state
        o Select one among all possible actions for the current state
        o Using this possible action, consider to go to the next state
        o Get maximum Q value of this next state based on all possible actions
        o Compute
        o Set the next state as the current state

        End Do

    End For

"Perfection" means all decision paths are tried often enough that the equations converge. Usual method is to explore only greedy paths, gamma = 1. Several approaches:

1. use greedy policy at every state-decision
2. use random actions alternating with 1.
3. use random action small percentage of the time and greedy the rest
4. choose a "temperature" to decide the probabilities of possible actions, low temp means high chance of greedy (reliable) action, high temperature corresponds to greater entropy and more likelihood some unreliable (ungreedy) action may be tested. P = exp(-Q/T)

Applications:

Iterated Prisoner's Dilemma
Towers of Hanoi example (Teknomo's state diagram is missing a couple of states between 6 & 7 but is otherwise accurate).
Distributed version for packet routing (Littman and Boyan)

References:

Watkins, C. 1989. Learning from Delayed Rewards. Thesis, Cambridge University.

Bertsekas, D.P. & Tsitsiklis, J. Neuro-Dynamic Programming, 1996

MIT OpenCourseWare 2.997: Decision Making in Large Scale Systems taught by Daniela Pucci De Farias. (http://ocw.mit.edu/OcwWeb/Mechanical-Engineering/2-997Spring2004/LectureNotes/index.htm) (Lecture 7)

Teknomo, Kardi: Tutorial, Q-learning by example. (http://people.revoledu.com/kardi/tutorial/ReinforcementLearning/index.html)

Hamzah, Zarul: "Are we learning now?" Surprise 96 Journal, Imperial College London, 1996 (http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol2/zah/article2.html)

Littman and Boyan: A Distributed Reinforcement Learning Scheme for Network Routing (http://www.machinelearning.net/reinforcement-learning/)