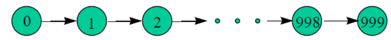
Temporal difference learning

- The number of training iterations necessary to sufficiently represent the optimal *Q-function* scales poorly with respect to the size of the time interval between states.
- The greater the number of actions per unit time, the greater the number of training iterations required to adequately represent the optimal Q-function.
- TD uses the concept of discounting the cumulative reinforcement versus the reinforcement from a single state transition.

Temporal difference learning

- Given the MDP with 1000 states.
- State 0 is initial state & state 999 is an absorbing state.
- Each state transition returns a cost (reinforcement) of 1, and the value for 999 is defined to be 1.



- Q-learning, value interaction can solve this problem.
- However TD(lambda) can solve it faster.
- TD(lambda) updates the value of the current state based on a weighted combination of the values of all future states versus only the value of the immediate successor state. Basic Bellman eqn:

$$V(\mathbf{x}_t \mathbf{w}_t) = r(\mathbf{x}_t) + \gamma V(\mathbf{x}_{t+1}, \mathbf{w}_t)$$

Temporal difference learning –

Update rule uses difference in utilities between successive states

Q learning: reduce discrepancy between successive Q estimates

One step time difference:

$$Q^{(1)}(s_t, a_t) \equiv r_t + \gamma \max_a \hat{Q}(s_{t+1}, a)$$

Why not two steps?

$$Q^{(2)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(s_{t+2}, a)$$

Or n?

$$Q^{(n)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \cdots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a \hat{Q}(s_{t+n}, a)$$

Blend all of these:

$$Q^{\lambda}(s_t, a_t) \equiv (1-\lambda) \left[Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) \right.$$

Temporal difference learning –

the reinforcement is the difference between the ideal prediction and the current prediction

$$Q^{\lambda}(s_t, a_t) \equiv (1-\lambda) \left[Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) \right.$$

Equivalent expression:

$$\begin{aligned} Q^{\lambda}(s_t, a_t) &= r_t + \gamma [\ (1-\lambda) \max_a \hat{Q}(s_t, a_t) \\ &\qquad + \lambda\ Q^{\lambda}(s_{t+1}, a_{t+1})] \end{aligned}$$

Ongoing research

- Replace Q approximation table with Neural Net, Bayes Net or other generalized classifier
- Handle case where state only partially observable
- Design optimal exploration strategies
- Extend to continuous action, state
- Learn and use delta function: *S x A->S*
- Relationship to dynamic programming
- Applications to dynamic markets (Stock, auctions, etc.)

Applications of Reinforcement Learning

- Intelligent Trading Agents for Massively Multi-player Game Economies, John Reeder, Gita Sukthankar, M. Georgiopoulos, G. Anagnostopoulos
- Learning to be a Bot: Reinforcement Learning in Shooter Games, Michelle McPartland, Marcus Gallagher
- Agent Learning using Action-Dependent Learning Rates in Computer Role-Playing Games, Maria Cutumisu, Duane Szafron, Michael Bowling, Richard S. Sutton
- Combining Model-Based Meta-Reasoning and Reinforcement Learning for Adapting Game-Playing Agents, Patrick Ulam, Joshua Jones, Ashok Goel

Demos

- **Demo - Control of an octopus arm using GPTD**
- http://videolectures.net/icml07_engel_demo/

- Reinforcement Learning Repository at UMass, Amherst
- http://www-all.cs.umass.edu/rlr/domains.html

- Robot arm
- http://iridia.ulb.ac.be/~fvandenb/qlearning/qlearning.html

- Cat-Mouse applet
- http://www.cse.unsw.edu.au/~cs9417ml/RL1/applet.html

RL Example – Tic Tac Toe

How might we construct a player that will find the imperfections in its opponent's play and learn to maximize its chances of winning?

X	O	O
O	X	X
		X

Tic Tac Toe – *Minimax*

- ***Minimax*** algorithm assumes a particular way of playing by the opponent – an infallible opponent.
- ***Minimax*** player would never reach a game state from which it could lose – even though this may be a valuable strategy.
- ***Minimax*** will not scale to complex environments with large state space.

X	O	O
O	X	X
		X

Tic Tac Toe – Dynamic Programming

- ***Dynamic programming***, can *compute* an optimal solution for any opponent, but requires as input a complete specification of that opponent – not realistic in many game playing settings.

Tic Tac Toe – Learning a Model

- Assume that information is not available ***a priori*** for this problem, as it is not for the vast majority of problems of practical interest.
- Such information can be estimated from experience playing many games against the opponent!
- ***Learn a model*** of the opponent's behavior up to some level of confidence, and then apply ***dynamic programming*** to compute an optimal solution given the approximate opponent model.