TD(0) Method for Policy Evaluation

(Sutton)

Initialize the value function V of policy π arbitrarily.

Repeat for each episode

- Initialize s (e.g. start state)
- **Repeat**
  - Choose $a \leftarrow \pi(s)$.
  - Do action a; Observe reward r and next state s'.
  - $V(s) \leftarrow V(s) + \alpha\,(r + \gamma V(s') - V(s))$.
  - Update $s \leftarrow s'$
- **Until s is terminal** or **MAX_STEPS**

Approximate VI by sampling and bootstrapping

The value function associated with a policy π is the expected discounted sum of rewards received following the policy.

$$V^\pi(s) = E_\pi\,(r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s)$$

Estimating the *expected* value of a random variable can be done by repeating the following *sampling* procedure:

- Carry out an action.
- Observe the actual next state and reward.
- Average over the observed values.

However, since the value $V^\pi(s_{t+1})$ is unknown, we use a *bootstrap* procedure of using the current estimate $V(s_{t+1})$ for the value.

TD(0) Method for Policy Evaluation

Initialize the value function V of policy π arbitrarily.

Repeat for each episode

- Initialize s (e.g. start state)
- **Repeat**
  - Choose $a \leftarrow \pi(s)$.
  - Do action a; Observe reward r and next state s'.
  - $V(s) \leftarrow V(s) + \alpha\,(r + \gamma V(s') - V(s))$.
  - Update $s \leftarrow s'$
- **Until s is terminal** or **MAX_STEPS**

Learning to Evaluate Policies vs. Learning Control

- TD methods can be used to learn the value function of a fixed policy π.
- Learning optimal control in unknown environments, however, requires learning the action value function $Q(x, a)$.
- Learning control requires addressing the exploration/exploitation tradeoff.
- At each step: the agent can choose the action for which $Q(s, a)$ is highest (exploitation) or it can choose a random action (exploration).
- Exploration strategies can be *directed* or *undirected*.

Exploration Strategies

- *Semi-uniform* or ϵ-greedy: Choose a random action with probability ϵ, otherwise choose the highest $Q(s,a)$ action.
- *Boltzmann exploration*: Choose the action a that maximizes the probability
$$\frac{e^{\frac{Q(s,a)}{\theta}}}{\sum_{a' \in A(s)} e^{\frac{Q(s,a')}{\theta}}}$$
- *Interval estimation*: Keep track of confidence intervals of the return resulting from choosing a particular state-action pair. Choose the action that has the highest upper bound.
- *Counter exploration:* Maintain a count of the number of steps each action was taken in every state. Choose the state-action pair that was performed least with some exploration probability.

SARSA: On Policy TD Control

- Initialize $Q(s,a)$ arbitrarily.
- Repeat (for each episode)
  - Initialize s
  - Choose a in s maximizing $Q(s,a)$ using ϵ-greedy exploration.
  - **Repeat** (for each episode step)
    - Take action a, observe reward r, new state s'.
    - Choose a' in s' maximizing $Q(s',a')$ using ϵ-greedy exploration.
    - $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma Q(s',a') - Q(s,a)\right)$
    - $s \leftarrow s'; \qquad a \leftarrow a'.$

    until s is terminal.

Q-learning: Off Policy TD Control

- Initialize $Q(s, a)$ arbitrarily.
- Repeat (for each episode)
  - Initialize s
  - **Repeat** (for each episode step)
    - Choose a in s maximizing $Q(s, a)$ using ϵ-greedy exploration (or any other method).
    - Take action a, observe reward r, new state s'.
    - $Q(s, a) \leftarrow Q(s, a) + \alpha\left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)$
    - $s \leftarrow s'$

    until s is terminal.

Convergence of Q-learning

- Q-learning learns the optimal action-value function, independent of the policy used to choose actions (can even be random).
- Q-learning converges to $Q^*(s, a)$ for any finite MDP, assuming
  - All actions are attempted in all states infinitely often
  - Learning rate α_n is decayed at each step n such that

$$\sum_{n=0}^{\infty} \alpha_n = \infty$$

$$\sum_{n=0}^{\infty} {\alpha_n}^2 < \infty$$

  - Action value function $Q(s, a)$ is stored as a table
- Convergence of SARSA is harder to prove (open question).

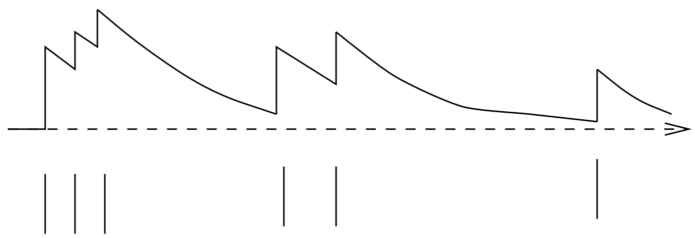
Problems with Discounting

- Causes an agent to sometimes prefer short-term mediocre reward over longer-term sustained reward.
- Arbitrary parameter that is not motivated by the problem.
- Most practical implementations of RL use discount factors very close to 1

Eligibility Traces

- Keep a (decaying) trace of the states most recently visited.
- Instead of modifying the value function just at the last state, modify over all states.
- The *eligibility* of a state is based on how recently the state was visited.
- Different trace update algorithms: replacing and accumulating.
- General form of TD(λ) and SARSA.

Accumulating Traces



Define $e_t(s)$ to be the eligibility of of state s at time t.

$$\begin{aligned} e_t(s) &= \gamma\lambda e_{t-1}(s) \ \text{ if } \ s \neq s_t \\ &= \gamma\lambda e_{t-1}(s) + 1 \ \text{ if } \ s = s_t \end{aligned}$$

Online Tabular TD(λ)

Initialize $V(s)$ arbitrarily and $e(s) = 0$ for all states s.
Repeat for each episode

- Initialize s (e.g. start state)
- **Repeat**
  - Choose $a \leftarrow \pi(s)$.
  - Do action a; Observe reward r and next state s'.
  - $\delta \leftarrow r + \gamma V(s') - V(s)$.
  - $e(s) \leftarrow e(s) + 1$
  - For all s:
    * $V(s) \leftarrow V(s) + \alpha\delta e(s)$
    * $e(s) \leftarrow \gamma\lambda e(s)$
  - $s \leftarrow s'$

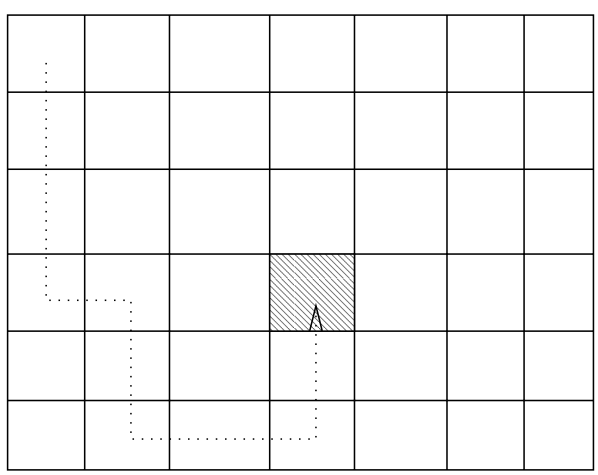
SARSA(λ)

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$ for all s, a.

Repeat for each episode

- Initialize s, a
- **Repeat**
  - Take action a, observe reward r and next state s'.
  - Choose action a' from s' that maximizes $Q(s', a')$
  - $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$.
  - $e(s,a) \leftarrow e(s,a) + 1$
  - For all s, a:
    - $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
    - $e(s,a) \leftarrow \gamma\lambda e(s,a)$
  - $s \leftarrow s'$, $a \leftarrow a'$

Grid World Example



TD(λ) Family of Learning Algorithms

Consider the parameterized update procedure (with parameter λ)

$$\Delta w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k$$

Note that when

- $\lambda = 1$: This results in pure supervised learning
- $\lambda = 0$: This results in one-step TD learning (Q-learning)
- General λ: Smooth interpolation between supervised and TD learning.

Scaling Reinforcement Learning

Issues:

- Large/continuous state spaces
- Impoverished feedback
- Transfer across tasks

Approaches:

- Function approximation
- Hierarchical methods and modularity
- Eligibility traces

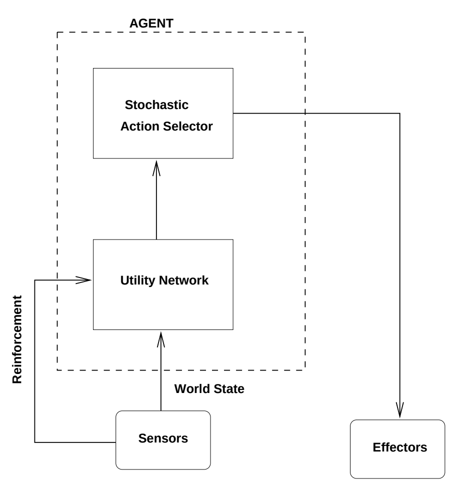
Function Approximation

Question: how to *compactly* approximate the value function over a large/infinite state space?

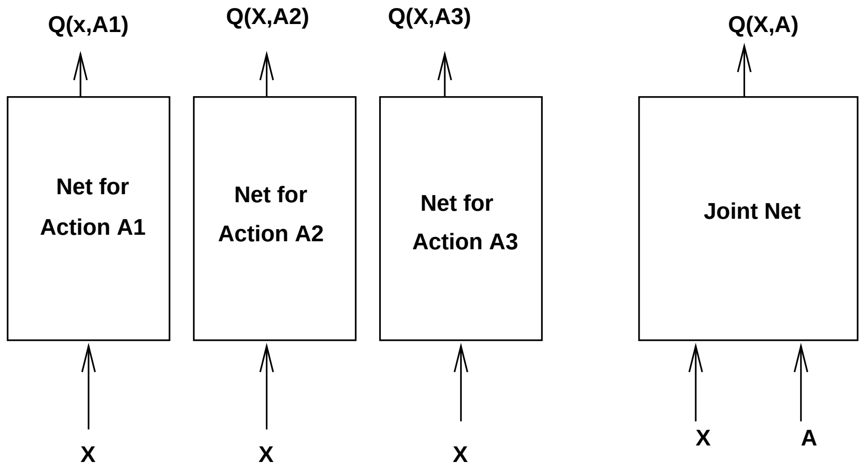
Some methods:

- Neural nets
- Clustering
- Decision trees
- Nearest-neighbor
- CMAC (sparse coarse coding)

A Model-free Framework



Some Neural Net Architectures



Q-learning with Neural Nets

1. Input $x \leftarrow$ current state; for each action i, compute $U_i \leftarrow Q(x, i)$ by forward prop.
2. Select $a \leftarrow \text{select}(U, T)$
3. Perform action a. New state $\leftarrow y$ and reinforcement $= r$.
4. TD error $u' \leftarrow r + \gamma * \max_{k \in A(y)} Q(y, k)$
5. Adjust neural net utility network by backpropagating ΔU through it where

$$\begin{aligned} \Delta U_i &= u' - U_i \ \text{ if } \ a = i \\ &= 0 \ \text{ otherwise} \end{aligned}$$

6. Go to 1

Successful Neural Net RL Systems

- Robotics (Lin '93, Rummery '96)
- Elevator control (Crites & Barto, '95)
- Backgammon (Tesauro '94)

Note: in each of these systems, additional scaling tricks were employed to build a succesful system

Pros and Cons of Neural Nets in RL

- Can deal with high-dimensional inputs
- Robust to sensor noise
- Convergence is slow
- Batch training is impossible
- Can fail to approximate value function