Learning Agents

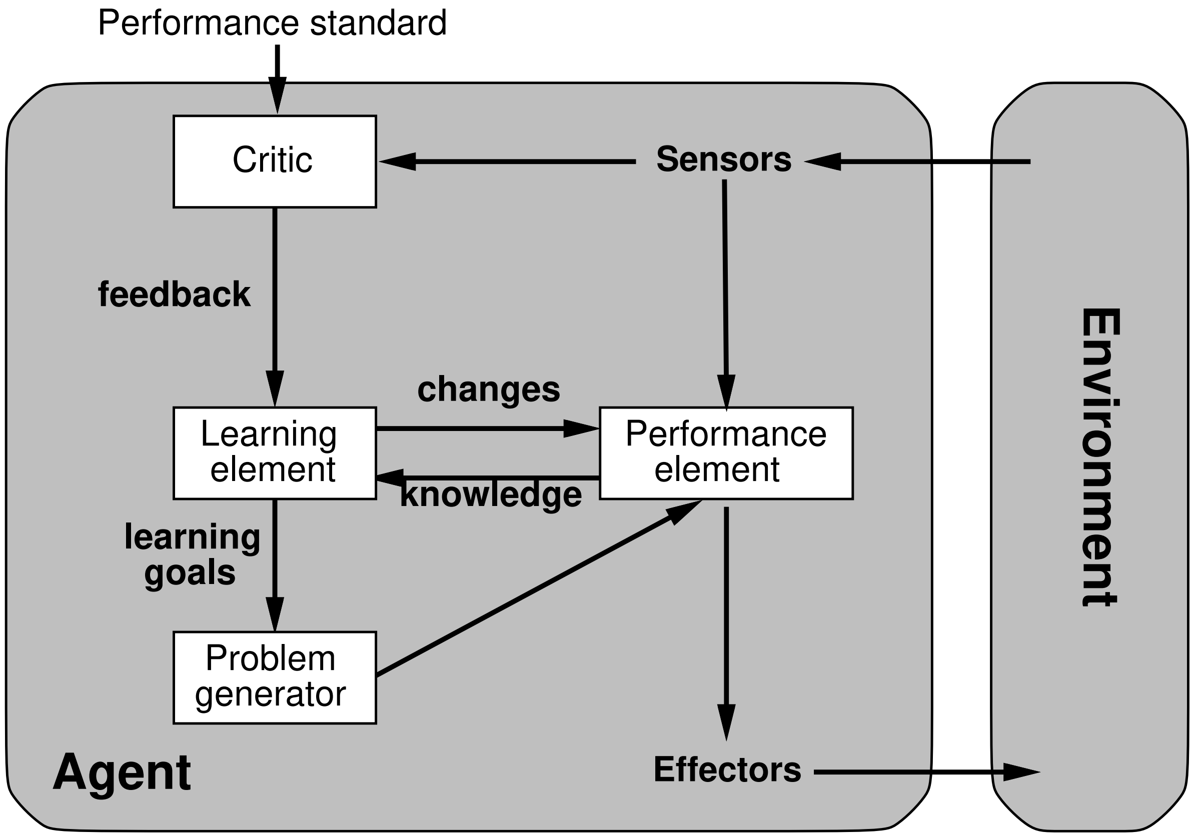
Any component of an agent can be learned

- **Policy:** $\pi(S)$
- **Sensor Model:** $P(O|S)$
- **Action Model:** $P(S'|S, A)$
- **Rewards or utilities:** $R(S'|S, A)$, $U(S)$
- **Action-Value information:** $Q(S, A)$ (explained later)

Each component can be represented in many different ways

- **Table**
- **Rules** (propositional logic or first-order logic)
- **Decision trees**
- **Neural networks**

Structure of a Learning Agent



Learning in Markov Decision Problems

Because the problem is Markov, we don't need to learn $P(O|S)$.

Two Major Approaches

- **Model-Based Methods**

  Experiment in the world to learn $R(S'|S, A)$ and $P(S'|S, A)$.

  Apply DP to compute optimal policy.

- **Model-Free Methods**

  Experiment in the world and directly learn an action-value function.

Q notation

Define

$$Q(s,a) = \sum_{s'} P(s'|s,a) \cdot [R(s'|s,a) + U(s')]$$

where

$$U(s) = \max_a \; Q(s,a)$$

This notation is useful for describing the methods that follow. It is also the heart of Q learning, which is the main model-free method.

Model-Based Methods

Learn a simple model by counting state transitions using the following arrays:

$r[s', s, a]$ = total reward experienced when action a was performed in state s and resulted in state s'.

$p[s', s, a]$ = number of state transitions observed from s to s' after performing a.

$$\hat{P}(s'|s,a) = \frac{p[s', s, a]}{\Sigma_{s'} \, p[s', s, a]}$$

$$\hat{R}(s'|s,a) = \frac{r[s', s, a]}{p[s', s, a]}$$

After each action, compute $\hat{P}$ and $\hat{R}$, compute π^t via dynamic programming.
(Or after every N actions...)

Exploration versus Exploitation Tradeoff

An action has two potential benefits:

- **Immediate or long-term payoff** (exploitation).

  We could perform the action in order to obtain an immediate reward or to put us into a state from which a reward can be more easily obtained.

- **Gathering information** (exploration).

  We could perform the action to learn more about the environment.

Multiple Slot Machines (2-armed bandit)

- Slot machine A pays a reward of \$100 with probability p_A.
- Slot machine B pays a reward of \$100 with probability p_B.

p_A and p_B are unknown. What strategy will maximize our total winnings?

- **Exploration:** play each machine several times, figure out which one has higher probability.
- **Exploitation:** play the machine with the higher probability.

Optimal "allocation of trials" has been studied. Answer: allocate trials to machine A according to $P(p_A > p_B)$.

In general, optimal solution of this tradeoff is intractable.

Heuristic Strategies for Exploration/Exploitation

- **Mix random policy with greedy policy**. Let $\pi^t(S)$ be the policy computed by DP after t interactions with the environment. Choose action a according to

$$a = \begin{cases} \text{random action} & \text{with probability } \beta \\ \pi^t(s) & \text{with probability } 1 - \beta \end{cases}$$

  The exploration probability β is gradually decreased to zero.

- **Boltzmann exploration**. Choose action randomly a according to

$$P(a) = \frac{e^{\hat{Q}(S,a)/T}}{\sum_{a'} e^{\hat{Q}(S,a')/T}}$$

  where $\hat{Q}(S, a) = \Sigma_{S'} \hat{P}(S'|S, a)[\hat{R}(S'|S, a) + \hat{U}(S')]$ The temperature T is gradually decreased to zero.

- **Optimism under uncertainty**.

Optimism Under Uncertainty

Define the OUU utility U^+ as follows:

$$U^+(s) = \max_a Q^+(s, a)$$

where

$$Q^+(s, a) = \begin{cases} \Sigma_{s'} \hat{P}(s'|s, a)[\hat{R}(s'|s, a) + U^+(s')] & \text{if } p[s, a] \geq N_e \\ R^+ & \text{if } p[s, a] < N_e \end{cases}$$

where

$$R^+ > U^*(s) \text{ for all } s.$$

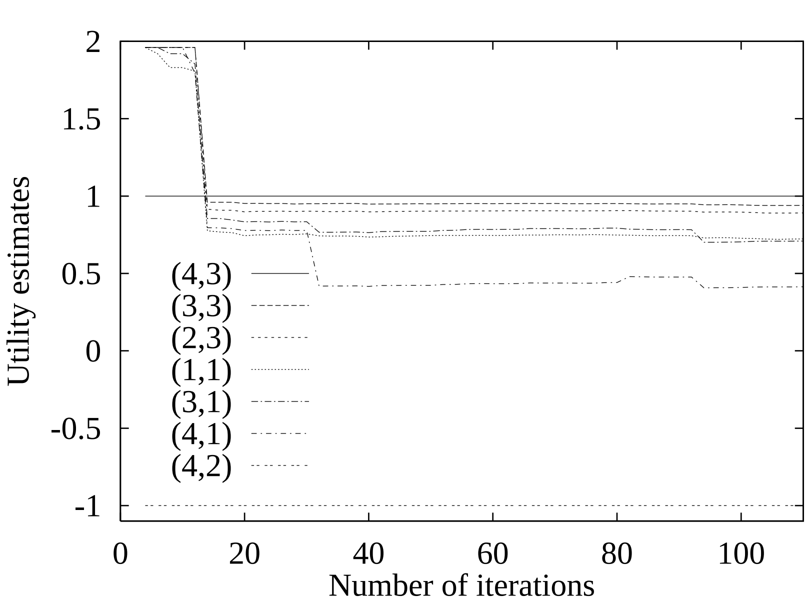
Choose the action a that maximizes $Q^+(s, a)$.

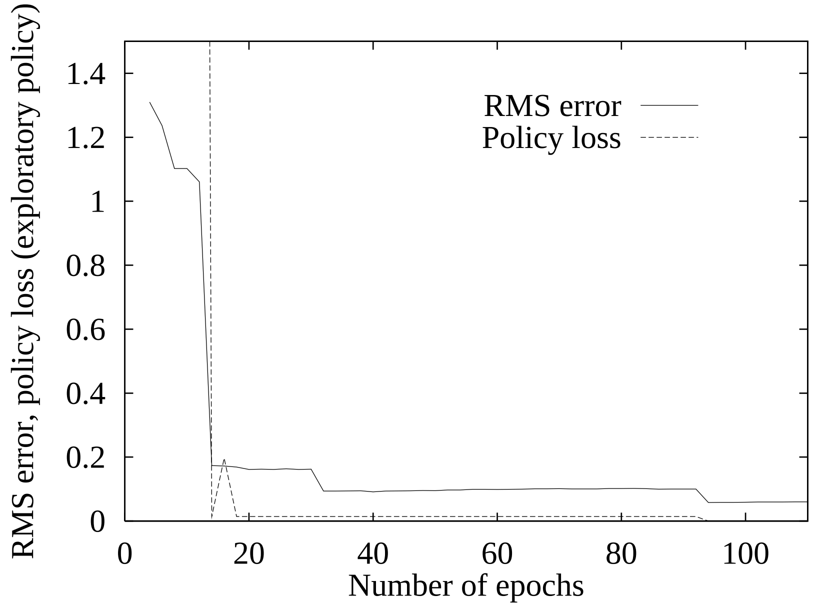
Every action will be tried in every state at least N_e times and use this to estimate P and R.

- **Plans to gain information**. The resulting policy seeks new experiences.
- **Ignores hopeless states**. However, it ignores states where even a reward of R^+ will not improve $U(s_0)$—the utility of the start state.

R^+ plays a role analogous to $h(n)$ in A^* search.

Example: 4x3 world





Model-Free Learning Agents

It is possible to learn an optimal policy and value function *without* learning P and R by combining the DP process with learning. To do this, we need two key ideas:

- **Temporal averaging**.
- **Using the environment as its own model**.

Temporal Averaging

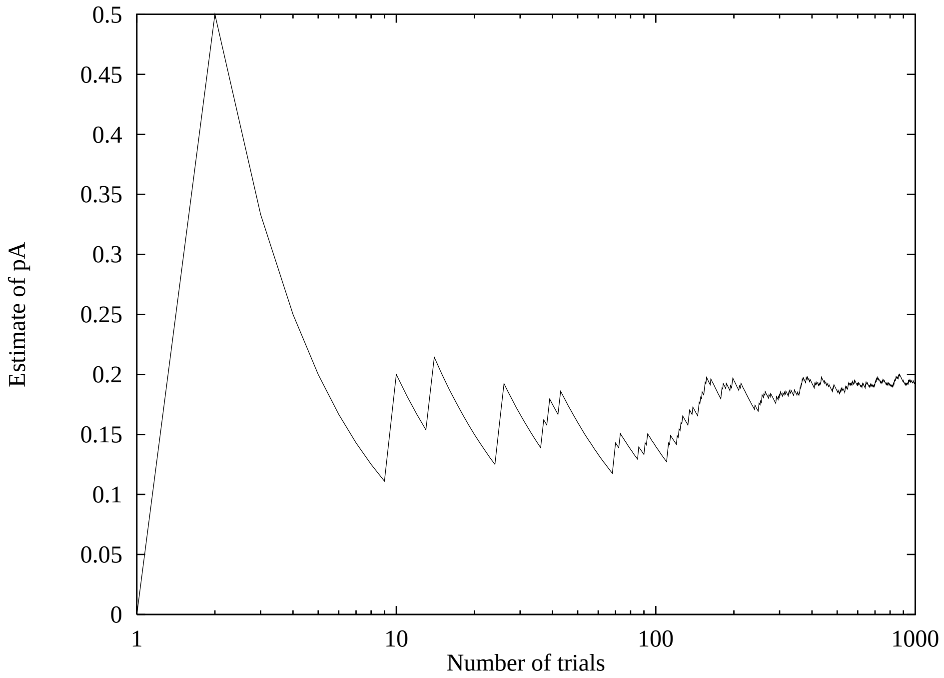
Suppose we want to compute p_A for slot machine A. One approach is to compute

$$\frac{\text{number of payoffs}}{\text{number of plays}}$$

An alternative way of computing the same thing is temporal averaging. Start with an initial estimate $p_A = 0.5$. In each trial, update p_A according to

$$p_A^t := (1 - \alpha^t)p_A^{t-1} + \alpha^t A^t$$

where $A^t = 1$ if the machine paid off at trial t and zero otherwise. Let $\alpha^t = 1/t$.



Using the Environment as Its Own Model: Q Learning

$\hat{Q}(s, a) := 0$ for all s, a

$s :=$ start state

do forever

  choose a according to exploration policy

  perform a

  observe s' and $R(s'|s, a)$

  $\hat{Q}(s, a) := (1 - \alpha)\hat{Q}(s, a) + \alpha[R(s'|s, a) + \max_{a'} \hat{Q}(s', a')]$

end

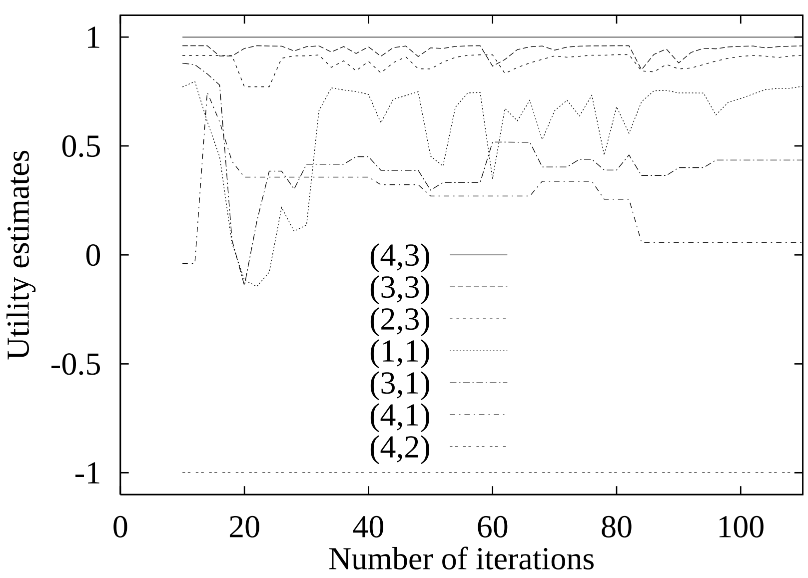
If exploration policy tries every action in every state infinitely often (as $t \to \infty$) and if α decreases as required, then $\hat{Q}(s, a) \to Q^*(s, a)$.

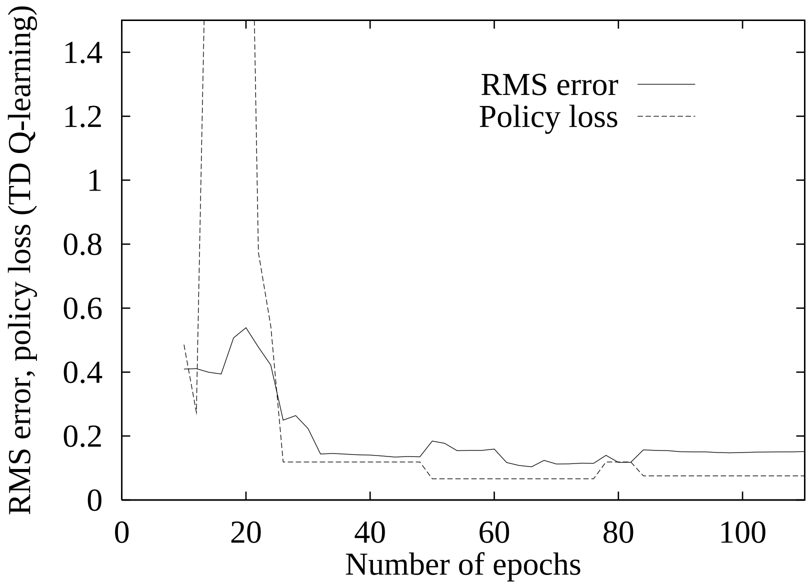
$$\pi(s) = \operatorname*{argmax}_a Q(s, a)$$

Transitions and rewards are observed from the environment. The expected backed-up value is computed by temporal averaging.

Example: Q learning with OUU

This requires keeping a count of the number of times each (s, a) pair has been tried.





Generalizing from Experience

The methods we have discussed so far assume that each action in each state could do something completely different. We'd like to learn something like

$$P(S' = [x, y+1] \mid S = [x, y], North) = 0.8$$

This kind of learning will be the final subject in the course.

Learning Agents for Markov Decision Problems
Summary

- **Model-Based Methods**

  Experiment in the world to learn $R(S'|S, A)$ and $P(S'|S, A)$.

  Apply DP to compute optimal policy.

- **Model-Free Methods**

  Experiment in the world and directly learn an action-value function. Q learning.

Without generalization, these methods do not scale to very large state spaces.