# Reinforcement Learning and Policy Reuse

**Manuela M. Veloso**

*PEL – Fall 2016*

Readings:

- Reinforcement Learning: An Introduction, R. Sutton and A. Barto
- Probabilistic policy reuse in a reinforcement learning agent, Fernando Fernandez and Manuela Veloso. In *Proceedings of AAMAS'06.* (Thanks to Fernando Fernandez)

## Learning

- Learning from experience
- Supervised learning
  – Labeled examples
- Reward/reinforcement
  – Something good/bad (positive/negative reward) happens
  – An agent gets reward as part of the "input" percept, but it is "programmed" to understand it as reward.
  – Reinforcement extensively studied by animal psychologists.

## Reinforcement Learning

- The problem of getting an agent to act in the world so as to maximize its rewards.
- Teaching a dog a new trick:
  - you cannot tell it what to do,
  - but you can reward/punish it if it does the right/wrong thing.
  - Learning: to figure out what it did that made it get the reward/ punishment: the **credit assignment** problem.
- RL: a similar method to train computers to do many tasks.

## Reinforcement Learning Task

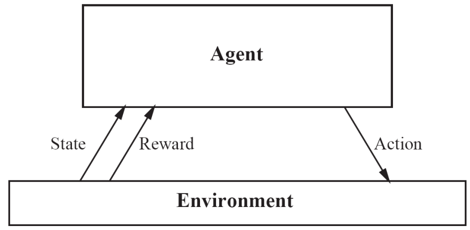
- Assume the world is a Markov Decision Process
  - States and actions known
  - Transitions and rewards unknown
  - Full observability
- Objective
  - **Learn action policy** $\pi : S \rightarrow A$
  - **Maximize expected reward**

$$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots]$$

*from any starting state in S.*

- $0 \leq \gamma < 1$, discount factor for future rewards

# Reinforcement Learning Problem




$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \ldots$$

Agent sees the state, selects and action, and gets reward

Goal: Learn to choose actions that maximize

$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$ , where $0 \leq \gamma < 1$

# Online Learning Approaches

- Capabilities
  - Execute actions in world
  - Observe state of world
- Two Learning Approaches
  - Model-based
  - Model-free

## Model-Based Reinforcement Learning

- Approach
  - Learn the MDP
  - Solve the MDP to determine optimal policy
- Appropriate when model is unknown, but small enough to solve feasibly

## Learning the MDP

- Estimate the rewards and transition distributions
  - Try every action some number of times
  - Keep counts (frequentist approach)
    - $R(s,a) = R^a_s / N^a_s$
    - $T(s',a,s) = N^a_{s,s'} / N^a_s$
  - Solve using value or policy iteration
- Iterative Learning and Action
  - Maintain statistics incrementally
  - Solve the model periodically

## Model-Free Reinforcement Learning

- Learn policy mapping ***directly***
- Appropriate when model is too large to store, solve, or learn
  - Do not need to try every state/action in order to get good policy
  - Converges to optimal policy

## Value Function

- For each possible policy π, define an *evaluation function over states*

$$V^{\pi}(s) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \ldots$$
$$\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where $r_t$, $r_{t+1}$,... are generated by following policy π starting at state $s$ $\boxed{\pi^* \equiv \mathrm{argmax}_{\pi} V^{\pi}(s), (\forall s)}$

- Learning task: Learn OPTIMAL policy

## Learn Value Function

- Learn the evaluation function $V^{\pi*}$ (i.e. $V^*$)
- Select the optimal action from any state $s$, i.e., have an optimal policy, by using $V^*$ with one step lookahead:

$$\pi^*(s) = \arg\max_a \left[ r(s,a) + \gamma V^*(\delta(s,a)) \right]$$

- But reward and transition functions are unknown

## Q Function

- Define new function very similar to $V^*$

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$$

**Learn $Q$ function – $Q$-learning**

- If agent learns $Q$, it can choose optimal action even without knowing $\delta$ or $r$

$$\pi^*(s) = \arg\max_a \left[ r(s,a) + \gamma V^*(\delta(s,a)) \right]$$

$$\pi^*(s) = \arg\max_a Q(s,a)$$

# Q-Learning

$Q$ and $V^*$:

$$V^*(s) = \max_{a'} Q(s, a')$$

We can write $Q$ recursively:

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) \\ &= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

$Q$-learning actively generates examples.
It "processes" examples by updating its $Q$ values.
*While* learning, $Q$ values are approximations.

# Training Rule to Learn Q (Deterministic Example)

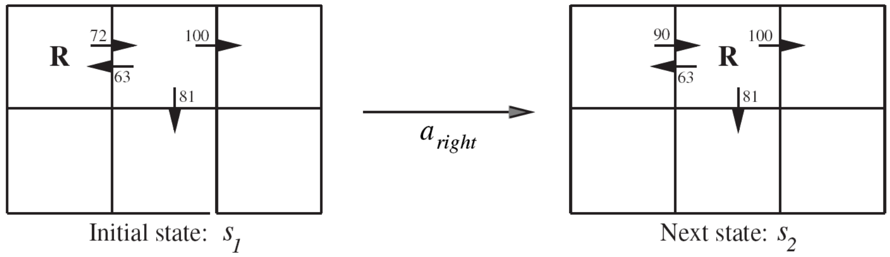
Let $\hat{Q}$ denote current approximation to $Q$.

Then Q-learning uses the following **training rule**:

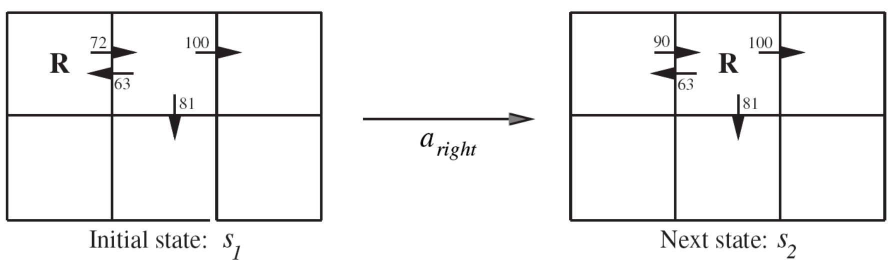
$$\boxed{\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')}$$

where $s'$ is the state resulting from applying action $a$ in state $s$, and $r$ is the reward that is returned.

## Deterministic Case – Example




## Deterministic Case – Example

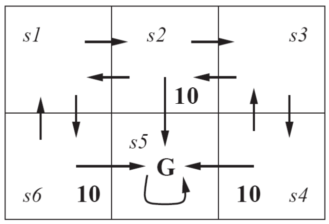



$$
\begin{aligned}
\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\
&\leftarrow 0 + 0.9 \max\{63, 81, 100\} \\
&\leftarrow 90
\end{aligned}
$$

# Q Learning Iterations

Start at top left corner with fixed policy – clockwise

Initially $Q(s,a) = 0$; $\gamma = 0.8$



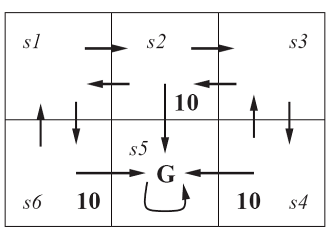

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$

Q (s1, E)  Q (s2, E)  Q (s3, S)  Q(s4, W)

# Q Learning Iterations

Starts at top left corner with fixed policy – clockwise

Initially $Q(s,a) = 0$; $\gamma = 0.8$




$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$

| Q(s1,E) | Q(s2,E) | Q(s3,S) | Q(s4,W) |
|---|---|---|---|
| 0 | 0 | 0 | r + $\gamma$ max{Q(s5,loop)}= 10 + 0.8 . 0 = 10 |
| 0 | 0 | r + $\gamma$ max{Q(s4,W),Q(s4,N)}= 0 + 0.8 max{10,0}= 8 | 10 |
| 0 | r + $\gamma$ max{Q(s3,W),Q(s3,S)}= 0 + 0.8 max{0,8}= 6.4 | 8 | 10 |

## Nondeterministic Case

- $Q$ learning in nondeterministic worlds
  - Redefine $V$, $Q$ by taking expected values:

$$V^{\pi}(s) \equiv E\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right]$$
$$\equiv E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]$$

$$Q(s,a) \equiv E\left[r(s,a) + \gamma V^*(\delta(s,a))\right]$$

## Nondeterministic Case

- $Q$ learning training rule:

$$\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_\mathrm{n}\left[r + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')\right],$$

where $\alpha_n = \frac{1}{1+visits_n(s,a)}$, and $s' = \delta(s,a)$.

$\hat{Q}$ still converges to $Q^*$ (Watkins and Dayan, 1992)

## Exploration vs Exploitation

- Tension between learning optimal strategy and using what you know, so far, to maximize expected reward
  - Convergence theorem depends on visiting each state sufficient number of times
  - Typically use reinforcement learning while performing tasks

## Exploration policy

- Wacky approach: act randomly in hopes of eventually exploring entire environment
- Greedy approach: act to maximize utility using current estimate
- Balanced approach: act “more” wacky when agent has not much knowledge of environment and “more” greedy when the agent has acted in the environment longer
- One-armed bandit problems

## Exploration Strategies

- ε-greedy
  - Exploit with probability 1-ε
  - Choose remaining actions uniformly
  - Adjust ε as learning continues
- Boltzman
  - Choose action with probability $p = \frac{e^{Q(s,a)/t}}{\sum_{a'} e^{Q(s,a')/t}}$
  - where $t$ "cools" over time (simulated annealing)

*All methods sensitive to parameter choices and changes*

## Policy Reuse

- Impact of change of reward function
  - Does not want to learn from scratch
- Transfer learning
  - Learn macros of the MPD – options
  - Value function transfer
  - Exploration bias
- Reuse complete policies

## Episodes

- MDP with absorbing goal states
  - Transition probability from a goal state to the same goal state is 1 (therefore to any other state is 0)
- Episode:
  - Start in random state, end in absorbing state
- Reward per episode (K episodes, H steps each):

$$W = \frac{1}{K}\sum_{k=0}^{K}\sum_{h=0}^{H}\gamma^{h} r_{k,h} \qquad (1)$$

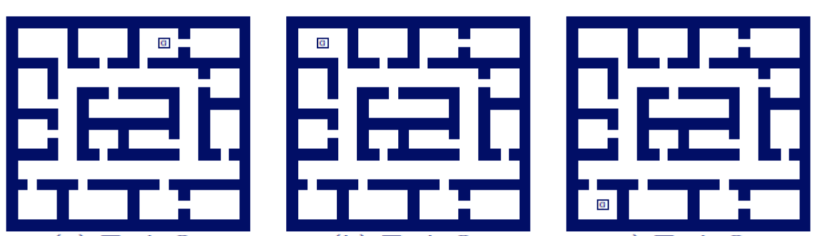
where $\gamma$ $(0 \leq \gamma \leq 1)$ reduces the importance of future rewards, and $r_{k,h}$ defines the immediate reward obtained in the step $h$ of the episode $k$, in a total of $K$ episodes.

## Domains and Tasks

A **domain** $\mathcal{D}$ is defined as a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T} >$, where $\mathcal{S}$ is the set o all possible states; $\mathcal{A}$ is the set of all possible actions; and $\mathcal{T}$ is a state transition function, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \Re$

A **task** $\Omega$ is defined as a tuple $< \mathcal{D}, \mathcal{R}_\Omega >$, where $\mathcal{D}$ is a domain; anc $\mathcal{R}_\Omega$ is the reward function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$

An **action policy** $\Pi_\Omega$ to solve a task $\Omega$ is a function $\Pi_\Omega : \mathcal{S} \rightarrow \mathcal{A}$.



(a) Task $\Omega_1$ (b) Task $\Omega_2$ c) Task $\Omega_3$

## Policy Library and Reuse

- **Policy Reuse**:
  - ⋆ We need to solve the task $\Omega$, i.e. learn $\Pi_\Omega$
  - ⋆ We have previously solved the set of tasks $\{\Omega_1, \ldots, \Omega_n\}$ so we have a Policy Library composed of the $n$ policies that solve them respectively, say $L = \{\Pi_1, \ldots, \Pi_n\}$
  - ⋆ How can we use the policy library, $L$, to learn the new policy, $\Pi_\Omega$?

## π-Reuse Exploration

Need to solve a task $\Omega$, i.e. learn $\Pi_{new}$.

Have a Policy Library, say $L = \{\Pi_1, \ldots, \Pi_n\}$

Let's assume that there is a supervisor who, given $\Omega$, tells us which is the most similar policy, say $\Pi_{past}$, to $\Pi_{new}$. Thus, we know that the policy to reuse is $\Pi_{past}$.

Integrate the past policy as a probabilistic bias in the exploration strategy of the new learning process

Define probabilities for exploiting the past policy, perform random exploration, or exploit the ongoing policy

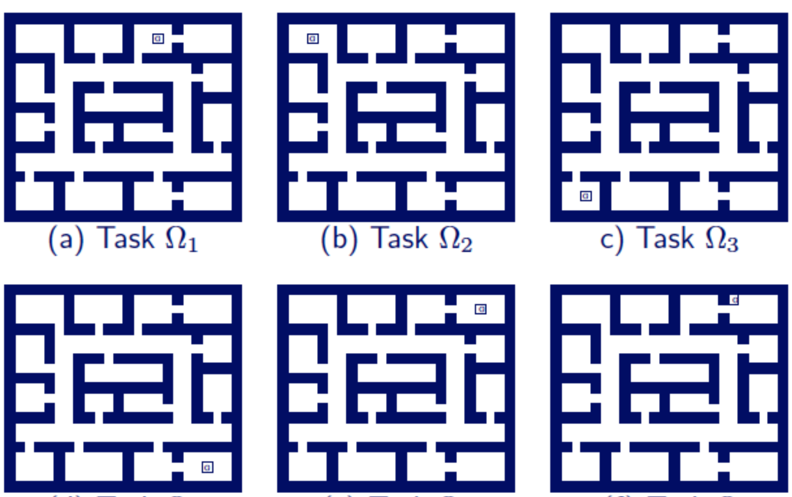
⋆ Select $a = \begin{cases} \Pi_{past}(s) & \text{w/prob. } \psi \\ \Pi_{new}(s)) & \text{w/prob. } (1-\psi)\epsilon \\ Random & \text{w/prob. } (1-\psi)(1-\epsilon) \end{cases}$

# π-Reuse Policy Learning

$\pi$-reuse $(\Pi_{past}, K, H, \psi, \upsilon, \gamma, \alpha)$.

Initialize $Q^{\Pi_{new}}(s, a) = 0$, $\forall s \in \mathcal{S}, a \in \mathcal{A}$
For $k = 1$ to $K$
 Set the initial state, $s$, randomly.
 Set $\psi_1 \leftarrow \psi$
 for $h = 1$ to $H$
  With a probability of $\psi_h$, $a = \Pi_{past}(s)$
  With a probability of $1 - \psi_h$, $a = \epsilon\text{-greedy}(\Pi_{new}(s))$
  Receive current state $s'$, and reward, $r_{k,h}$
  Update $Q^{\Pi_{new}}(s, a)$, and therefore, $\Pi_{new}$, using the Q-Learning update function:
   $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \,\text{máx}_{a'} Q(s', a')]$
  Set $\psi_{h+1} \leftarrow \psi_h \upsilon$
  Set $s \leftarrow s'$
$W = \frac{1}{K} \sum_{k=0}^{K} \sum_{h=0}^{H} \gamma^h r_{k,h}$
Return $W$, $Q^{\Pi_{new}}(s, a)$ and $\Pi_{new}$

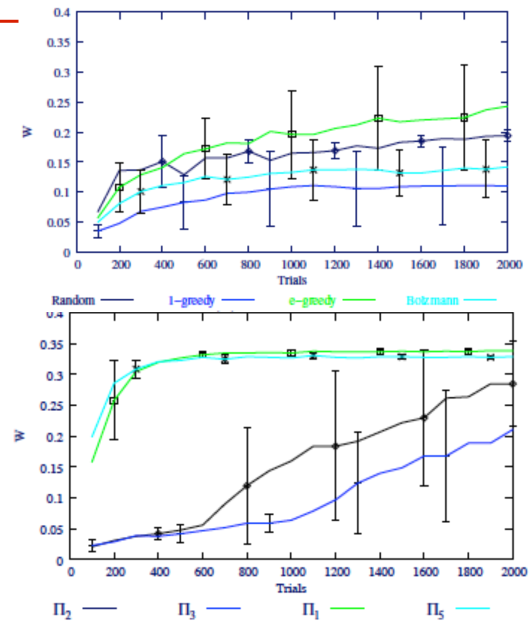
# Experimental Results



(a) Task $\Omega_1$ (b) Task $\Omega_2$ c) Task $\Omega_3$

(d) Task $\Omega_4$ (e) Task $\Omega_5$ (f) Task $\Omega$

# Results




# Policy Reuse in Q-Learning

- Interestingly, the pi-reuse strategy also contributes a *similarity metric* between policies
  - The gain Wi obtained while executing the pi-reuse exploration strategy, reusing the past policy i.
- Wi is an estimation of how similar the policy i is to the new one!
- The set of Wi values for each of the policies in the library is unknown a priori, but it can be estimated on-line while the new policy is computed in the different episodes.

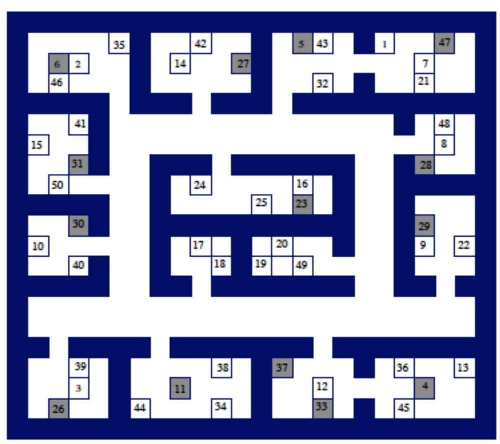
PRQ-Learning.

**PRQ-Learning ($\Omega, L, K, H$)**

- Given:
(1) A new task $\Omega$ we want to solve.
(2) A Policy Library $L = \{\Pi_1, \ldots, \Pi_n\}$.
(3) A maximum number of episodes to execute, $K$.
(4) A maximum number of steps per episode, $H$.
- Initialize:
(1) $Q_\Omega(s, a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$.
(2) $W_\Omega = W_i = 0$, for $i = 1, \ldots, n$.
- For $k = 1$ to $K$ do
  – Choose an action policy, $\Pi_k$, assigning to each policy the probability of being selected computed by the following equation:
  $$P(\Pi_j) = \frac{e^{\tau W_j}}{\sum_{p=0}^{n} e^{\tau W_p}}$$
  where $W_0$ is set to $W_\Omega$.
  – Execute the learning episode $k$.
    If $\Pi_k = \Pi_\Omega$, execute a $Q$-Learning episode following a fully greedy strategy.
    Otherwise, call $\pi$-reuse ($\Pi_k, 1, H, \psi, \upsilon$).
    In any case, receive the reward obtained in that episode, say $R$, and the updated $Q$ function, $Q_\Omega(s, a)$.
  – Recompute $W_k$ using $R$.
- Return the policy derived from $Q_\Omega(s, a)$.

# Learning to Use a Policy Library

- Similarity between policies can be learned
- Gain of using each policy
- Explore different policies
- Learn domain structure: “eigen” policies



## Summary

- Reinforcement learning
  - Q-learning
- Policy Reuse

- Next class:
  - Other reinforcement learning algorithms
  - (There are many…)