BLG521E
Artificial Intelligence

Lecture 14: Reinforcement Learning

Reinforcement Learning

- No supervision

- Use observed rewards to learn an optimal policy

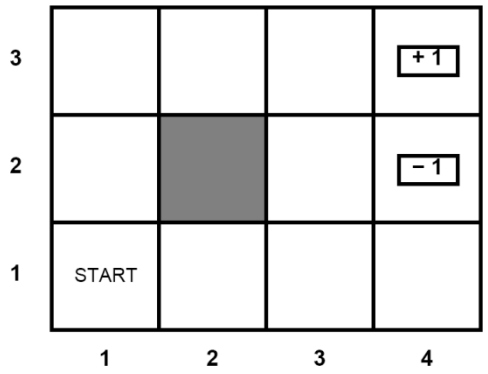
Passive vs. Active Learning

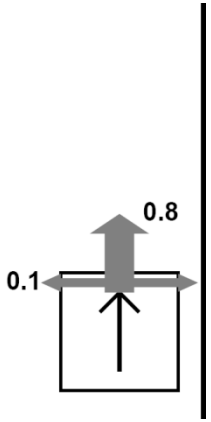
- Passive learning
  - Agent's policy is fixed
  - The task is learning the utilities of states
  - Learning a model of the world

- Active learning
  - The agent also must learn what to do

Passive learning agent

- Agent's policy is fixed
- The task is learning the utilities of states: $U^{\pi}(s)$
- States $s \in S$, *actions* $a \in A$, S_0
- Model *T(s, a, s') is not known*
- Reward function *R(s) is not known*





Passive learning agent

- Epoch = training sequences:

(1,1)→(1,2)→(1,3)→(1,2)→(1,3)→(1,2)→(1,1)→(1,2)→(2,2)→(3,2) –1
(1,1)→(1,2)→(1,3)→(2,3)→(2,2)→(2,3)→(3,3) +1
(1,1)→(1,2)→(1,1)→(1,2)→(1,1)→(2,1)→(2,2)→(2,3)→(3,3) +1
(1,1)→(1,2)→(2,2)→(1,2)→(1,3)→(2,3)→(1,3)→(2,3)→(3,3) +1
(1,1)→(2,1)→(2,2)→(2,1)→(1,1)→(1,2)→(1,3)→(2,3)→(2,2)→(3,2) -1
(1,1)→(2,1)→(1,1)→(1,2)→(2,2)→(3,2) -1

- The utilities are to be learned: the expected sum of rewards using policy π

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^{t} R(s_t) \mid \pi, s_0 = s\right]$$

Adaptive Dynamic Programming

- Widrow & Hoff (1960): Direct Utility Estimation
  - Utility is the expected total reward onward
  - Each trial provides a sample of this value
  - Directly calculated as supervised learning

- Bellman Equation

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U^{\pi}(s')$$

Adaptive Dynamic Programming

function PASSIVE-ADP-AGENT($percept$) **returns** an action
  inputs: $percept$, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
    mdp, an MDP with model P, rewards R, discount γ
    U, a table of utilities, initially empty
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    $N_{s'|sa}$, a table of outcome frequencies given state–action pairs, initially zero
    s, a, the previous state and action, initially null

  if s' is new **then** $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$ and $N_{s'|sa}[s', s, a]$
    for each t such that $N_{s'|sa}[t, s, a]$ is nonzero **do**
      $P(t \mid s, a) \leftarrow N_{s'|sa}[t, s, a] \;/\; N_{sa}[s, a]$
  $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
  if s'.TERMINAL? **then** $s, a \leftarrow$ null **else** $s, a \leftarrow s', \pi[s']$
  return a

- Intractable for large state spaces

Temporal Difference (TD) Learning

- Do ADP backups on a per move basis, not for the whole state space

$$U^{\pi}(s) = U^{\pi}(s) + \alpha\left[R(s) + \gamma U^{\pi}(s') - U^{\pi}(s)\right]$$

- Average value of $U^{\pi}(s)$ converges to the correct value

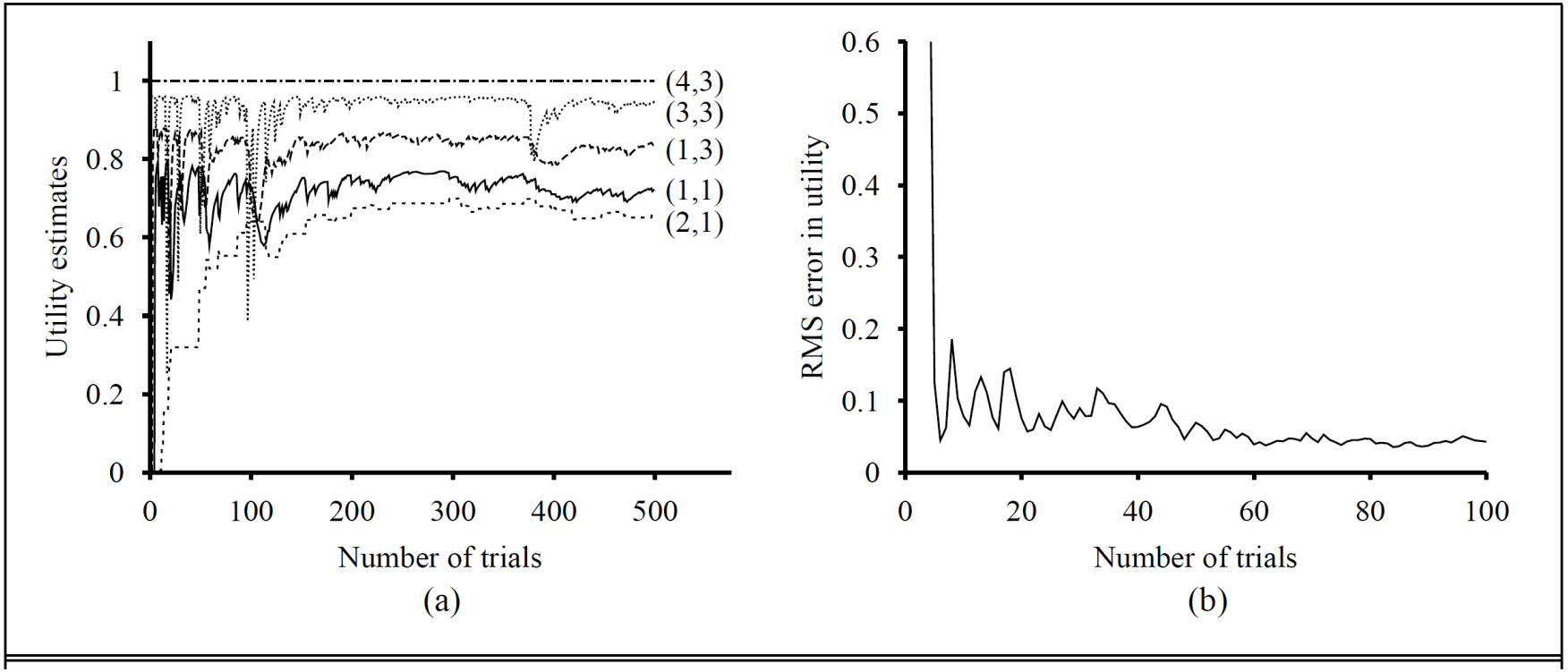
Temporal Difference (TD) Learning

function PASSIVE-TD-AGENT($percept$) **returns** an action
  inputs: $percept$, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
    U, a table of utilities, initially empty
    N_s, a table of frequencies for states, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if s' is new **then** $U[s'] \leftarrow r'$
  if s is not null **then**
    increment $N_s[s]$
    $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma\, U[s'] - U[s])$
  if s'.TERMINAL? **then** $s, a, r \leftarrow$ null **else** $s, a, r \leftarrow s', \pi[s'], r'$
  return a

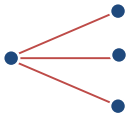
- Does not need a model!
- If α is appropriately decreased as a function of times a state is visited, then $U^{\pi}(s)$ converges to the correct value

Temporal Difference (TD) Learning Results



ADP vs. TD Learning

- ADP: full backup



- TD: one experience back up per observed transitions

Active learning in an unknown environment

- Agent considers what actions to take

$$U(s) = R(s) + \gamma \max \sum_{s'} T(s, a, s') U(s')$$

- Model-based (learn *T*)
- Model-free (Q-learning)

Exploitation vs. Exploration

- Exploitation: maximize reward
- Exploration: maximize the long-term well being

- A greedy agent favors exploitation
  - Exploration component can also be added
    - Choose a random action a fraction 1/t of the time

- Assign weights to actions
  - Tend to avoid actions believed to be of low utility
  - Assign a higher utility estimate to unexplored state-actions

Q-learning

- Learning action-value representation

$$Q(a,s)$$

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

$$U(s) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(a',s')$$

- Update after each state transition:

$$Q(a,s) \leftarrow Q(a,s) + \alpha[R(s) + \gamma \max_{a'} Q(a',s') - Q(a,s)]$$

Q-learning success

Q-learning converges to optimal Q-values if

- Every state is visited infinitely often (due to exploration)
- The action selection becomes greedy as time approaches infinity, and
- The learning rate α is decreased fast enough but not too fast

Algorithm

- Observe the current state
- Do forever
  - Select an action *a* and execute it.
  - Receive immediate reward *r.*
  - Observe the new state *s'*
  - Update the table entry for $\hat{Q}(s,a)$

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

  - $s \leftarrow s'$

What makes RL special?

- Delayed reward
  - Reward can be delayed. e.g. only when reaching the goal state, the agent is given a reward.

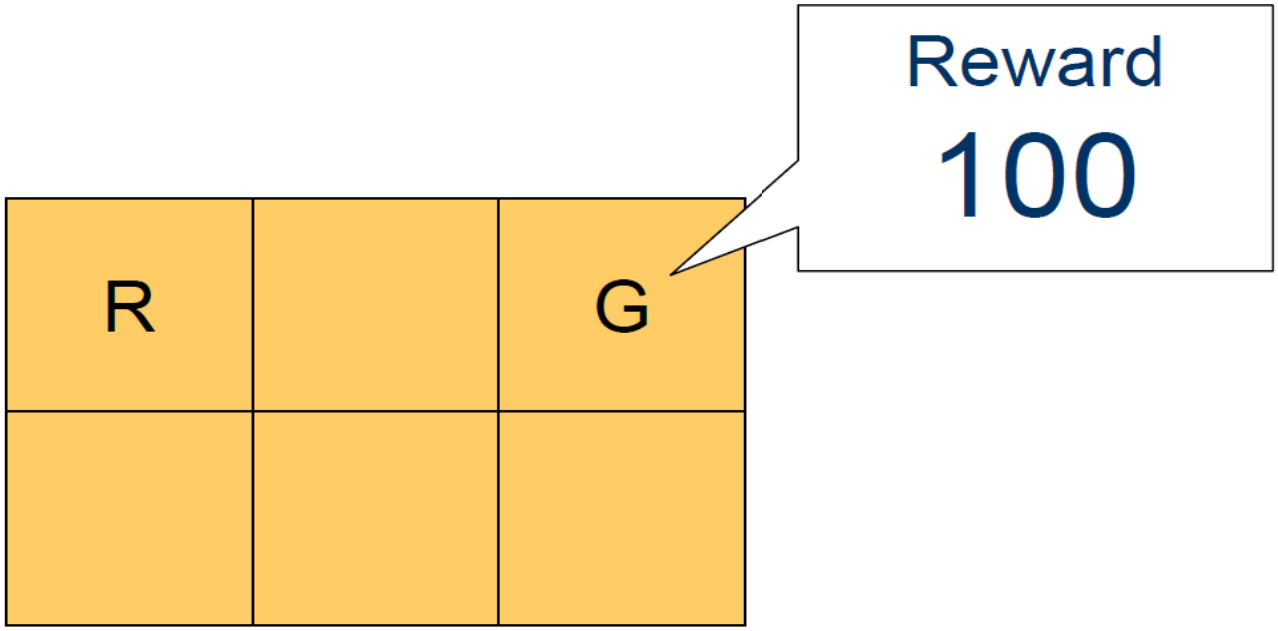
- Exploration/exploitation
  - The agent can explore its environment but can also decide to exploit what is has learned. This can happen at the same time. This is different from learning methods which need a training phase

What makes RL special?, con't

- Partially observable states
  - Sensors provide only partial information about the environmental state, policy can bring the robot to states that provide extra information.

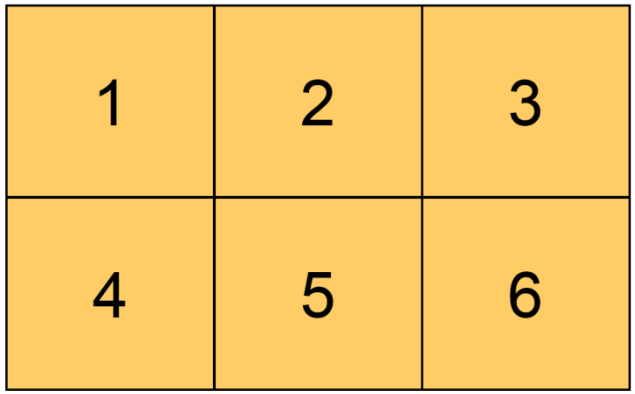
- Life-long learning
  - RL can accumulate knowledge and use this to solve new tasks, or to cope with changing environments.

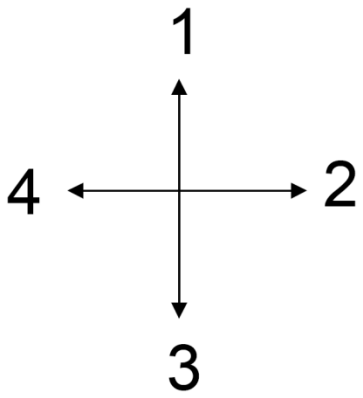
Example



Example

```
% 1 = N, 2 = E, 3 = S, 4 = W
% [currentState action nextState reward]
sa = [1 2 2 0
    1 3 4 0
    2 2 3 100
    2 3 5 0
    2 4 1 0
    3 1 3 0
    3 2 3 0
    3 3 3 0
    3 4 3 0
    4 1 1 0
    4 2 5 0
    5 1 2 0
    5 2 6 0
    5 4 4 0
    6 2 5 0
    6 1 3 100];
```

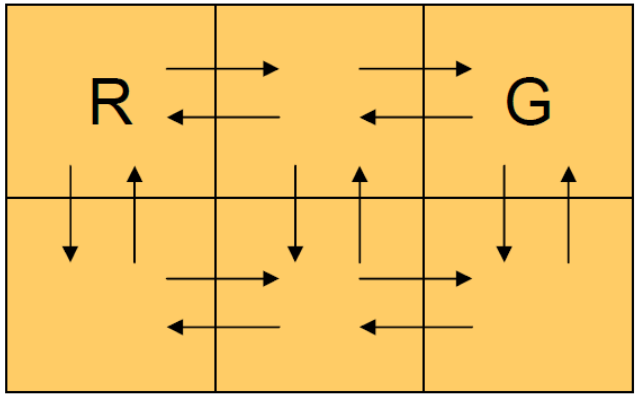


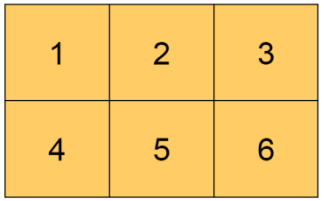


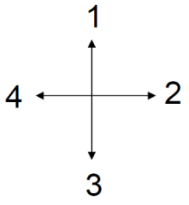
Example

- With $\gamma = 0.9$ the Q values are

```
% state action Q-value
Q= [1.0000    3.0000   72.9000
    4.0000    2.0000   81.0000
    5.0000    2.0000   90.0000
    6.0000    2.0000   81.0000
    5.0000    1.0000   90.0000
    2.0000    2.0000  100.0000
    5.0000    4.0000   72.9000
    6.0000    1.0000  100.0000
    4.0000    1.0000   81.0000
    1.0000    2.0000   90.0000
    2.0000    3.0000   81.0000
    2.0000    4.0000   81.0000
    3.0000    2.0000         0
    3.0000    1.0000         0
    3.0000    4.0000         0
    3.0000    3.0000         0]
```

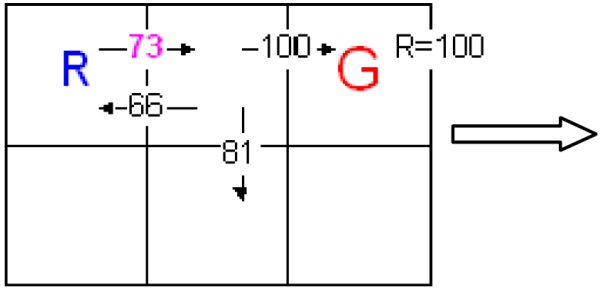




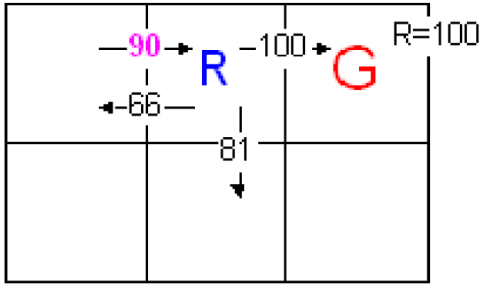


Example

Initial configuration (S1):



Next Iteration after taking action = *right* (S2):



$$\gamma = 0.9$$

$$Q(a,s) = Right$$

$$Q(a,s) = 0.9 \max(66,81,100) = 90$$

Example

- Robot is allowed to do some “episodes”: walking around randomly until it hits the goal state. After that, robot is placed at a different position and starts again.
- The $\widehat{Q}(s,a)$ table will receive its first non-zero value when the robot hits the goal.
- After that, the table gets refined and is eventually completely filled with $\widehat{Q}(s,a)$ values.

Properties of the algorithm

- $\widehat{Q}(s,a)$ alues never decrease during training.
- The $\widehat{Q}(s,a)$ values will remain between 0 and their real Q values.
- The $\widehat{Q}(s,a)$ values will equal the Q values when
  - The system is a deterministic MDP process.
  - When the reward values are bounded.
  - When the agent visits every state-action pair infinitely often.
  - Not very realistic conditions, but still Q-learning works very well.

Variations on Q learning

- Non-deterministic environments, when doing an action:
  - the next state is reached with a certain probability,
  - the reward may change over time.
- Often more realistic for robots with noisy sensors and actuators.
- Update rule for the table

$$\hat{Q}_n(s,a) = (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n\left(r + \max_{a'}\hat{Q}_{n-1}(s',a')\right)$$

$$\alpha_n = \frac{1}{1 + visits(s,a)}$$

Continuous states and actions

- Divide the continuous state space into discrete intervals.
- Make actions discrete.
  - Instead of motor value ∈[0,100], motor value ∈ {stop, slow, mid, fast}.
- Use a different structure for the $\widehat{Q}(s,a)$ table,
- Not a table anymore, but a structure which interpolates between states.

Light seeking robot

- A robot with two light sensors and front and back bumpers.
- The robot needs to find a light, but without bumping into obstacles.
- rewards?
  - **Approach light:** reward is equal to current light level – prev light level. Moving towards the light gives a positive reward.
  - **Avoid obstacles:** if a bumper is pressed, the reward is –2.
  - **Avoid staying still:** if the light level hasn't changed in the last 5 steps, the reward is –2.

Light seeking robot

- How to translate the perception of the robot into states?
- There are two light sensors, and two bumpers.
- table of “percepts”

Percept	1	2	3
Difference in light	L>R	L<R	L==R
Front bumper	Pressed	Unpressed	
Back bumper	Pressed	Unpressed	
No change in light level for 5 seconds	Yes	No	

Light seeking robot

State	Percepts
1	0,0,0,0
2	1,0,0,0
3	2,0,0,0
4	0,1,0,0
...	

Percept	1	2	3
Difference in light	L>R	L<R	L==R
Front bumper	Pressed	Unpressed	
Back bumper	Pressed	Unpressed	
No change in light level for 5 seconds	Yes	No	

Light seeking robot

- Actions

Action	Motors left, right
1	Stop, stop
2	Stop, fwd
3	Stop, rvs
4	Fwd, stop
5	Rvs, stop
6	Fwd, fwd
7	Rvs, rvs
8	Fwd, rvs
9	Rvs, fwd

Light seeking robot

- With 24 states and 9 actions, we will have a Q-table containing 216 Q-values.

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

- Set the learning rate γ to a value which gives a balance between speed and good learning from experience.
- Let agent explore with a certain probability (e.g. 0.2) and exploit at other times.