Reinforcement Learning Part 1

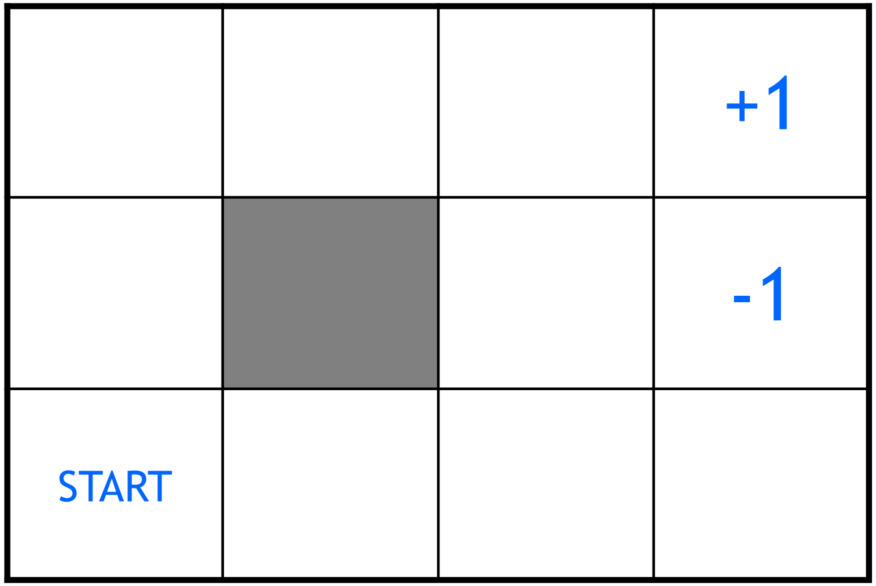
CPSC 470 – Artificial Intelligence

Brian Scassellati

From Supervised Learning to Reinforcement Learning

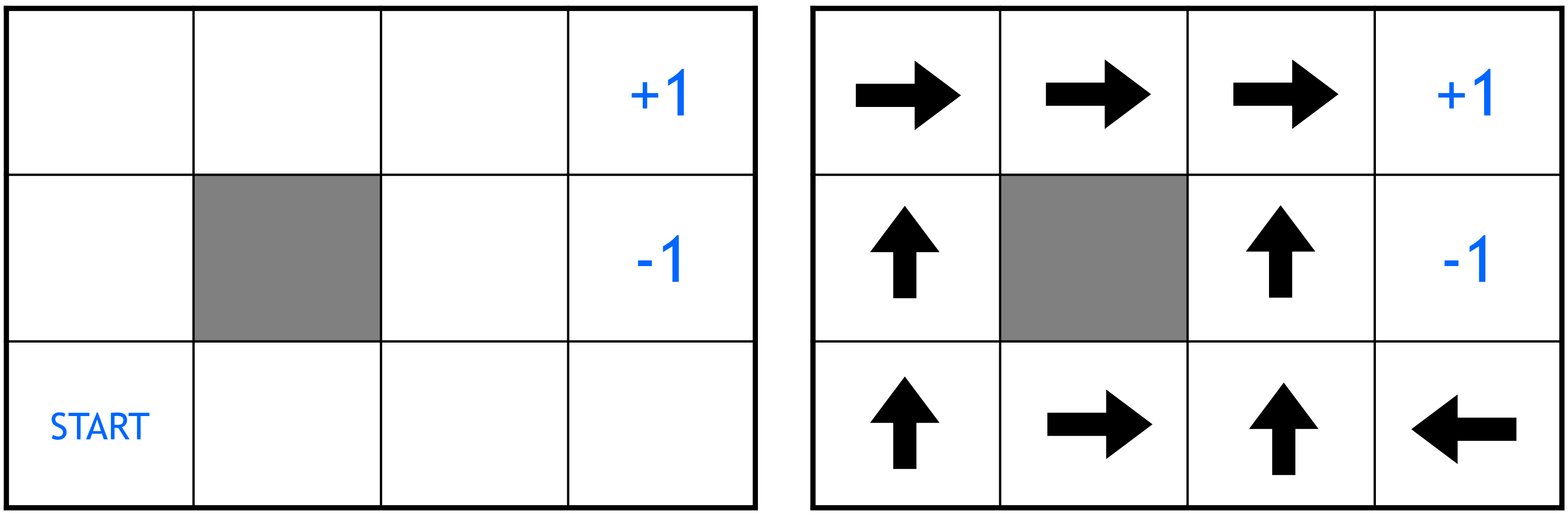
- In supervised learning, when the agent makes a mistake, it is immediately given feedback in the form of the correct response
- In reinforcement learning, when the agent makes a mistake, it will later be given feedback in the form of a punishment or reward

Deterministic Agent, Known Environment



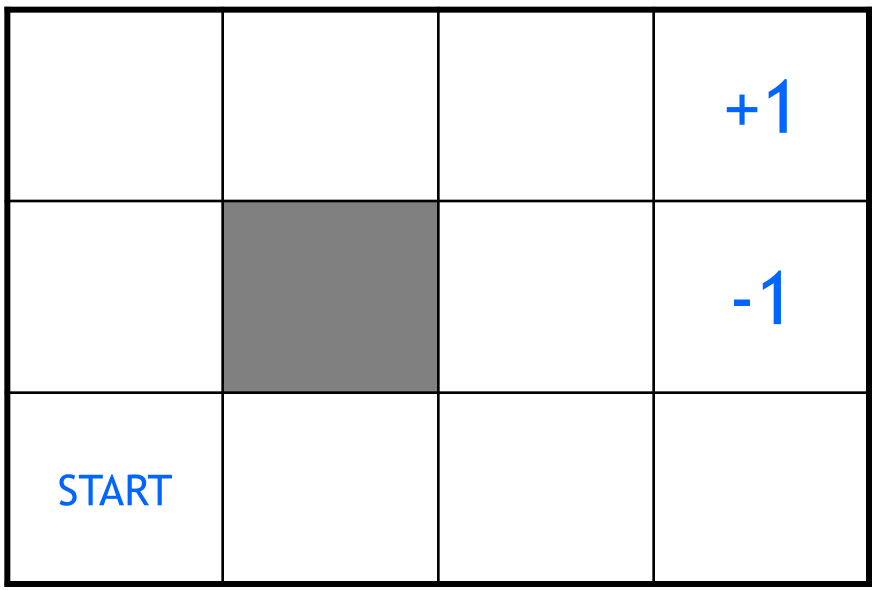
- reward +1 at [4,3], -1 at [4,2]

Optimal Deterministic Policy, Known Environment



- reward +1 at [4,3], -1 at [4,2]

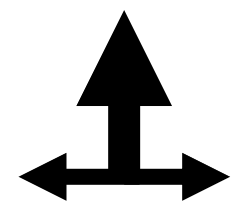
Non-deterministic Agent, Known Environment



actions: UP, DOWN, LEFT, RIGHT

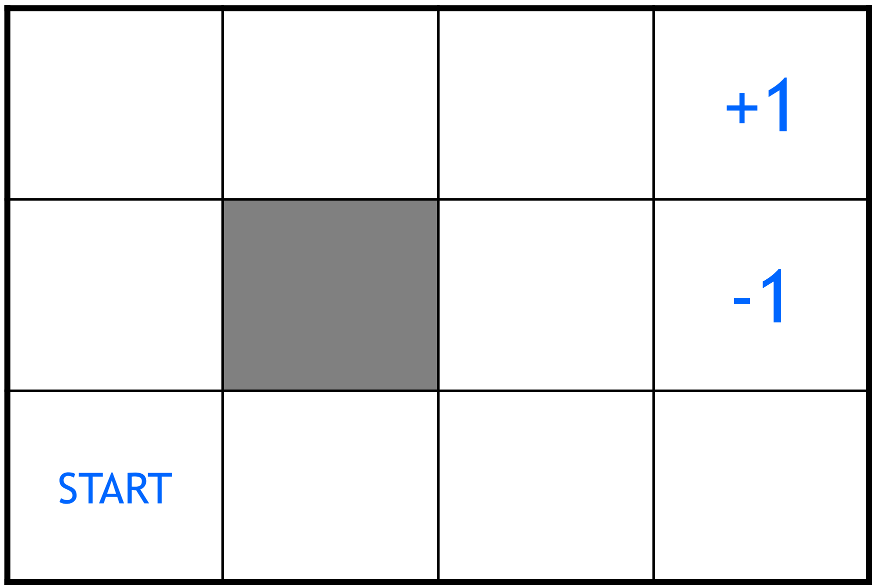
UP

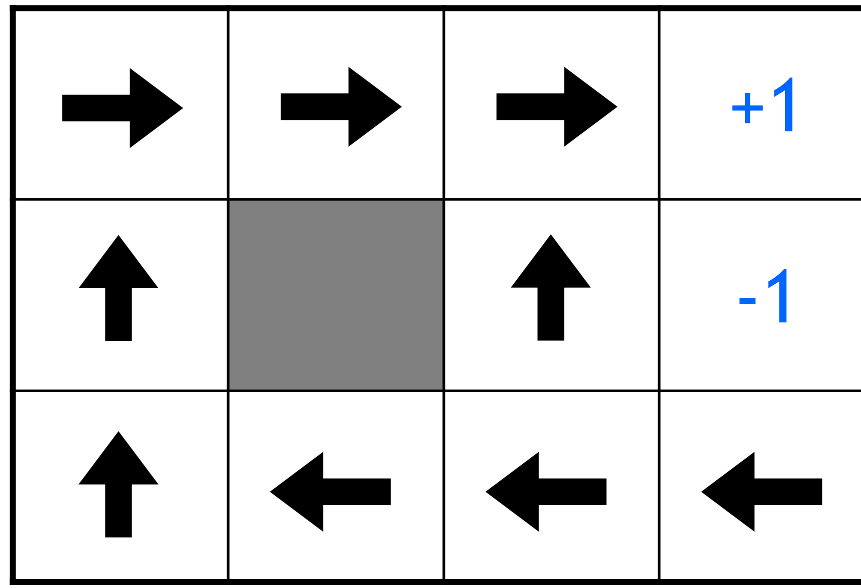
80%	move UP
10%	move LEFT
10%	move RIGHT



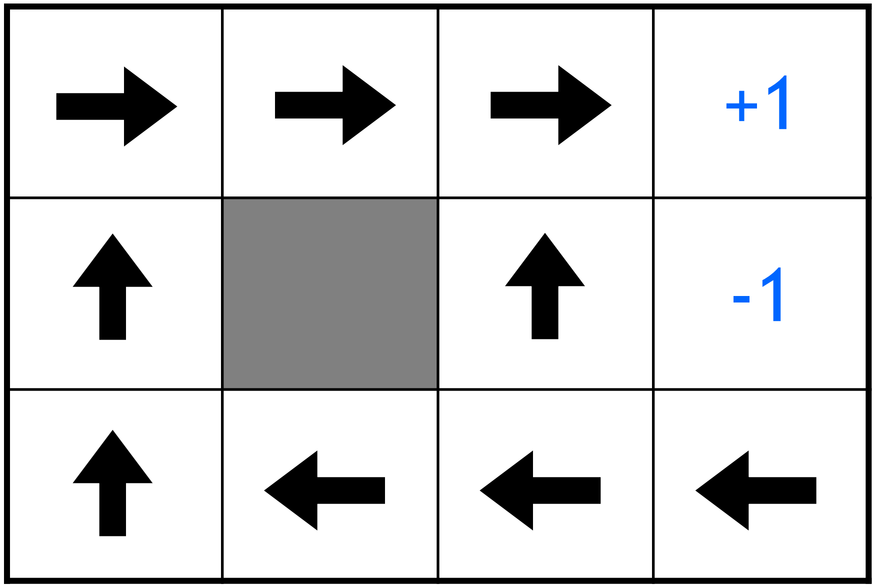
- reward +1 at [4,3], -1 at [4,2]
- what is the strategy to achieve max reward?
- reward -0.04 for each step

Non-deterministic Agent, Known Environment

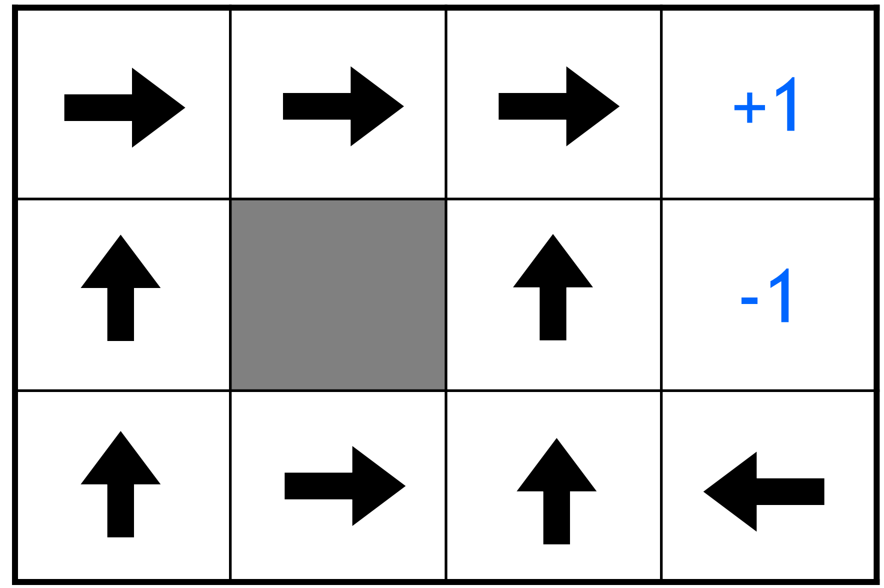




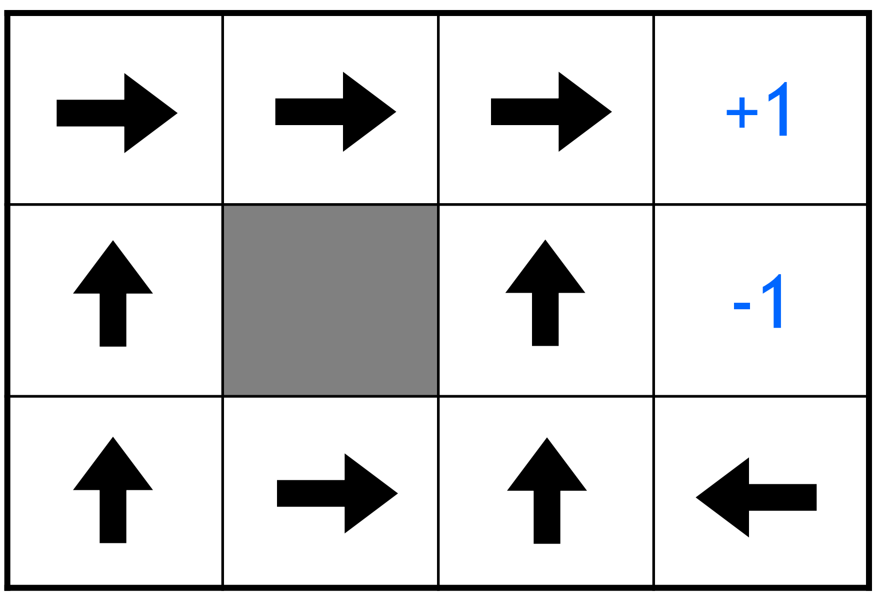
- reward +1 at [4,3], -1 at [4,2]
- what is the strategy to achieve max reward?
- reward -0.04 for each step



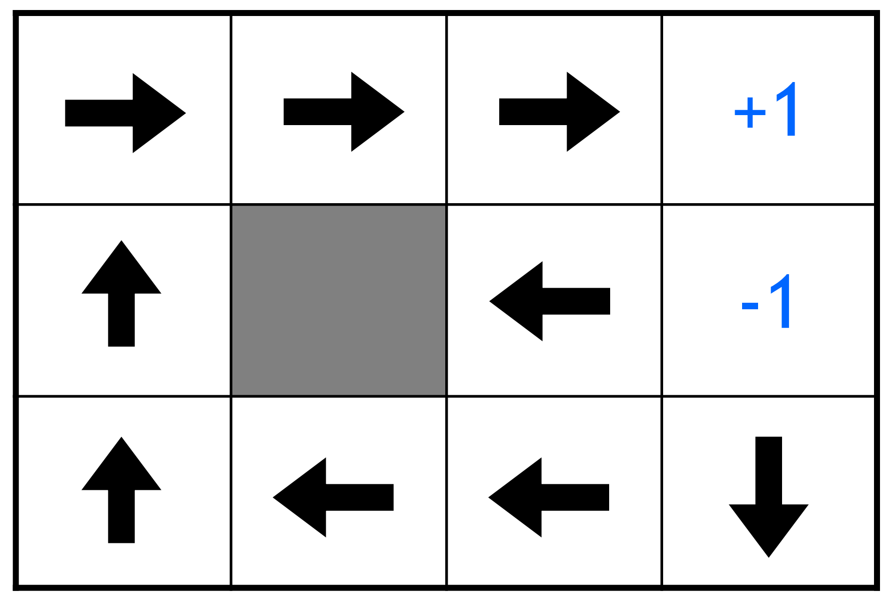
Reward for each step: -0.04



Deterministic policy



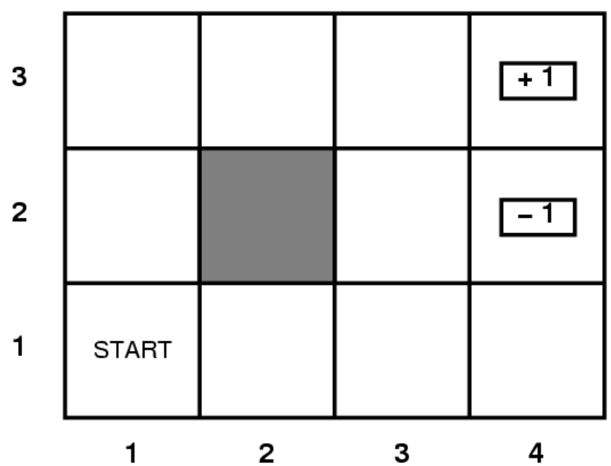
Reward for each step: -0.1



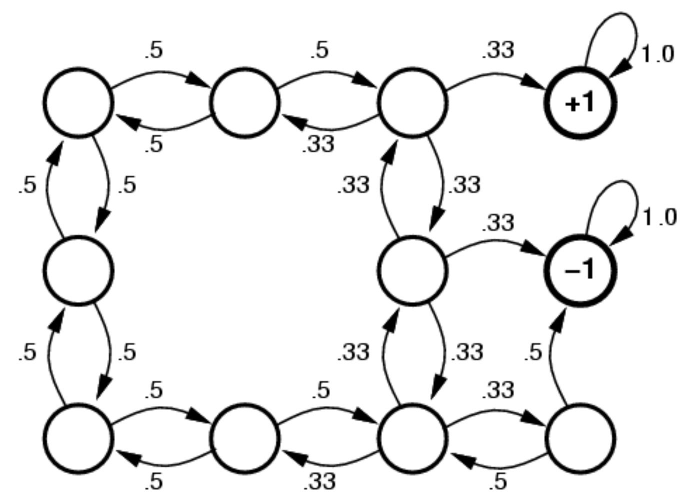
Reward for each step: -0.01

Sample Environment for Today

Simplified Wumpus World



State Transitions



- Reward function only defined at terminal states
- Equal probability transitions among neighboring states

Passive Learning in Known Environments

Actual Utility Values

3	−0.0380	0.0886	0.2152	+ 1
2	−0.1646		−0.4430	− 1
1	−0.2911	−0.0380	−0.5443	−0.7722
	1	2	3	4

Estimated Utility Values

3	-0.1	0.1	0.2	+ 1
2	-0.2		-0.4	− 1
1	-0.3	0	-0.5	-0.7
	1	2	3	4

- Given a set of training sequences that end in a terminal state (with a reward)

  (1,1) → (2,1) → (3,1) → (3,2) → (3,3) → (4,3) → **+1**

  (1,1) → (2,1) → (3,1) → (3,2) → (4,2) → **-1**

  (1,1) → (1,2) → (1,3) → (2,3) → (3,3) → (4,3) → **+1**

  (1,1) → (2,1) → (3,1) → (4,1) → (3,1) → (3,2) → (4,2) → **-1**

- Determine the expected utility ***U(i)*** associated with each non-terminal state ***i***

Passive Reinforcement Learning Agent

- Maintain
  - An estimate *U(i)* for all of the states *i*
  - The number of times you have visited each state
  - Table of transition properties between states
- How do we update our estimate?
  - Naïve updating: Least-Mean Squares
  - Temporal Difference Learning
  - Adaptive Dynamic Programming

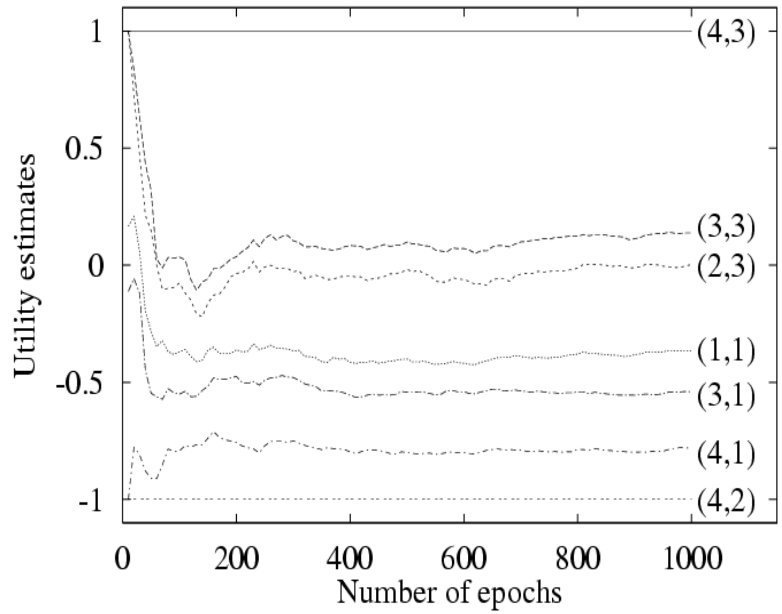
Updating via Least Mean Squares

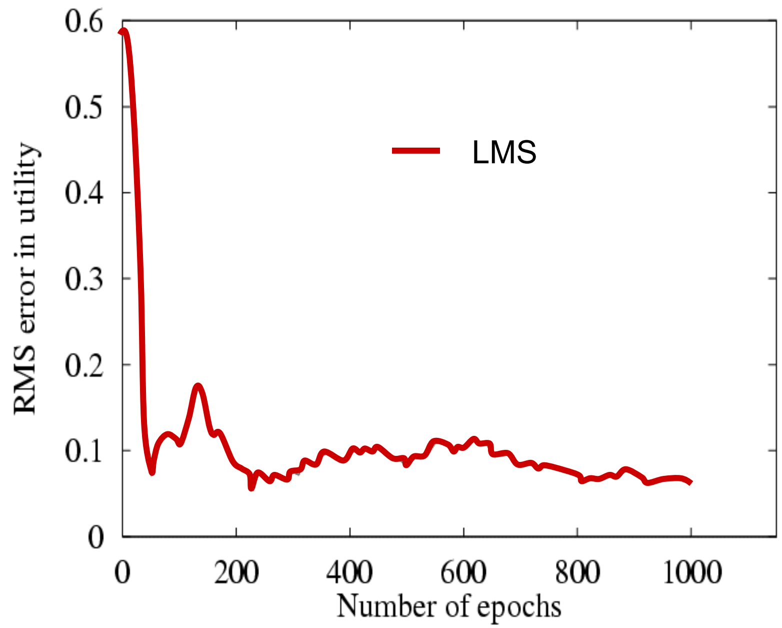
function LMS-UPDATE(U, e, *percepts*, M, N) **returns** an updated U

if TERMINAL?[e] **then** *reward-to-go* $\leftarrow$ 0
for each e_i **in** *percepts* (starting at end) **do**
  reward-to-go $\leftarrow$ *reward-to-go* + REWARD[e_i]
  U[STATE[e_i]] $\leftarrow$ RUNNING-AVERAGE(U[STATE[e_i]], *reward-to-go*, N[STATE[e_i]])
end

- (also known as Adaptive Control Theory)
- Define ***reward-to-go*** as the sum of the rewards from a state until a terminal state is reached
- Expected utility is the expected reward-to-go
- Estimate utility in order to minimize the mean square error among the observed sequence data

Updating via Least Mean Squares





Treats each utility measurement as independent… misses an important constraint!

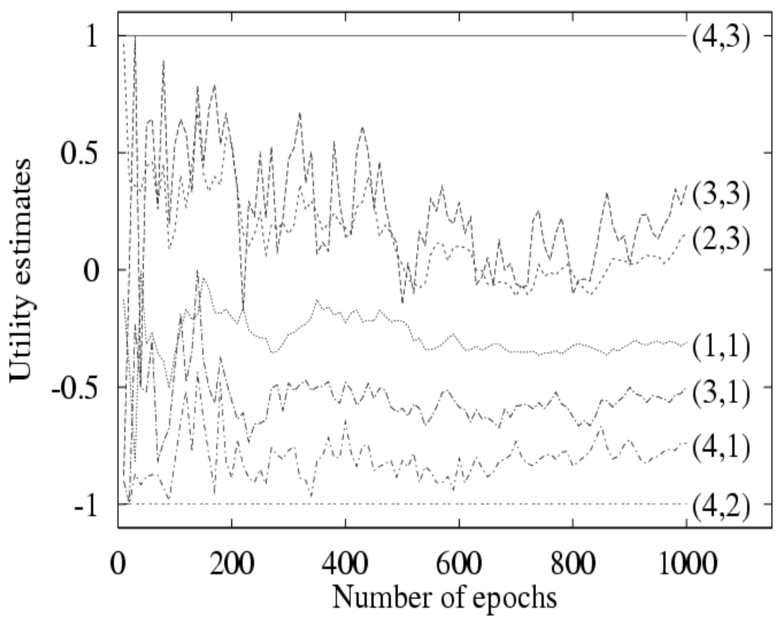
Updating via Temporal Difference

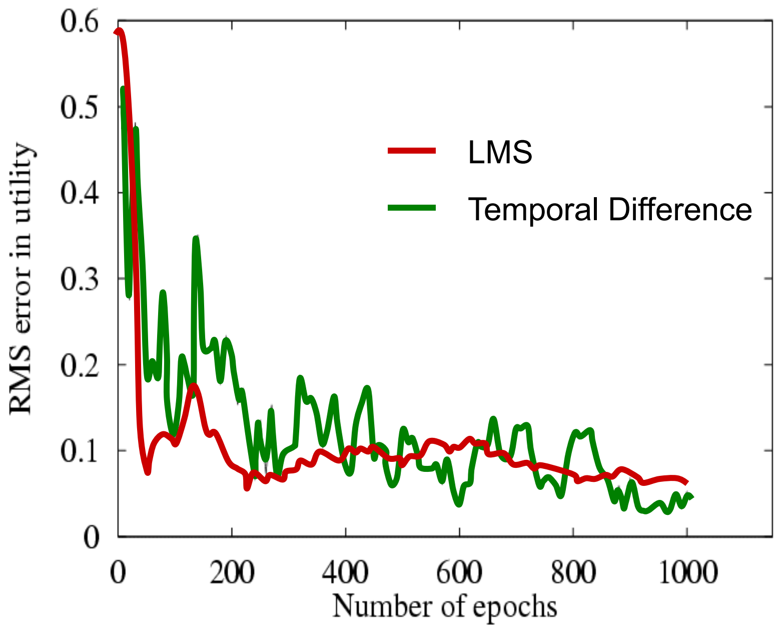
- Try to get the best of both worlds
  - Approximate the constraint equations between neighboring states
  - Provide a solution without computing all these equations
- Suppose that we often see a transition from U(i)=-0.5 and U(j)=+0.5
  - then we should increase U(i) to reflect the fact that it often leads to U(j)
- Update rule

$$U(i) \leftarrow U(i) + \alpha(N(i))[R(i) + U(j) - U(i)]$$

- Parameterize the learning rate by the number of times we have visited that state

Updating via Temporal Difference



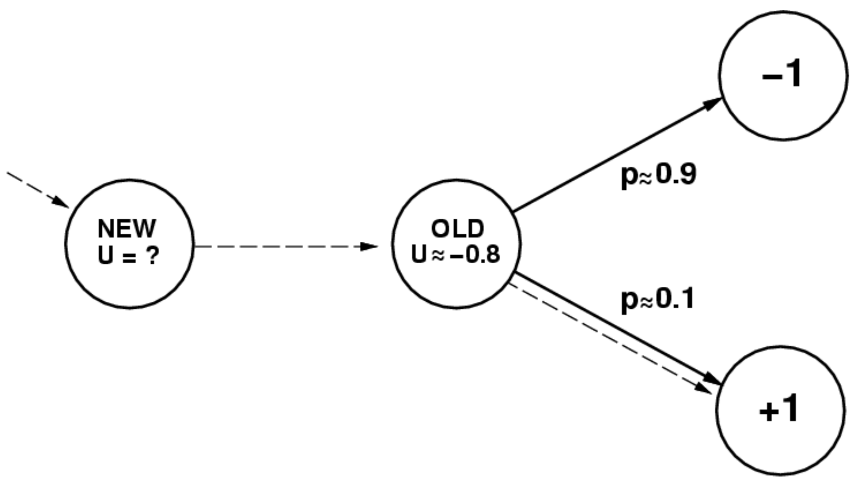


TD generates noisier values, but results in a lower RMS utility error

Do we need a Complete Model of the World?

- What information is required about the world state?
  - LMS makes no use of connectivity between states … it will work in an unknown environment
  - Temporal Difference makes use of connectivity, but only as much as is generated by the training sequences… it will also work in an unknown environment
- Look at an algorithm that does require a model of the world: Adaptive Dynamic Programming

Updating via Adaptive Dynamic Programming

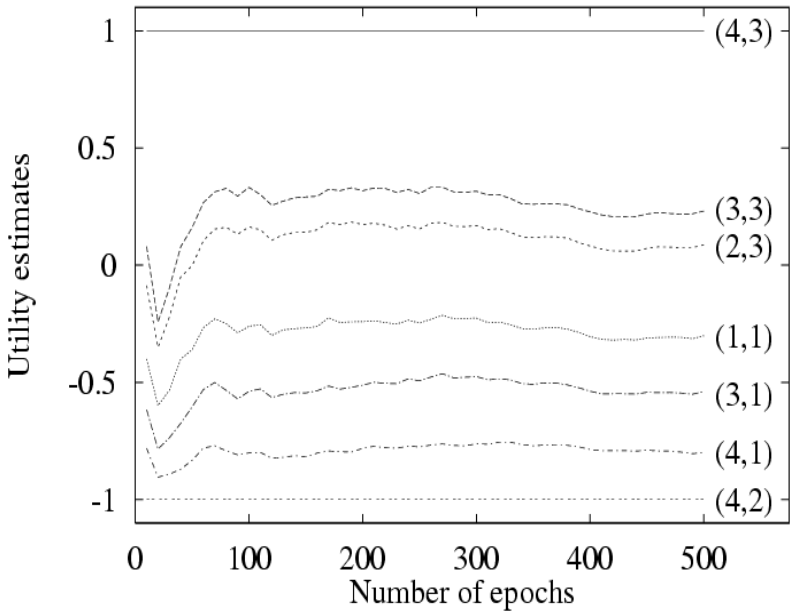


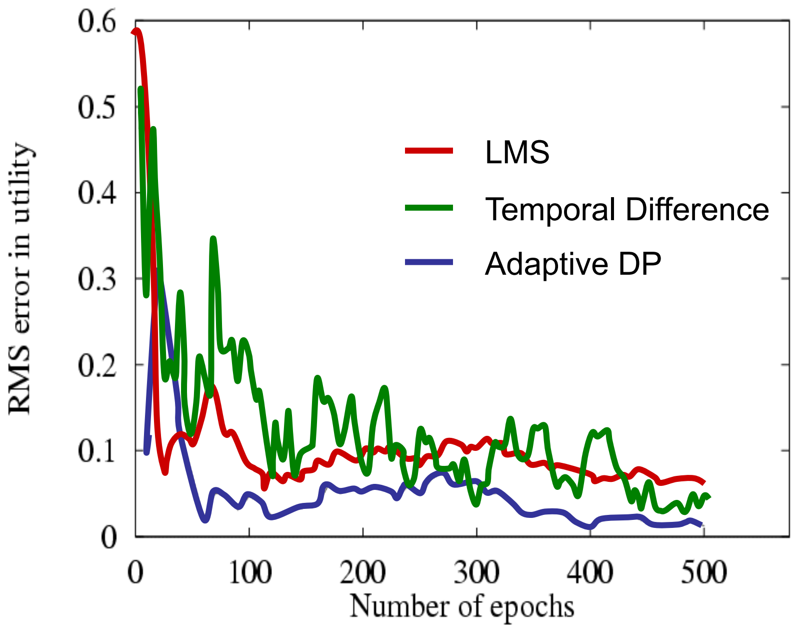
- Key idea: use knowledge of the structure of the environment to aid future decisions
- The actual utility of a state is constrained to be the probability-weighted average of its successors' utilities (plus its own reward)

$$U(i) = R(i) + \sum_j M_{ij} U(j)$$

- ADP solves these utility equations simultaneously using dynamic programming (equivalent to value determination)

Updating via Adaptive Dynamic Programming





Adaptive DP gives a very fast convergence at the expense of large compute costs (can be intractable for large search spaces)

Can we do better if the agent can actively explore the world?

- Need two changes to our existing algorithms
  - Environment model must incorporate the idea that transition probabilities are dependent on the action that we take
  - Utility must be based on choosing the action that maximizes the expected reward

$$U(i) \leftarrow R(i) + \max_{action} \sum_j M_{ij}^{action(i)} U(j)$$

Active Learning in an Unknown Environment

- Action has two kinds of outcomes
  - It gains rewards on the current sequence
  - It affects the percepts received and thus the ability of the agent to learn (and receive future reward)
- Trade-off between immediate gains (rewards) and long-term gains
- Range of learning approaches
  - Act randomly: explore as much as possible
  - Act greedy: always grab the immediate gain
  - … and everything in between

Exploration

- Is there an optimal exploration policy?
- Is there a reasonable exploration policy?
  - Main idea: give weight to actions that have not been tried very often
  - Example update rule:

$$U^+(i) \leftarrow R(i) + \max_{action} f\Big(\sum_j M_{ij}^{action} U^+(j)\ ,\ N(a,i)\Big)$$

Optimistic estimate of utility

Exploration function

Number of Times visited

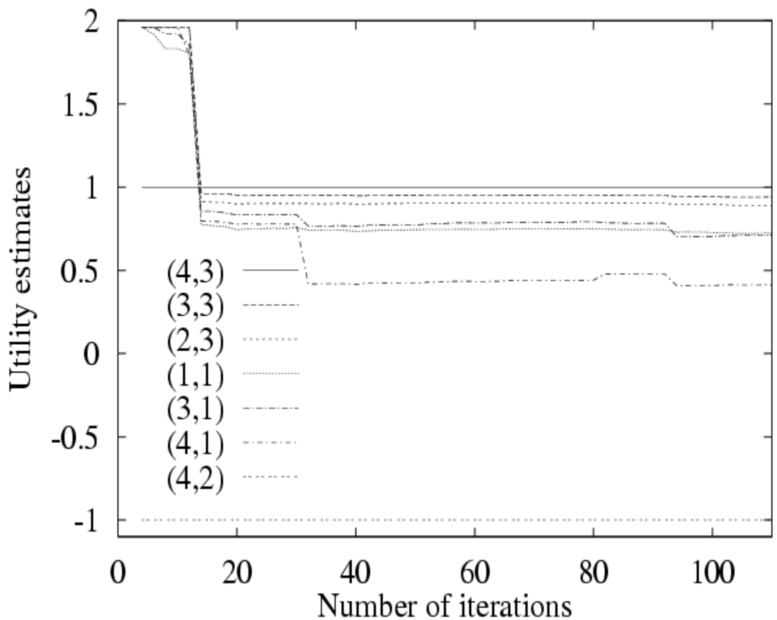
  - Exploration function

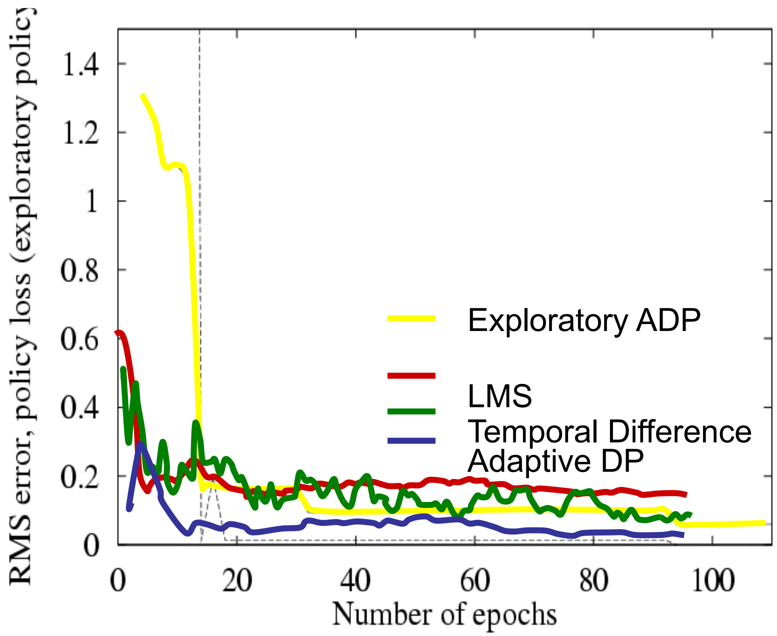
Optimistic estimate of reward

$$f(u,n) = \begin{cases} R^+ & \text{if n} < \mathrm{N_e} \\ u & \text{otherwise} \end{cases}$$

Fixed parameter

Exploratory ADP Agent (R^+=2 and N_e=5)





Exploratory ADP initially gives states an exploration bonus (high valued states quickly reach their correct values). Low-valued states take longer to adapt because they are seldom visited.

Administrivia

- Monday:
  - end of reinforcement learning
  - (Q-learning)