Reinforcement Learning

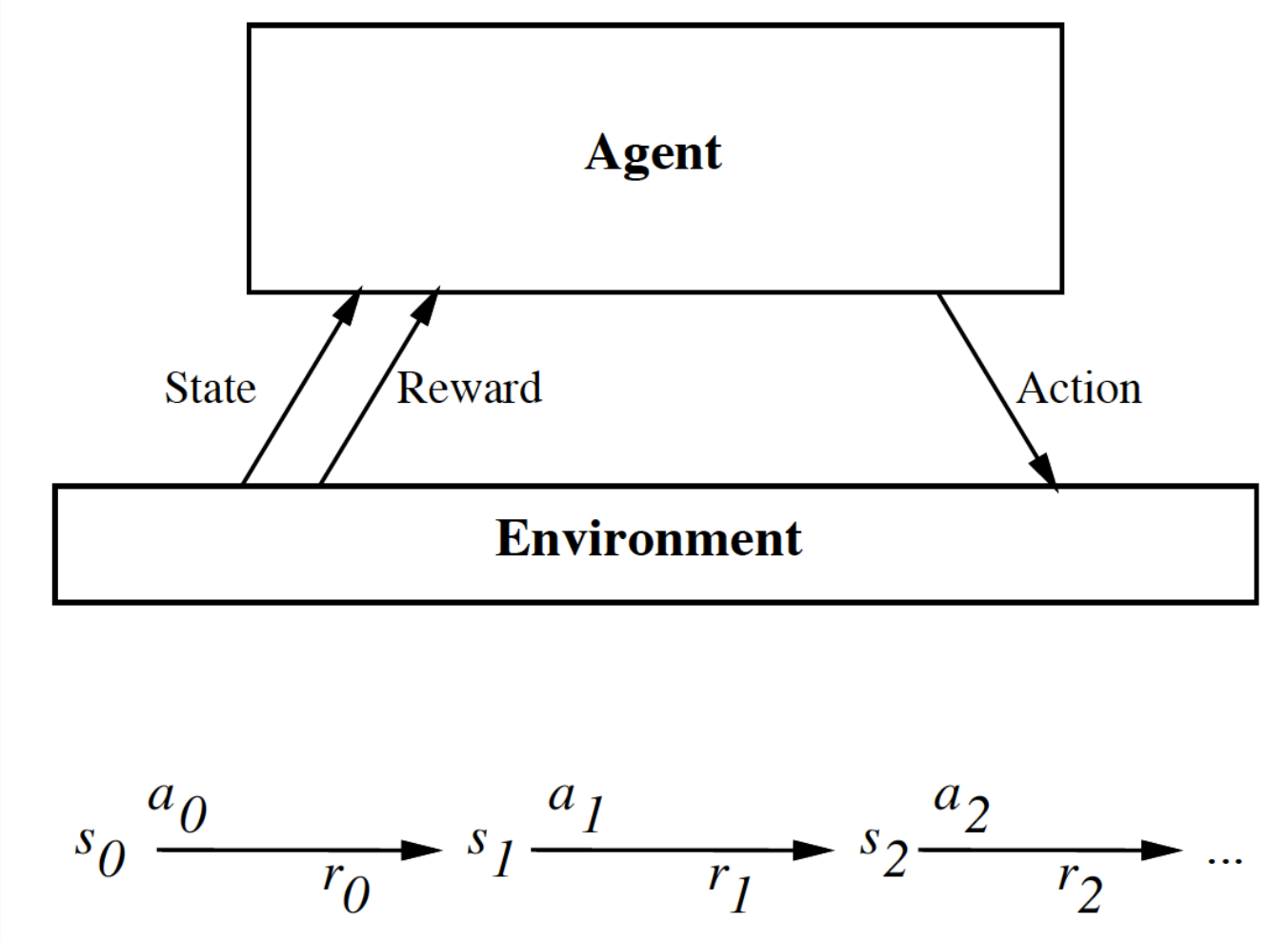
Instructor: Max Welling

Source: T. Mitchell, Machine Learning, Chapter 13.

Overview

- Supervised Learning: Immediate feedback (labels provided for every input).
- Unsupervised Learning: No feedback (no labels provided).
- Reinforcement Learning: Delayed scalar feedback (a number called reward).
- RL deals with agents that must sense & act upon their environment.
  This is combines classical AI and machine learning techniques.
  It the most comprehensive problem setting.
- Examples:
  - A robot cleaning my room and recharging its battery
  - Robot-soccer
  - How to invest in shares
  - Modeling the economy through rational agents
  - Learning how to fly a helicopter
  - Scheduling planes to their destinations
  - and so on

The Big Picture



Your action influences the state of the world which determines its reward

Complications

- The outcome of your actions may be uncertain
- You may not be able to perfectly sense the state of the world
- The reward may be stochastic.
- Reward is delayed (i.e. finding food in a maze)
- You may have no clue (model) about how the world responds to your actions.
- You may have no clue (model) of how rewards are being paid off.
- The world may change while you try to learn it
- How much time do you need to explore uncharted territory before you exploit what you have learned?

The Task

- To learn an optimal *policy* that maps states of the world to actions of the agent. I.e., if this patch of room is dirty, I clean it. If my battery is empty, I recharge it.

$$\pi : S \rightarrow A$$

- What is it that the agent tries to optimize?
  Answer: the total future discounted reward:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$

$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i} \qquad 0 \leq \gamma < 1$$

Note: immediate reward is worth more than future reward.

What would happen to mouse in a maze with gamma = 0 ?

Value Function

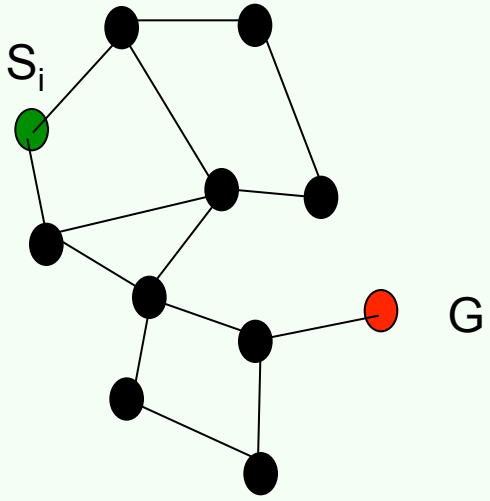
- Let's say we have access to the optimal value function that computes the total future discounted reward $V^*(s)$
- What would be the optimal policy $\pi^*(s)$?
- Answer: we choose the action that maximizes:

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

- We assume that we know what the reward will be if we perform action "a" in state "s": $r(s,a)$
- We also assume we know what the next state of the world will be if we perform action "a" in state "s": $s_{t+1} = \delta(s_t, a)$

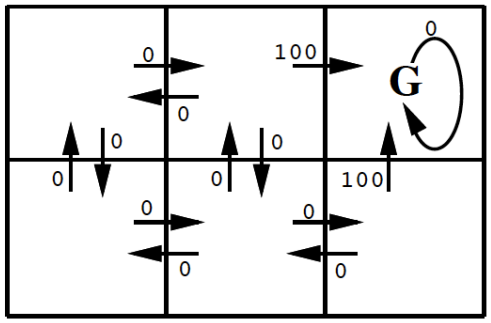
Example I

- Consider some complicated graph, and we would like to find the shortest path from a node Si to a goal node G.
- Traversing an edge will cost you "length edge" dollars.
- The value function encodes the total remaining distance to the goal node from any node s, i.e. V(s) = "*1 / distance*" to goal from s.
- If you know V(s), the problem is trivial. You simply choose the node that has highest V(s).

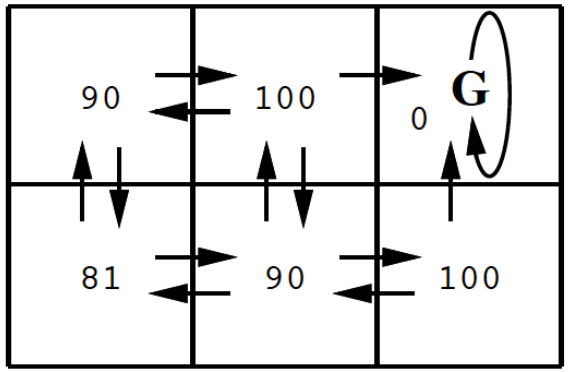


Example II

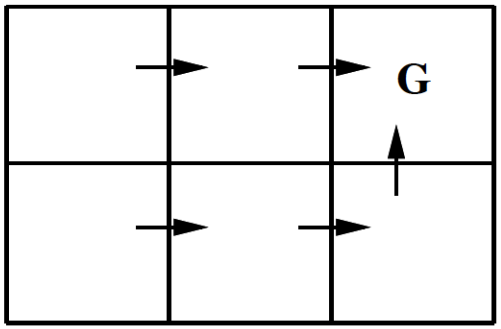
Find your way to the goal.



$r(s, a)$ (immediate reward) values



$V^*(s)$ values



One optimal policy

Q-Function

Bellman Equation:

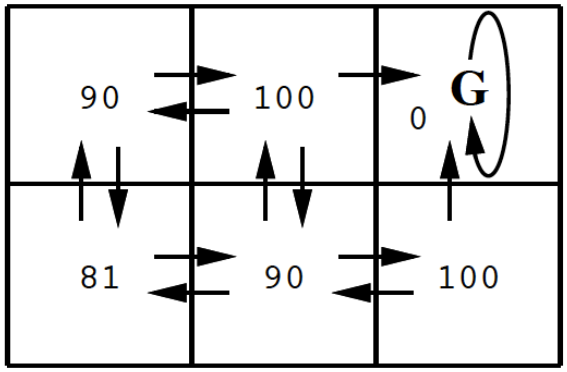
$$V^*(s) \leftarrow \max_a \left[r(s,a) + \gamma V^*(\delta(s,a)) \right]$$

- One approach to RL is then to try to estimate V*(s).
- However, this approach requires you to know r(s,a) and delta(s,a).
- This is unrealistic in many real problems. What is the reward if a robot is exploring mars and decides to take a right turn?
- Fortunately we can circumvent this problem by exploring and experiencing how the world reacts to our actions. We need to *learn* r & delta.
- We want a function that directly learns good state-action pairs, i.e. what action should I take in this state. We call this Q(s,a).
- Given Q(s,a) it is now trivial to execute the optimal policy, *without knowing r(s,a) and delta(s,a).* We have:

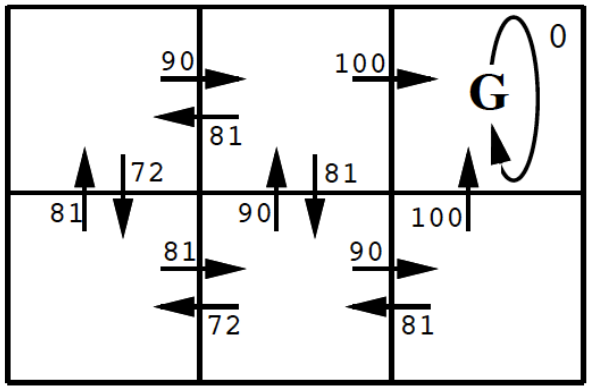
$$\pi^*(s) = \arg\max_a Q(s,a)$$

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

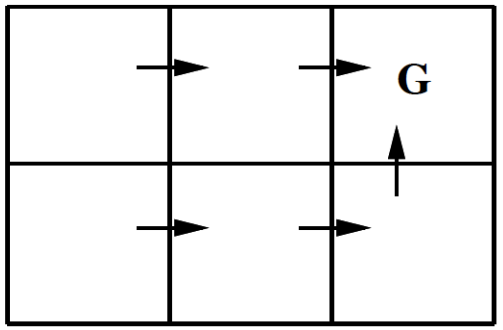
Example II



$V^*(s)$ values



$Q(s,a)$ values



One optimal policy

Check that

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

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

Q-Learning

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$$
$$= r(s,a) + \gamma \max_{a'} Q(\delta(s,a),a')$$

- This still depends on r(s,a) and delta(s,a).
- However, imagine the robot is exploring its environment, trying new actions as it goes.
- At every step it receives some reward "r", and it observes the environment change into a new state s' for action a.
  How can we use these observations, (s,a,s',r) to learn a model?

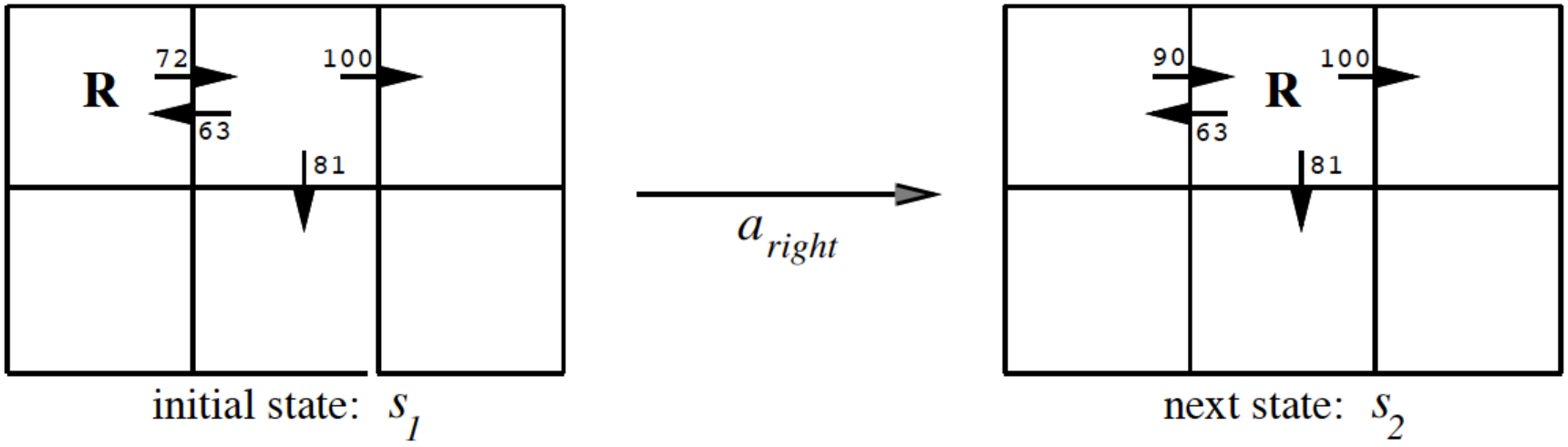
$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a') \qquad s' = s_{t+1}$$

Q-Learning

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a') \qquad s' = s_{t+1}$$

- This equation continually estimates Q at state s consistent with an estimate of Q at state s', one step in the future: temporal difference (TD) learning.
- Note that s' is closer to goal, and hence more "reliable", but still an estimate itself.
- Updating estimates based on other estimates is called bootstrapping.
- We do an update after each state-action pair. I.e., we are learning online!
- We are learning useful things about explored state-action pairs. These are typically most useful because they are likely to be encountered again.
- Under suitable conditions, these updates can actually be proved to converge to the real answer.

Example Q-Learning



$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$$

$$\leftarrow 0 + 0.9 \max\{66, 81, 100\}$$

$$\leftarrow 90$$

Q-learning propagates Q-estimates 1-step backwards

Exploration / Exploitation

- It is very important that the agent does not simply follow the current policy when learning Q. (off-policy learning).The reason is that you may get stuck in a suboptimal solution. I.e. there may be other solutions out there that you have never seen.

- Hence it is good to try new things so now and then, e.g. $P(a \mid s) \propto e^{\hat{Q}(s,a)/T}$
  If T large lots of exploring, if T small follow current policy.
  One can decrease T over time.

Improvements

- One can trade-off memory and computation by cashing (s,s’,r) for observed transitions. After a while, as Q(s’,a’) has changed, you can “replay” the update:

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

- One can actively search for state-action pairs for which Q(s,a) is expected to change a lot (prioritized sweeping).

- One can do updates along the sampled path much further back than just one step ($TD(\lambda)$ learning).

Extensions

- To deal with stochastic environments, we need to maximize *expected* future discounted reward:

$$Q(s,a) = E[r(s,a)] + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a')$$

- Often the state space is too large to deal with all states. In this case we need to learn a function: $Q(s,a) \approx f_\theta(s,a)$

- Neural network with back-propagation have been quite successful.

- For instance, TD-Gammon is a back-gammon program that plays at expert level. state-space very large, trained by playing against itself, uses NN to approximate value function, uses TD(lambda) for learning.

More on Function Approximation

- For instance: linear function: $Q(s,a) \approx f_\theta(s,a) = \sum_{k=1}^{K} \theta_k^a \Phi_k(s)$

The features Phi are fixed measurements of the state (e.g. # stones on the board). We only learn the parameters theta.

- Update rule: (start in state s, take action a, observe reward r and end up in state s')

$$\theta_k^a \leftarrow \theta_k^a + \alpha \underbrace{\left(r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a) \right)}_{\text{change in Q}} \Phi_k(s)$$

Conclusion

- Reinforcement learning addresses a very broad and relevant question: How can we learn to survive in our environment?
- We have looked at Q-learning, which simply learns from experience. No model of the world is needed.
- We made simplifying assumptions: e.g. state of the world only depends on last state and action. This is the *Markov* assumption. The model is called a *Markov Decision Process (MDP).*
- We assumed deterministic dynamics, reward function, but the world really is stochastic.
- There are many extensions to speed up learning.
- There have been many successful real world applications.

http://elsy.gdan.pl/index.php?option=com_content&task=view&id=20&Itemid=39