Reinforcement Learning Part I

Volker Tresp
2021

Introduction

- Some researchers distinguish between supervised learning and unsupervised learning

- Supervised learning: We typically have an input $\mathbf{x}$ and an output y and in the training data both are known; in the test data, only $\mathbf{x}$ is known

- In unsupervised learning, only $\mathbf{x}$ is known in training and testing. Maybe a better definition, which would include Bayesian networks, is to say that one does not distinguish between input and output. One might be interested in understanding structure in the data (as in clustering); another case is that in test data different variables can be inputs (known) or output (variable of interest to predict)

Introduction (cont'd)

- But in both cases, the learner is not really smart in the sense that the agent can only learn to describe what is in the data
- In a clinical setting, we might learn to imitate the doctor, but how can we become smarter than the doctor? Reinforcement learning is a learning-based optimization approach, which tries to achieve that
- In reinforcement learning, an agent acts in an environment and receives rewards and punishments as feedback and as a result of its actions
- The agent should learn, based on interactions with the environment, and from rewards (or punishments) to optimize its behavior
- Reward can be delayed (end of a board game)
- Actions may have long term consequences

Introduction (cont'd)

- Parents reward their child for doing something well; the child learns to repeat what is well done (for more rewards to obtain)
- Small children reward and punish their parents with a smile or by uncontrolled screaming; parents learn to optimize their behavior so that rewards are maximized and punishments are minimized; here it is obvious that children only give unspecific signals, and parents are trained, to find out which action is optimal (change diaper, feed, put to sleep, carry around ...)
- Technically we are considering Markov decision processes (MDPs) and we are solving multistage optimization problems
- The first part of the lecture is about the derivation of optimal actions with perfect knowledge about the system one is trying to optimize (keywords are: model-based, planning, dynamic programming)

Introduction (cont'd)

- The second part is about the derivation of optimal actions; you derive the policy by observing the system or a simulation (off-policy) or by experimenting with the system or a simulation (on-policy) or by learning a model from observed data
- In the third part we discuss the role of function approximation (e.g., neural networks) and a number of advanced topics

Literature

- An excellent book and the standard is: An Introduction to Reinforcement Learning, Sutton and Barto (SB)
- Excellent slide set and video: https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver (S); this lecture uses the structure and content of this slide set

Preliminary: Expected Values are Additive

- Consider random variables X and Y with a joint $P(X, Y)$
- We have $\mathbb{E}(X) = \sum_x xP(x)$ and $\mathbb{E}(Y) = \sum_y yP(y)$
- Let $Z = X + Y$; then

$$\mathbb{E}(Z) = \sum_{x,y}(x+y)P(x,y) = \sum_{x,y} xP(y|x)P(x)$$

$$+\sum_{x,y} yP(x|y)P(y) = \mathbb{E}(X) + \mathbb{E}(Y)$$

- So even when X and Y are not independent, expected values are additive; also,

$$\mathbb{E}(X) \approx \frac{1}{N_s}\sum_{i=1}^{N_s} x_i$$

with N_s samples drawn either from $P(X)$ or from $P(X, Y)$

Preliminary: An Important Identity used in Policy Gradient

- We know

$$\frac{\partial \log f_w(x)}{\partial w} = \frac{1}{f_w(x)} \frac{\partial f_w(x)}{\partial w}$$

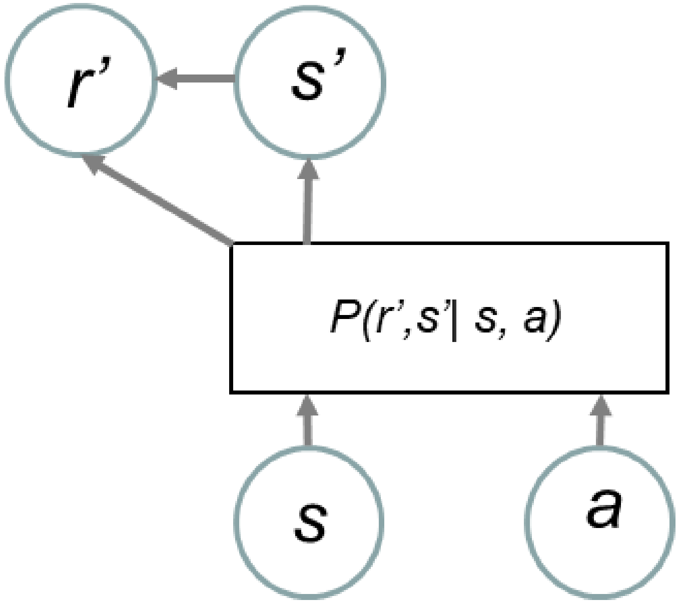
- This implies

$$\frac{\partial f_w(x)}{\partial w} = f_w(x) \frac{\partial \log f_w(x)}{\partial w}$$

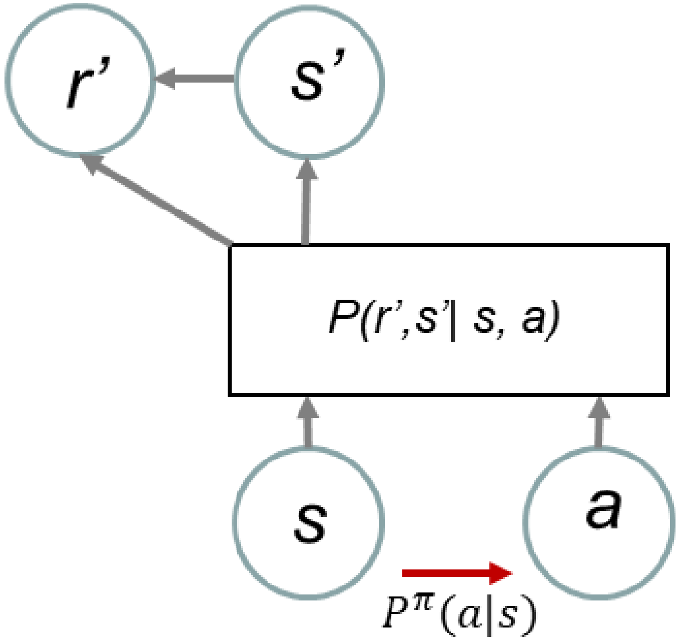
One Stage Problem

- We begin very simple: there is only one action at one instance
- The next figure shows the random variables and their dependencies as a Bayes net
- $S = s$ means that the world is in state s; the state can also represent the history of what had happened to the agent
- $A = a$ means that action a is executed
- $S' = s'$ is the state of the world in the next instance
- $R' = r'$ means that the agent obtained reward r' (e.g., $r' = 1$, or $r' = 0$) “at” s'; the state is labelled by the actual reward value

- Structure relevant for calculating the Q-function (action-value function)



- Structure relevant for calculating the state-value function
- We assume a policy $P^{\pi}(a|s)$



One Stage: Dependencies and Probabilities for Action-Value Setting

- Left: The probability distribution, conditioned on s and a, is

$$P(r', s'|s, a) = P(s'|a, s)P(r'|s', s, a)$$

- $P(s'|s, a)$ is the probability of the next state, given the current state and given that action a is executed; it can be represented by a 3-way array of size $|S| \times |S| \times |A|$

- $P(r'|s', s, a)$ is the probability for reward r' when we observe a transition from s to s' under a; note that we include in our model the possibility, that reward is probabilistic, even when s', s, a is given! it can be represented by a 4-way array of size $|S| \times |S| \times |S| \times |A|$

One Stage: Dependencies and Probabilities for Value Setting

- Right: The probability distribution, conditioned on s, is

$$P^{\pi}(r', s', a|s) = P^{\pi}(a|s)P(s'|a, s)P(r'|s', s, a)$$

- $P^{\pi}(a|s)$ is called the **policy** π (also called control law); a policy can be represented as an array of size $|A| \times |S|$

- If we have a deterministic policy, we write $a = \pi(s)$; an optimal policy is indicated by an asterisk π^*; in fully observed MDPs, optimal policies are deterministic!

One Stage: Reward Function and Q-Function

- We define the **reward function**

  $$\mathcal{R}(s,a) = \sum_{s'} \sum_{r'} r' P(r'|s,s',a) P(s'|s,a)$$

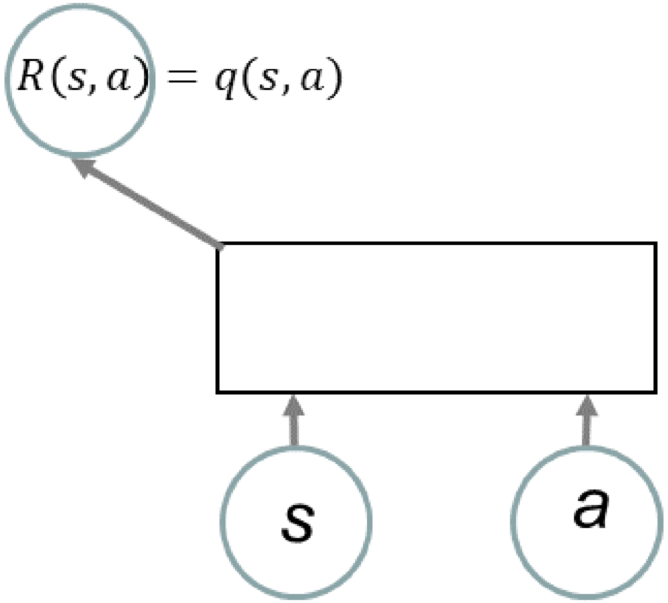
  This is the expected instantaneous reward, when action a is executed in state s; $\mathcal{R}(s,a)$ can be represented by a 2-way array of size $|S| \times |A|$; note that the expected reward at s' is associated with state s

- The **action-value function** (also called **Q-function**) in this case is defined as

  $$q(s,a) \doteq \mathbb{E}(r'|s,a) = \sum_{s'} \sum_{r'} r' P(r'|s,s',a) P(s'|s,a)$$

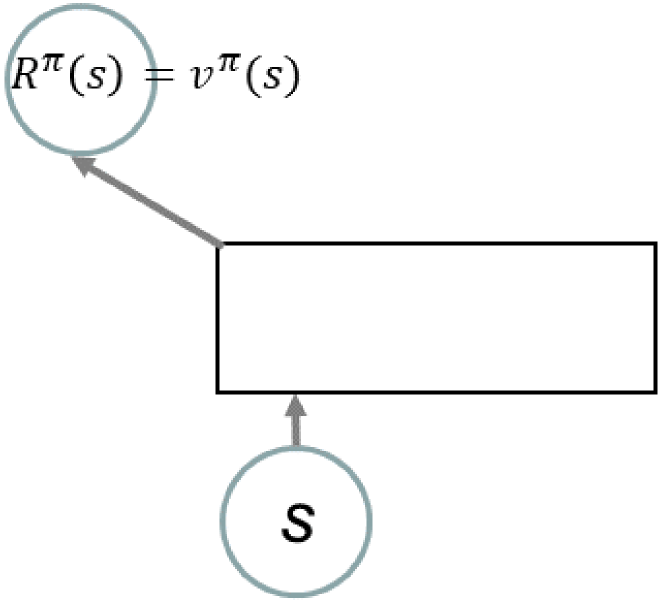
  It is the expected reward, given that we are in state s and perform action a; in the one-step case, $q(s,a) = \mathcal{R}(s,a)$

- Here the reward function is the Q-function
- We integrate out *s*'



$R(s,a) = q(s,a)$

- For a given policy, the reward function is the value-function
- We integrate out *a* and *s'*



$R^{\pi}(s) = v^{\pi}(s)$

One Stage: Value Function

- In addition, we can integrate out the action. We define,

  $$\mathcal{R}^\pi(s) = \sum_a P^\pi(a|s) \sum_{s'} \sum_{r'} r' P(r'|s, s', a) P(s'|s, a)$$

  This is the expected instantaneous reward, under policy π; $\mathcal{R}^\pi(:)$ can be represented as a 1-way array of size $|S|$; note that $\mathcal{R}^\pi(s)$ can be considered a function of s (and not s')

- We define the **state-value function** (also simply called **value function**) as

  $$v^\pi(s) \doteq \mathbb{E}^\pi(r'|s) = \sum_a P^\pi(a|s) \sum_{s'} \sum_{r'} r' P(r'|s, s', a) P(s'|s, a)$$

  It is the expected reward, given that we are in state s and follow policy π; $v^\pi(s)$ can be represented as a 1-way array of size $|S|$; in the one-step case, $v^\pi(s) = \mathcal{R}^\pi(s)$

One Stage: Relationship Between Action-Value Function and Value Funciton

- We can calculate the value function from the Q-function as

$$v^{\pi}(s) = \sum_{a} P^{\pi}(a|s) q(s, a)$$

One Stage: Optimal Control

- We are now interested to perform the action that gives us the maximum expected reward

- The optimal action for state s is

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

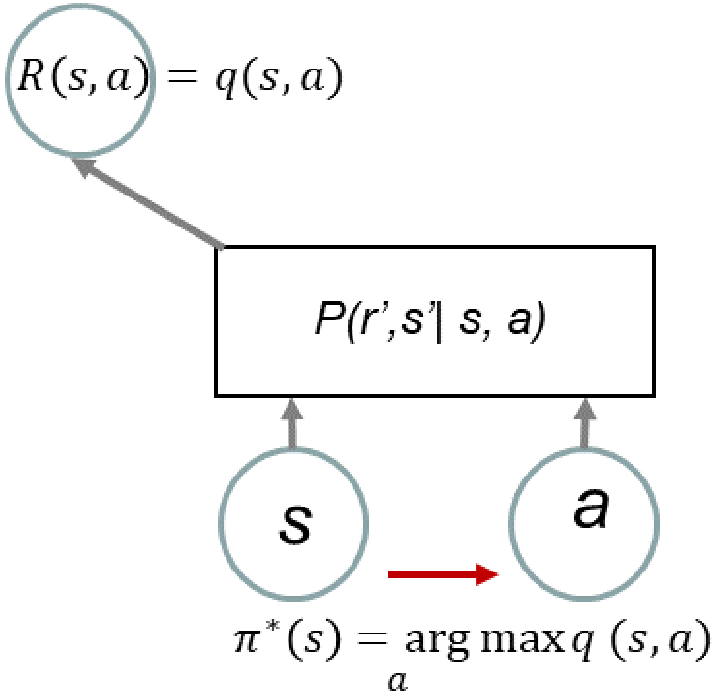
- Thus the **optimal policy** is deterministic with

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

- The **value of the optimal policy** is

$$v^*(s) = \max_a q(s,a)$$

- Optimal policy

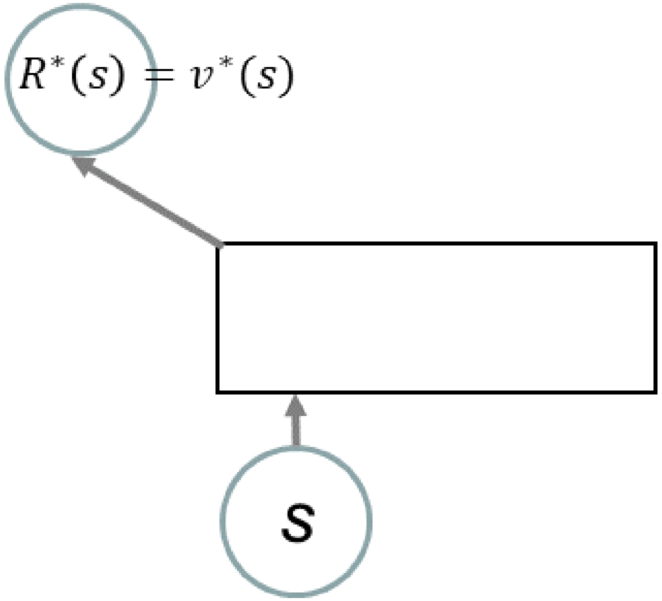


$R(s,a) = q(s,a)$

P(r’,s’| s, a)

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

- The value function for the optimal policy



$R^*(s) = v^*(s)$

Optimizing an Input

- The optimal policy is found by optimizing the Q-function with respect to an input (the action)

One Stage: Contextual Bandits

- One stage reinforcement learning problems (i.e., with unknown models) are called contextual multi-armed bandit problems and are special cases of learning automata
- Contextual: s is observed
- Las Vegas-style slot machines are sometimes called "one-armed bandits"

Two Stages: Dependencies and Probabilities for Action-Value Setting

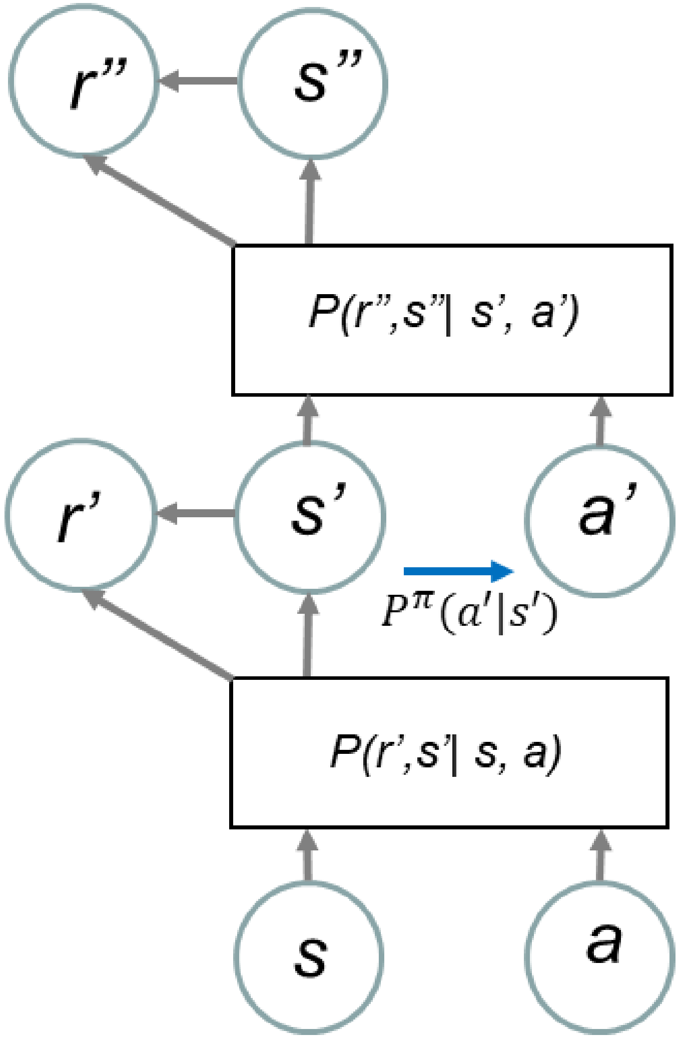
- We now make everything slightly more complex by considering two steps (see figure)
- Left: The probability distribution, conditioned on s and a, is

$$P^{\pi}(r'', r', s'', s', a'|s, a) = P(s'|s, a)P(r'|s', s, a)$$

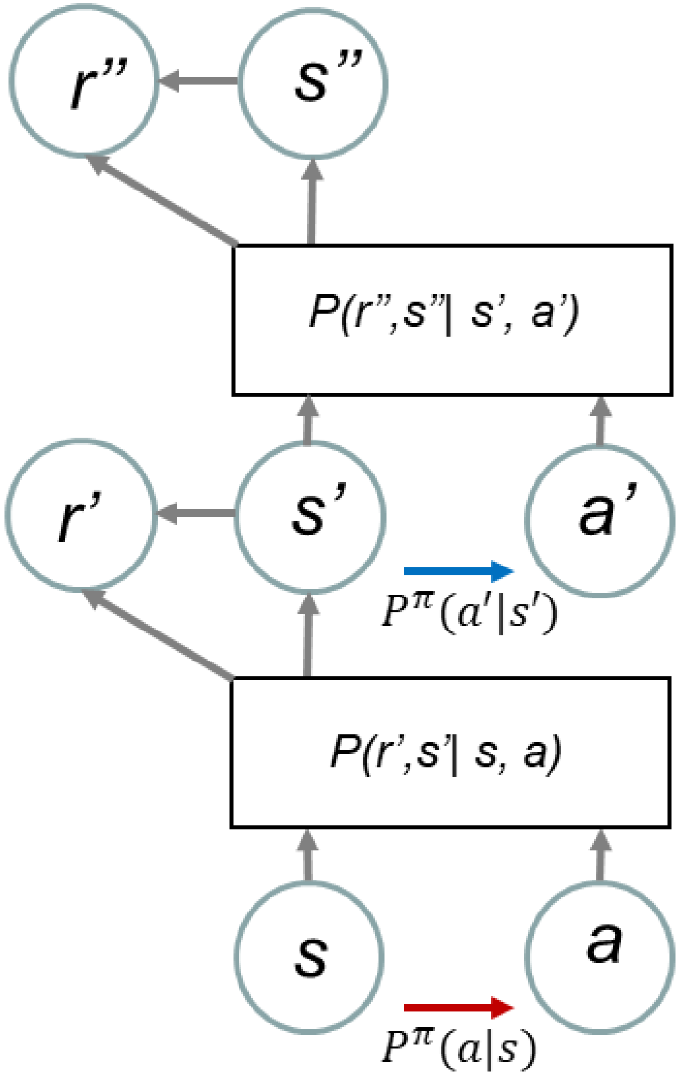
$$\times P^{\pi}(a'|s')P(s''|s', a')P(r''|s'', s', a')$$

- Note that we have a $P^{\pi}(a'|s')$ but NOT a $P^{\pi}(a|s)$

- Structure relevant for calculating the Q-function (action-value function)
- Note: at the second stage a policy is applied



- Structure relevant for calculating the state-value function
- We assume a policy $P^{\pi}(a|s)$



Two Stages: Dependencies and Probabilities for Value Setting

- Right: The probability distribution, conditioned on s, is

$$P^{\pi}(r'', r', s'', s', a', a|s) =$$

$$P^{\pi}(a|s)P(s'|s, a)P(r'|s', s, a)$$

$$\times P^{\pi}(a'|s')P(s''|s', a')P(r''|s'', s', a')$$

- Note that we both a $P^{\pi}(a'|s')$ and a $P^{\pi}(a|s)$

Two Stages: Calculating the Q-Function

- Consider we are in state s'; the Q-function concerns only reward r'' and is

  $$q'(s', a') = \mathbb{E}(r''|s', a') = \sum_{s''} \sum_{r''} r'' P(r''|s', s'', a') P(s''|s', a') = \mathcal{R}(s', a')$$

- The expected reward contributed by r'', given initial state s and initial action a is

  $$\mathbb{E}^{\pi}(r''|s, a) = \sum_{r''} \sum_{s''} \sum_{a'} \sum_{s'} r'' P(r''|s', s'', a') P(s''|s', a') P^{\pi}(a'|s') P(s'|s, a)$$

  $$= \sum_{s'} P(s'|s, a) \sum_{a'} P^{\pi}(a'|s')\, q'(s', a')$$

- Note that this is a backpropagation of the Q-of s' to s

(Q-PE) Two Stages: Policy Evaluation using Action-Value Function

- We are interested in the action a that maximizes the sum of the rewards, and also considers expected reward in the future!

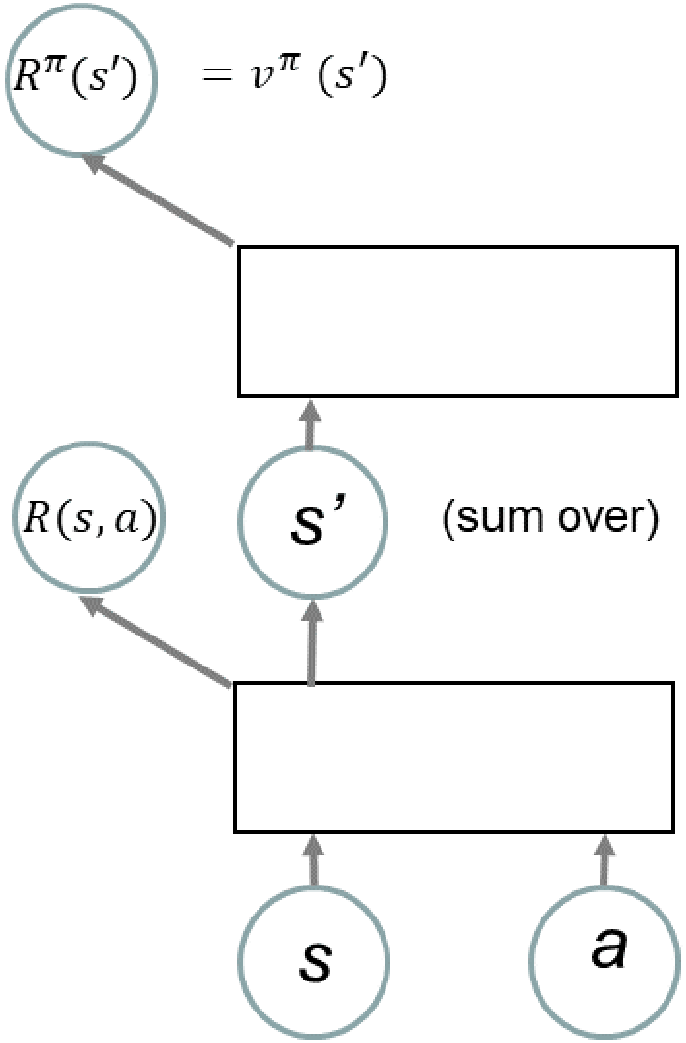
- Q-evaluation 2-step: Finally the **Q-function** at s with contributions from r' and r'' is

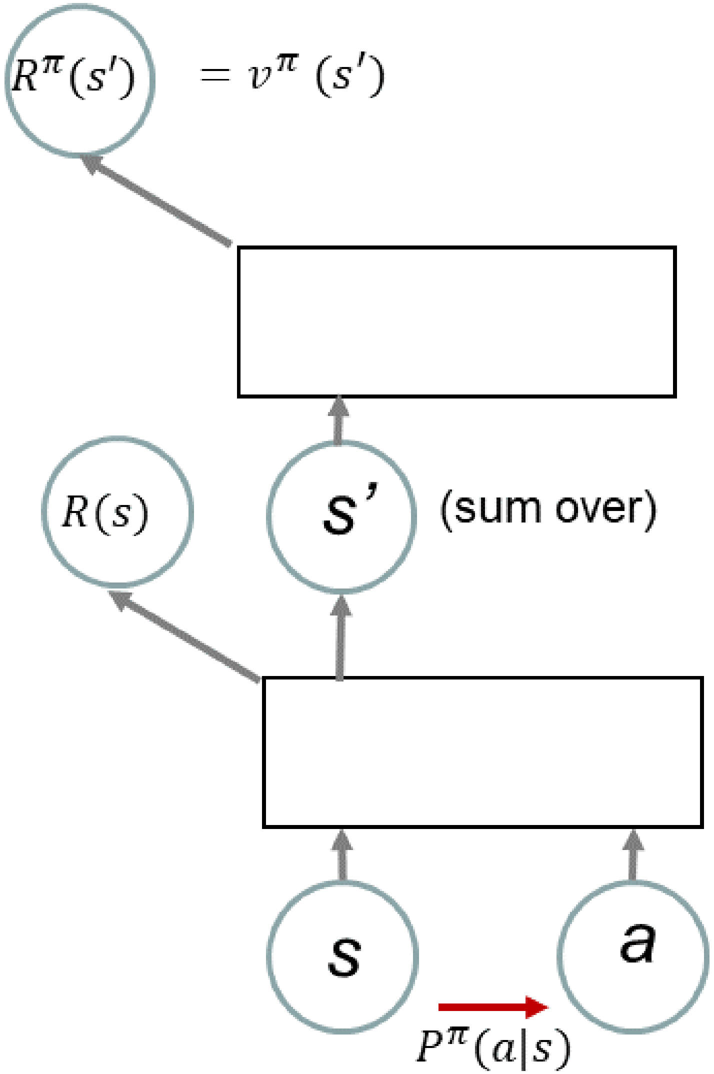
$$q^{\pi}(s,a) = \mathbb{E}^{\pi}(r' + r''|s,a) = \mathbb{E}^{\pi}(r'|s,a) + \mathbb{E}^{\pi}(r''|s,a)$$

$$= \mathcal{R}(s,a) + \sum_{s'} P(s'|s,a) \sum_{a'} P^{\pi}(a'|s') q'(s',a')$$

  This includes the backpropagation of $q'(s',a')$

- Note that a can be freely chosen, but a' is selected according to policy $P^{\pi}(a'|s')$


$R^{\pi}(s')$
$= v^{\pi}(s')$
$R(s,a)$
s'
(sum over)
s
a


$R^{\pi}(s')$
$= v^{\pi}(s')$
$R(s)$
s'
(sum over)
s
a
$P^{\pi}(a|s)$

(PE) Two Stages: Policy Evaluation

- The **value function** at s is

$$v^{\pi}(s) = \sum_{a} P^{\pi}(a|s) q^{\pi}(s,a) = \mathcal{R}^{\pi}(s) + \sum_{s'} P^{\pi}(s'|s)\, v'^{\pi}(s')$$

Here we have used that

$$P^{\pi}(s'|s) = \sum_{a} P^{\pi}(a|s) P(s'|s,a)$$

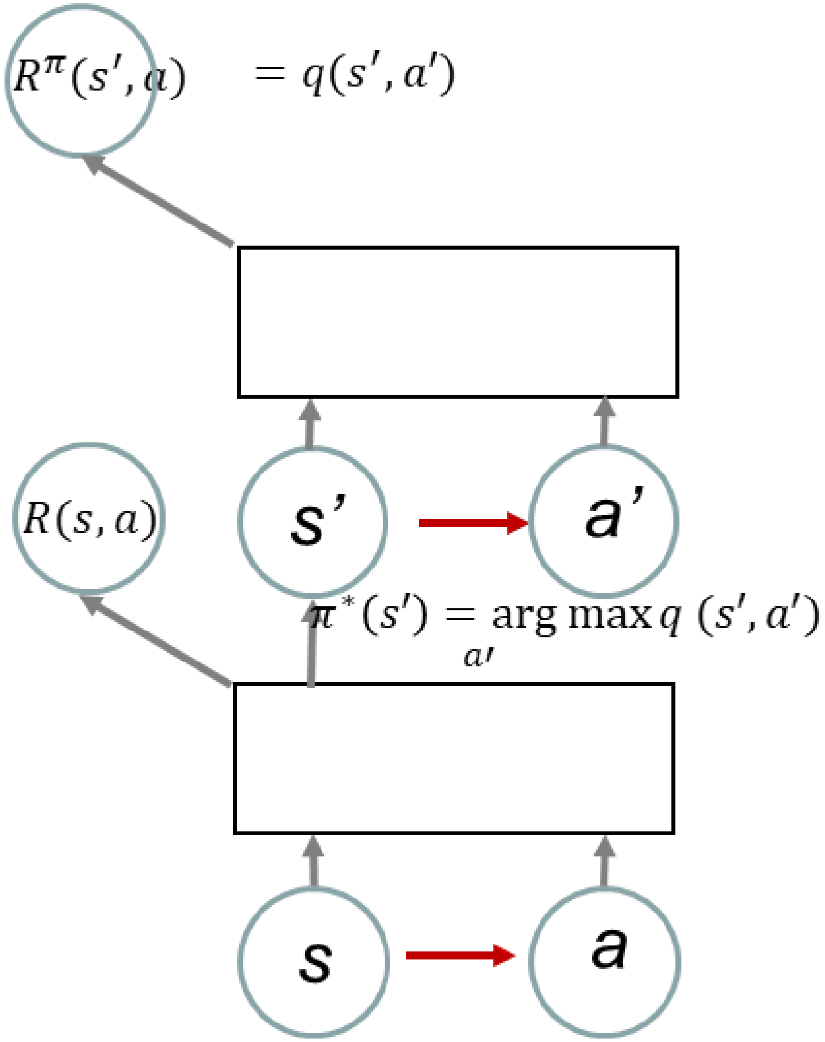
Two Stage: Relationship Between Action-Value Function and Value Function

- We can calculate the value function from the Q-function as

$$v^{\pi}(s) = \sum_{a} P^{\pi}(a|s) q^{\pi}(s,a)$$

- We can calculate the value function from the Q-function as

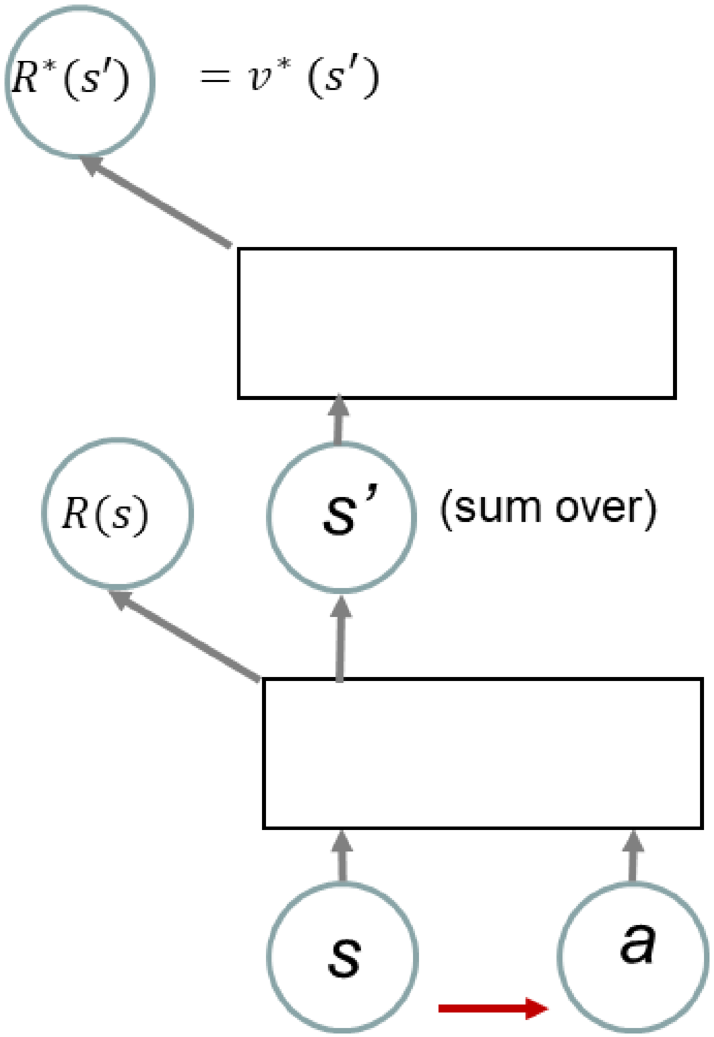
$$q^{\pi}(s,a) = \mathcal{R}(s,a) + \sum_{s'} P(s'|s,a)\; v'^{\pi}(s')$$



$R^{\pi}(s',a) \quad = q(s',a')$

$R(s,a)$

$$\pi^*(s') = \arg\max_{a'} q\ (s',a')$$



$R^*(s') \quad = v^*\ (s')$

$R(s)$

$$\pi^*(s) \leftarrow \arg\max_a \left(\mathcal{R}(s,a) + \sum_{s'} P(s'|s,a)\ v'^*(s') \right)$$

(PI) Two Stages: Policy Iteration to Obtain An Optimal Policy

- The value function of the optimal policy at the second step s' is

$$v'^*(s') = \max_{a'} q(s', a')$$

  Selecting any other action can only decrease the total value

- The optimal policy thus must be,

$$\pi^*(s) \leftarrow \arg\max_{a} \left(\mathcal{R}(s,a) + \sum_{s'} P(s'|s,a)\; v'^*(s') \right)$$

(VI) Two Stages: Value Iteration to Obtain An Optimal Policy

- The value function of the optimal policy at initial step s is

$$v^*(s) = \max_a \left(\mathcal{R}(s,a) + \sum_{s'} P(s'|s,a)\; v'^*(s') \right)$$

(Q-VI) Two Stages: Action-Value Iteration to Obtain An Optimal Policy

- We also get

$$q^*(s,a) = \mathcal{R}(s,a) + \sum_{s'} P(s'|s,a) \max_{a'} q'(s',a')$$

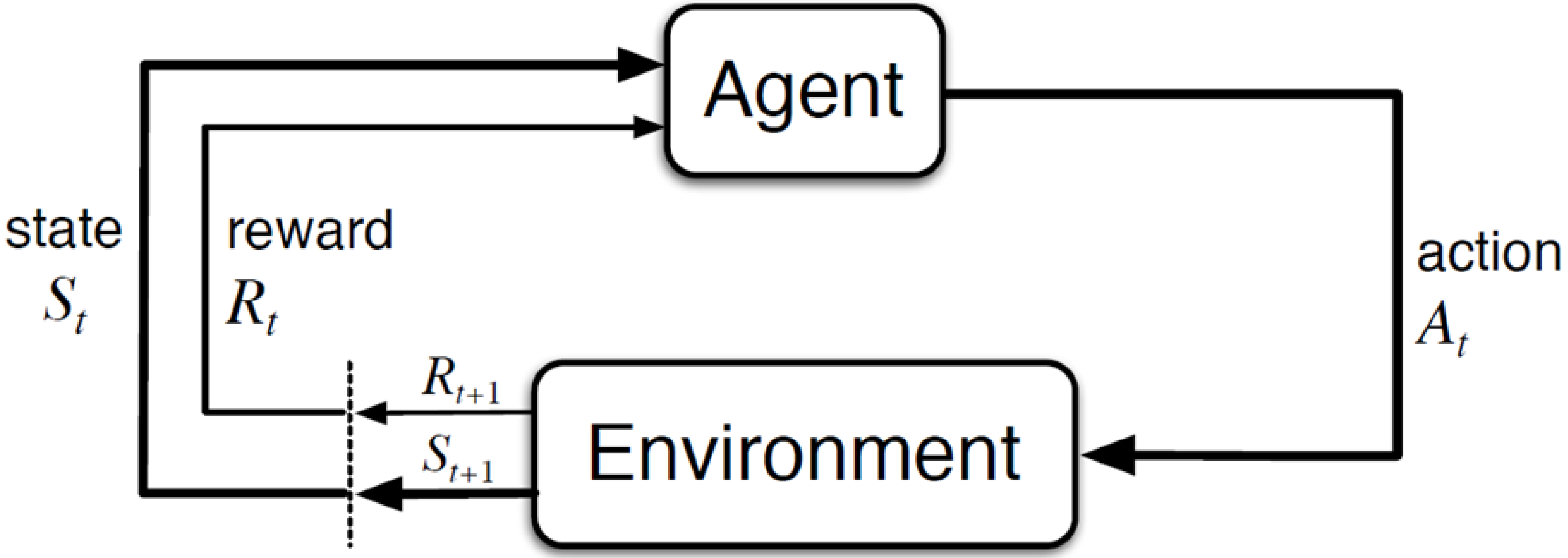
Infinite Horizon

- Note that we essentially obtained a structure suitable for Dynamic Programming; to make a decision *now*, we need information from the *future*!

- We are interested in infinite horizon problems

- We are looking for a time-invariant solution; then the value function and the Q-function are independent of the instance (in particular $v(\cdot) = v'(\cdot)$ and $q(\cdot) = q'(\cdot)$)

- We introduce discount factor γ to reduce the influence of events far in the future $0 \leq \gamma \leq 1$; in other words, the agent's decision now is a bit more optimizing immediate reward than a reward far in the future


Agent
state
S_t
reward
R_t
R_{t+1}
S_{t+1}
Environment
action
A_t

Bellman Equation

- Bellman equation for the value function is

$$v^{\pi}(s) = \mathcal{R}^{\pi}(s) + \gamma \sum_{s'} P^{\pi}(s'|s) v^{\pi}(s')$$

- In vector notation

$$\mathbf{v}^{\pi} = \mathbf{r}^{\pi} + \gamma P^{\pi} \mathbf{v}^{\pi}$$

- The leads to the explicit solution

$$\mathbf{v}^{\pi} = (I - \gamma P^{\pi})^{-1} \mathbf{r}^{\pi}$$

- Direct inversion is almost never used; computational complexity scales as $O(|S|^3)$; consider that $|S|$ can be quite large!

Bellman Equation (cont'd)

- To some degree, the remaining part of the lecture is about methods which calculate the value function and the q-function more efficiently:
  - Iterative solutions to the Bellman equation (in the remaining of Part I)
  - Replacing the model probabilities by simulation (Part II)
  - Approximating the value function, the q-function and the policy by function approximators (e.g., neural networks) (Part III)

(PE) Policy Evaluation

- As an immediate generalization of the 2-stage case, we get with fixed policy π

$$v^\pi(s) \leftarrow \mathcal{R}^\pi(s) + \gamma \sum_{s'} P^\pi(s'|s)\, v^\pi(s')$$

- Note that this is an update equation: pick any s and update $v^\pi(s)$ using this equation! One should be smart about with s to pick in which order!
- In the Appendix we provide some background on policy evaluation

(Q-PE) Policy Evaluation using Action-Value Function

- As a generalization to the 2-stage case, we can update the Q-function

$$q^\pi(s,a) \leftarrow \mathcal{R}(s,a) + \sum_{s'} P(s'|s,a) \sum_{a'} P^\pi(a'|s')\, q^\pi(s',a')$$

- As a reminder: we continue to assume that the system $P(s'|s,a))$ is known and does not need to be learned!

- Note that this is an update equation: pick any s, a and update $q^\pi(s,a)$ using this equation! One should be smart about with s, a to pick in which order!

Relationship Between Action-Value Function and Value Function

- We can calculate the value function from the Q-function as

$$v^{\pi}(s) = \sum_{a} P^{\pi}(a|s) q^{\pi}(s, a)$$

- We can calculate the value function from the Q-function as

$$q^{\pi}(s, a) = \mathcal{R}(s, a) + \sum_{s'} P(s'|s, a)\, v^{\pi}(s')$$

(PI) Policy Iteration to Obtain An Optimal Policy

- Policy iteration
  - We calculate $v^{\pi}(s)$ with policy evaluation (as discussed) until some form of convergence is reached
  - Then we calculate a new deterministic policy as

$$\pi_{new}(s) \leftarrow \arg\max_a \left(\mathcal{R}(s,a) + \gamma \sum_{s'} P(s'|s,a)\, v^{\pi}(s') \right)$$

- The two steps are repeated until convergence
- Note that in the second step, we only go through all states s once

(Q-PI) Q-Policy Iteration to Obtain An Optimal Policy

- Policy iteration
  - We calculate $q^{\pi}(s,a)$ with Q-PE(as discussed) until some form of convergence is reached
  - Then we calculate a new deterministic policy as

$$\pi_{new}(s) \leftarrow \arg\max_a \left(q^{\pi}(s,a)\right)$$

- The two steps are repeated until convergence
- Note that in the second step, we only go through all states s once

(VI) Value Iteration to Obtain An Optimal Policy

- In generalization of the two-stage case, we get

$$v(s) \leftarrow \max_a \left(\mathcal{R}(s,a) + \gamma \sum_{s'} P(s'|s,a)v(s') \right)$$

- No policy needs to be stored; convergence can be faster than in (PI)

(Q-VI): Action-Value Iteration to Obtain An Optimal Policy

- We also get

$$q(s,a) \leftarrow \mathcal{R}(s,a) + \sum_{s'} P(s'|s,a) \max_{a'} q(s',a')$$

- No policy needs to be stored; convergence can be faster than in (Q-PI)

Optimal Policy

- The optimal policy is deterministic, with

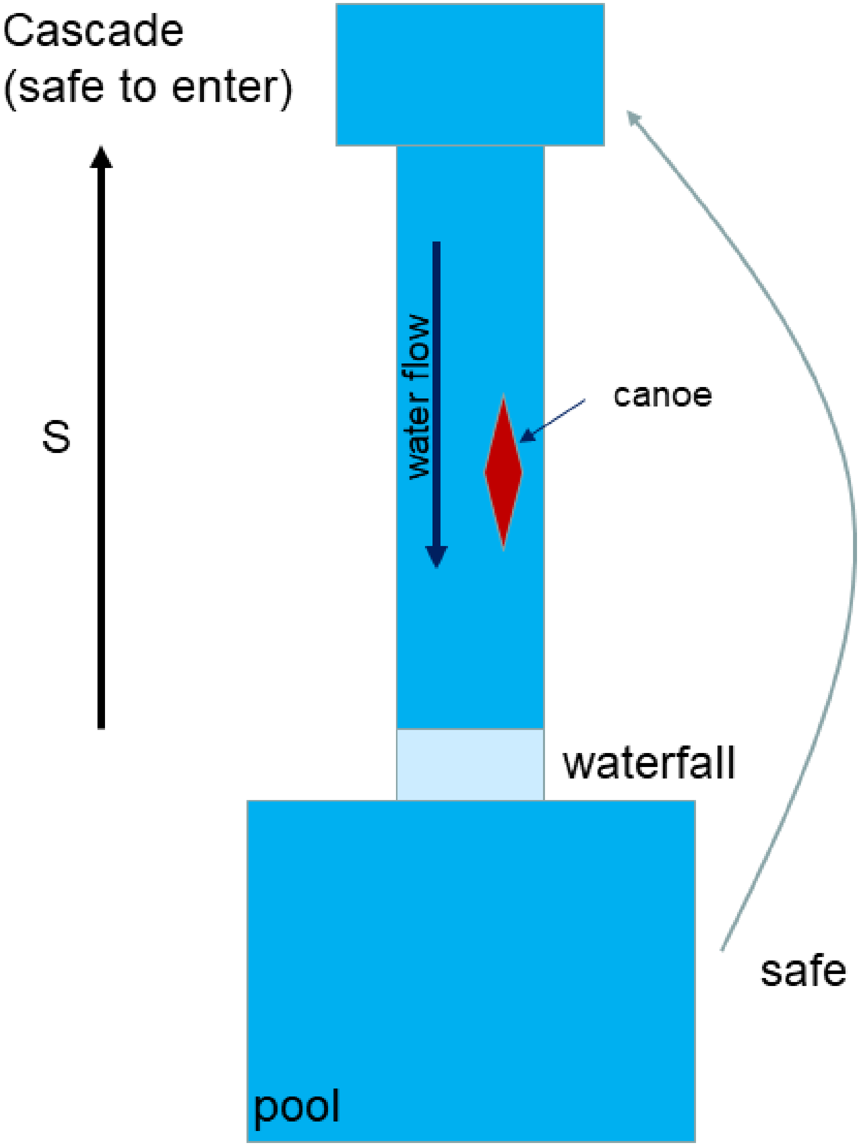
$$\pi^*(s) \leftarrow \arg\max_a q^*(s,a) = \arg\max_a \left(\mathcal{R}(s,a) + \gamma \sum_{s'} P(s'|s,a) v^*(s') \right)$$

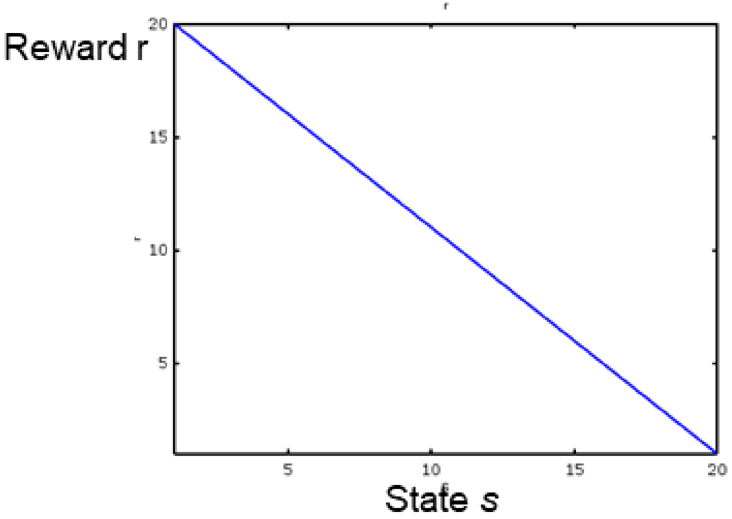
A Wet Game of Chicken (Tresp, 1994)

- The agent is a paddler in a canoe
- The agent can just float in which case in **drifts** towards the waterfall ($a = -1$); the agent can try to **balance** the drift ($a = 0$); the agent can **paddle** upstream ($a = 1$)
- The agent get's a larger reward closer to the waterfall
- If the agent's goes down the waterfall it has to get up to the cascade and can again enter the river
- We have

$$s_{t+1} = s_t + round(a_t + \epsilon_t)$$

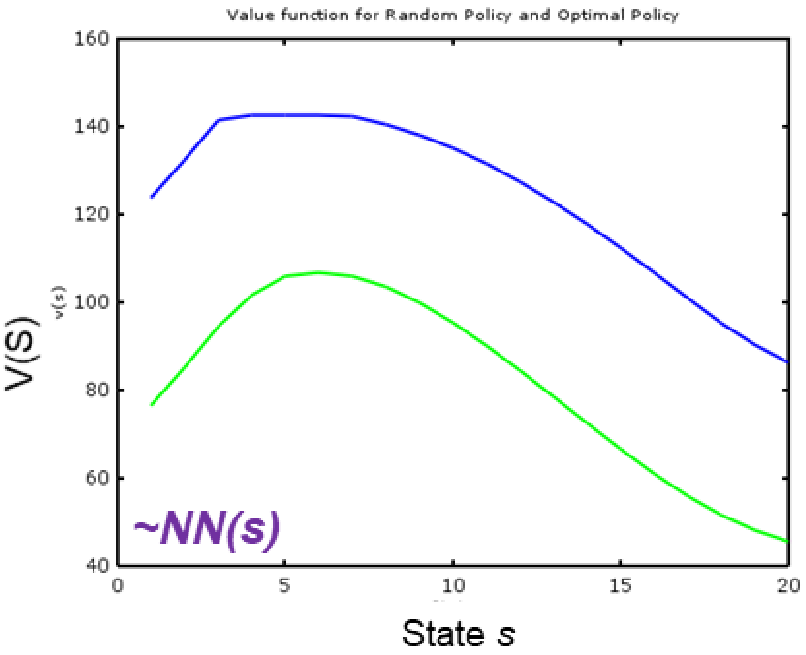
  If $s_t \leq 1$, then $s_t = 20$; ϵ is independent Gaussian noise

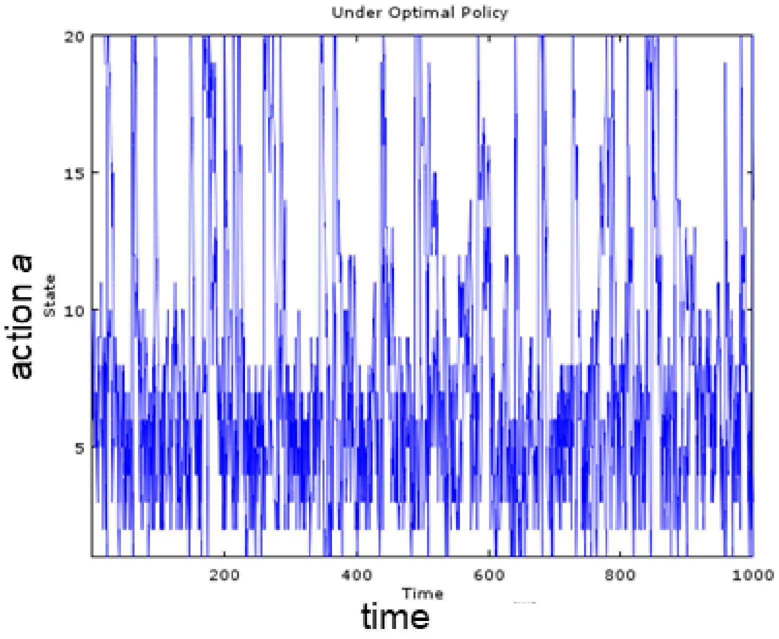

Cascade
(safe to enter)
S
water flow
canoe
waterfall
pool
safe to leave

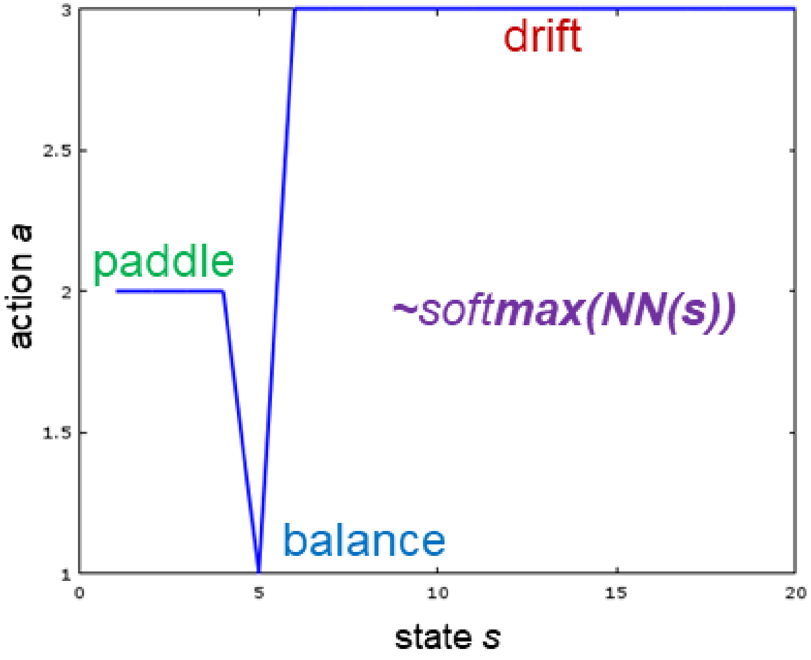

Reward r
20
15
10
5
5
10
15
20
State s

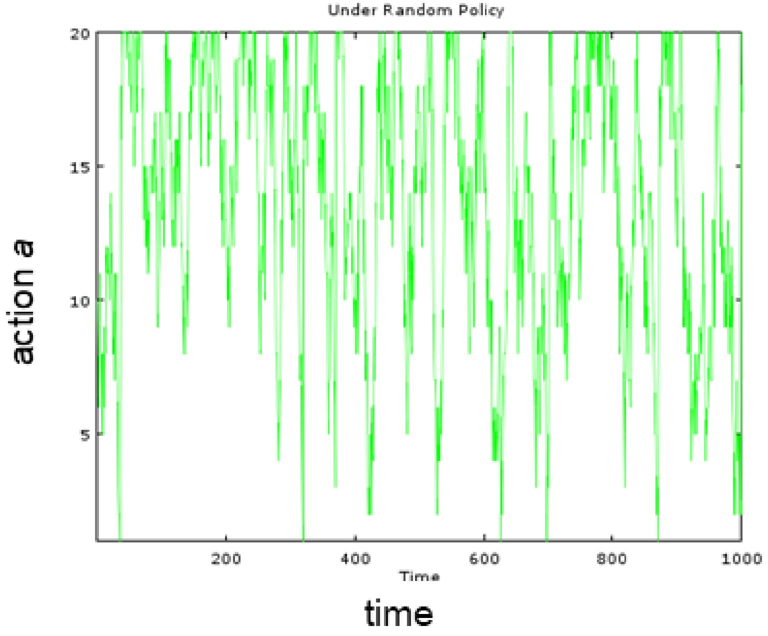
A Wet Game of Chicken (cont'd)

- We see the optimal policy: drift if the agent is far away from the waterfall, paddle, if $s < 5$ and try to balance when $s = 5$
- If compared to a random policy, the value function of the optimal policy is much larger and the agent spends more time close to the waterfall to collect more reward


Value function for Random Policy and Optimal Policy
V(S)
v(s)
160
140
120
100
80
60
40
0
5
10
15
20
State *s*
~NN(s)

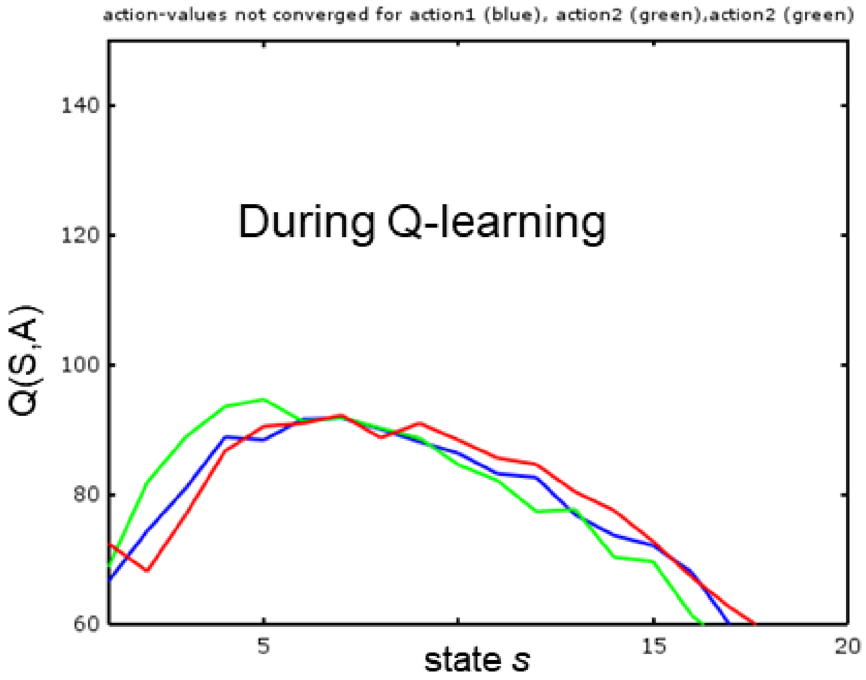

Under Optimal Policy
action *a*
State
20
15
10
5
200
400
600
800
1000
Time
time

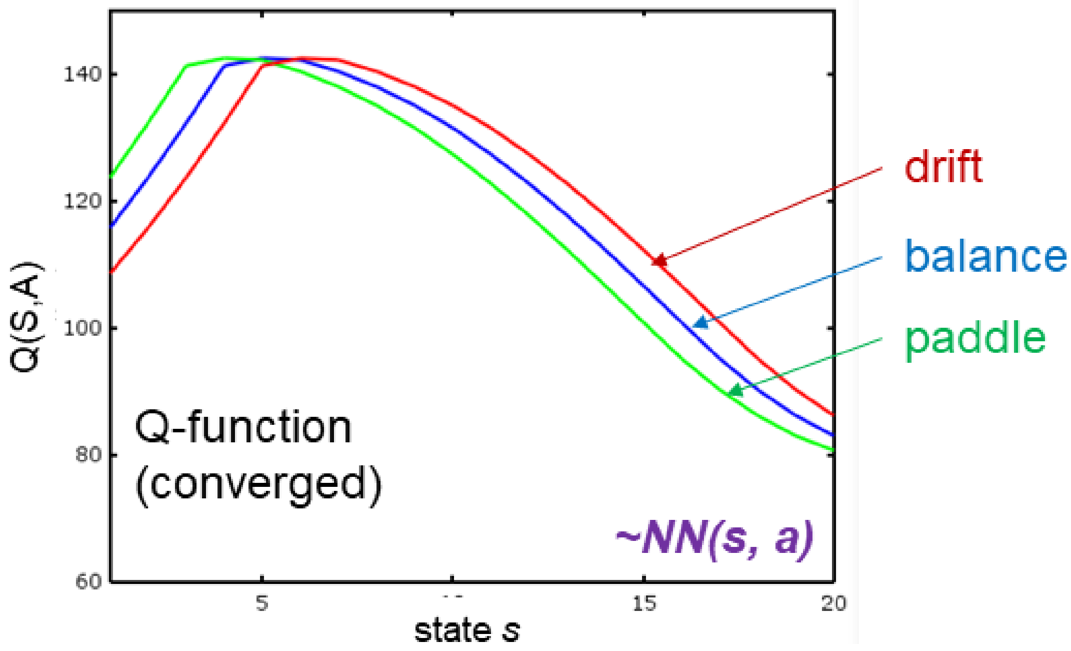

3
2.5
2
1.5
1
0
5
10
15
20
action *a*
state *s*
paddle
balance
drift
~softmax(NN(s))


Under Random Policy
action *a*
20
15
10
5
200
400
600
800
1000
Time
time

A Wet Game of Chicken (cont'd)

- The Q-function during iterations and after convergence
- Paddling leads to a higher value when close to the waterfall


action-values not converged for action1 (blue), action2 (green),action2 (green)
During Q-learning
Q(S,A)
140
120
100
80
60
5
15
20
state *s*


Q(S,A)
140
120
100
80
60
5
15
20
state *s*
drift
balance
paddle
Q-function
(converged)
~NN(s, a)

Summary

- Evaluation/prediction
  - (PE) Policy evaluation with value function or Q-function
- Optimal control
  - (PI) Policy iteration with value function or Q-function
  - (VI) Value iteration with value function or Q-function

Discussion

- So far, we covered S, lectures 1-3
- The discusses approaches were **model based**: we assumed that the model $P(s'|s,a)$ and $P(r'|s,s',a)$ are known
- Classically the algorithms were derived in the field of optimal control and **dynamic programming**; Silver calls this also **planning**
- It might be unusual to consider an infinite future; consider a finite board game; after the end of the game, formally the states simply do not change anymore

Discussion (cont'd)

- For these algorithms to work, $v(s)$ or $q(s, a)$ must be well estimated for all relevant states s; there are different strategies:
- Which states are updated in which order? Here algorithms differ: there are **synchronous** updates (iterate through all s) or **asynchronous** updates; a special case is **prioritized sweeping** which prioritizes states with large error; other approaches are **real-time dynamic programming** and full width backup
- The iterations are related to the Jacobi method and the Gauss–Seidel algorithm for solving systems of linear equations (see Appendix)
- In **approximate dynamic programming** the value function is approximated by a function approximator (more on this later)
- The iterative algorithms discussed so far used **bootstrapping:** using the current value function (resp., Q-function) estimate instead of the true ones on the right side of the updates

- We now discuss model-free approaches where the models $P(s'|s,a)$ and $P(r'|s,s',a)$ are unknown; we start with model-free evaluation (prediction) and then discuss model-free control