Artificial Neural Networks: Lecture 2 (RL1)

Reinforcement Learning and SARSA

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Objectives for today:

- Reinforcement Learning (RL) is learning by rewards
- Agents and actions
- Exploration vs Exploitation
- Bellman equation
- SARSA algorithm

Reading for this week:

Sutton and Barto, Reinforcement Learning (MIT Press, 2nd edition 2018, also online)

Chapters: 1.1-1.4; 2.1-2.6; 3.1-3.5; 6.4

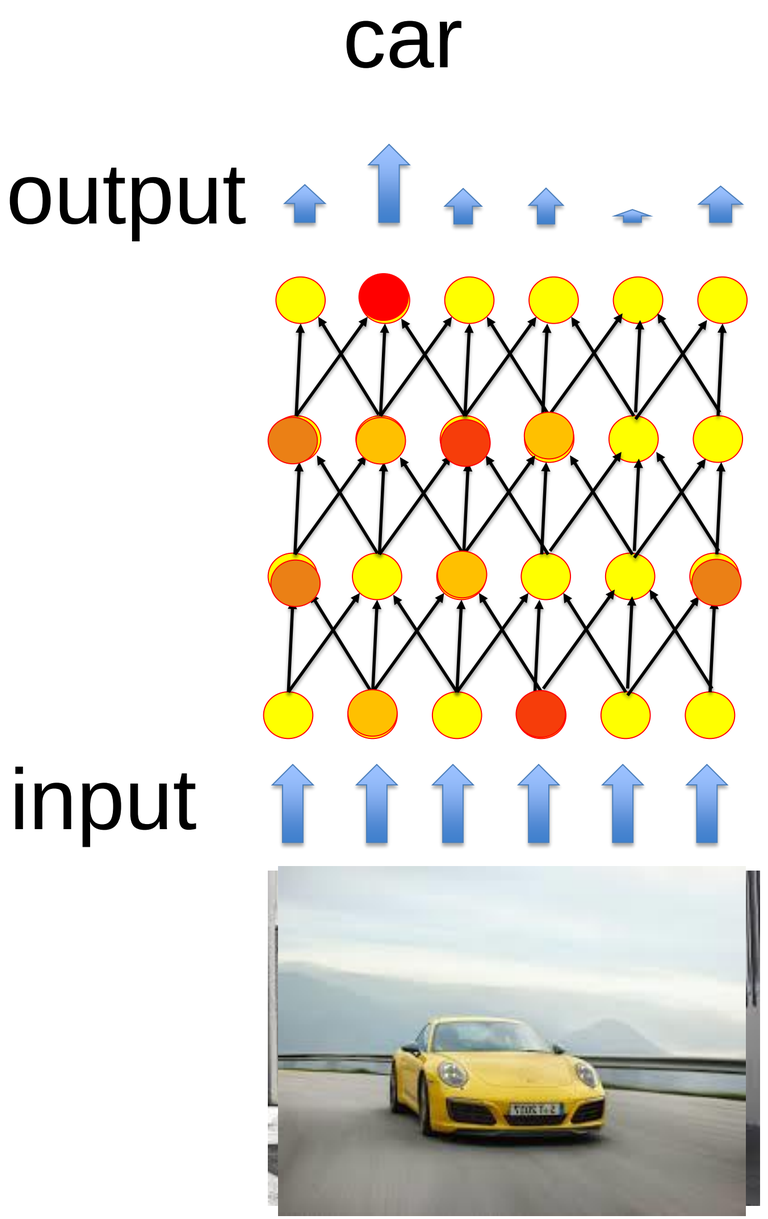
Background reading:

Silver et al. 2017, Archive

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

Review: Artificial Neural Networks for classification

feedforward network



review: Artificial Neural Networks for classification

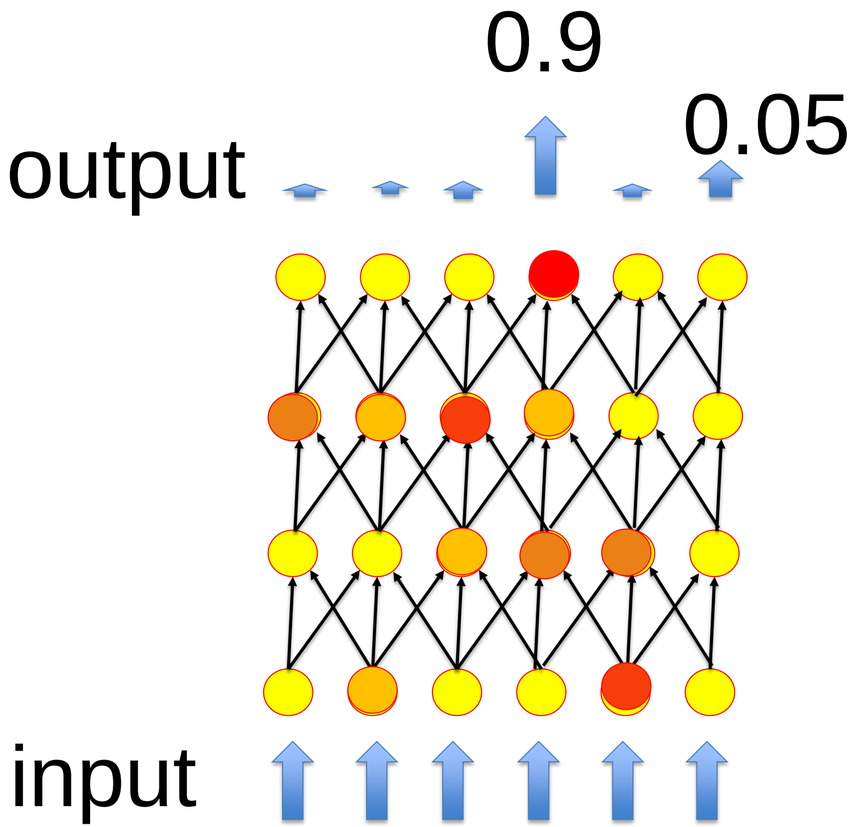
Prerequisite for learning:
labeled data base

$$\{ \quad (\boldsymbol{x}^{\mu}, \boldsymbol{t}^{\mu}) \quad , \qquad 1 \leq \mu \leq P \quad \};$$

Aim of learning:
Adjust connections such that output $\boldsymbol{y}^{\mu}$ is correct

$$\boldsymbol{y}^{\mu} = \boldsymbol{t}^{\mu}$$

(for each static input image, $\boldsymbol{x}^{\mu}$)



review: Artificial Neural Networks for classification

Prerequisite for learning:

labeled data base

$$\{ \quad (\boldsymbol{x}^{\mu}, \boldsymbol{t}^{\mu}) \quad , \qquad 1 \leq \mu \leq P \quad \};$$

Question: Is this realistic?

Previous slide.

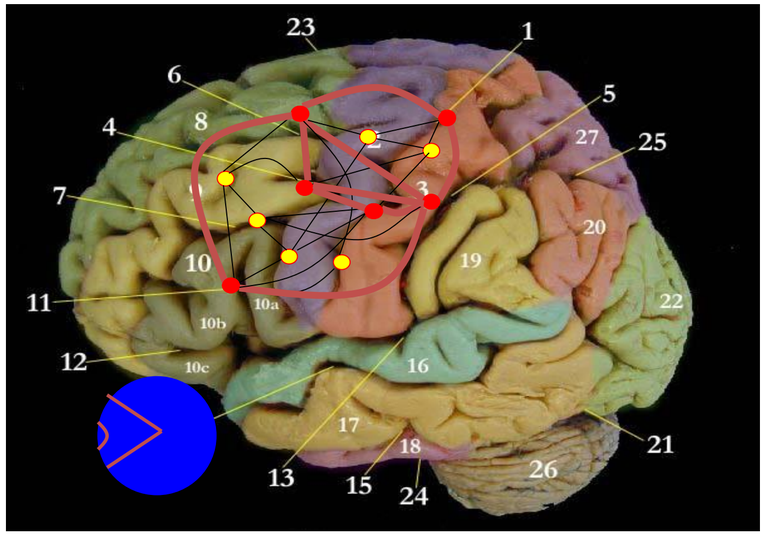
In the previous lecture (on Perceptron Learning) we started with a data base consisting of a large number of patterns, each one with its label.

But a question remains: Is this realistic? Do we ‘normally’ have a training base with labeled data? Where should this come from?

A first answer is no: there are lots of examples of sequences in the world around us. If the aim is to predict the next step of the sequence, then we have lots of labeled data (because we just have to compare the prediction of the network with what actually happens in the next step).
In this case, we can still use the framework of ‘supervised learning’ where the next step (e.g., next frame of video) is the supervisor (=label). We will come to sequences at the very end of the semester.

In the following, however, we focus on a completely different scenario. The paradigm of reinforcement learning.

1. Artificial Neural Networks for action learning



Where is the supervisor?
Where is the labeled data?

Replaced by:
'Value of action'
- 'goodie' for dog
- 'success'
- 'compliment'

BUT:
Reward is rare:
'sparse feedback' after a long action sequence

Previous slide.

How does a human learn to play table tennis: How does a child learn to play the piano? How does a dog learn to perform tricks?

In all these cases there is no supervisor. No master guides the hand of the players during the learning phase. Rather the player ‘discovers’ good movements by rather coarse feedback. For example,
the ball in table tennis does not land on the table as it should. That is bad (negative feedback). The ball has a great spin so that the opponent does not get. This is good (positive feedback).

Similarly, it is hard to tell a dog what to do. But if you reinforce the dog’s behavior by giving a ‘goodie’ at the moment when it spontaneously performs a nice action, then it can learn quite amazing things.

In all these cases it is the ‘reward’ that guides the learning. Rewards can be the goodie for the dog, or just the feeling ‘now I did well’ for humans.

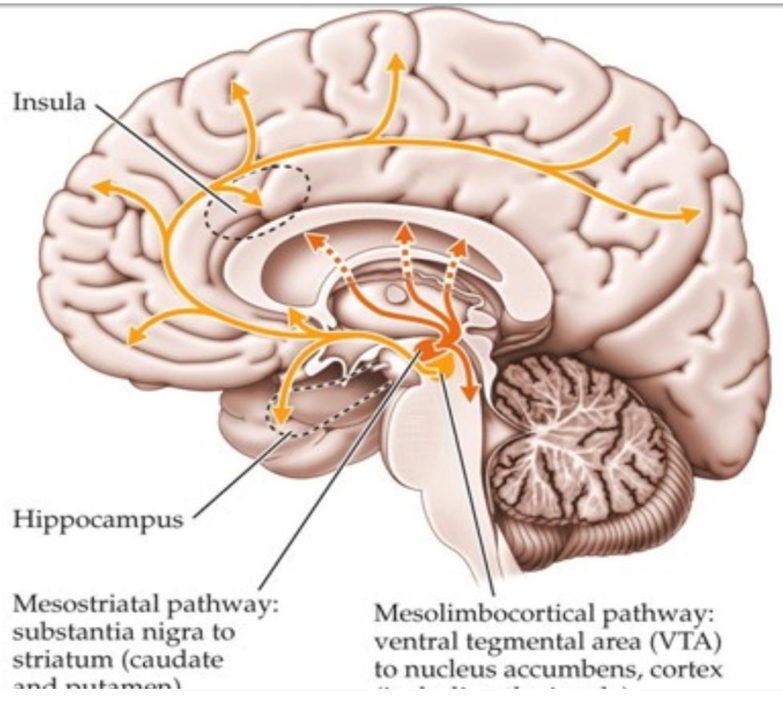
Reward information is available in the brain

Neuromodulator **dopamine**:
Signals reward minus
expected reward

Schultz et al., 1997,
Waelti et al., 2001
Schultz, 2002

‘success signal’

Dopamine

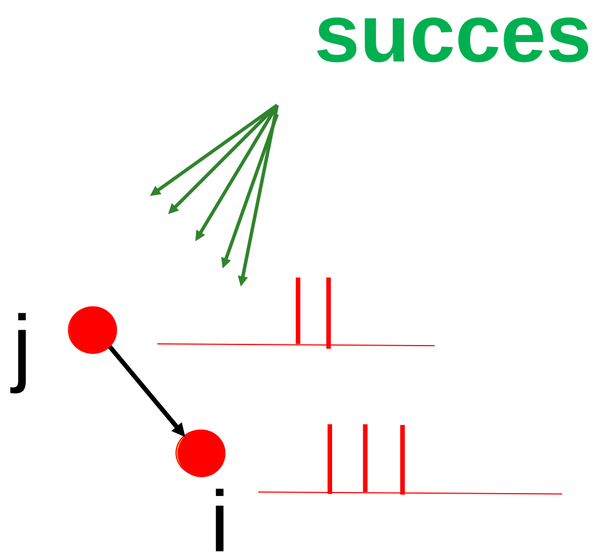


Previous slide.

Inside the brain, reward information is transmitted by the neuromodulator dopamine. Neurons that use dopamine as their chemical transission signal are situated in nuclei below the cortex and have cables (axons) that reach out to vast areas of the brain.

As we will see later, neurons that communicate with the neuromodulator dopamine transmit a generic success signal that is not just reward, but something like ‘reward minus expected reward’.

Review: Modeling – the role of reward



Three factors for changing a connection
- activity of neuron j
- activity of neuron i
- success

Barto 1985, Schultz et al. 1997; Waelti et al., 2001;
Reynolds and Wickens 2002;
Lisman et al. 2011

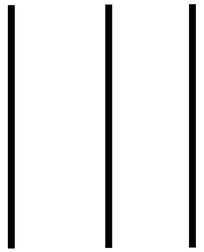
Reinforcement learning = learning based on reward

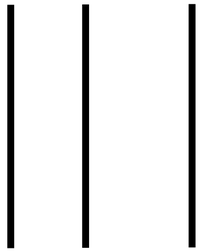
Previous slide (not shown in 2020)

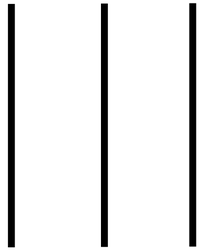
Connections in the brain are changed if three factors come together:
-activity of the sending neuron j
-some form of activity of the receiving neuron i
-and the success signal (such as the neuromodulator dopamine)

1. Examples of reinforcment learning

Middle bar: shifted left or shifted right?

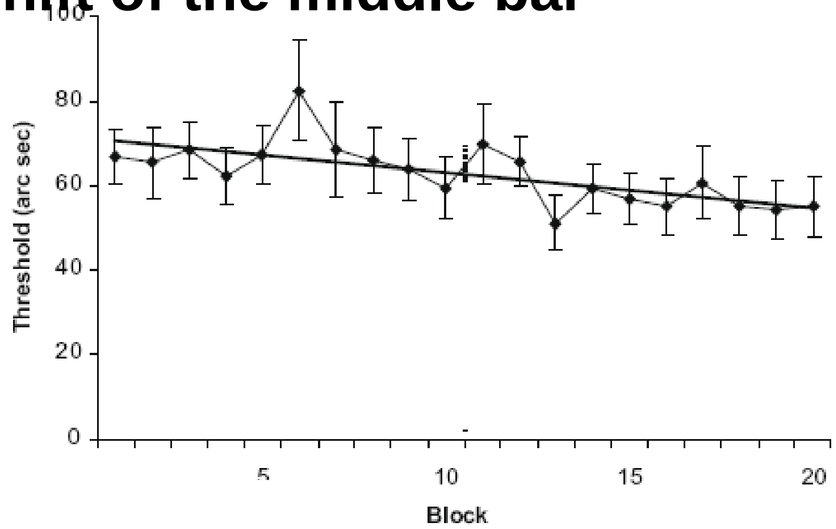






Observers get better at seeing the shift of the middle bar

Min. shift



Feedback:
tone for wrong response

Tartaglia,Aberg,Herzog 2009

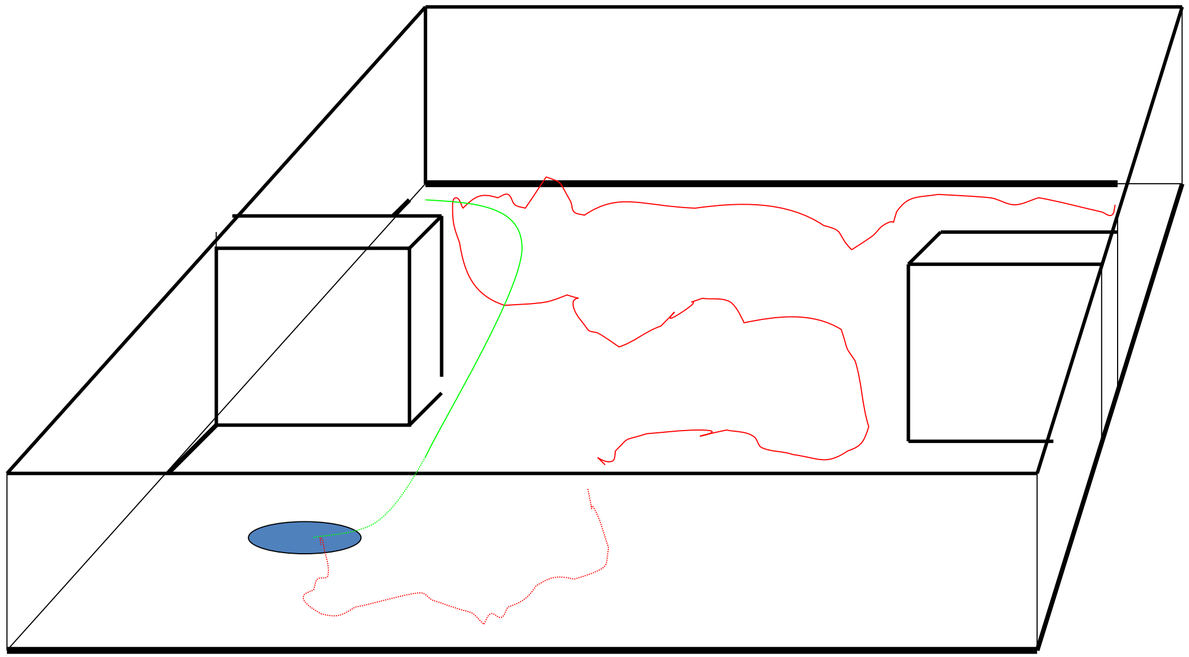
Previous slide (not shown in 2020)

Let us look at a few additional examples, beyond table tennis.

Humans can get, by practice and feedback, better at recognizing a visual pattern with three bars. The task is to distinguish cases where the middle bar is shifted to the left from those where it is shifted to the right.

Bottom right:
The minimal shift that is just recognizable decreases over time (1 block = 1 practice session) indicating learning.

1. Examples of reinforcement learning: animal conditioning

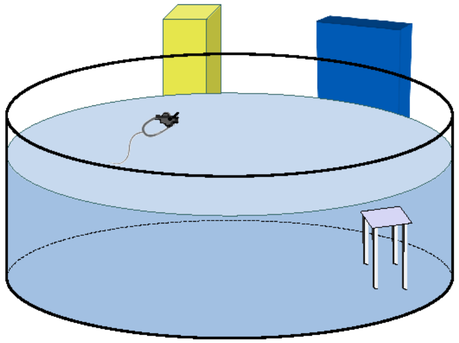


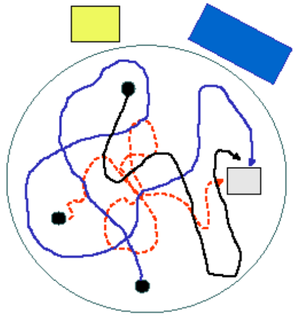
Previous slide.

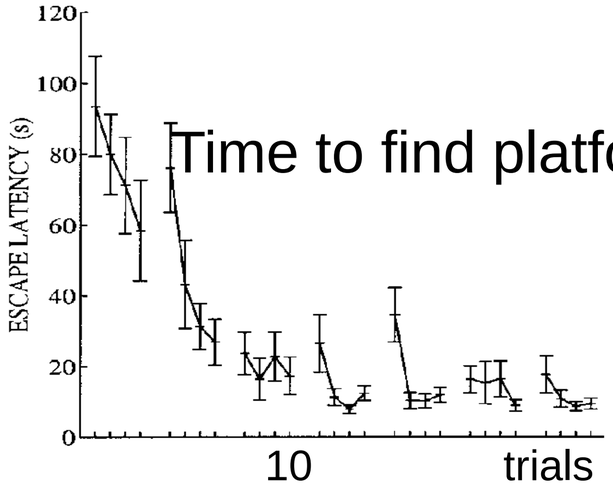
If you put a rat into an environment it will wander around. Suppose that, at some place, it discovers a food source hidden below the sand of the surface.
After a couple of trials it will go straight to the location of the food source which implies that it has learned the appropriate sequence of actions in the environment to find the food source.

1. Examples of reinforcement learning: animal conditioning

Morris Water Maze







Time to find platform

Rats learn to find
the hidden platform

(Because they like to
get out of the cold water)

Foster, Morris, Dayan 2000

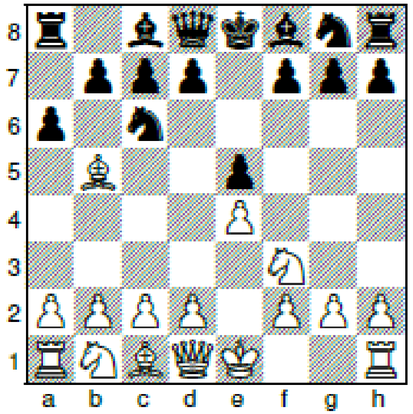
Previous slide.

Actual experiments for location learning are often performed in a Morris water maze. In the maze, there are 4 starting points and one target location which is a platform hidden (in milky water) just below the water surface. The rat does not like to swim in cold water and therefore tries to find the platform.

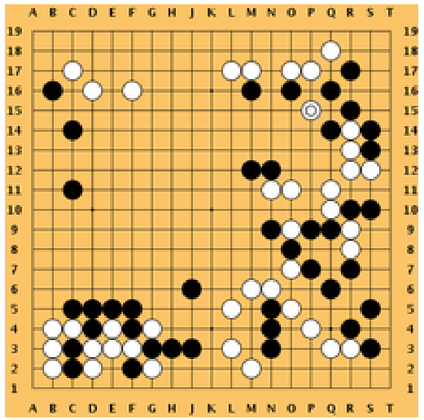
After a few trials it swims straight to the platform.
Bottom right: the time to reach the platform decreases over trials, indicating learning.

1. Deep reinforcement learning

Chess



Go



Artificial neural network (*AlphaZero*) discovers different strategies by playing against itself.

In Go, it beats Lee Sedol

Previous slide.

In chess a neural network trained by reinforcement learning discovers winning strategies by playing against itself. Similarly, a neural network playing Go against itself learns to play at a level so as to beat one of the world champions.

The aim of the class is to arrive at Deep Reinforcement Learning (Deep RL): Today we start with (standard) RL, in a few weeks we turn to deep networks, and in May we will turn to Deep RL.

1. Deep reinforcement learning

Network for choosing action

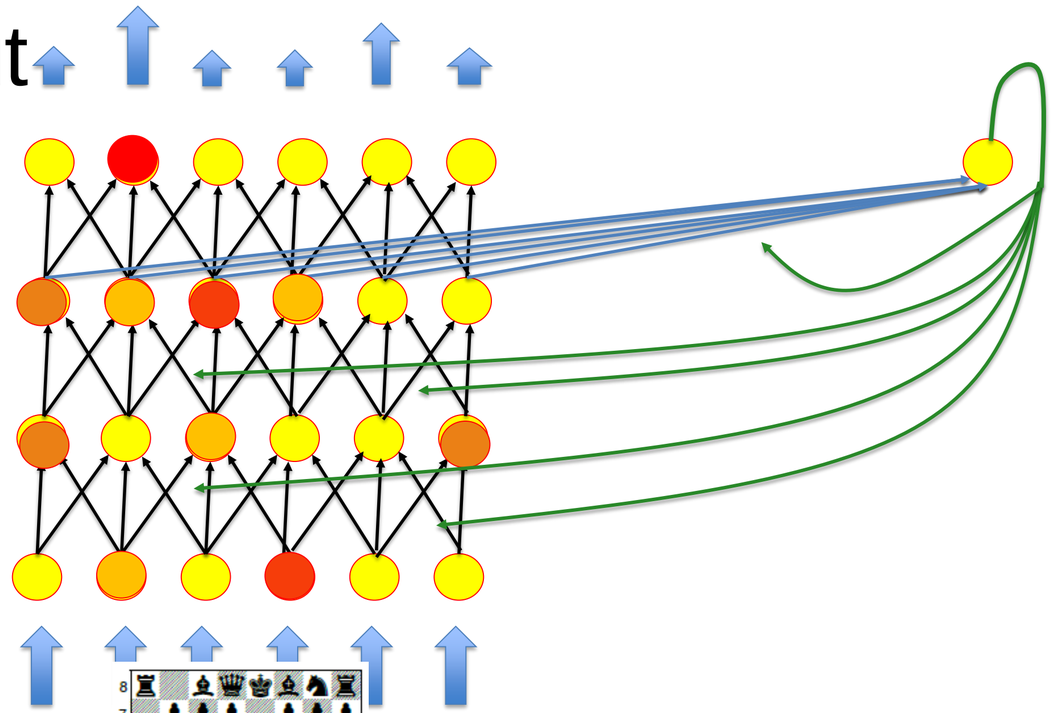
action:

Advance king

2e output for **value** of state:

probability to win

output



input

learning:

- change connections

aim:

- Choose next action to win

aim for value unit:

- Predict value of current position

Previous slide.

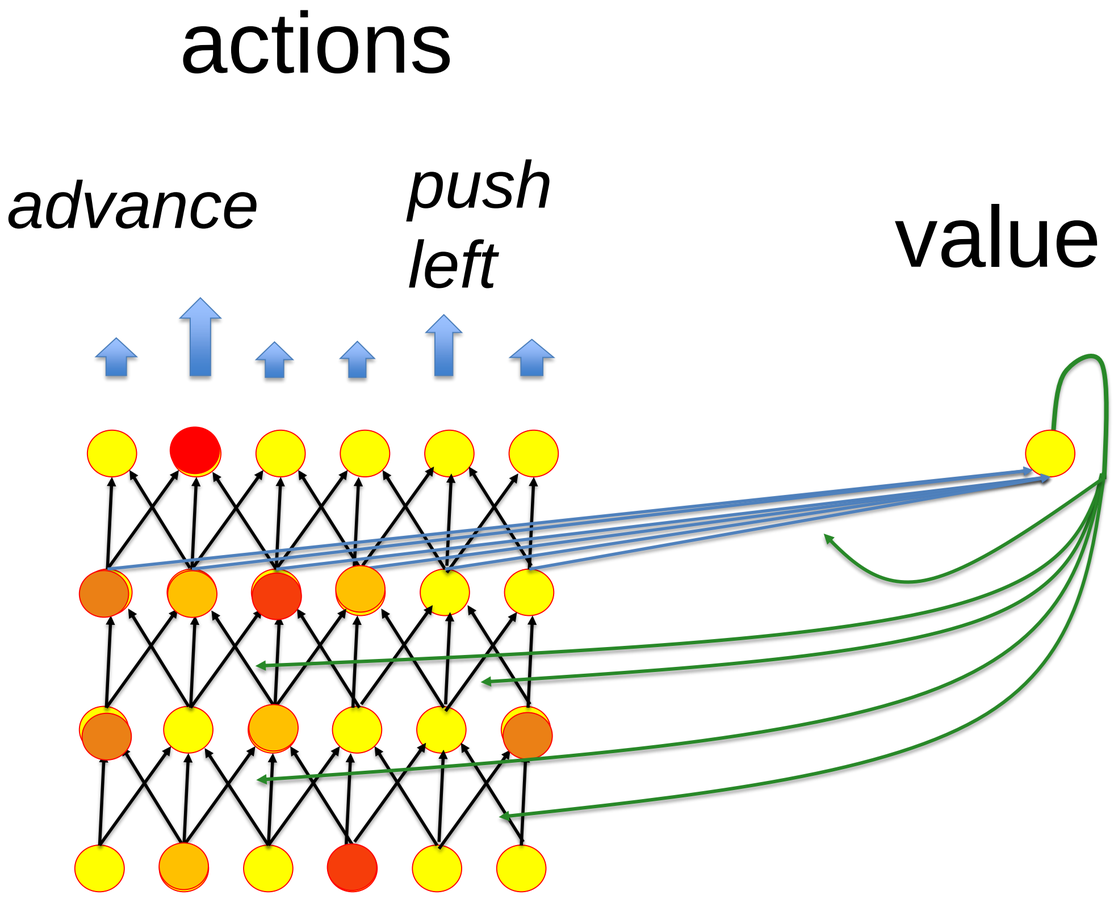
At the end of this semester, you will be able to understand the algorithms and network structure used to achieve these astonishing performances. Important are two types of outputs.

Left: different output neurons represent different actions.
Right: an additional output neuron represents the value of the present state; we can loosely define the value as the probability to win.

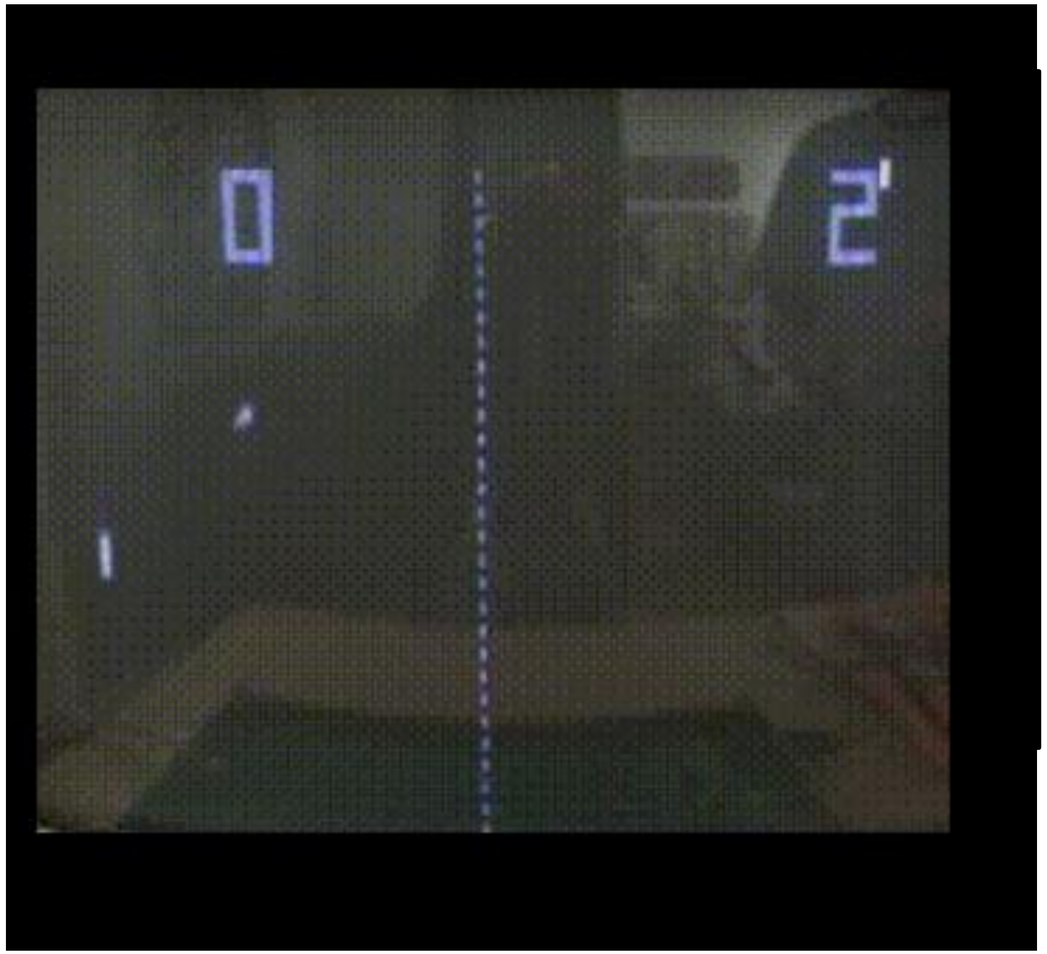
The input is a representation of the present state of the game.

Details will become clear toward the end of the semester; at the moment the aim is just to give you a flavor of the high-level concepts.

1. Deep Reinforcement Learning: games



Aim: Play Pong (Atari game)



Previous slide.

In the miniproject on RL, you will train a game such as a moon-lander to land between the two flag poles, or a car to stay on the street, or a tennis racket to hit the ball (pong). Training will be based on reward: successful behavior will give positive rewards.

Quiz: Rewards in Reinforcement Learning

[ ] Reinforcement learning is based on rewards
[ ] Reinforcement learning aims at optimal action choices
[ ] In chess, the player gets an external reward after every move
[ ] In table tennis, the player gets a reward when he makes a point
[ ] A dog can learn to do tricks if you give it rewards at appropriate moments

Previous slide.

Artificial Neural Networks: RL1

Reinforcement Learning and SARSA

1. Learning by Reward: Reinforcement Learning
2. Elements of Reinforcement Learning

Previous slide.
We now start with the formalization of reinforcement learning

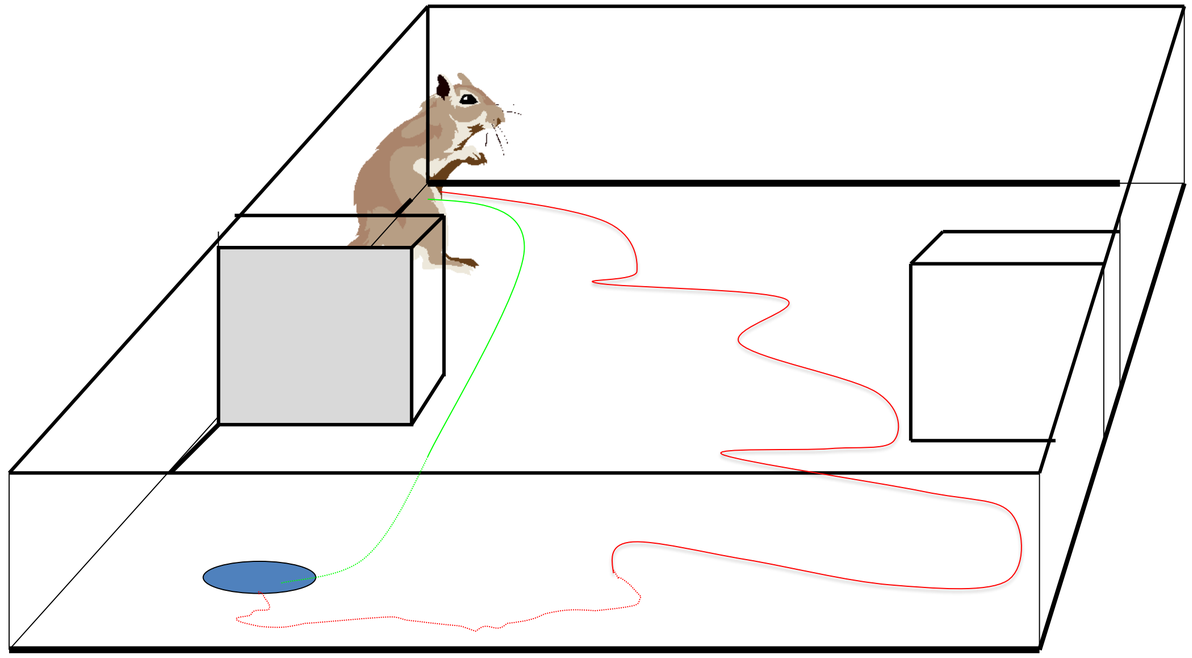
2. Elements of Reinforcement Learning:

-states
-actions
-rewards

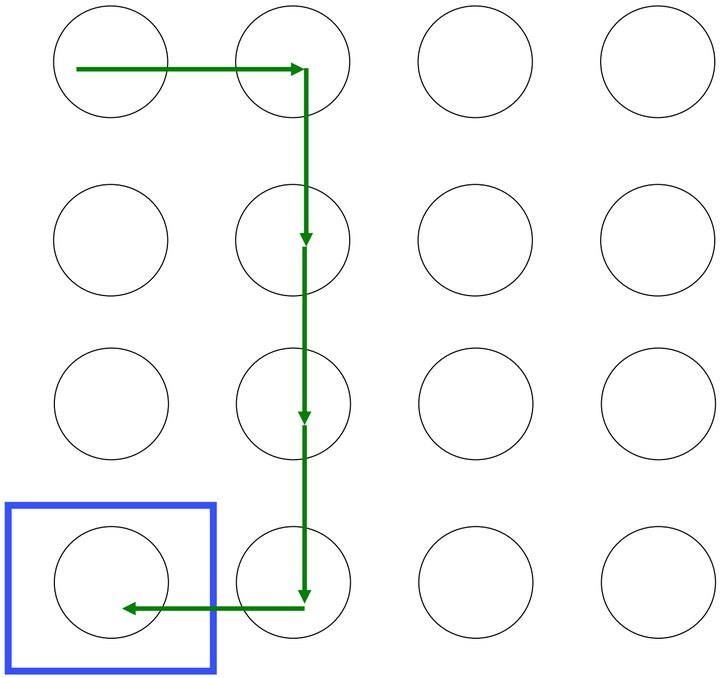
Previous slide.

Reinforcement learning needs states, actions, and rewards.

2. Elements of Reinforcement Learning:



- discrete states
- discrete actions
- sparse rewards



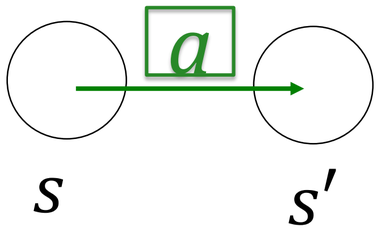
Previous slide.

Note that, for standard formulations of Reinforcement Learning Theories this (normally) implies discretizing space and actions.

We will study continuous-space formulations only next week.

2. Elements of Reinforcement Learning:

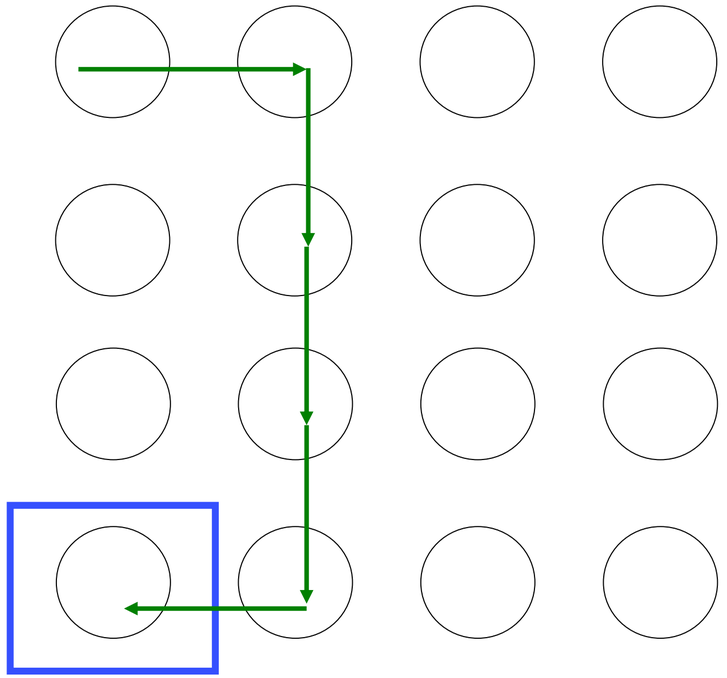
- discrete states:
  - old state s
  - new state s'



- current state: s_t
- discrete actions: a
- Mean rewards for transitions:

$$R^{a}_{s \to s'}$$

- current reward: r_t



often most transitions have zero reward

Previous slide.

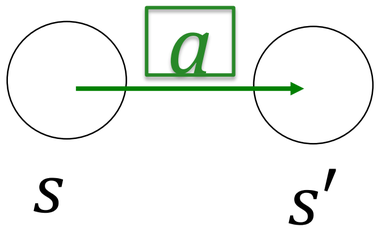
The elementary step is:
The agent starts in state s.
It takes action a
It arrives in a new state s’
Potentially receiving reward r (during the transition or upon arrival at s’).

Since rewards are stochastic we have to distinguish the mean reward at the transition (capital R with indices identifying the transition) from the actual reward (lower-case r with index t) that is received at time t on a transition.

Note that in many practical situations most transitions or states have zero rewards, except a single ‘goal’ state at the end.

2. States Reinforcement Learning:

- discrete states:
  - old state s
  - new state s'
- current state: s_t



state = current configuration/well-defined situation
= generalized ‘location’ of actor in environment

Previous slide.

What are these discrete states?
Loosely speaking a state is the current configuration that **uniquely** describes the momentary situation. We can think of the generalized ‘location’ of the actor in the environment

To get acquainted with this, let us look at an example.

2. Reinforcement Learning: Example Acrobot

3 actions: $a1$ = no torque,

$a2$ = torque +1 at elbow,

$a3$ = torque -1 at elbow

States?

reward if tip above line

→ discretize!

Suppose 5 states per dimension,
How many states in total?

[ ] 5
[ ] 25
[ ] 125
[ ] 625

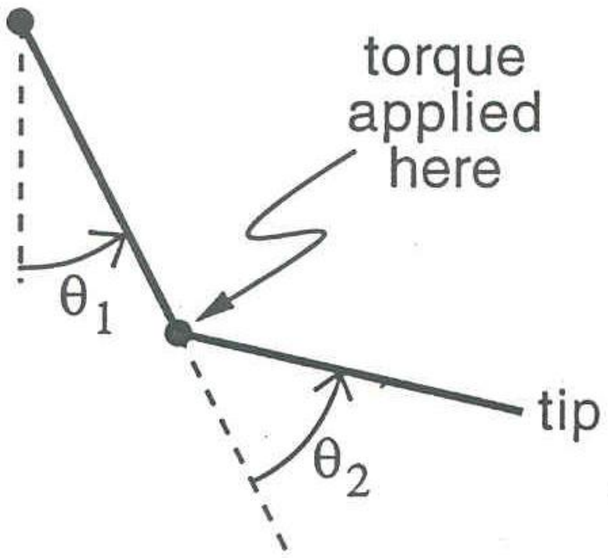


Figure 11.4 The acrobot.

From Book:
Sutton and Barto

Previous slide.
The aim of the acrobat is to move the tip above the blue line. To achieve this torque can be applied at the ‘elbow’ link. The second link is the ‘shoulder’.

There are three possible actions.
But what are the states? How many states do we have?

2. Reinforcement Learning: Example Acrobot

1st episode: long sequence of random actions
400th episode: short sequence of ‘smart’ actions

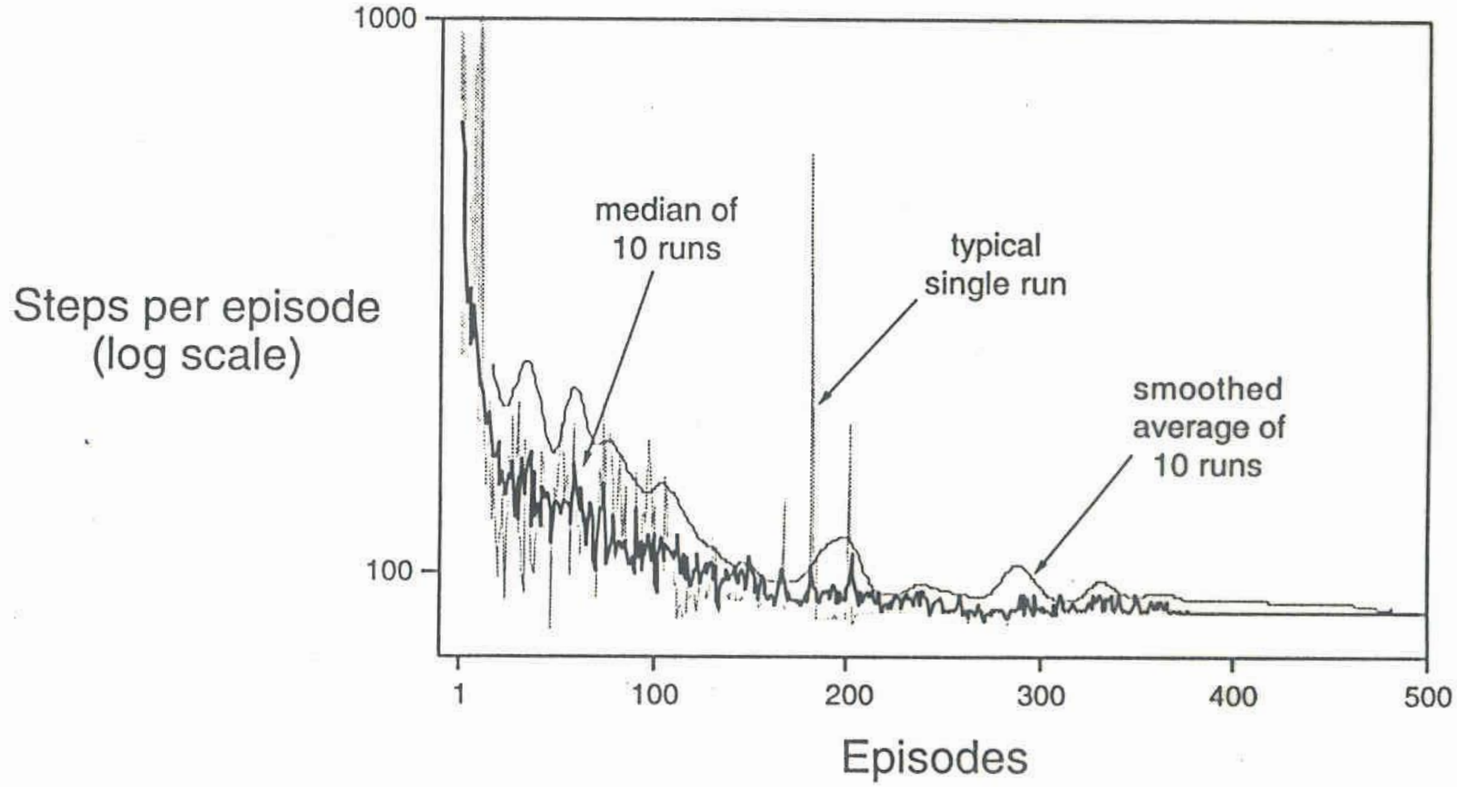


Figure 11.6 Learning curves for Sarsa(λ) on the acrobot task.

From Book:
Sutton and Barto

Previous slide.

An episode finishes if the target is reached. Over time episodes get shorter and shorter indicating that the acrobat has discovered (via reinforcement learning) a smart sequence of actions so as to reach the target (i.e., move the tip above the reference line)

2. Reinforcement Learning: Example Acrobot

after 400 episodes

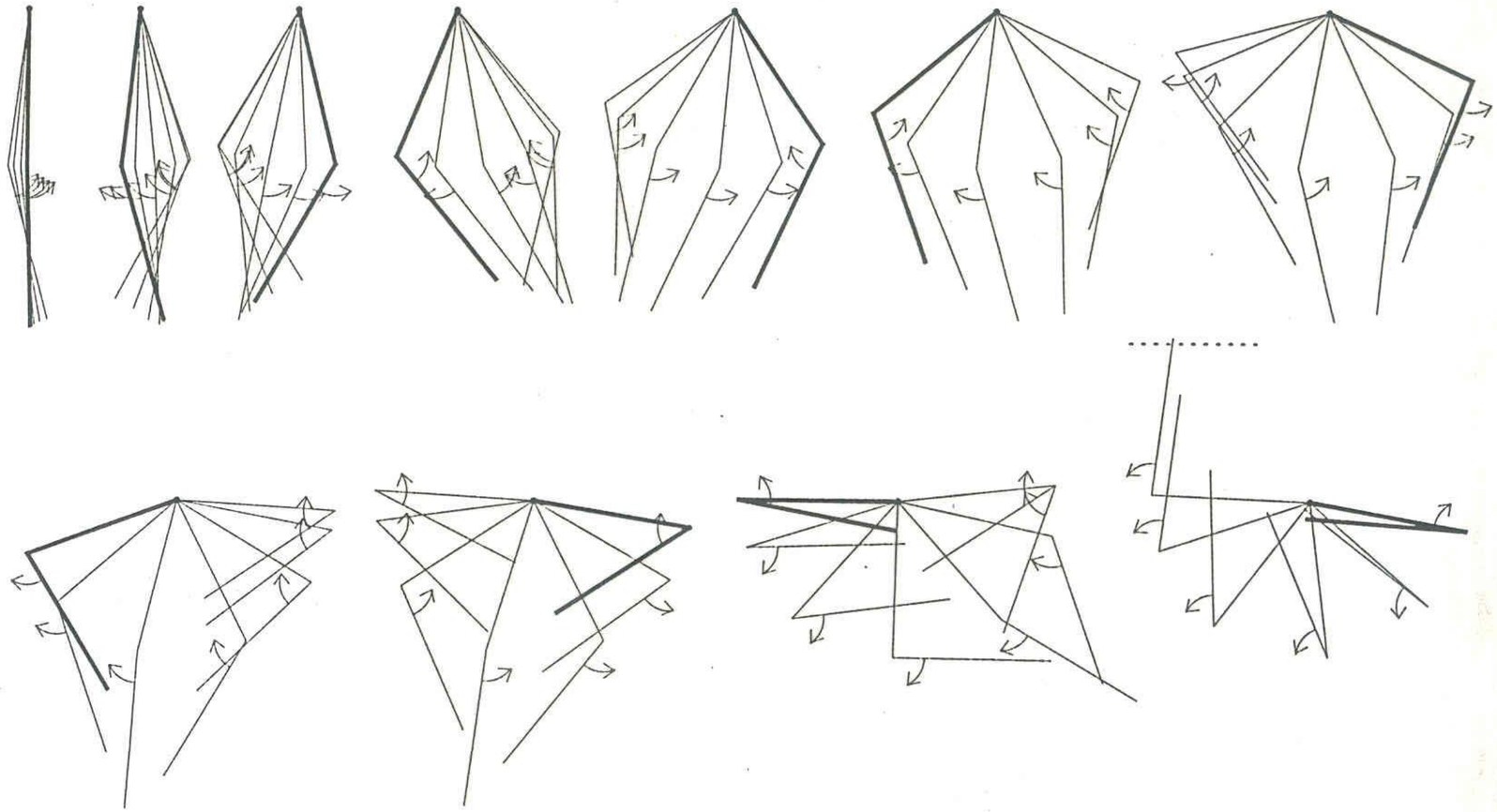


Figure 11.7 A typical learned behavior of the acrobot. Each group is a series of consecutive positions, the thicker line being the first. The arrow indicates the torque applied at the second joint.

From Book:
Sutton and Barto

Previous slide.
One example of an action sequence, after learning, is shown.

2. Reinforcement Learning: Example backgammon

Case Studies

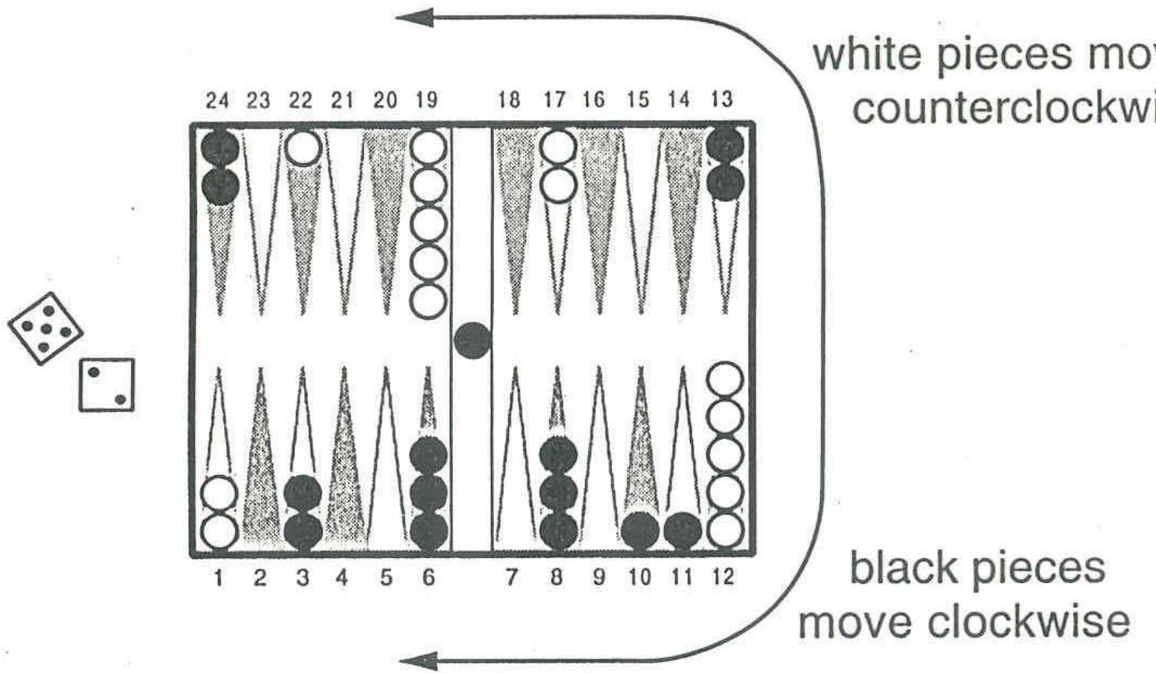


Figure 11.1 A backgammon position.

Game position = discrete states!

Suppose 2 pieces per player,
How many states in total?

[ ] 100<n<500
[ ] 500<n<5000
[ ] 5 000<n<50 000
[ ] n>50 000

From Book:
Sutton and Barto

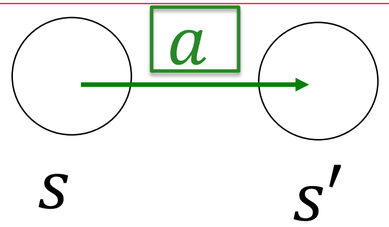
Previous slide.

Backgammon game. There are 24 fields on the board. Players have several pieces. Pieces are protected if there are two of the same color on the same field.

To make it simply, we now consider that both players have two pieces each left. How many different states are there in total?

2. Elements of Reinforcement Learning: Summary

There can be MANY states
Often need to discretize first
(→ next week we try to model in continuum)

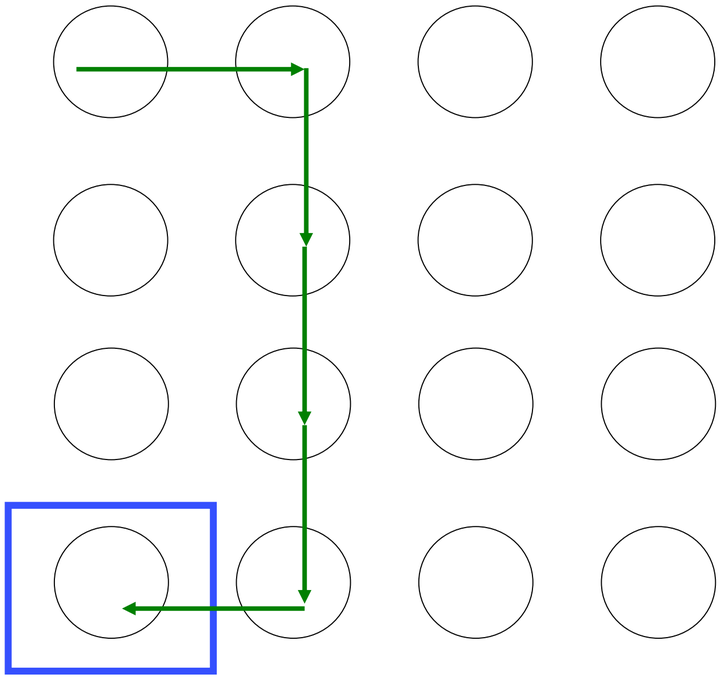


- discrete actions: a

- Mean reward for transition:

$$R^{a}_{s \to s'} = E\left(r \middle| s, a, s' \right)$$

- current actual reward: r_t



often most transitions have zero reward

Previous slide.
Conclusion: In all practical situations, there is an enormous number of states.

In many situations we can think of the actions as discrete. For the moment we also think of the states as discrete (but next week we will go to continuous state space)

Artificial Neural Networks: RL1

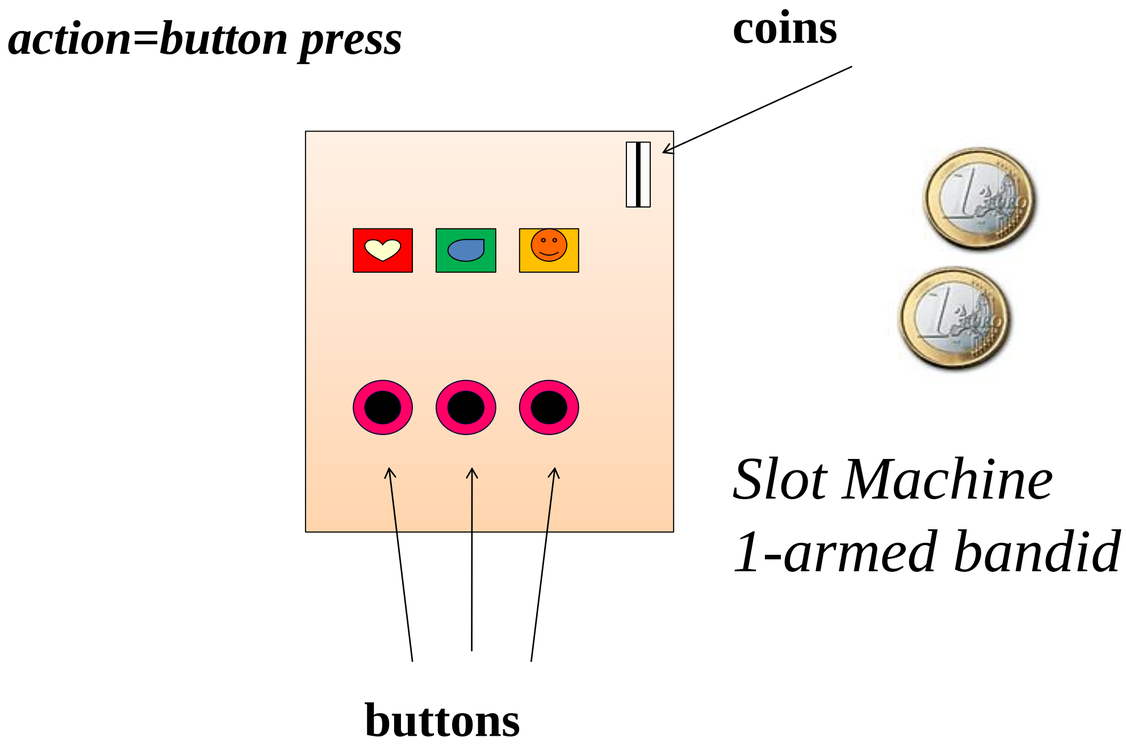
Reinforcement Learning and SARSA

1. Learning by Reward: Reinforcement Learning
2. Elements of Reinforcement Learning
3. One-step horizon (bandit problems)

Previous slide.

We start with the simplest discrete example: the game is over and reward is given after a single step.

2. One-step horizon games (bandit)

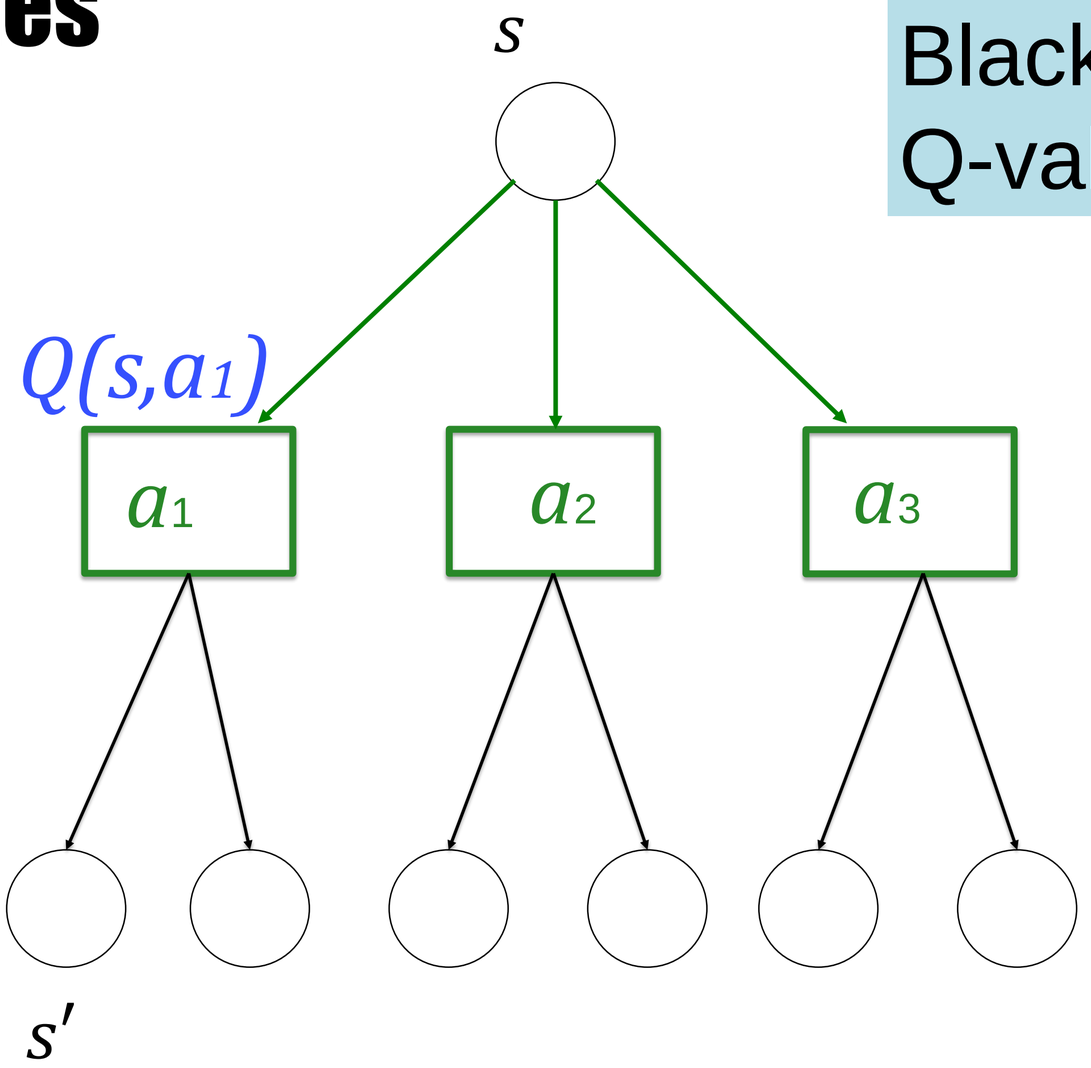


Slot Machine
1-armed bandid

Previous slide.
The standard example is a one-armed bandit, or slot machine: you have to choose between a few actions, and once you have pressed the button you can just wait and see whether you get reward or not.

2. One-step horizon games

Q-value: $Q(s,a)$
Expected reward for action a starting from s



Blackboard1:
Q-values

Previous slide.

One of the most central notion in reinforcement learning is the Q-value.
Q(s,a) has two indices: you start in state s and take action a.
The Q-value Q(s,a) is (an estimate of) the mean expected reward that you will get if you take action a starting from state s.

2. One-step horizon games

Blackboard1:
Q-values

Your notes.

2. One-step horizon games: Q-value

Q-value $Q(s,a)$

Expected reward for action a starting from s

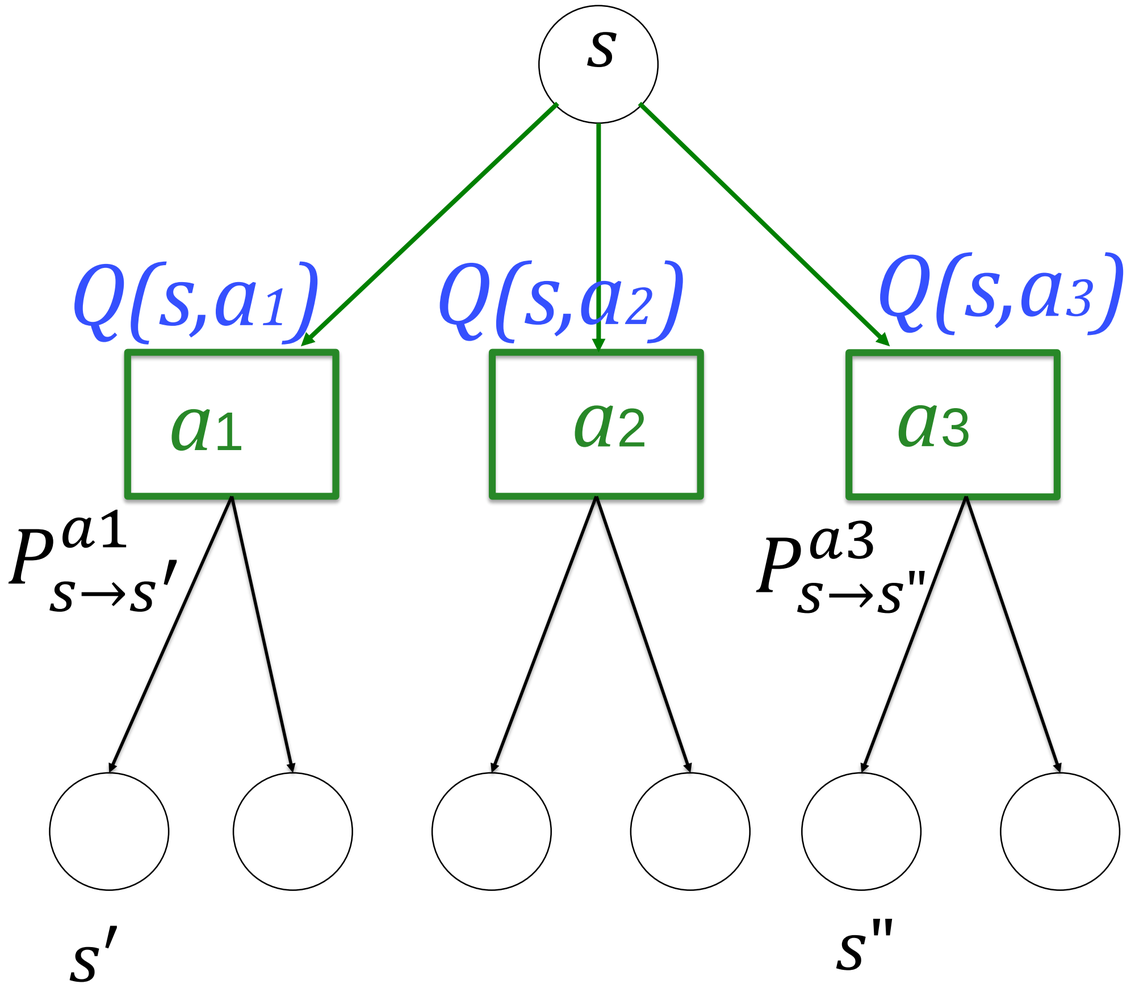
$$Q(s,a) = \sum_{s'} P^{a}_{s\to s'} R^{a}_{s\to s'}$$

Reminder:

$$R^{a}_{s\to s'} = E\left(r \middle| s', a, s\right)$$

Similarly:

$$Q(s,a) = E(r|s,a)$$



Now we know the Q-values: which action should you choose?

Previous slide.

$P_{s \to s'}^{a1}$ is the probability that you end up in a specific state s' if you take action a1 in state s.
We refer to this sometimes as the 'branching ratio' below the 'actions'.

Q(s,a) is attached to the branches linking the state s with the actions.

actions are indicated by green boxes; states are indicated by black circles.

The mean reward $R_{s \to s'}^{a}$ is defined as the expected reward given that you start in state s with action a and end up in state s' (see Blackboard 1).

Given the branching ratio and the mean rewards, it is easy to calculate the Q-values (Blackboard 1).

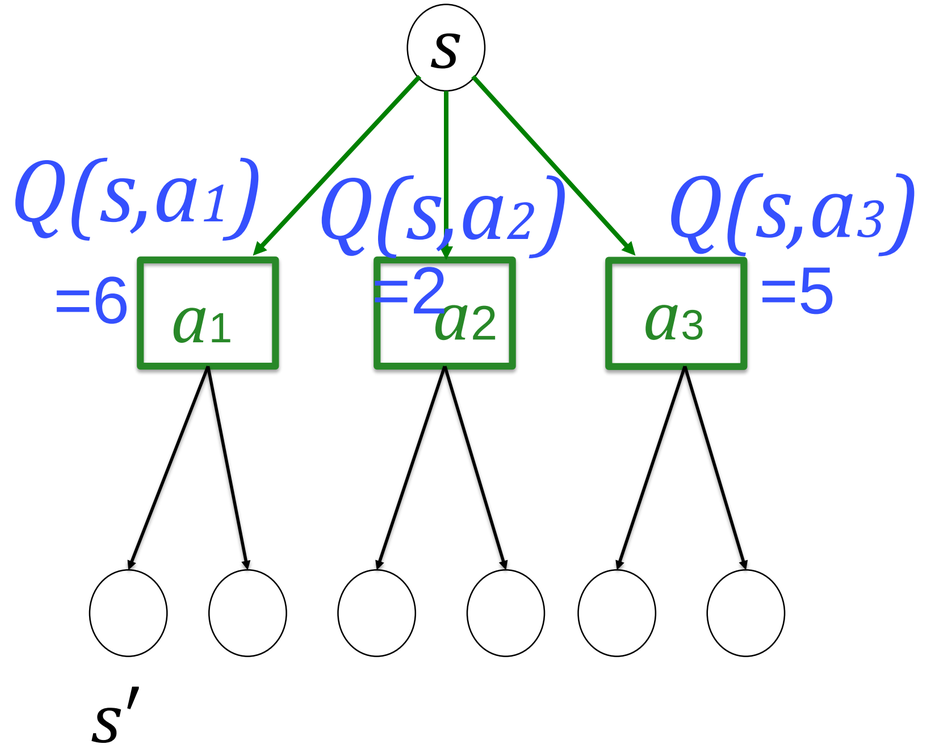
2. Optimal policy (greedy)

Suppose all Q-values are known:

take action a with*

$$Q(s,a^*) \geq Q(s,a_j)$$

other actions



optimal action:

$a^* = \text{argmax}_a\ [Q(s,a)]$

Optimal policy is also called ‘greedy policy’

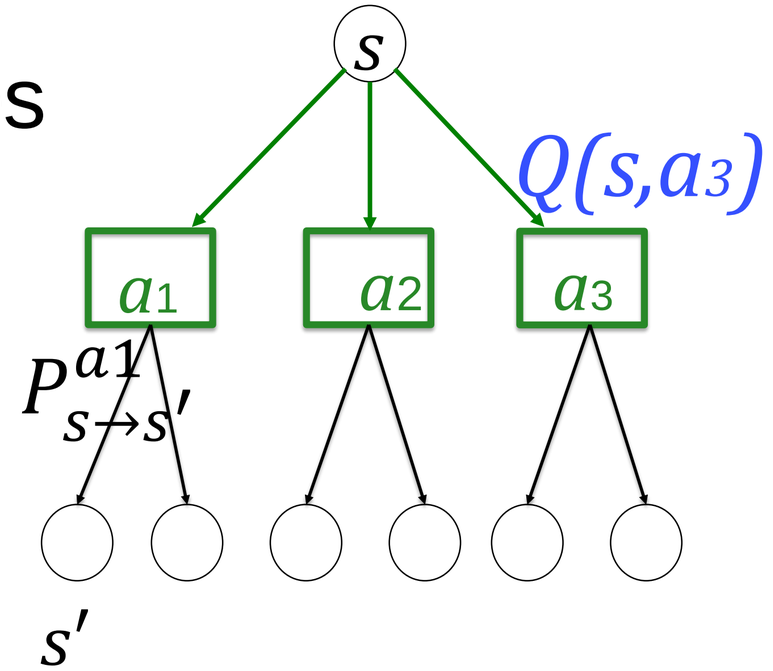
Previous slide.
And once you have the Q-values it is easy to choose the optimal action:
Just take the one with maximal Q-value.

2. One-step horizon games

Q-value = expected reward for state-action pair

If Q-value is known, choice of action is simple
→ take action with highest Q-value

BUT: we normally do not know the Q-values
→ estimate by trial and error



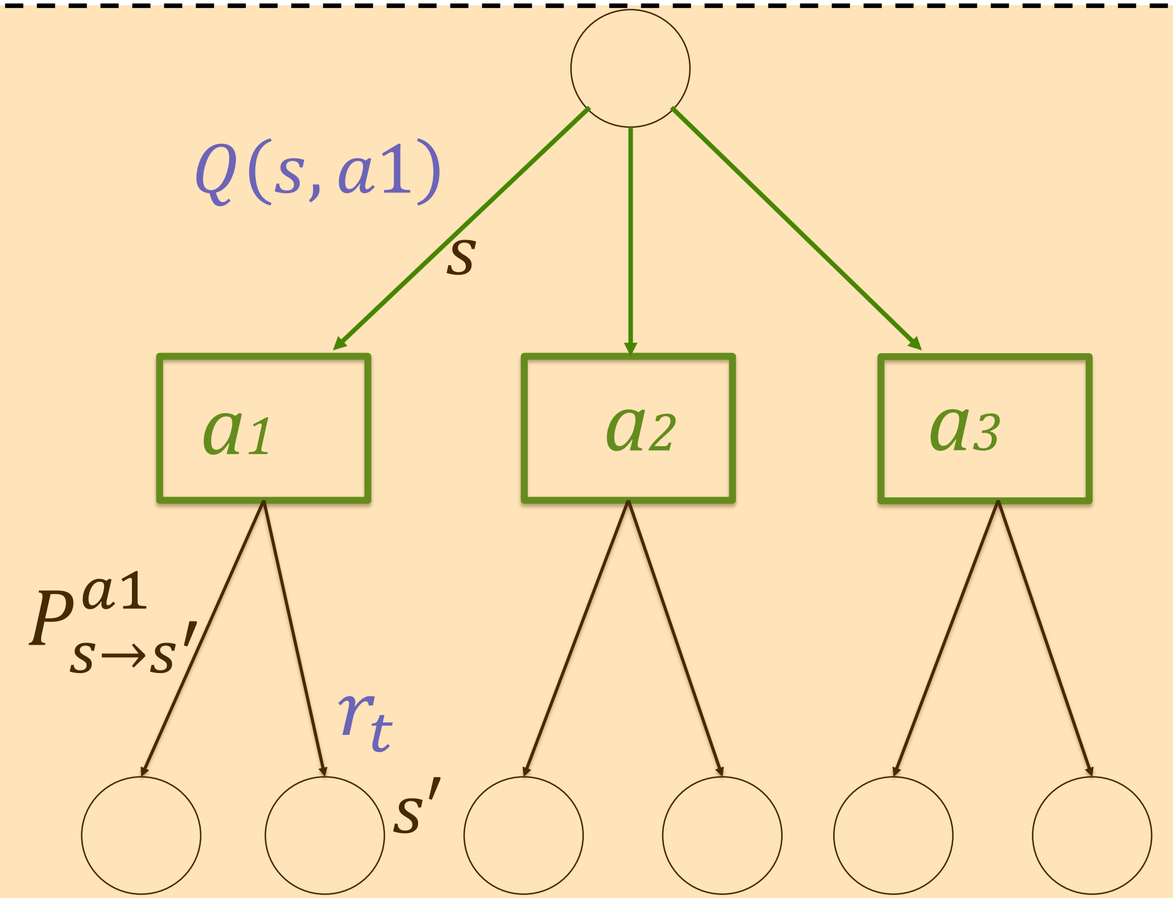
Previous slide.
The only remaining problem is that we do not know the Q-values, because the casino gives you neither the branching ratio nor the reward scheme.

Hence the only way to find out is by trial and error (that is, by playing many times – the casino will love this!).

Exercise 1 now (preparation)

Expected reward

$$Q(s,a) = \sum_{s'} P^{a}_{s \to s'} R^{a}_{s \to s'}$$



Show that empirical averaging over k trials gives an update rule

$$\Delta Q(s,a) = \eta\,[r_t - Q(s,a)]$$

Exercise 1 now (in class)

10 min, now!

Exercise 1. Iterative update (in class)

We consider an empirical evaluation of $Q(s, a)$ by averaging the rewards for action a over the first k trials:

$$Q_k = \frac{1}{k} \sum_{i=1}^{k} r_i.$$

We now include an additional trial and average over all $k + 1$ trials.

a. Show that this procedure leads to an iterative update rule of the form

$$\Delta Q_k = \eta(r_k - Q_{k-1}),$$

(assuming $Q_0 = 0$).

b. What is the value of η?

c. Give an intuitive explanation of the update rule. *Hint: Think of the following: If the actual reward is larger then my estimate, then I should ...*

Blackboard2: Exercise 1

Your notes.

2. One-step horizon: summary

Q-value = expected reward for state-action pair

If Q-value is known, choice of action is simple

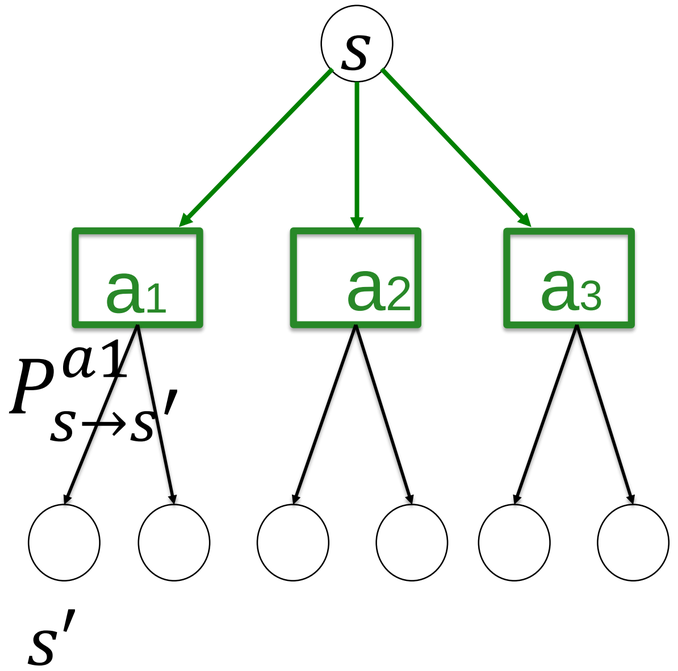
→ take action with highest Q-value

If Q-value not known:

→ estimate $\hat{Q}$ by trial and error

→ update with rule

$$\Delta\hat{Q}(s,a) = \eta\,[r_t - \hat{Q}(s,a)\,]$$



Let learning rate η decrease over time

Previous slide.
Let us distinguish the ESTIMATE $\hat{Q}(s,a)$ from the real Q-value $Q(s,a)$

The update rule can be interpreted as follows:
if the actual reward is larger than (my estimate of) the expected reward, then I should increase (a little bit) my expectations.

The learning rate η :

In the example, we found a rather specific scheme for how to reduce the learning rate over time. But many other schemes also work in practice. For example you keep η constant for a block of time, and then you decrease it for the next block.

The exact value of η is not relevant, as discussed in the following theorem. Important is that η is small at the end of learning so as to limit the amount of fluctuations.

2. Convergence in Expectation

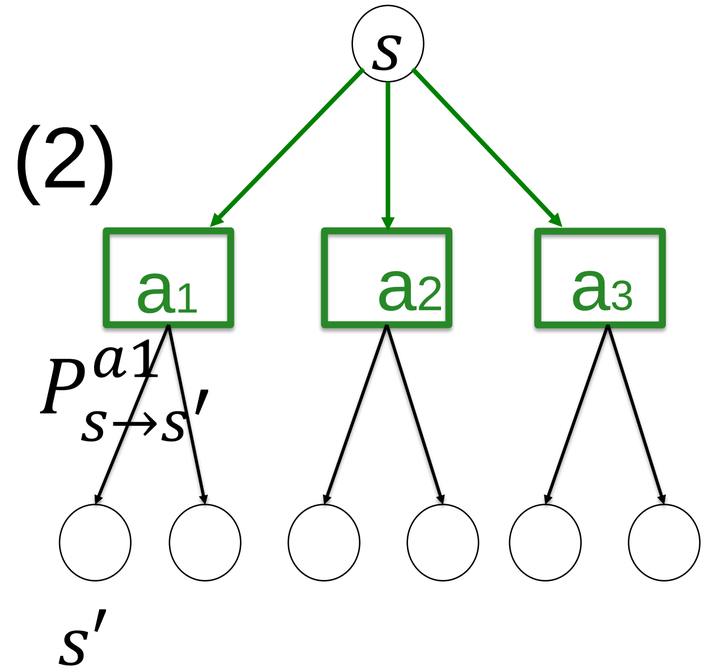
Blackboard3:
Proof of (i).

After taking action a in state s, we update with

$$\Delta\hat{Q}(s,a) = \eta\,[r_t - \hat{Q}(s,a)] \qquad (1)$$

(i) If (1) converges in expectation, then $\hat{Q}(s,a)$ fluctuates around,

$$E\left[\widehat{Q}(s,a)\right] = \sum_{s'} P^{a}_{s\to s'}\, R^{a}_{s\to s'} = Q(s,a) \qquad (2)$$



(ii) If the learning rate η decreases, fluctuations around $E[\widehat{Q}(s,a)]$ decrease.

Your notes.

Artificial Neural Networks: RL1

Reinforcement Learning and SARSA

1. Learning by Reward: Reinforcement Learning
2. Elements of Reinforcement Learning
3. Exploration vs Exploitation

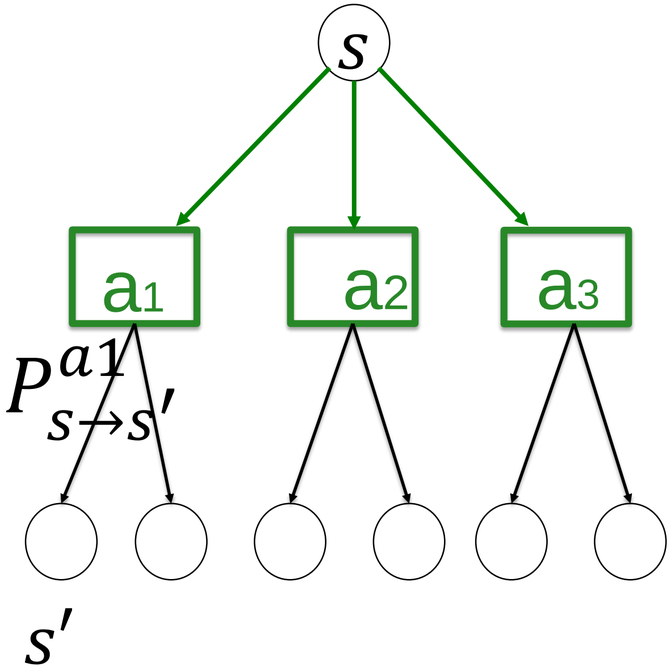
Previous slide.

To estimate the Q-values you have to play all the different actions several times. However, if you know the Q-values you should only play the best action.

3. Exploration – Exploitation dilemma

Ideal: take action with maximal $Q(s,a)$

Problem: correct Q values not known
(since reward probabilities and branching probabilities unknown)



Exploration versus exploitation

Explore so as to estimate reward probabilities

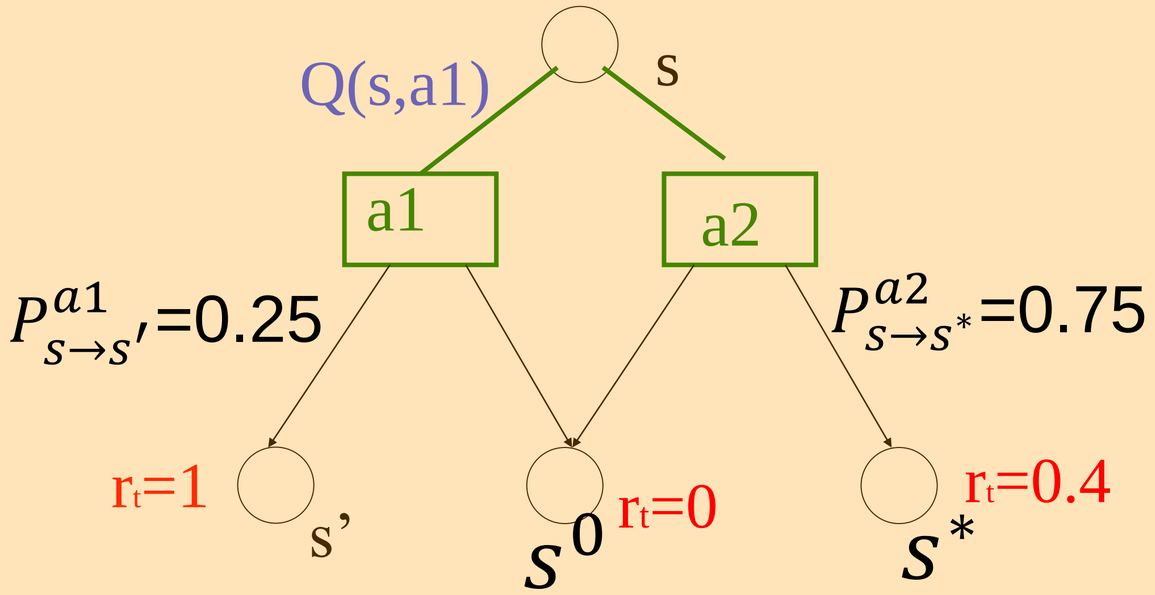
Take action which looks optimal, so as to maximize reward

Previous slide.

Since Q-values are not known, you are always in the situation of an exploration-exploitation dilemma.

Note: All estimates of Q will be empirical estimates. To simplify, I write for the empirical estimate Q(s,a) without the hat.

Exercise 2.a - 2.c: Exploration-Exploitation



s=state

a2 action

r_t=1 actual reward

s'=new state

2.a Assume that you initialize all Q values with zero; set η =0.2 (constant)

Trial 1: you choose action a1, you get r_t=1

Trial 2: you choose action a2, you get r_t=0.4

$$\Delta Q(s,a) = \eta\,[r_t - Q(s,a)]$$

2.b Trial 3 – 5: continue 'greedy' (assume that you do not get rewards), update Q

2.c Calculate for both actions the expected reward $Q(s,a)=\sum_{s'} P^a_{s\to s'} R^a_{s\to s'}$

Exercise 2.a and 2.c now: Exploration-Exploitation

6 min,
Get started now!

Exercise 2. Greedy policy and the two-armed bandit (in class)

In the "2-armed bandit" problem, one has to choose one of 2 actions. Assume action a_1 yields a reward of $r = 1$ with probability $p = 0.25$ and 0 otherwise. If you take action a_2, you will receive a reward of $r = 0.4$ with probability $p = 0.75$ and 0 otherwise. The "2-armed bandit" game is played several times.

a. Assume that you initialize all Q values at zero. You first try both actions: in trial 1 you choose a_1 and get $r = 1$; in trial 2 you choose a_2 and get $r = 0.4$. Update your Q values ($\eta = 0.2$).

b. In trials 3 to 5, you play greedy and always choose the action which looks best (i.e., has the highest Q-value). What are the Q-values after trial 5? Assume actual reward r =0 in trials 3-5

c. Calculate the expected reward for both actions. Which one is the best?

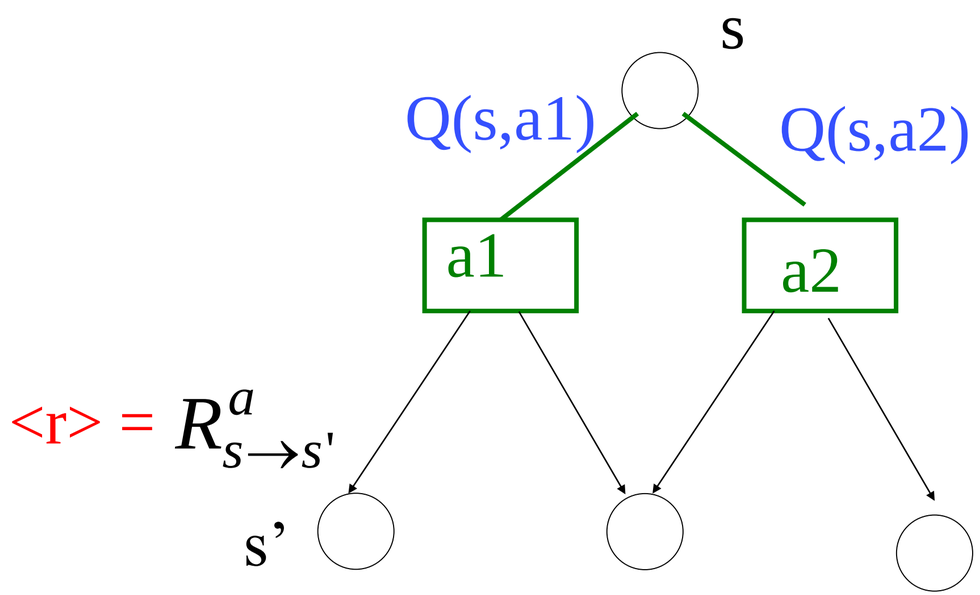
Update rule in a and b is $\Delta Q(s,a) = 0.2 \quad [r_t - Q(s,a)]$

η=0.2 is a constant

Blackboard4:
Exercise 2a-2c

Your comments.

3. Exploration and Exploitation



Problem: correct Q values not known

greedy strategy:
- take action a* which looks best

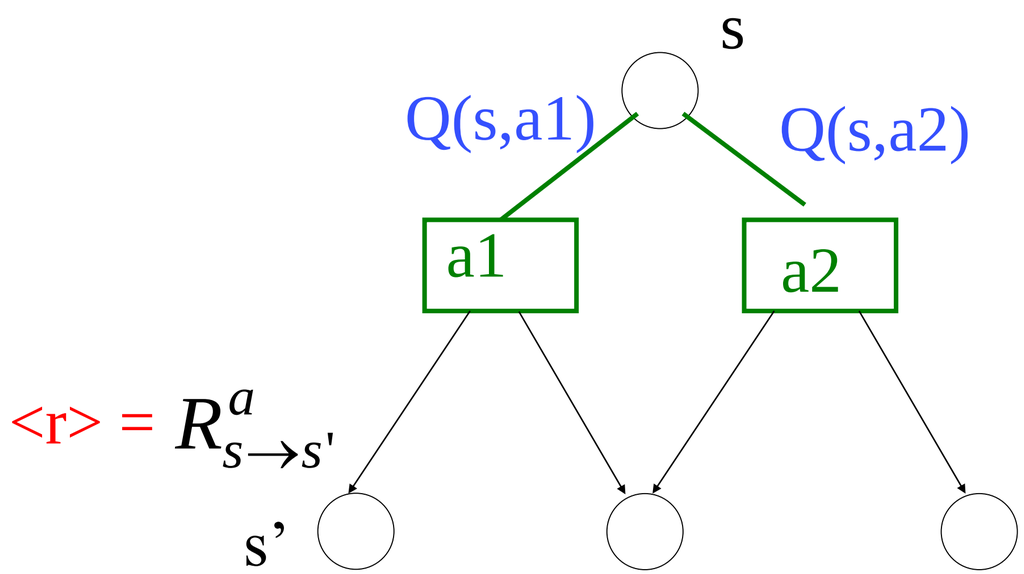
$Q(s,a^*) > Q(s,a_j)$

ATTENTION:
with ‘greedy’ you may get
stuck with a sub-optimal strategy

Previous slide.

If you know the correct Q-values, the best choice would be to choose the action with maximal Q-value (called ‘greedy’ action). But since you don’t know the Q-values it is risky to choose the greedy action because you may get stuck with a suboptimal choice.

3. Exploration and Exploitation: practical approach



$$\Delta Q(s,a) = \eta\,[r_t - Q(s,a)]$$

Problem: correct Q values not known

greedy strategy:
- take action a* which looks best

$$Q(s,a^*) > Q(s,a_j)$$

$\mathcal{E}$greedy strategy:
- take action a* which looks best with prob $P = 1-\varepsilon$

Softmax strategy: take action a' with prob

$$P(a') = \frac{\exp[\beta Q(a')]}{\sum_a \exp[\beta Q(a)]}$$

Optimistic greedy:
initialize with Q values that are too big

Previous slide.

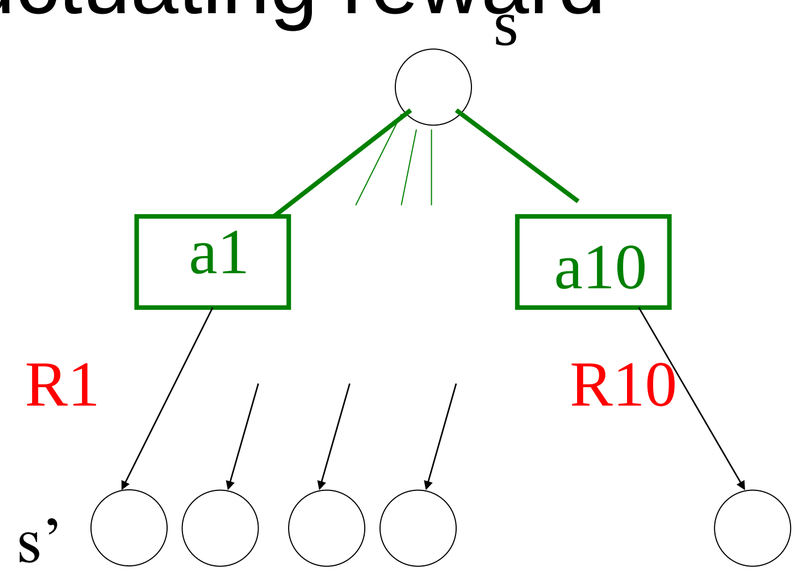
Softer versions of greedy allow you to choose occasionally an action which looks suboptimal, but which allows you to further explore the Q-values of other options.

Epsilon-greedy and softmax are examples following this idea.
Note that ‘softmax’ is a function that one also encounters in multiclass tasks with 1-hot coding (see course of ‘machine learning’; also later lecture on deep learning)

A radically different approach is optimistic greedy. If you initialize all Q-values at the same value, but clearly too high (compared to maximal reward that you can get in the scheme), then the Q-value of action a1 decreases initially each time you play a1, which in turn favors other actions that you have not yet played.

3. Exploration and Exploitation: practical approach

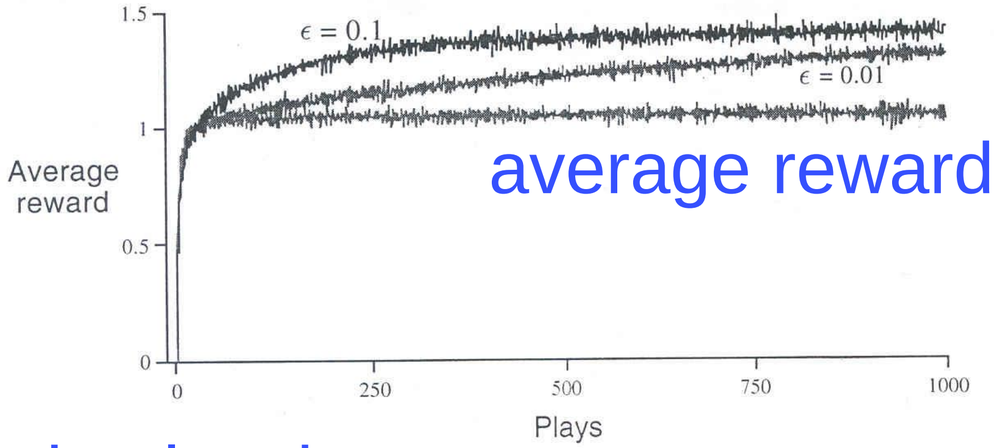
Example: 10-armed bandit with fluctuating reward



in each action, actual rewards fluctuate around a mean

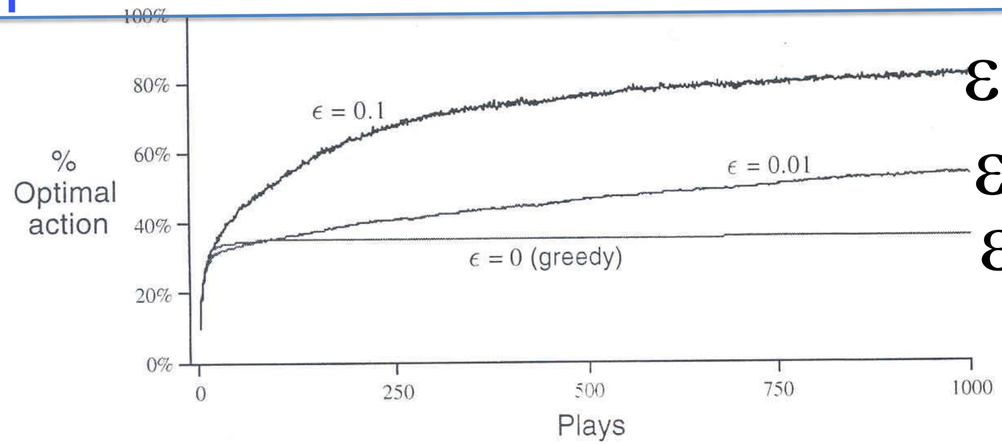
Rk= $R_{s \to s'}^{ak}$

Epsilon-greedy: simulation



average reward

Optimal action



ε=0.1

ε=0.01

ε=0

Figure 2.1 Average performance of ϵ-greedy action-value methods on the 10-armed testbed. These data are averages over 2000 tasks. All methods used sample averages as their action-value estimates.

book: Sutton and Barto

Previous slide.

Computer simulation of a situation where actual rewards r fluctuate around the mean reward R. There are 10 different actions a1, …, a10 each with a different mean reward R1, …, R10.

There exist two different ways to evaluate the performance.
Top: what is the average reward that you get by playing epsilon-greedy?
Bottom: what is the fraction of times that you play the optimal action, by playing epsilon-greedy.

Three different values of epsilon are used.

3. Exploration and Exploitation: practical approach

Epsilon-greedy, combined with iterative update of Q-values

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Repeat forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

Sutton and Barto, ch. 2

Previous slide.

This is the style of pseudo-code that we will see a lot over the next few weeks. It is taken from the book of Sutton and Barto (MIT Press, 2018).

N(a) is a counter of how many times the agent has taken action a.

In this specific example the learning rate eta is the inverse of the count N(a) (see earlier exercise); but in the more general setting we would remove the counter and just use some heuristic reduction scheme for eta.

3. Quiz: Exploration – Exploitation dilemma

We use an iterative method and update Q-values with **eta=0.1**

[ ] With a greedy policy the agent uses the best possible action

[ ] Using an epsilon-greedy method with epsilon = 0.1
means that, even after convergence of Q-values,
in at least 10 percent of cases a suboptimal action is chosen.

[ ] If the rewards in the system are between 0 and 1 and Q-values are initialized with Q=2, then each action is played at least 5 times before exploitation starts.

Previous slide.

3. Quiz: Exploration – Exploitation dilemma

All Q values are initialized with the same value Q=0.1

Rewards in the system are r =0.5 for action 1 (always)
and r=1.0 for action 2 (always)

We use an iterative method and update Q-values with **eta=0.1**

[ ] if we use softmax with **beta = 10**, then, after 100 steps, action 2 is chosen almost always

[ ] if we use softmax with **beta = 0.1**, then, after 100 steps action 2 is taken about twice as often as action 1.

Softmax strategy: take action a'
with prob

$$P(a') = \frac{\exp[\beta Q(a')]}{\sum_{a} \exp[\beta Q(a)]}$$

Your notes.

Teaching monitoring – monitoring of understanding

[ ] today, up to here, at least 60% of material was new to me.

[ ] up to here, I have the feeling that I have been able to follow (at least) 80% of the lecture.

Remark.

1. As an EPFL teacher, my aim is to teach such that 80 percent of the students in the classroom are able to follow at least 80 percent of the lecture.

Sometimes I succeed, sometimes I don’t.

2. Since most courses in the master are optional, overlap is sometimes unavoidable. However, my aim as an EPFL teacher is to still present in each lecture 60 percent of material that is new to 60 percent of the students.

If part of a lecture is novel for less than 20 percent of the students, I am happy to remove that material from the in-class presentation (it could still stay in the notes).

Artificial Neural Networks: RL1

Reinforcement Learning and SARSA

1. Learning by Reward: Reinforcement Learning
2. Elements of Reinforcement Learning
3. Exploration vs Exploitation
4. Bellman equation

Previous slide.
So far our Q-values were limited to situations with a 1-step horizon. Now we will get more general.

4. Multistep horizon

Policy $\pi(s,a)$

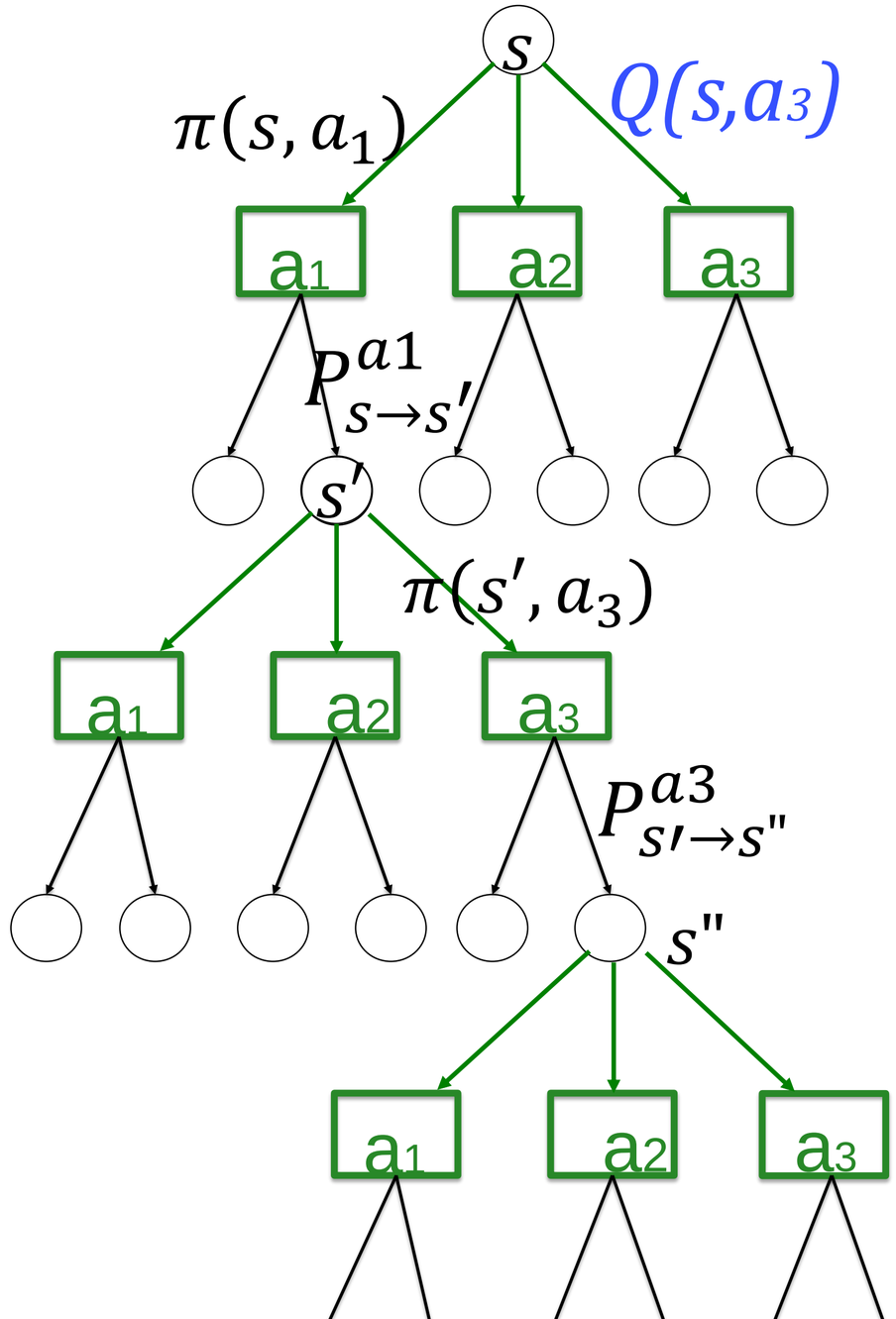
probability to choose action a in state s

$1=\sum_{a'}\pi(s,a')$

Examples of policy:
-epsilon-greedy
-softmax

Stochasticity $P^{a}_{s\to s'}$

probability to end in state s' taking action a in state s



Previous slide.

After a first action that leads to state s' starting from state s , the agent can now take a second action starting from s'.

Note that there are two different types of branching ratio:

$\pi(s, a_1)$ describes the probability that the agent uses action a1 when it is in state s – based on the agent's policy (such as epsilon-greedy)

$P_{s \to s'}^{a1}$ describes as before the probability that the agent arrives in state s' given that it chooses action a1 in state s.

As before we are interested in the expected reward. The Q value Q(s,a) describes the total accumulated reward the agent can get starting in state s with action a.

Next slide: rewards that are n steps away are discounted with a factor γ^{n}

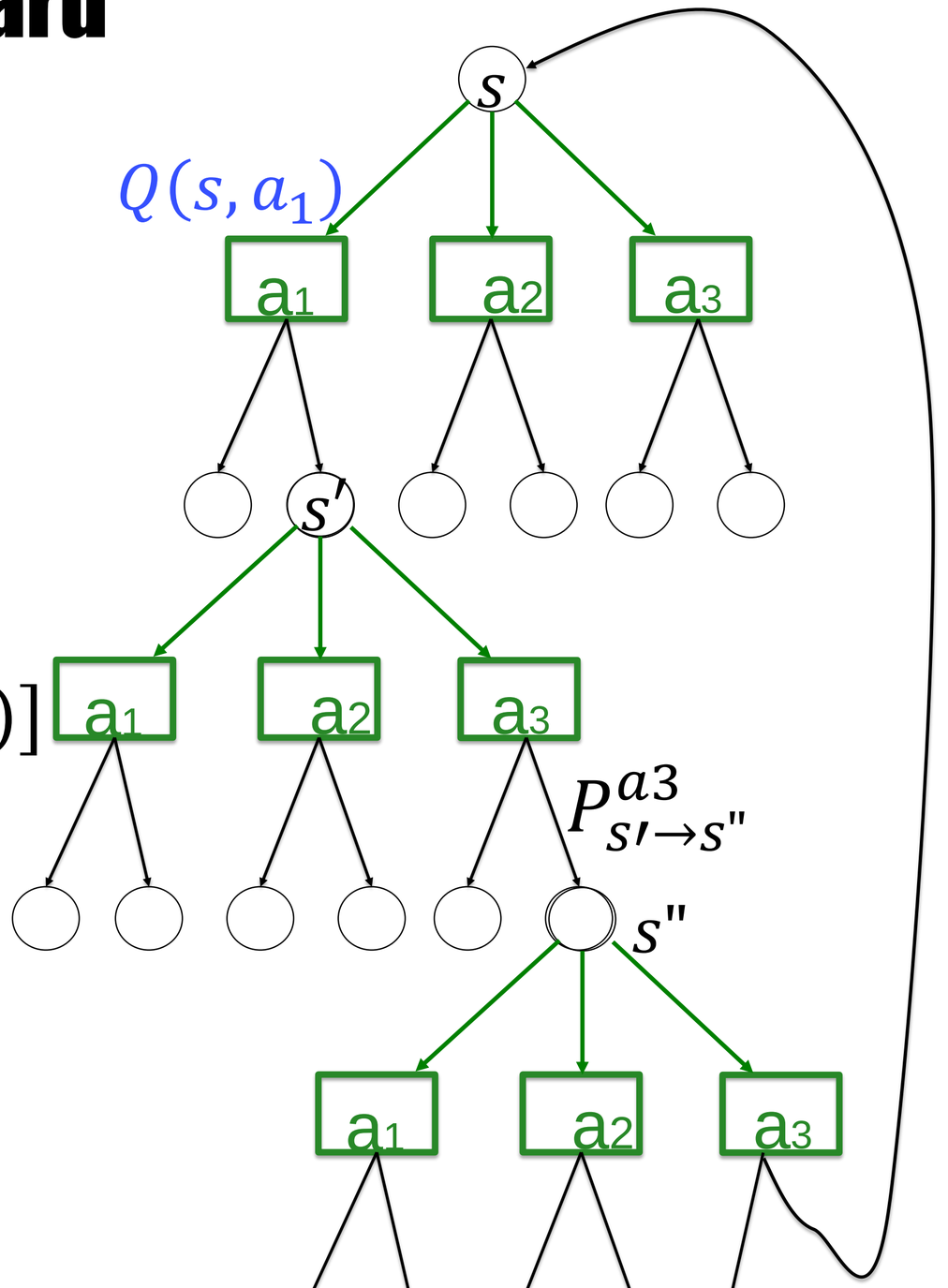
4. Total expected (discounted) reward

Starting in state s with action a

$Q(s,a)$ =

$$= \left\langle r_t \quad + \gamma\, r_{t+1} + \gamma^2 r_{t+2} + \gamma^3\, r_{t+3} + \cdots \right\rangle$$

$$= E[r_t + \gamma\, r_{t+1} + \gamma^2 r_{t+2} + \gamma^3\, r_{t+3} + \ldots | s, a)]$$



Discount factor: γ <1

-important for recurrent networks!

-avoids blow-up of summation

-gives less weight to reward in **far** future

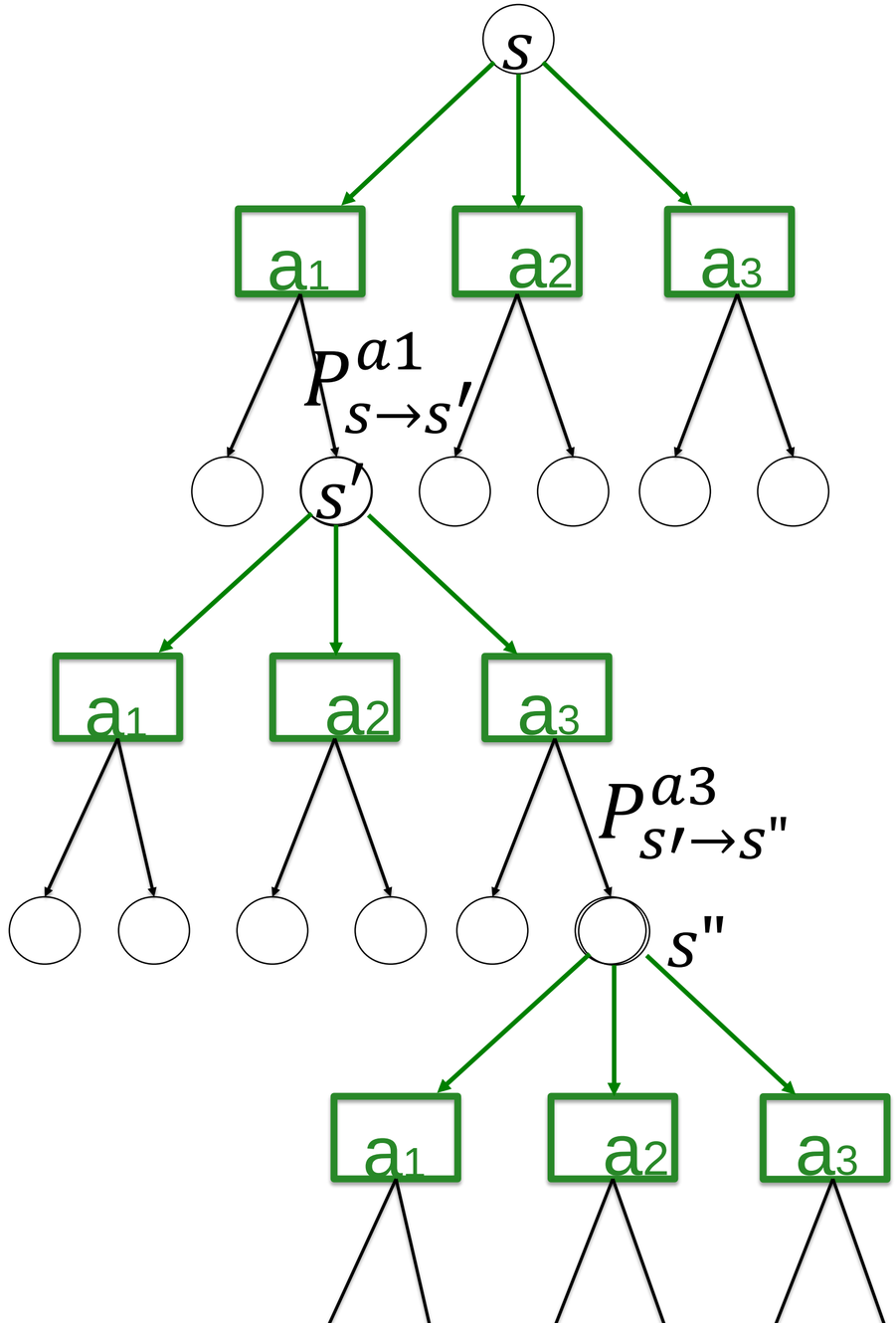
Previous slide.

Angular brackets denote expectation.
Red-font lower-case r indicates the reward collected over multiple time steps in one episode, starting in state s with action a.

Expectation means that we have to take the average over all possible future paths giving each path its correct probabilistic weight.

4. Bellman equation

Blackboard5:
Bellman eq.

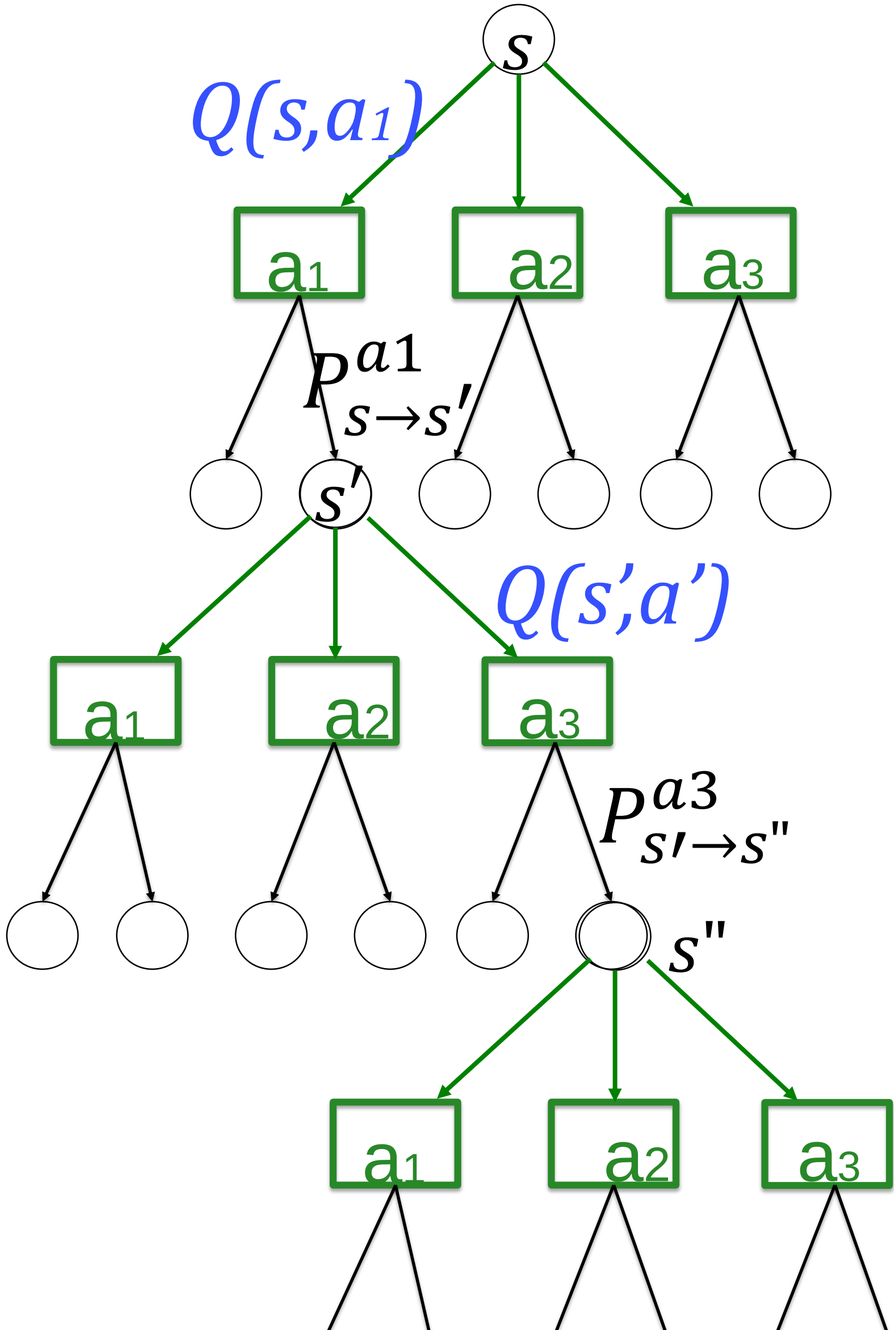


Space for calculations.

4. Bellman equation with policy π

$$Q(s,a) = \sum_{s'} P^{a}_{s\to s'} \left[R^{a}_{s\to s'} + \gamma \sum_{a'} \pi(s',a') Q(s',a') \right]$$

Bellman equation = value consistency of neighboring states



Remark:

Sometimes Bellman equation is written for greedy policy: $\pi(s,a) = \delta_{a,a*}$

with action $a^* = \underset{a'}{\operatorname{argmax}}\, Q(s,a')$

Previous slide.

The Bellman equation relates the Q-value for state s and action a with the Q-values of the neighboring states.
Note that the two different types of branching ratio both enter the equation.

Bottom: in the case of a greedy policy, the Bellman equation simplifies

4. Bellman equation (for optimal actions)

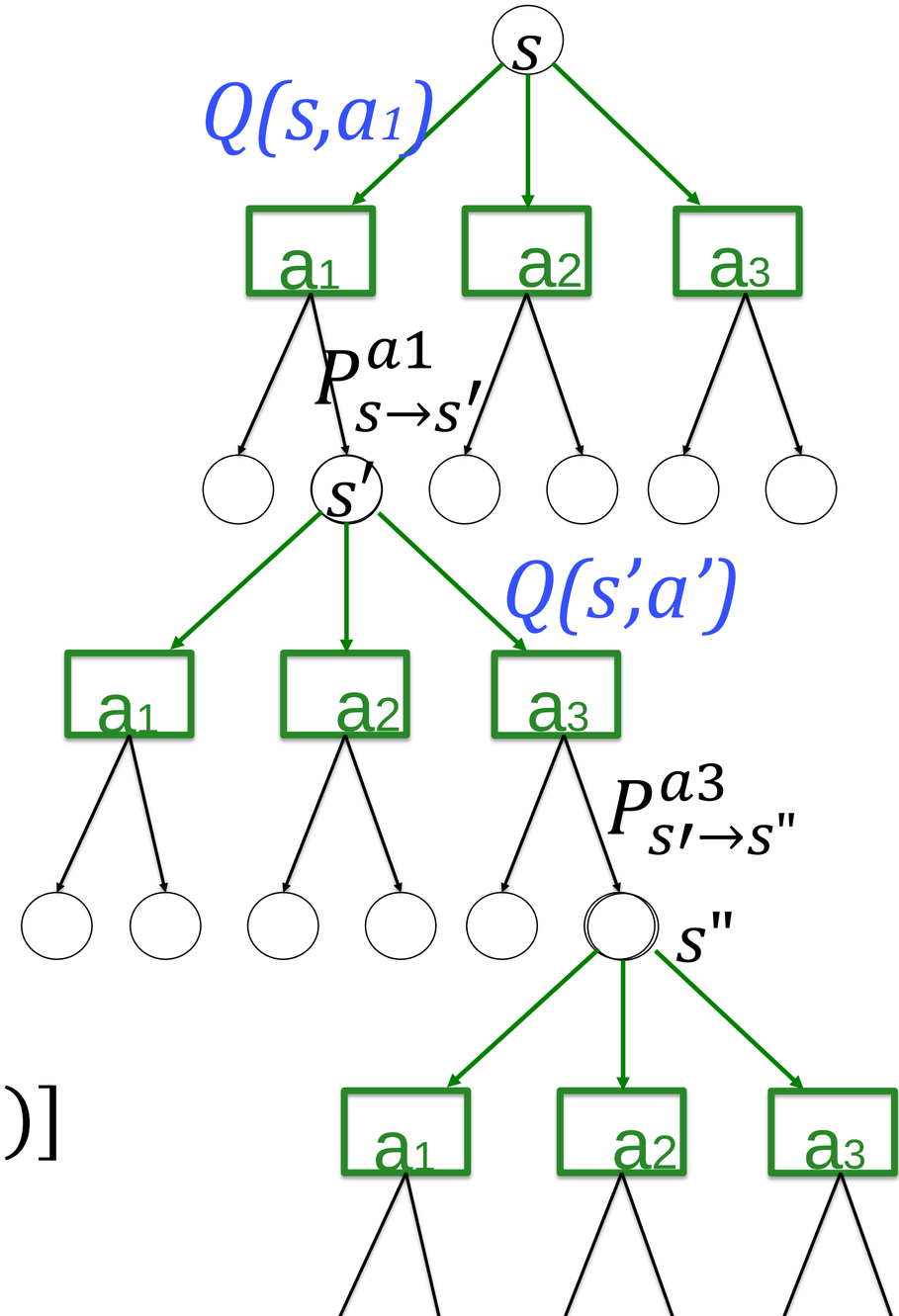
$$Q(s,a) = \sum_{s'} P^{a}_{s\to s'} \left[R^{a}_{s\to s'} + \gamma \sum_{a'} \pi(s',a') Q(s',a') \right]$$

for greedy policy:

$$\pi(s,a) = \delta_{a,a*}$$

with action $a^* = \underset{a'}{\operatorname{argmax}}\, Q(s,a')$

$$Q(s,a) = \sum_{s'} P^{a}_{s'\to s"} \; [\, R^{a}_{s'\to s"} + \gamma max_{a'} Q(s'a')]$$



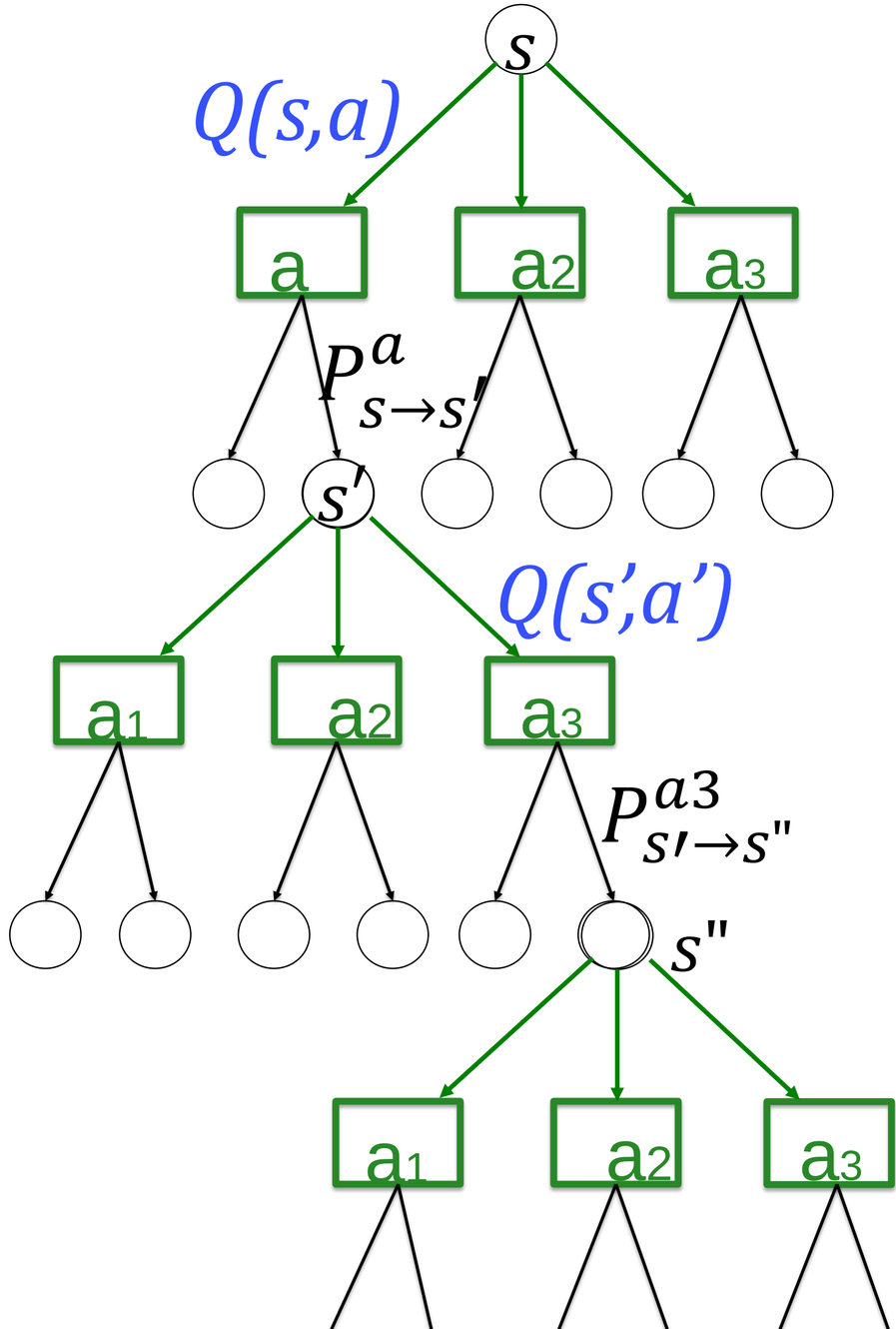
Previous slide.
For a greedy policy, the sum over actions disappears from the Bellman equation and is replaced by the max-sign.

4. Quiz: Bellman equation with policy π

$$Q(s,a) = \sum_{s'} P^{a}_{s \to s'} \left[R^{a}_{s \to s'} + \gamma \sum_{a'} \pi(s',a') Q(s',a') \right]$$

[ ] The Bellman equation is linear in the variables $Q(s'a')$

[ ] The set of variables $Q(s',a')$ that solve the Bellman equation is unique and does not depend on the policy



Your comments.

Artificial Neural Networks: RL1

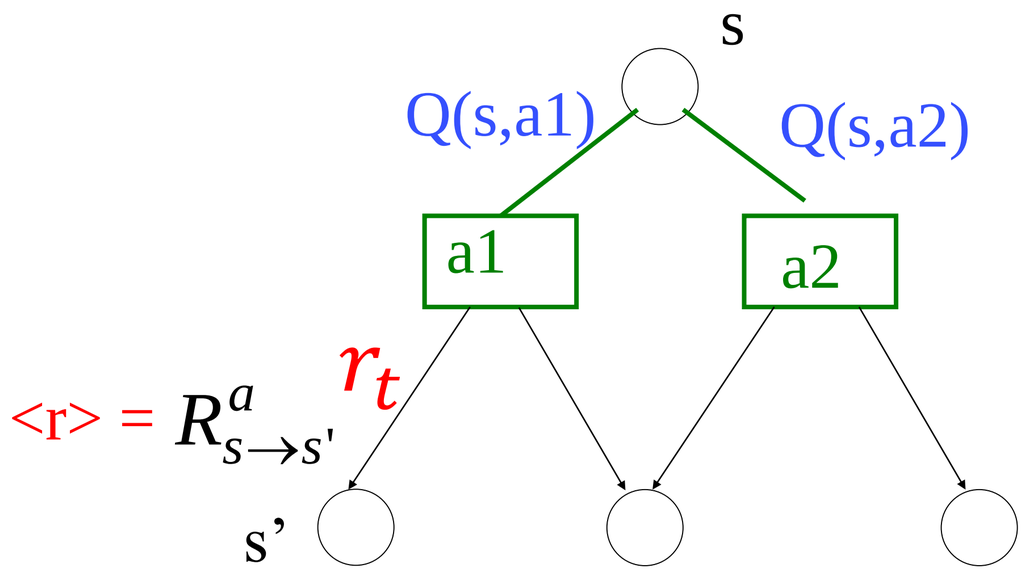
Reinforcement Learning and SARSA

1. Learning by Reward: Reinforcement Learning
2. Elements of Reinforcement Learning
3. Exploration vs Exploitation
4. Bellman equation
5. SARSA algorithm

Previous slide.
We not turn to the first practical algorithm, called SARSA.

3. Iterative update of Q-values

Problem: Q-values not given



Solution: iterative update

$$\Delta Q(s,a) = \eta\ [r_t - Q(s,a)]$$

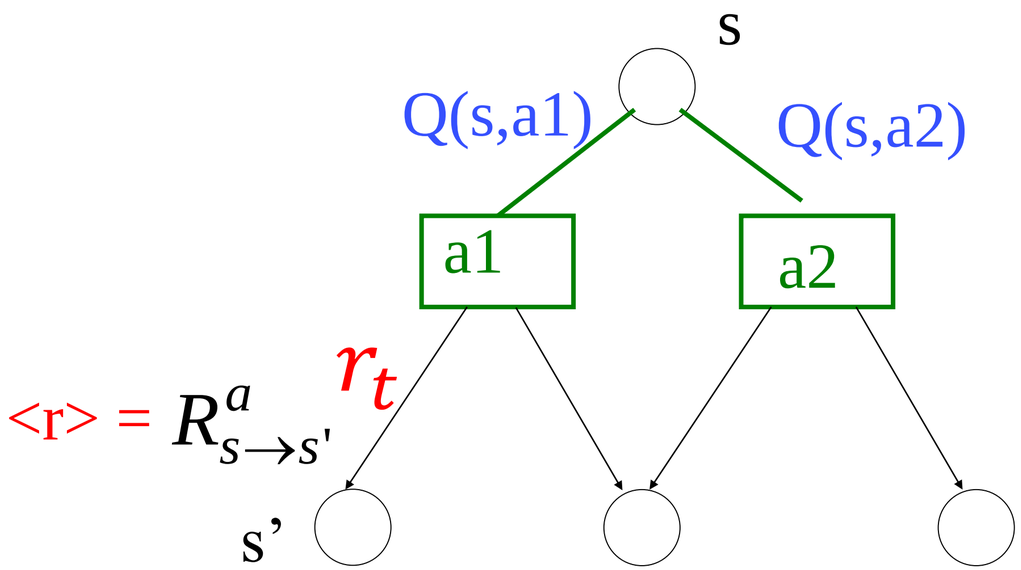
while playing with policy $\pi(s,a)$

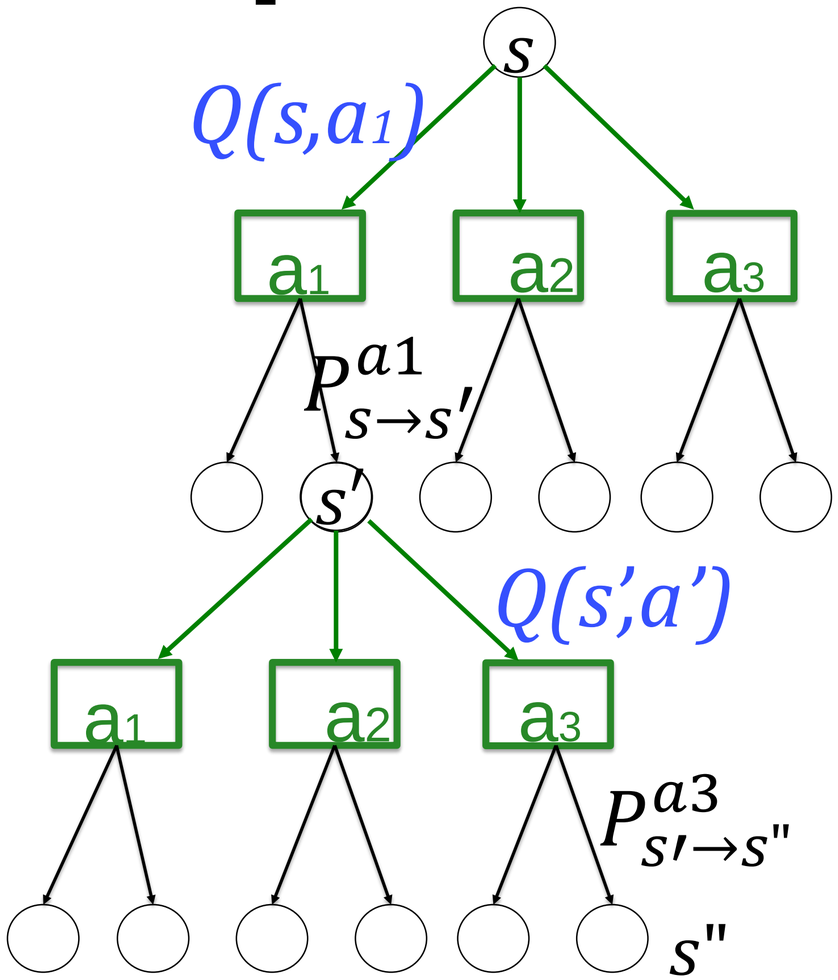
Previous slide.
Reminder: for the 1-step horizon scenario we found that we could calculate the Q-values iteratively. We increase the Q-value by a small amount (with learning rate eta>0) if the reward observed at time t is larger than our current estimate of Q. And we decrease the Q-value by a small amount if the reward observed at time t is smaller than our current estimate of Q.

Iterative updates with one data point at a time are also called ‘online algorithms’. Thus our update rule is an online algorithm for the estimation of Q-values.

5. Iterative update of Q-values for multistep environments

Problem: Q-values not given





Solution: iterative update

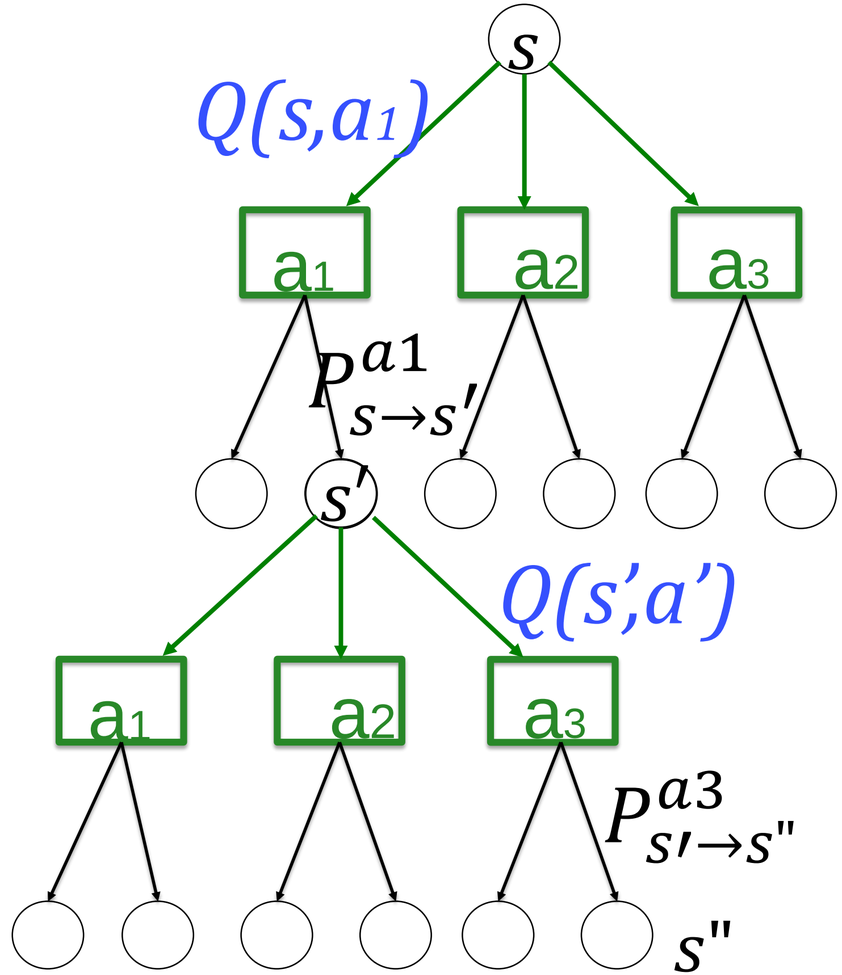
$$\Delta\hat{Q}(s,a) = \eta\,[r_t - \hat{Q}(s,a)]$$

$$\Delta\hat{Q}(s,a) = \;?$$

while playing with policy $\pi(s,a)$

Previous slide.
The question now is: can we have a similar iterative update scheme also for the multi-step horizon?

Blackboard6: SARSA update



Your notes.

5. Iterative update of Q-values for multistep environments

Bellman equation:

$$Q(s,a) = \sum_{s'} P^{a}_{s\to s'}\left[R^{a}_{s\to s'} + \gamma \sum_{a'} \pi(s',a')Q(s',a')\right]$$

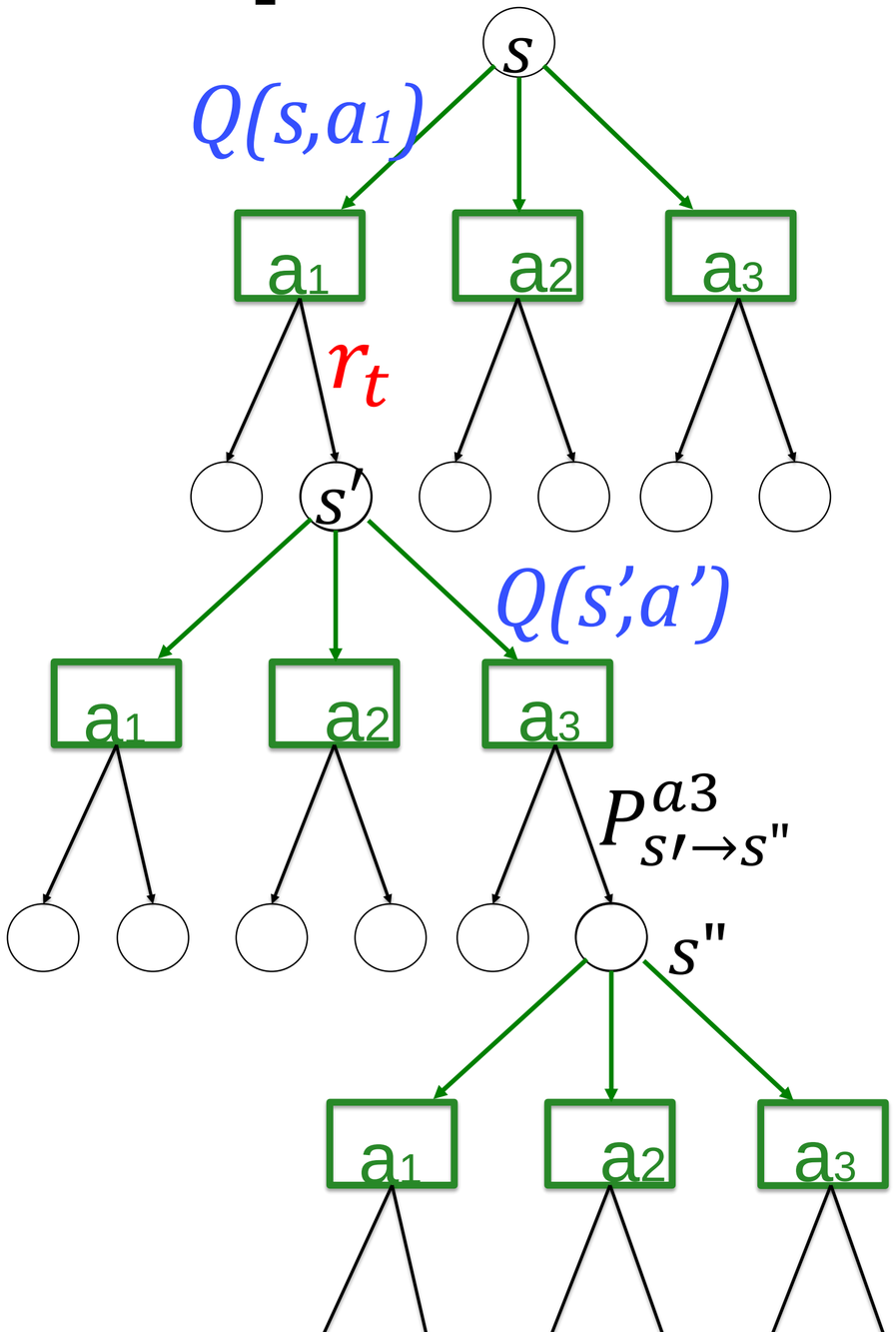
Problem:

- Q-values not given
- branching probabilities not given
- reward probabilities not given

Solution: iterative update

$$\Delta\hat{Q}(s,a) = \eta\ [r_t + \gamma\hat{Q}(s',a') - \hat{Q}(s,a)]$$

while playing with policy $\pi(s,a)$



Previous slide.
Even for the case of the multi-step horizon, we can estimate the Q-values by an interative update:

The Q-values Q(s,a) is increase by a small amount if the sum of (reward observed at time t plus discounted Q-value in the next step) is larger than our current estimate of Q(s,a).

This iterative update gives rise to an online algorithm.

NOTE: in the following we always work with empirical estimates, and drop the ‘hat’ of the variable Q.

5. SARSA vs. Bellman equation

$$Q(s,a) = \sum_{s'} P^{a}_{s \to s'} \left[R^{a}_{s \to s'} + \gamma \sum_{a'} \pi(s',a') Q(s',a') \right]$$

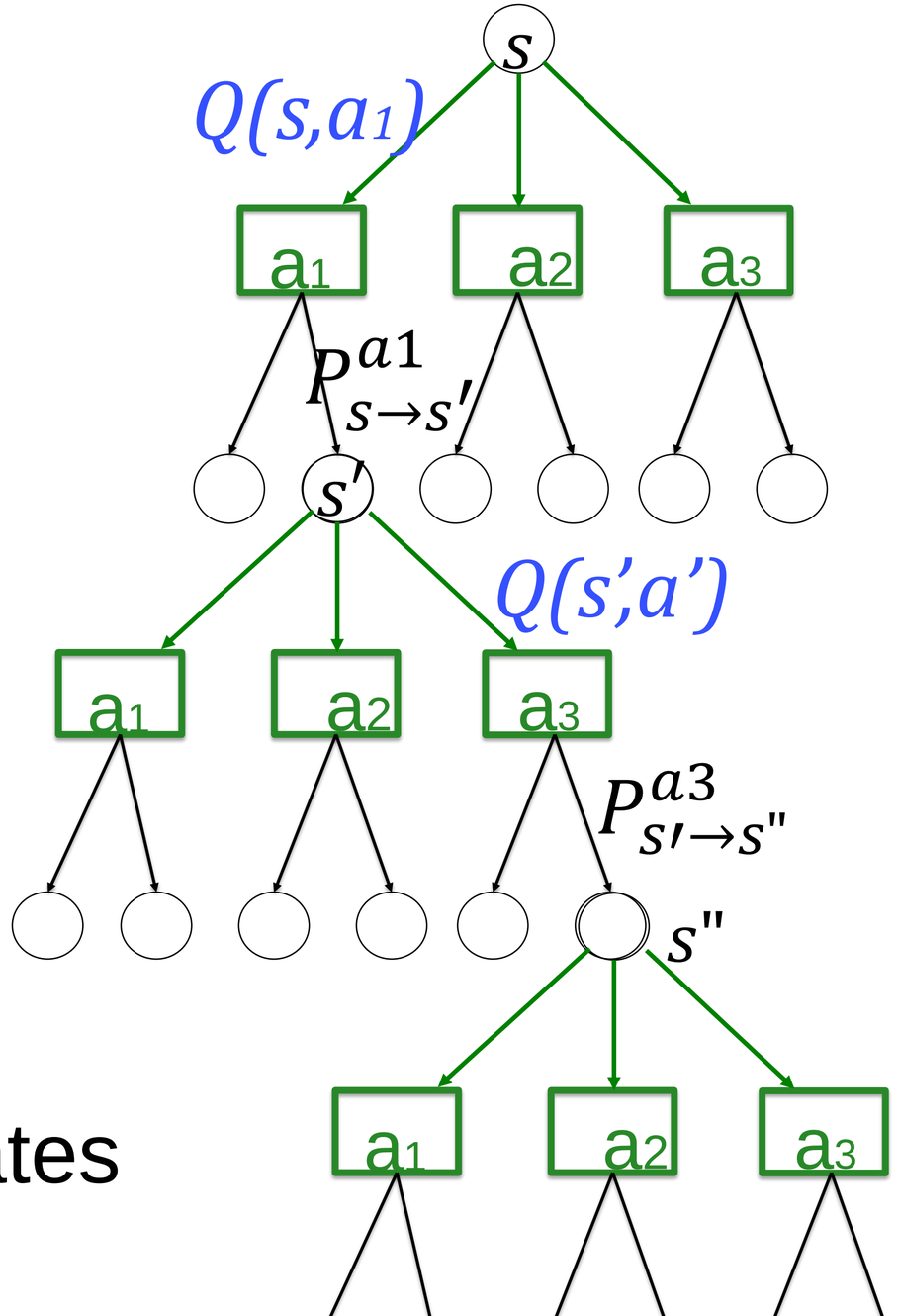
Bellman equation

= consistency of Q-values
across neighboring states

SARSA update rule

$$\Delta Q(s,a) = \eta [r_t + \gamma Q(s',a') - Q(s,a)]$$

= make Q-values of neighboring states
more consistent



Previous slide.
The Bellman equation summarizes the consistency condition:
The (average) rewards must explain the difference between Q(s,a) and Q(s’,a’) averaged over all s’ and a’.

The iterative update state that Q(s,a) needs to be adapted so the current reward explains the difference between Q(s,a) and Q(s’,a’).

5. SARSA algorithm

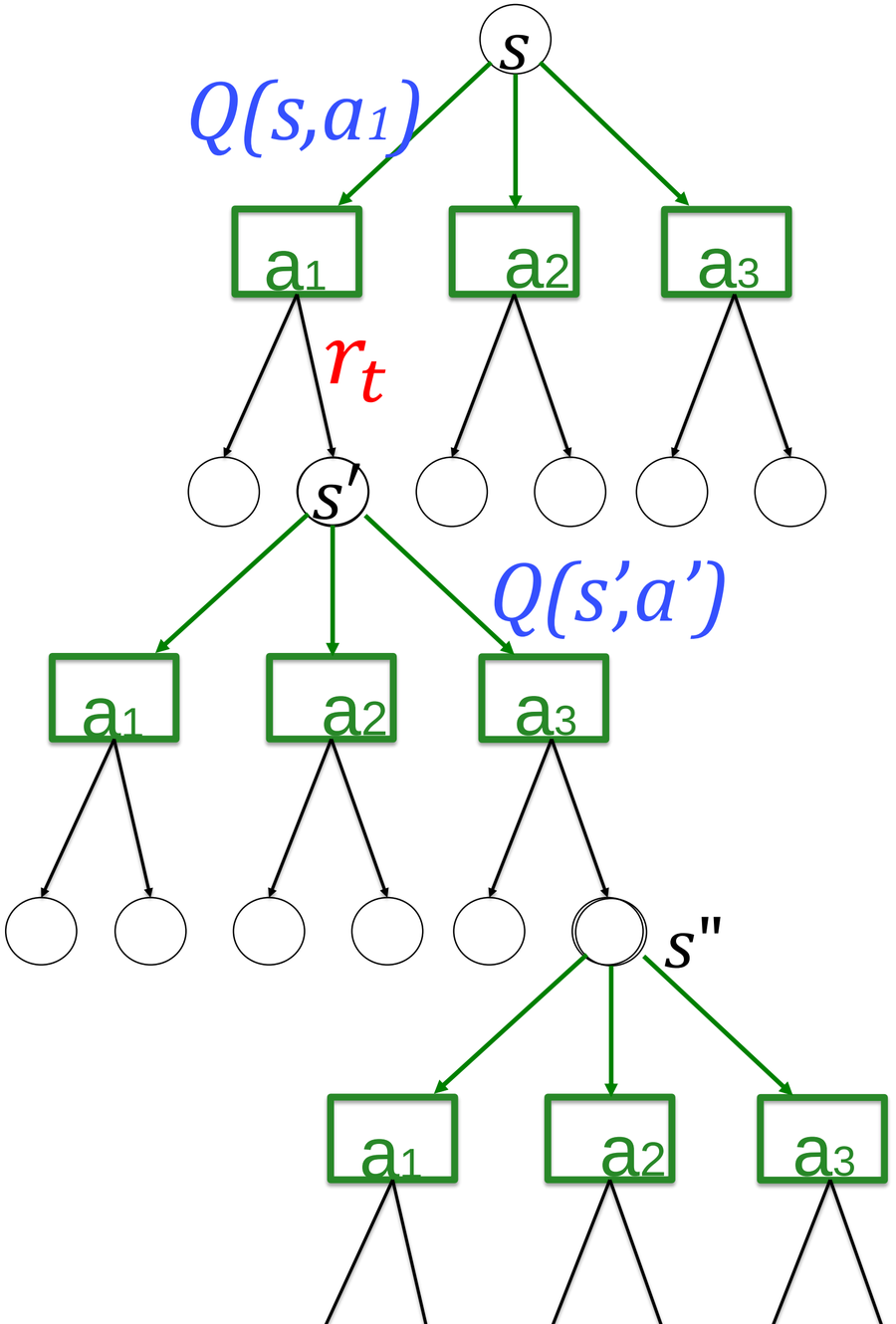
Initialise Q values
Start from initial state **s**

1) being in state **s**
   choose action **a**
   [according to policy $\pi(s,a)$]
2) Observe reward **r**
   and next state **s'**
3) Choose action **a'** in state s'
   [according to policy $\pi(s,a)$]
4) Update with SARSA update rule

$$\Delta Q(s,a) = [r_t + \gamma Q(s',a') - Q(s,a)]$$

5) set: s ← s'; a ← a'
6) Goto 1)

Stop when all Q-values have converged



Previous slide.
The update rule gives immediately rise to an online algorithm. You play the game. While you run through one of the episodes you observe the state s, choose action a, observe reward r, observe next state s’ and choose next action a’. At this point in time (and not earlier) you have all the information to update the Q-value Q(s,a).

The name SARSA comes from this sequence state-action-reward-state-action.

Exercise 4 (preparation)

- **Update of Q values in SARSA**

$$\Delta Q(s,a) = \eta \; [r - (Q(s,a) - Q(s',a'))]$$

- **policy for action choice:**

  Pick most often action

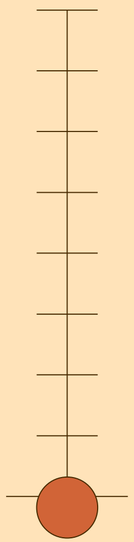
$$a_t^* = \arg\max_a Q_a(s,a)$$

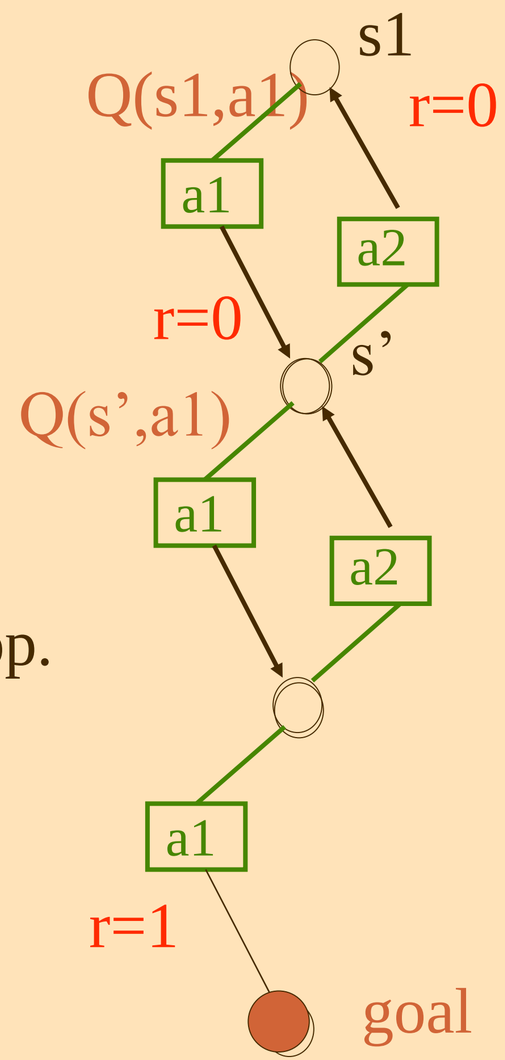
Consider a linear sequence of states. Reward only at goal.

Actions are up or down.

a) Initialise Q values at 0. Start at top.
   How do Q values develop?

b) Q values after 2 complete trials?





Exercise 4: SARSA for Linear Track.

Exercise 4 now (10min)

Exercise 4. SARSA algorithm

In the lecture, we introduced the SARSA (state-action-reward-state-action) algorithm, which (for discount factor $\gamma = 1$) is defined by the update rule

$$\Delta Q(s,a) = \eta \left[r - \left(Q(s,a) - Q(s',a') \right) \right] , \tag{1}$$

where s' and a' are the state and action subsequent to s and a. In this exercise, we apply a greedy policy, i.e., at each time step, the action chosen is the one with maximal expected reward, i.e.,

$$a_t^* = \arg\max_a Q_a(s,a) . \tag{2}$$

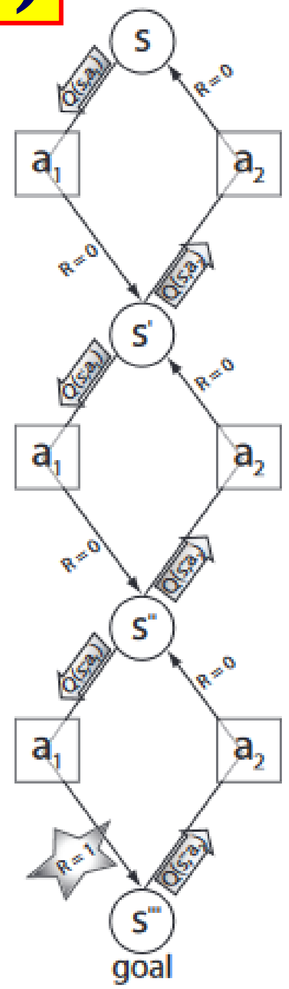
If the available actions have the same Q-value, we take both actions with probability 0.5.

Consider a rat navigating in a 1-armed maze (=linear track). The rat is initially placed at the upper end of the maze (state s), with a food reward at the other end. This can be modeled as a one-dimensional sequence of states with a unique reward ($r = 1$) as the goal is reached. For each state, the possible actions are going up or going down (Fig. 2). When the goal is reached, the trial is over, and the rat is picked up by the experimentalist and placed back in the initial position s and the exploration starts again.

a. Suppose we discretize the linear track by 6 states, $s_1, \ldots, s_6$. Initialize all the Q-values at zero. How do the Q-values develop as the rat walks down the maze in the first trial?

b. Calculate the Q-values after 3 complete trials. How many Q-value values are non-zero? How many trial do we need so that information about the reward has arrived in the state just 'below' the starting state?

c. What happens to the learning speed if the number of states increases from 6 to 12? How many Q-values are non-zero after 3 trials? How many trial do we need so that information about the reward has arrived in the state just 'below' the starting state?



ure 2: A linear maze.

Your calculations.

Your calculations.

5. Convergence in expectation of SARSA: theorem

Assumption:

The SARSA algo has been applied for a very long time, using updates

$$\Delta Q(s,a) = \eta\,[r_t + \gamma Q(s',a') - Q(s,a)]$$

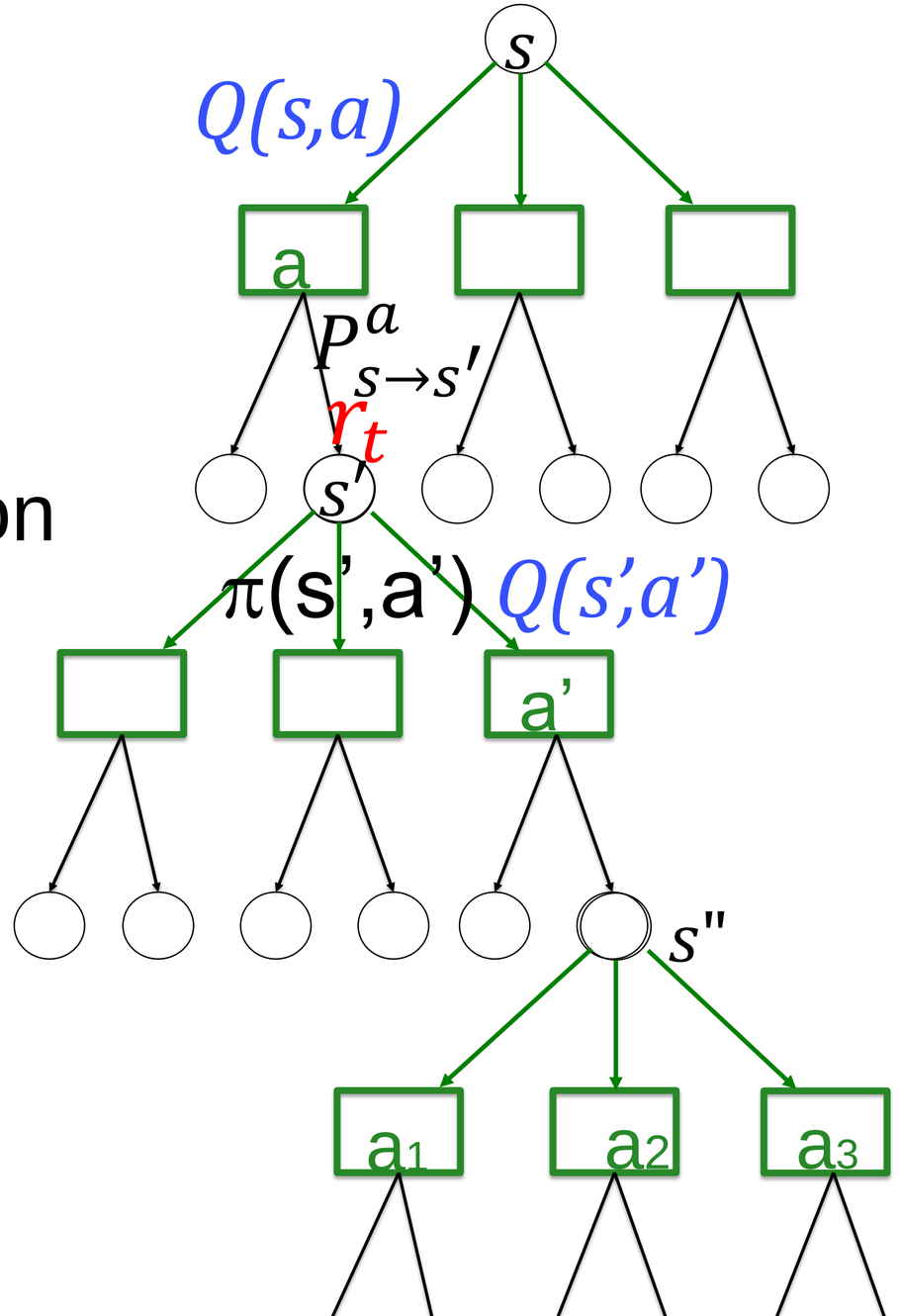
IF all Q-values have converged in expectation

$$\langle \Delta Q(s,a) \rangle = 0$$

THEN

The set of Q-values solves the Bellman eq.

$$Q(s,a) = \sum_{s'} P^{a}_{s\to s'} \left[R^{a}_{s\to s'} + \gamma \sum_{a'} \pi(s',a') Q(s',a') \right]$$



Previous slide.
Similar to the case of the 1-step horizon, we will show now for the multi-step horizon that the iterative update rule SARSA makes the estimated Q-values converge to the correct ones, i.e., the one predicted by the self-consistent solution of the Bellman equation

5. Convergence in expectation of SARSA: theorem

Blackboard7:
SARSA convergence

Look at graph to take expectations:

- if algo in state s, all remaining expectations are “given s”
- if algo on a branch (s,a), all remaining exp. are “given s and a”

Your notes.

5. SARSA algorithm

We have initialized SARSA and played for n>2 steps.
Is the following true for the next steps?

[ ] in SARSA, updates are applied after each move.

[ ] in SARSA, the agent updates the Q-value $Q(s_{(t)}, a_{(t)})$ related to the **current** state $s_{(t)}$

[ ] in SARSA, the agent updates the Q-value $Q(s_{(t-1)}, a_{(t-1)})$ related to the **previous** state, when it is in state $s_{(t)}$

[ ] in SARSA, the agent moves in the environment using the policy $\pi(s, a)$

[ ] SARSA is an online algorithm

Your comments.

Reinforcement Learning and SARSA

Learning outcome and conclusions:

- **Reinforcement Learning is learning by rewards**
  → world is full of rewards (but not full of labels)
- **Agents and actions**
  → agent learns by interacting with the environment
  → state s, action a, reward r
- **Exploration vs Exploitation**
  → optimal actions are easy if we know reward probabilities
  → since we don't know the probabilities we need to explore
- **Bellman equation**
  → self-consistency condition for Q-values
- **SARSA algorithm:** state-action-reward-state-action
  → update while exploring environment with current policy

Your comments.

The END

Teaching monitoring – monitoring of understanding

[ ] today, in the second part, at least 60% of material was new to me.

[ ] in the second part, I have the feeling that I have been able to follow (at least) 80% of the lecture.