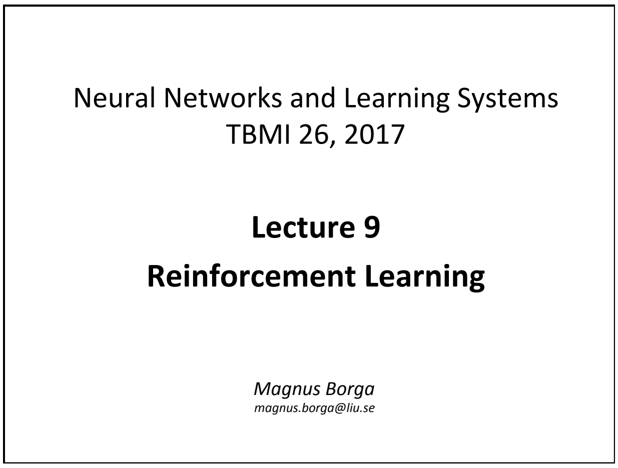
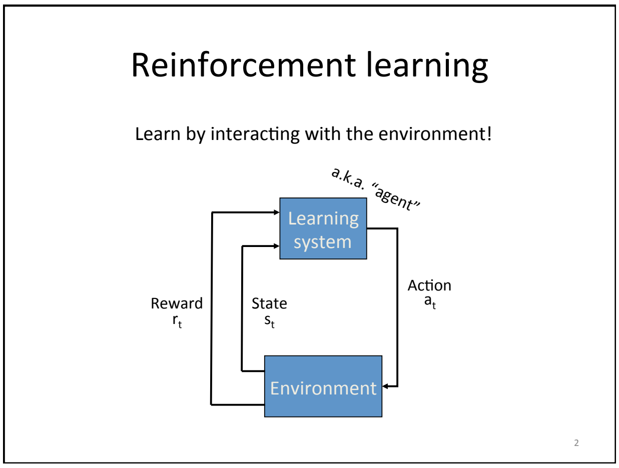


Neural Networks and Learning Systems
TBMI 26, 2017

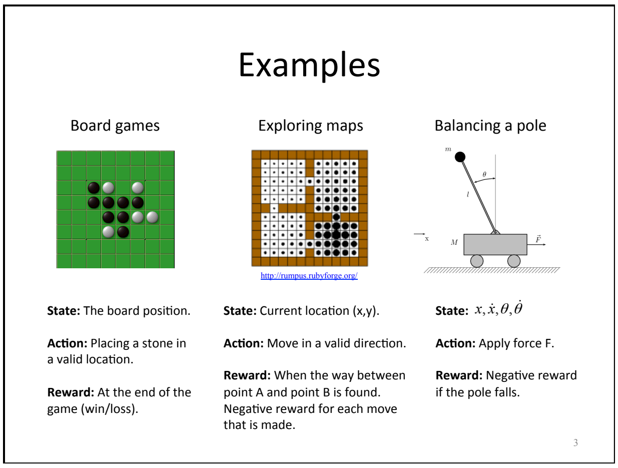
Lecture 9
Reinforcement Learning

Magnus Borga
magnus.borga@liu.se



Reinforcement learning

Learn by interacting with the environment!



Examples

Board games

Exploring maps

Balancing a pole

State: The board position.

Action: Placing a stone in a valid location.

Reward: At the end of the game (win/loss).

State: Current location (x,y).

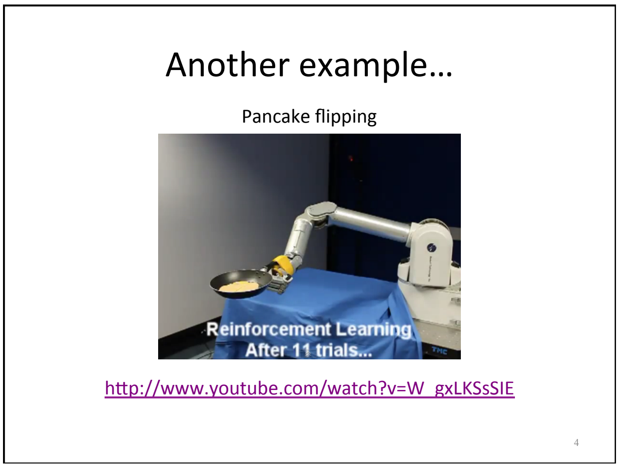
Action: Move in a valid direction.

Reward: When the way between point A and point B is found. Negative reward for each move that is made.

State: $x, \dot{x}, \theta, \dot{\theta}$

Action: Apply force F.

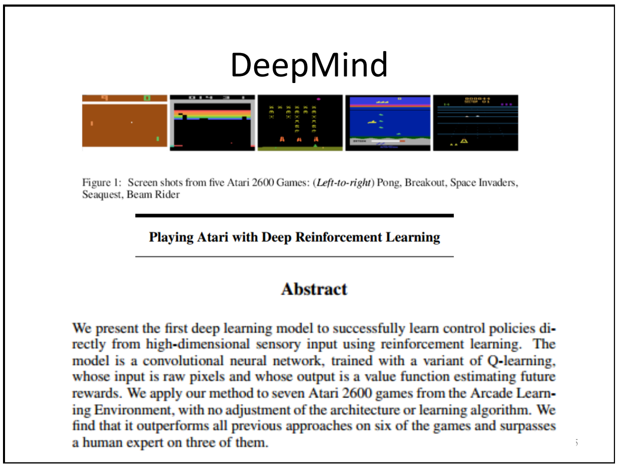
Reward: Negative reward if the pole falls.



Another example…

Pancake flipping

http://www.youtube.com/watch?v=W_gxLKSsSIE



DeepMind

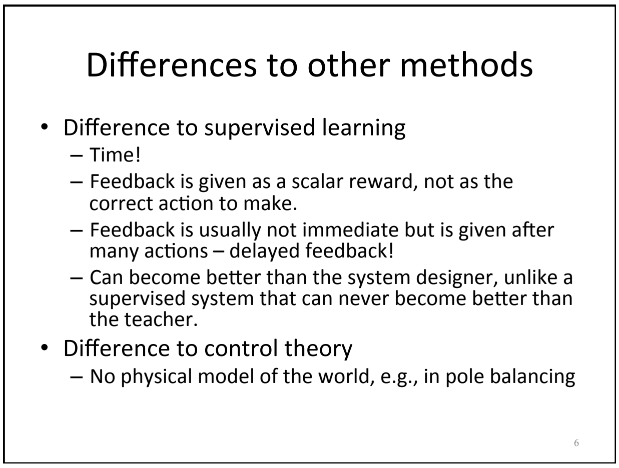
Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

Playing Atari with Deep Reinforcement Learning

Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

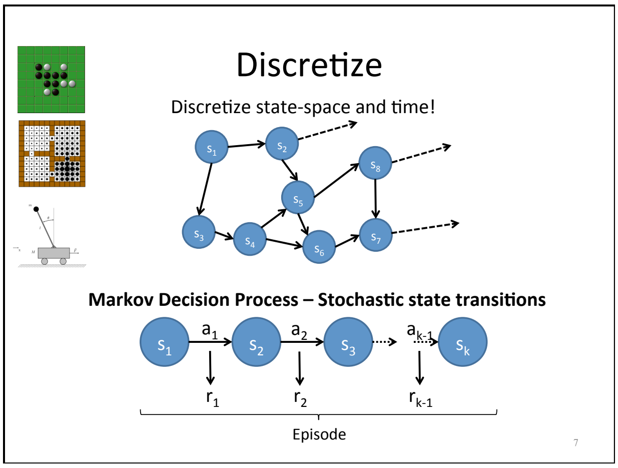
5



Differences to other methods

- Difference to supervised learning
  - Time!
  - Feedback is given as a scalar reward, not as the correct action to make.
  - Feedback is usually not immediate but is given after many actions – delayed feedback!
  - Can become better than the system designer, unlike a supervised system that can never become better than the teacher.
- Difference to control theory
  - No physical model of the world, e.g., in pole balancing

6



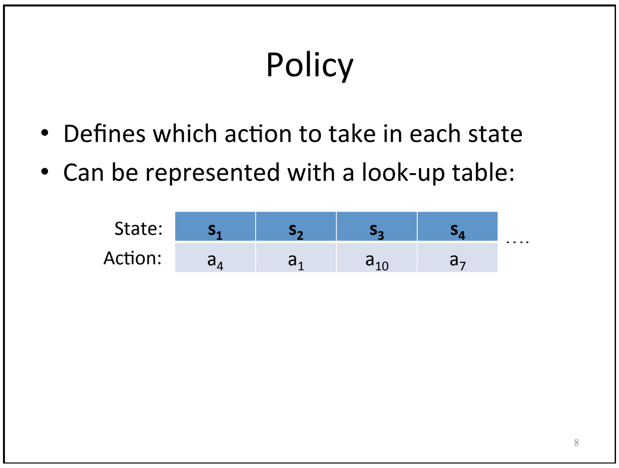
Discretize

Discretize state-space and time!

Markov Decision Process – Stochastic state transitions

Episode

7

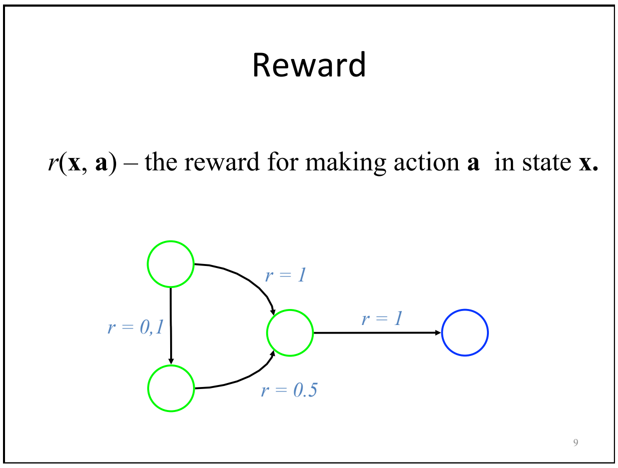


Policy

- Defines which action to take in each state
- Can be represented with a look-up table:

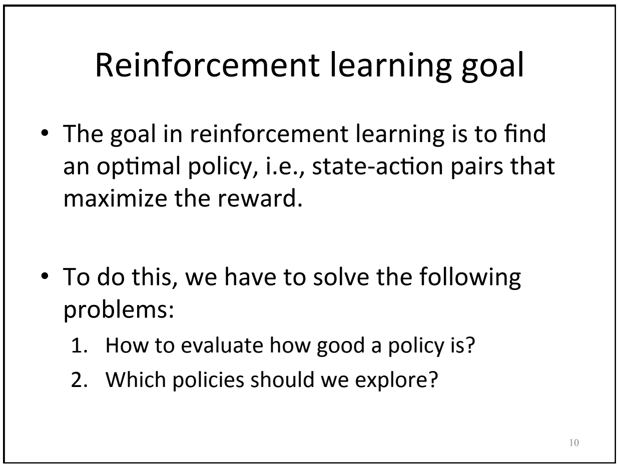
State:	s_1	s_2	s_3	s_4	….
Action:	a_4	a_1	a_{10}	a_7	

8



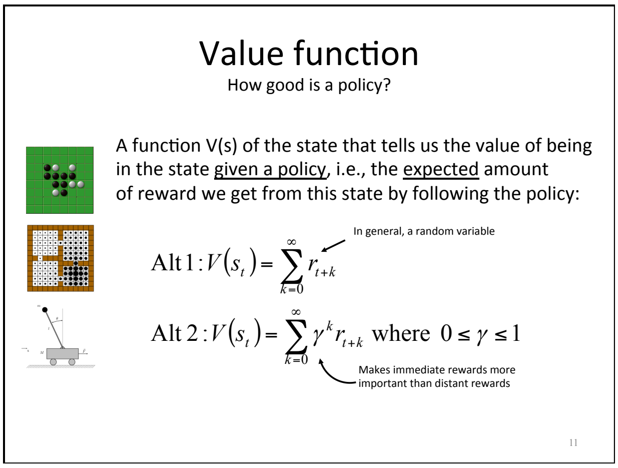
Reward

$r(\mathbf{x}, \mathbf{a})$ – the reward for making action $\mathbf{a}$ in state $\mathbf{x}$.



Reinforcement learning goal

- The goal in reinforcement learning is to find an optimal policy, i.e., state-action pairs that maximize the reward.
- To do this, we have to solve the following problems:
  1. How to evaluate how good a policy is?
  2. Which policies should we explore?



Value function

How good is a policy?

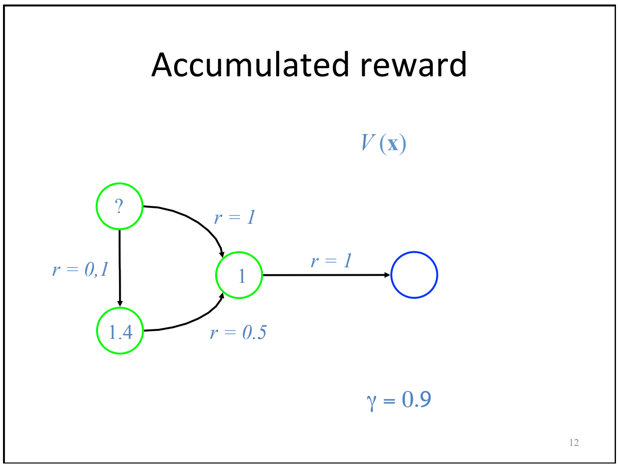
A function V(s) of the state that tells us the value of being in the state given a policy, i.e., the expected amount of reward we get from this state by following the policy:

$$\text{Alt 1}: V(s_t) = \sum_{k=0}^{\infty} r_{t+k}$$

In general, a random variable

$$\text{Alt 2}: V(s_t) = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \quad \text{where} \quad 0 \le \gamma \le 1$$

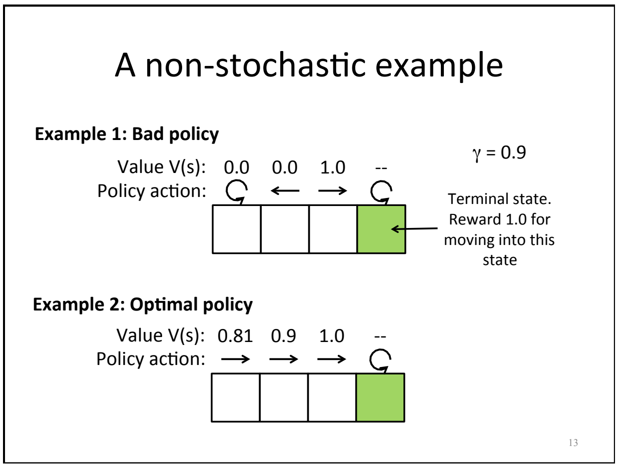
Makes immediate rewards more important than distant rewards



Accumulated reward

$V(\mathbf{x})$

$\gamma = 0.9$



A non-stochastic example

$\gamma = 0.9$

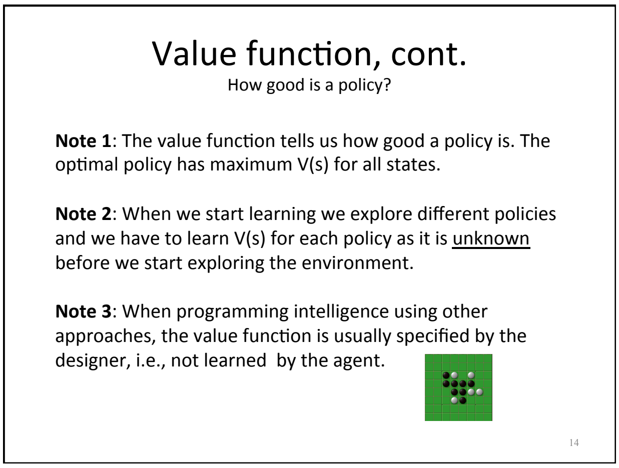
Example 1: Bad policy

Value V(s):	0.0	0.0	1.0	--
Policy action:	↻	←	→	↻

Terminal state. Reward 1.0 for moving into this state

Example 2: Optimal policy

Value V(s):	0.81	0.9	1.0	--
Policy action:	→	→	→	↻



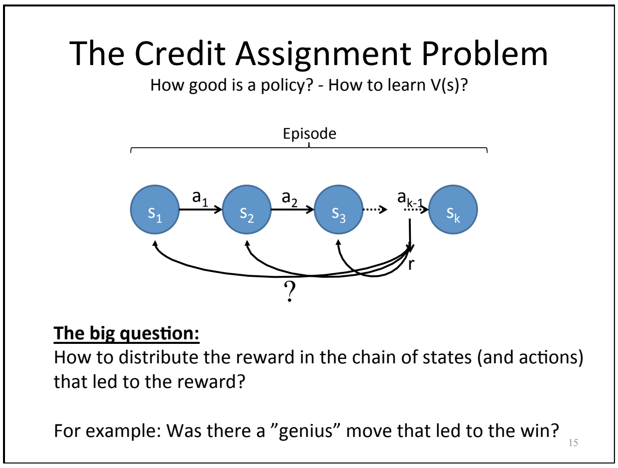
Value function, cont.

How good is a policy?

Note 1: The value function tells us how good a policy is. The optimal policy has maximum V(s) for all states.

Note 2: When we start learning we explore different policies and we have to learn V(s) for each policy as it is unknown before we start exploring the environment.

Note 3: When programming intelligence using other approaches, the value function is usually specified by the designer, i.e., not learned by the agent.



The Credit Assignment Problem

How good is a policy? - How to learn V(s)?

Episode

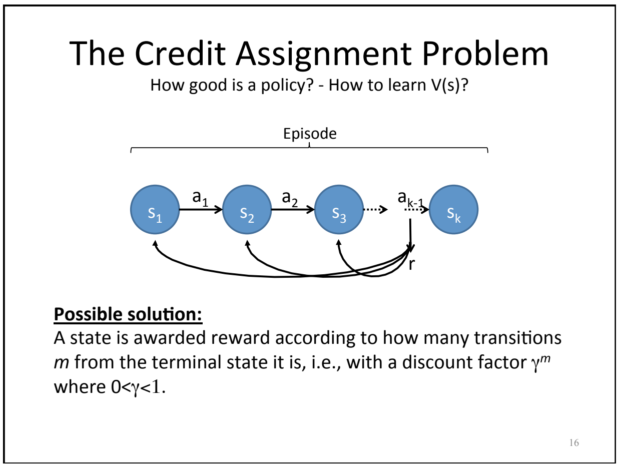
s_1 a_1 s_2 a_2 s_3 a_{k-1} s_k

r

?

The big question:
How to distribute the reward in the chain of states (and actions) that led to the reward?

For example: Was there a ”genius” move that led to the win?



The Credit Assignment Problem

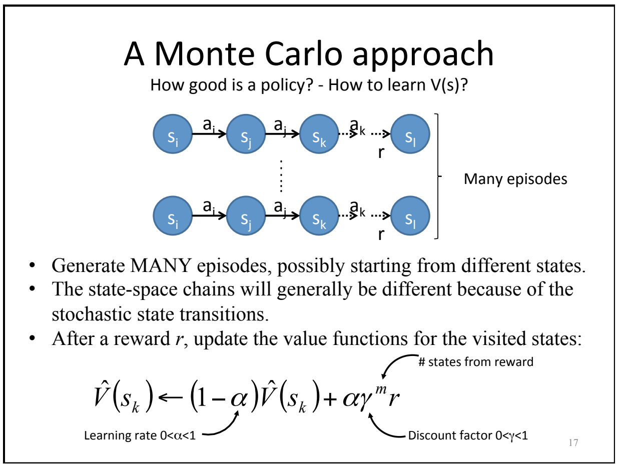
How good is a policy? - How to learn V(s)?

Episode

s_1 a_1 s_2 a_2 s_3 a_{k-1} s_k

r

Possible solution:
A state is awarded reward according to how many transitions *m* from the terminal state it is, i.e., with a discount factor γ^m where $0<\gamma<1$.



A Monte Carlo approach

How good is a policy? - How to learn V(s)?

$s_i \xrightarrow{a_i} s_j \xrightarrow{a_j} s_k \xrightarrow{a_k} \dots \rightarrow s_l$, r

⋮

$s_i \xrightarrow{a_i} s_j \xrightarrow{a_j} s_k \xrightarrow{a_k} \dots \rightarrow s_l$, r

Many episodes

- Generate MANY episodes, possibly starting from different states.
- The state-space chains will generally be different because of the stochastic state transitions.
- After a reward r, update the value functions for the visited states:

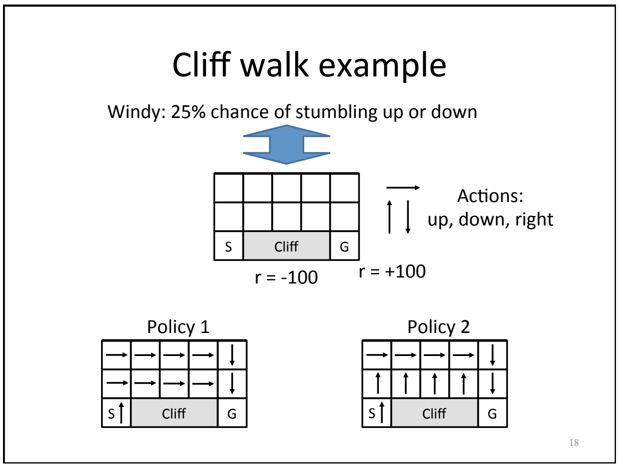
# states from reward

$$\hat{V}(s_k) \leftarrow (1-\alpha)\hat{V}(s_k) + \alpha\gamma^m r$$

Learning rate 0<α<1

Discount factor 0<γ<1

17



Cliff walk example

Windy: 25% chance of stumbling up or down

S	Cliff			G

r = -100 r = +100

→ ↑ ↓ Actions: up, down, right

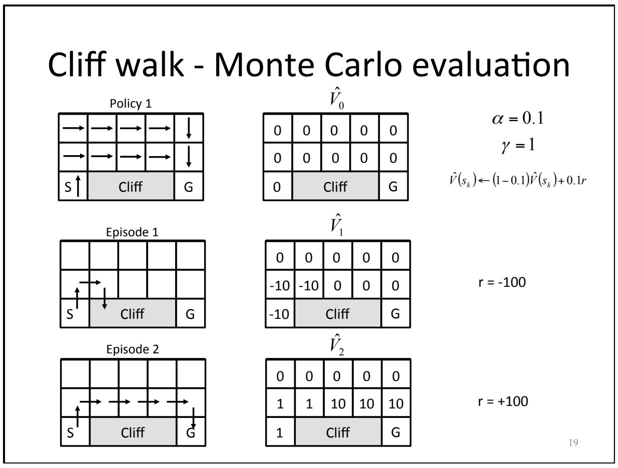
Policy 1

→	→	→	→	↓
→	→	→	→	↓
S ↑	Cliff			G

Policy 2

→	→	→	→	↓
↑	↑	↑	↑	↓
S ↑	Cliff			G

18



Cliff walk - Monte Carlo evaluation

Policy 1

→	→	→	→	↓
→	→	→	→	↓
S ↑	Cliff			G

$\hat{V}_0$

0	0	0	0	0
0	0	0	0	0
0	Cliff			G

$\alpha = 0.1$

$\gamma = 1$

$\hat{V}(s_k) \leftarrow (1-0.1)\hat{V}(s_k) + 0.1r$

Episode 1

↑→	↓			
S	Cliff			G

$\hat{V}_1$

0	0	0	0	0
-10	-10	0	0	0
-10	Cliff			G

r = -100

Episode 2

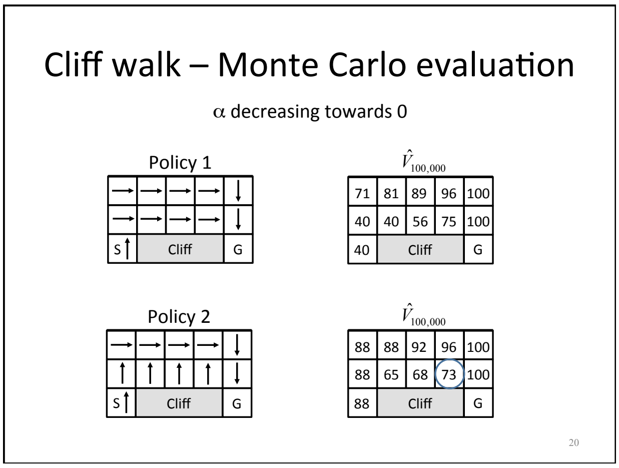
↑→	→	→	→	↓
S	Cliff			G

$\hat{V}_2$

0	0	0	0	0
1	1	10	10	10
1	Cliff			G

r = +100

19



Cliff walk – Monte Carlo evaluation

α decreasing towards 0

Policy 1

→	→	→	→	↓
→	→	→	→	↓
S ↑	Cliff			G

$\hat{V}_{100,000}$

71	81	89	96	100
40	40	56	75	100
40	Cliff			G

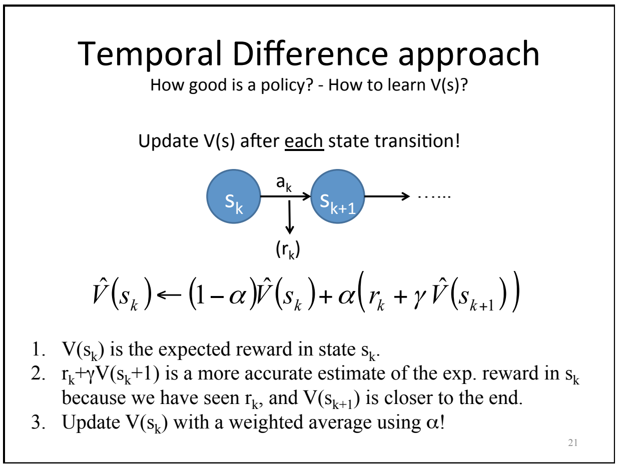
Policy 2

→	→	→	→	↓
↑	↑	↑	↑	↓
S ↑	Cliff			G

$\hat{V}_{100,000}$

88	88	92	96	100
88	65	68	73	100
88	Cliff			G

20



Temporal Difference approach

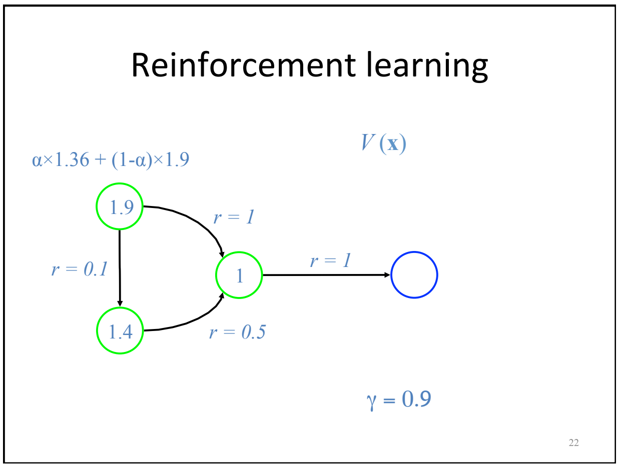
How good is a policy? - How to learn V(s)?

Update V(s) after <u>each</u> state transition!

$$\hat{V}(s_k) \leftarrow (1-\alpha)\hat{V}(s_k) + \alpha\left(r_k + \gamma\hat{V}(s_{k+1})\right)$$

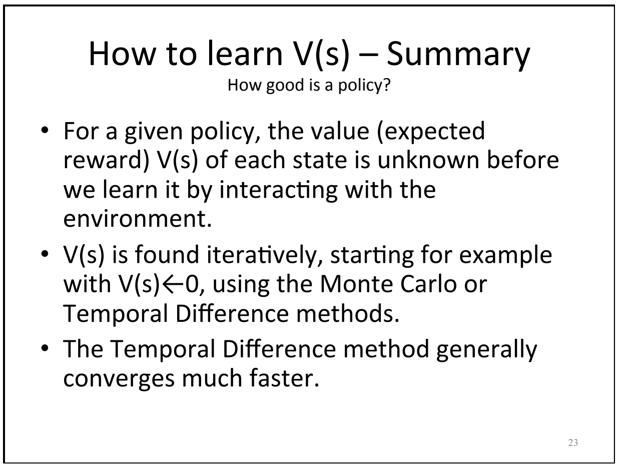
1. $V(s_k)$ is the expected reward in state s_k.
2. $r_k+\gamma V(s_k+1)$ is a more accurate estimate of the exp. reward in s_k because we have seen r_k, and $V(s_{k+1})$ is closer to the end.
3. Update $V(s_k)$ with a weighted average using α!

21



Reinforcement learning

22

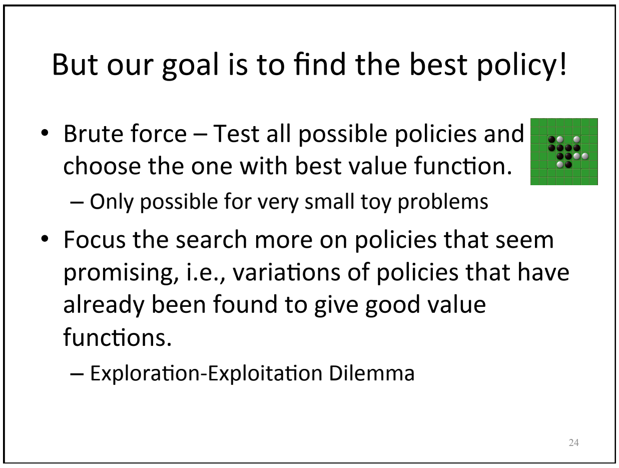


How to learn V(s) – Summary

How good is a policy?

- For a given policy, the value (expected reward) V(s) of each state is unknown before we learn it by interacting with the environment.
- V(s) is found iteratively, starting for example with V(s)←0, using the Monte Carlo or Temporal Difference methods.
- The Temporal Difference method generally converges much faster.

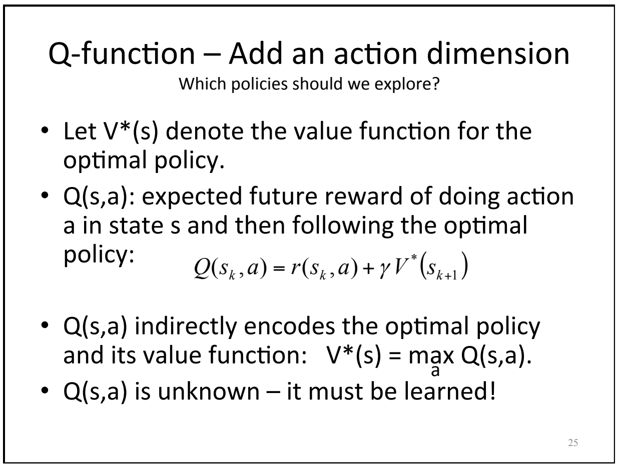
23



But our goal is to find the best policy!

- Brute force – Test all possible policies and choose the one with best value function.
  - Only possible for very small toy problems
- Focus the search more on policies that seem promising, i.e., variations of policies that have already been found to give good value functions.
  - Exploration-Exploitation Dilemma

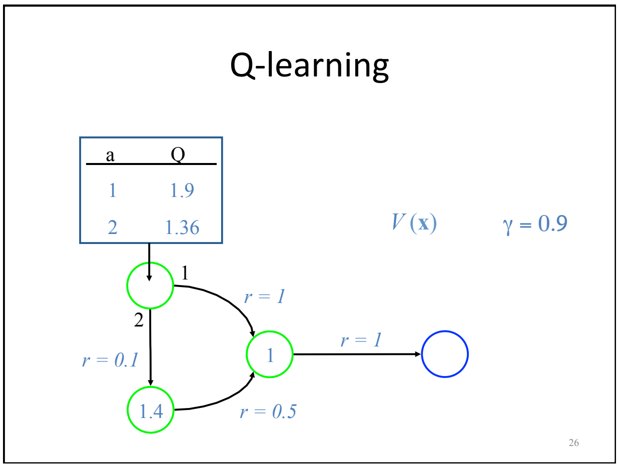
24



Q-function – Add an action dimension

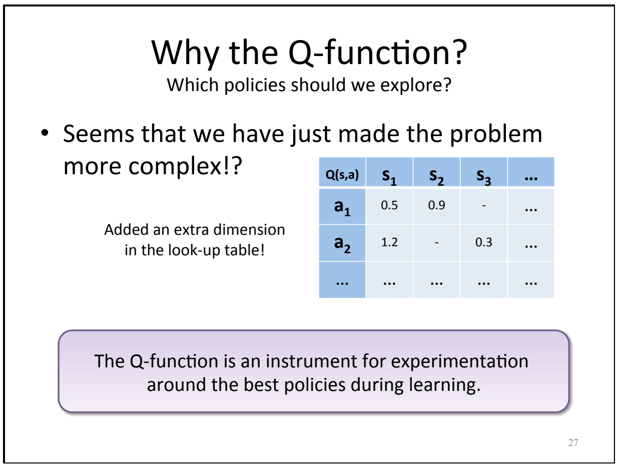
Which policies should we explore?

- Let V*(s) denote the value function for the optimal policy.
- Q(s,a): expected future reward of doing action a in state s and then following the optimal policy: $Q(s_k,a) = r(s_k,a) + \gamma V^*(s_{k+1})$
- Q(s,a) indirectly encodes the optimal policy and its value function: $V^*(s) = \max_a Q(s,a)$.
- Q(s,a) is unknown – it must be learned!



Q-learning

a	Q
1	1.9
2	1.36



Why the Q-function?

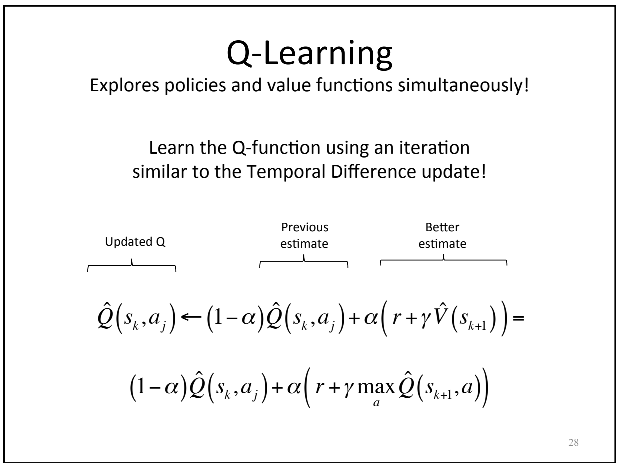
Which policies should we explore?

- Seems that we have just made the problem more complex!?

Added an extra dimension in the look-up table!

Q(s,a)	s_1	s_2	s_3	…
a_1	0.5	0.9	-	…
a_2	1.2	-	0.3	…
…	…	…	…	…

The Q-function is an instrument for experimentation around the best policies during learning.



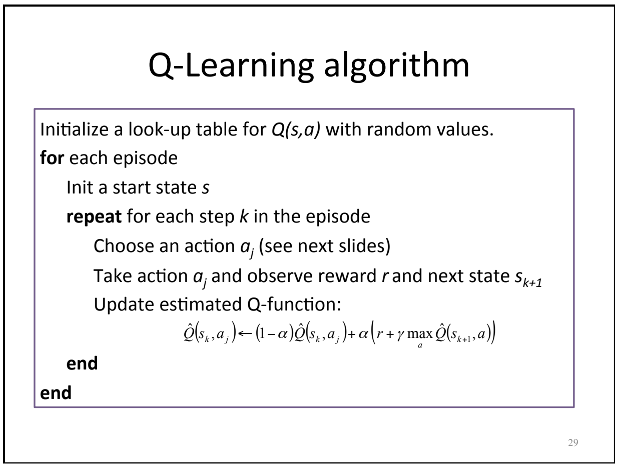
Q-Learning

Explores policies and value functions simultaneously!

Learn the Q-function using an iteration similar to the Temporal Difference update!

$$\overbrace{\hat{Q}(s_k,a_j)}^{\text{Updated Q}} \leftarrow (1-\alpha)\overbrace{\hat{Q}(s_k,a_j)}^{\text{Previous estimate}} + \alpha\overbrace{\left(r+\gamma\hat{V}(s_{k+1})\right)}^{\text{Better estimate}} =$$

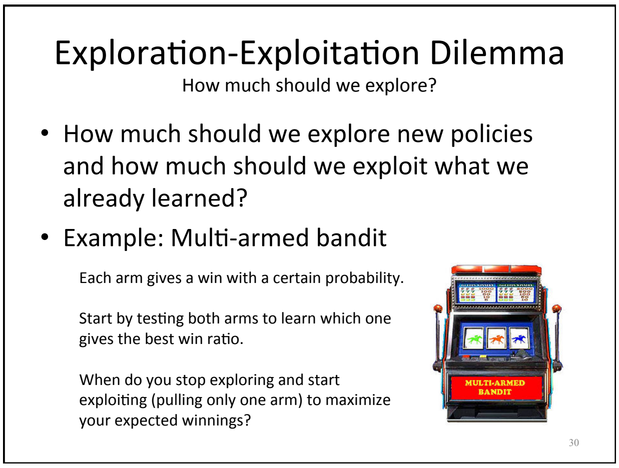
$$(1-\alpha)\hat{Q}(s_k,a_j) + \alpha\left(r+\gamma\max_a \hat{Q}(s_{k+1},a)\right)$$



Q-Learning algorithm

```
Initialize a look-up table for Q(s,a) with random values.
for each episode
    Init a start state s
    repeat for each step k in the episode
        Choose an action $a_j$ (see next slides)
        Take action $a_j$ and observe reward r and next state $s_{k+1}$
        Update estimated Q-function:
            $\hat{Q}(s_k,a_j) \leftarrow (1-\alpha)\hat{Q}(s_k,a_j) + \alpha\left(r + \gamma \max_a \hat{Q}(s_{k+1},a)\right)$
    end
end
```

29



Exploration-Exploitation Dilemma

How much should we explore?

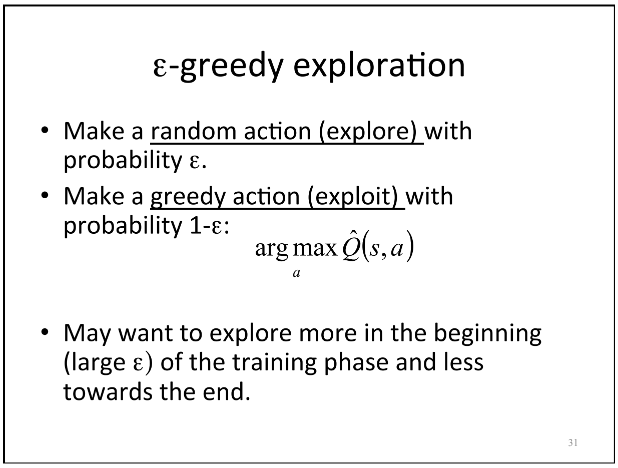
- How much should we explore new policies and how much should we exploit what we already learned?
- Example: Multi-armed bandit

Each arm gives a win with a certain probability.

Start by testing both arms to learn which one gives the best win ratio.

When do you stop exploring and start exploiting (pulling only one arm) to maximize your expected winnings?

30



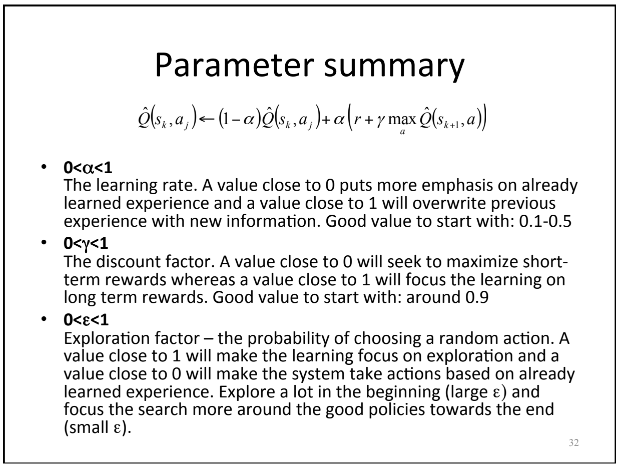
ε-greedy exploration

- Make a random action (explore) with probability ε.
- Make a greedy action (exploit) with probability 1-ε:

$$\arg\max_a \hat{Q}(s,a)$$

- May want to explore more in the beginning (large ε) of the training phase and less towards the end.

31



Parameter summary

$$\hat{Q}(s_k,a_j) \leftarrow (1-\alpha)\hat{Q}(s_k,a_j) + \alpha\left(r + \gamma \max_a \hat{Q}(s_{k+1},a)\right)$$

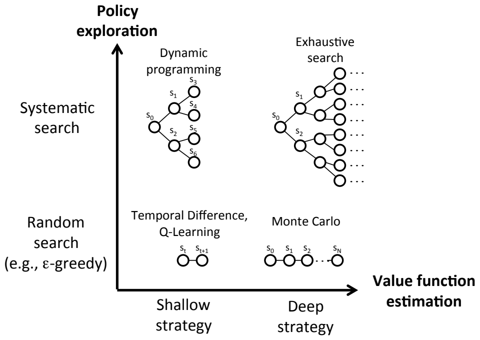
- **0<α<1**
  The learning rate. A value close to 0 puts more emphasis on already learned experience and a value close to 1 will overwrite previous experience with new information. Good value to start with: 0.1-0.5
- **0<γ<1**
  The discount factor. A value close to 0 will seek to maximize short-term rewards whereas a value close to 1 will focus the learning on long term rewards. Good value to start with: around 0.9
- **0<ε<1**
  Exploration factor – the probability of choosing a random action. A value close to 1 will make the learning focus on exploration and a value close to 0 will make the system take actions based on already learned experience. Explore a lot in the beginning (large ε) and focus the search more around the good policies towards the end (small ε).

32

Summary Q-Learning

- Solves the two problems simultaneously:
  1. How to evaluate how good a policy is?
  2. Which policies should we explore?
- The look-up table quickly becomes (too) big.
- Lots of demos and applets on the internet.

Method summary



Policy exploration

Systematic search

Random search (e.g., ε-greedy)

Dynamic programming

Exhaustive search

Temporal Difference, Q-Learning

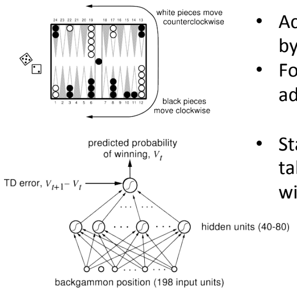
Monte Carlo

Value function estimation

Shallow strategy

Deep strategy

Application: TD-Gammon



white pieces move counterclockwise
black pieces move clockwise
predicted probability of winning, V_t
TD error, $V_{t+1} - V_t$
hidden units (40-80)
backgammon position (198 input units)

- Achieved world-class performance by playing against itself
- Found moves that now have been adopted by expert humans.

- State-space too big for a look-up table. Value function approximated with a neural network.

Figures from *Reinforcement Learning: An Introduction* by R. Sutton.

Reinforcement Learning - Summary

- Learning by trial-and-error
- Often hard to tell *how* a task should be solved but easy to tell *if* and *how well* it has been solved.
- Still mainly a research field
  - Many principally important small-scale applications and demonstrations.
  - Fewer large-scale, real-life applications.
- Strong links to dynamic optimization and optimal control fields.