Reinforcement learning

Applied Machine Learning (EDAN95)
Lectures13 and 14
2018-12-17 and 2018-12-19
Elin A. Topp

Material based on "Hands-on Machine Learning with SciKit-learn and TensorFlow" (course book, chapter 16),
and on lecture "Belöningsbaserad inlärning / Reinforcement learning"
by Örjan Ekeberg, CSC/Nada, KTH, autumn term 2006 (in Swedish)

Outline

- Reinforcement learning
  - Problem definition
    - Learning situation
    - Role of the reward
    - Simplified assumptions
    - Central concepts and terms
  - Known environment
    - Bellman’s equation
    - Approaches to solutions
  - Unknown environment
    - Temporal-Difference learning
    - Q-Learning
    - Sarsa-Learning
  - Improvements
    - The usefulness of making mistakes
    - Eligibility Trace

Outline

- Reinforcement learning
  - Problem definition
    - Learning situation
    - Role of the reward
    - Simplified assumptions
    - Central concepts and terms
  - Known environment
    - Bellman's equation
    - Approaches to solutions
    - Outlook: unknown environments, Monte Carlo method and policy gradients
  - Unknown environment
    - Temporal-Difference learning
    - Q-Learning
    - Sarsa-Learning
  - Improvements
    - The usefulness of making mistakes
    - Eligibility Trace

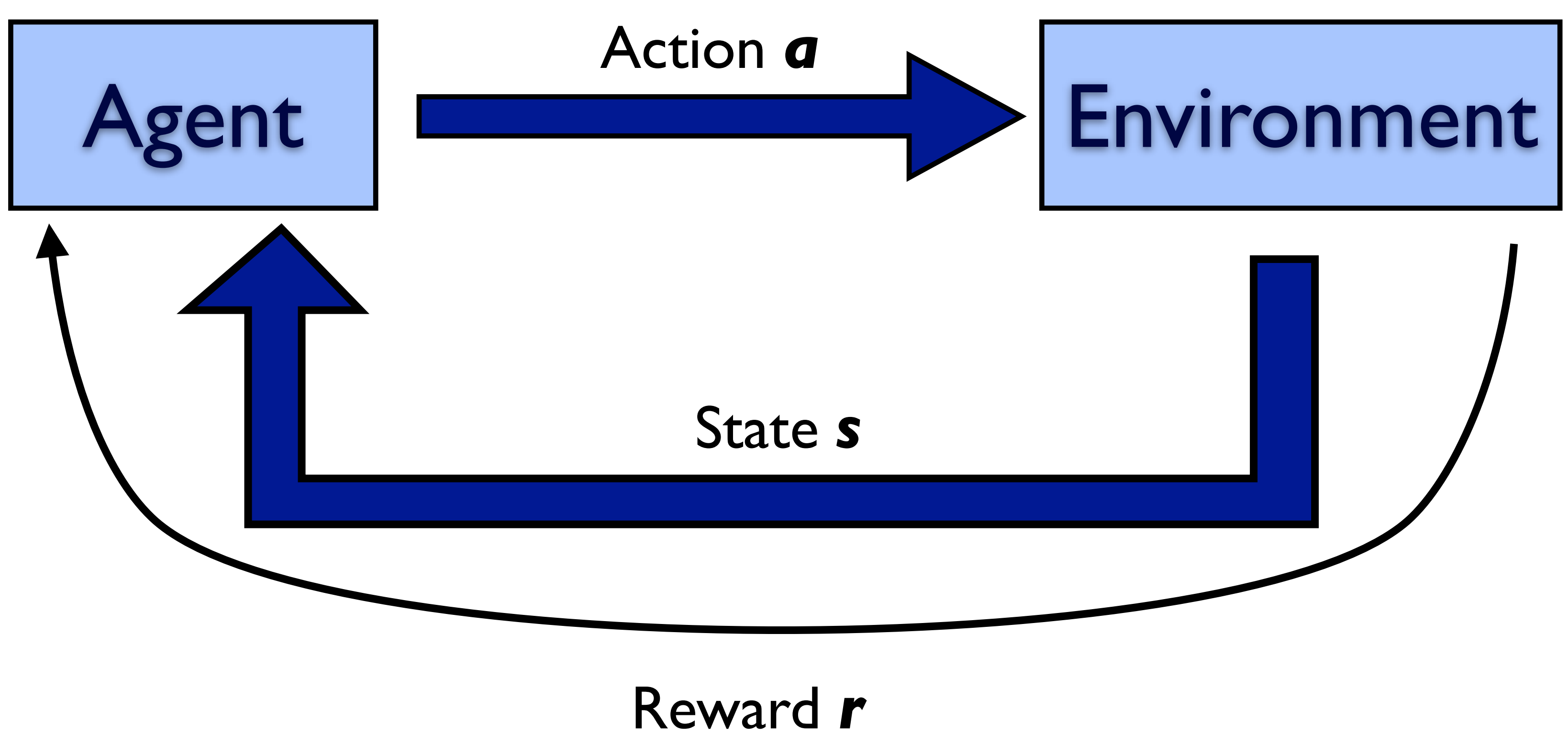
Learning situation: A model

An *agent* interacts with its *environment*

The agent performs *actions*

Actions have *influence* on the environment's *state*

The agent *observes* the environment's *state* and *receives* a *reward* from the environment



Real life examples

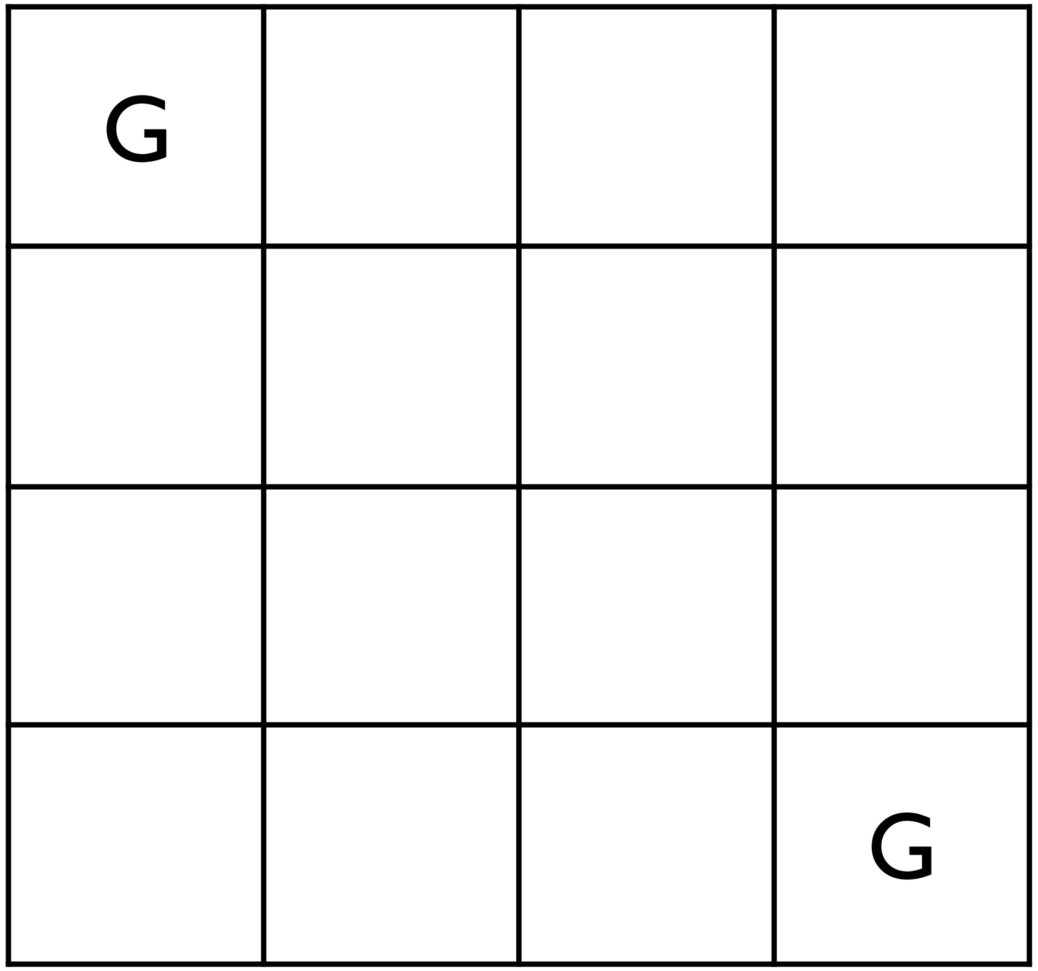
Real life examples

Riding a bicycle

Powder skiing

A classic example: Grid World

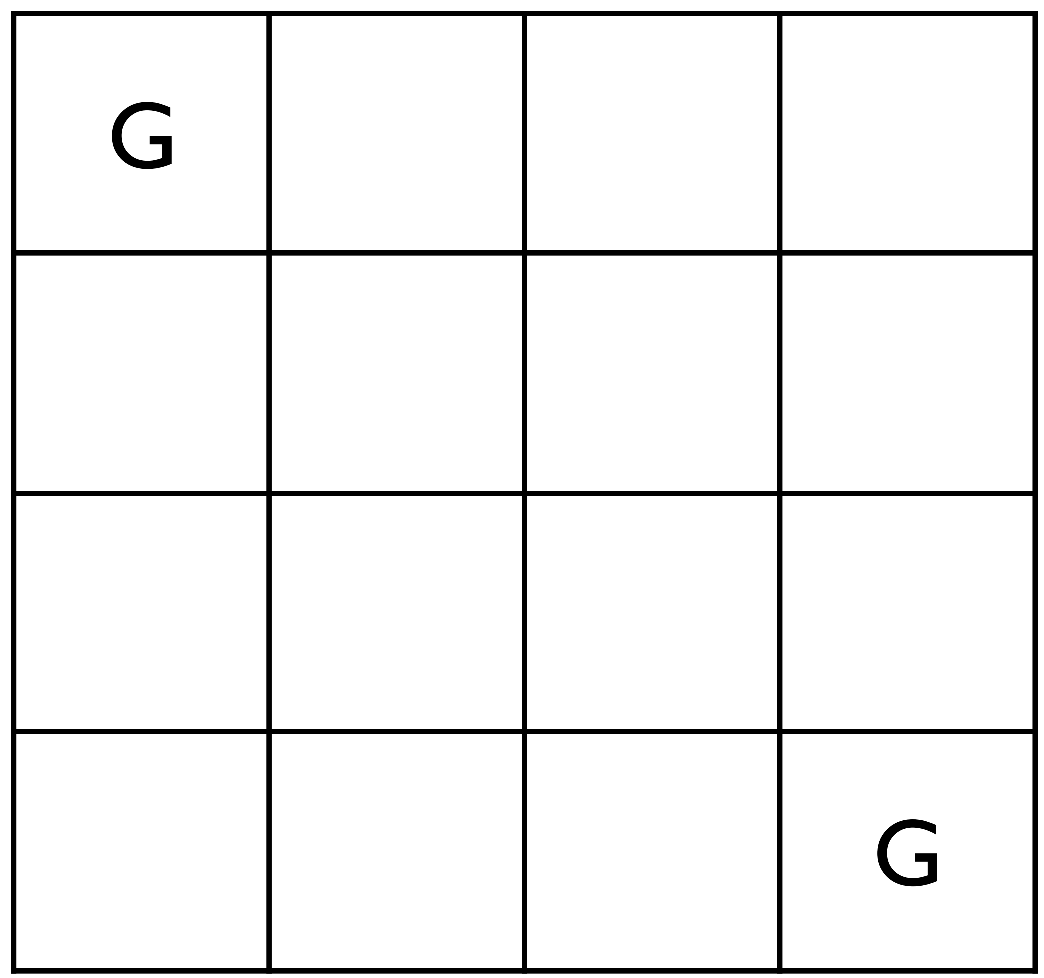
Simplified "Wumpus world" with just two gold pieces



A classic example: Grid World

Simplified "Wumpus world" with just two gold pieces

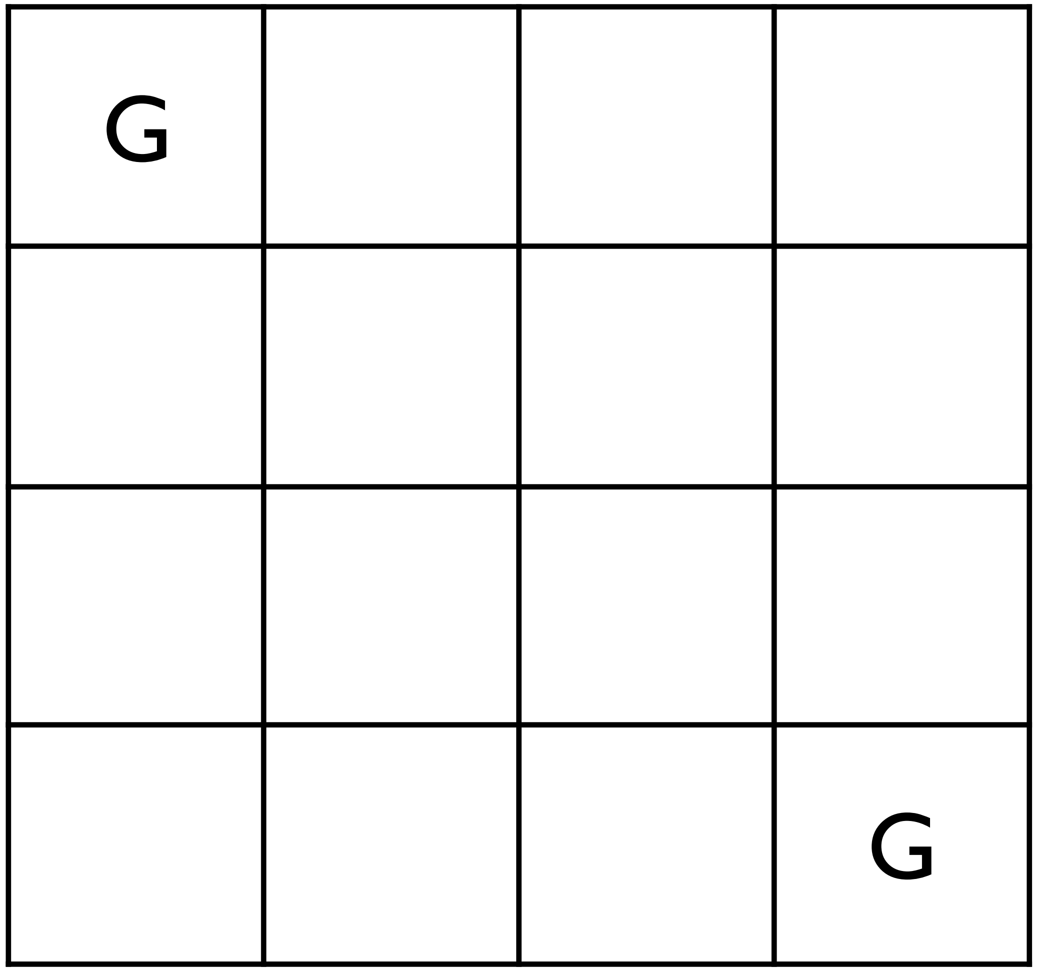
- Every state s_j is represented by a field in the grid



A classic example: Grid World

Simplified "Wumpus world" with just two gold pieces

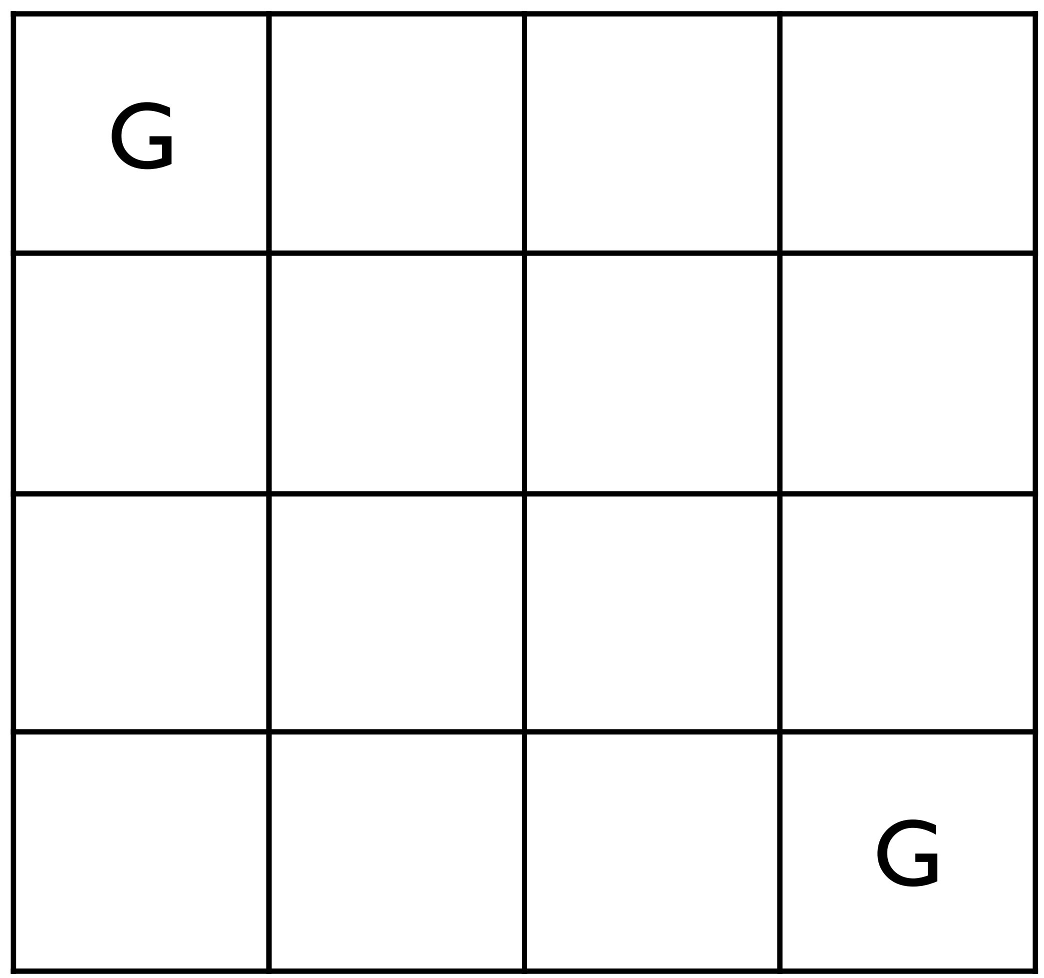
- Every state s_j is represented by a field in the grid
- Action a the agent can choose consists of moving one step to a neighbouring field



A classic example: Grid World

Simplified "Wumpus world" with just two gold pieces

- Every state s_j is represented by a field in the grid
- Action *a* the agent can choose consists of moving one step to a neighbouring field
- *Reward*: -1 in every step until one of the goals (G) is reached.



Reinforcement learning

Learning of a behaviour (a strategy, a skill) without access to a right / wrong measure for actions and decisions taken.

Reinforcement learning

Learning of a behaviour (a strategy, a skill) without access to a right / wrong measure for actions and decisions taken.

With the help of a *reward*, a measure is given, of *how well things are going*

Reinforcement learning

Learning of a behaviour (a strategy, a skill) without access to a right / wrong measure for actions and decisions taken.

With the help of a *reward*, a measure is given, of *how well things are going*

Note: The reward is not given in direct connection with a good choice of action *(temporal credit assignment)*

Reinforcement learning

Learning of a behaviour (a strategy, a skill) without access to a right / wrong measure for actions and decisions taken.

With the help of a *reward*, a measure is given, of *how well things are going*

Note: The reward is not given in direct connection with a good choice of action *(temporal credit assignment)*

Note: The reward does not tell what exactly it was, that made the action "good" *(structural credit assignment)*

Learning situation: The agent's task

The task:

Find a behaviour (action sequence) that maximises the overall reward

How long into the future should we spy?

Finite time horizon:

$$max\ E\left[\ \sum_{t=0}^{h} r_t\right]$$

Infinite time horizon:

$$max\ E\left[\ \sum_{t=0}^{\infty} \gamma^t r_t\right]$$

with γ being a discount factor for future rewards ($0 < \gamma < 1$)

The reward function's role

The reward function depends on the type of task

The reward function's role

The reward function depends on the type of task

- Game (Chess, Backgammon, Go): Reward is given only in the end of the game, +1 for "win", -1 for "loose"

The reward function's role

The reward function depends on the type of task

- Game (Chess, Backgammon, Go): Reward is given only in the end of the game, +1 for "win", -1 for "loose"
- Avoid mistakes (Riding a bike, Learning to fly according to Hitchhiker's Guide to the Galaxy): Reward -1 when failing (falling)

The reward function's role

The reward function depends on the type of task

- Game (Chess, Backgammon, Go): Reward is given only in the end of the game, +1 for "win", -1 for "loose"
- Avoid mistakes (Riding a bike, Learning to fly according to Hitchhiker's Guide to the Galaxy): Reward -1 when failing (falling)
- Avoid mistakes and try to do something useful (Learning to walk towards a goal): Reward -10 when failing (falling) or -5 when moving backwards, +5 when an action leads to a forward movement

The reward function's role

The reward function depends on the type of task

- Game (Chess, Backgammon, Go): Reward is given only in the end of the game, +1 for "win", -1 for "loose"
- Avoid mistakes (Riding a bike, Learning to fly according to Hitchhiker's Guide to the Galaxy): Reward -1 when failing (falling)
- Avoid mistakes and try to do something useful (Learning to walk towards a goal): Reward -10 when failing (falling) or -5 when moving backwards, +5 when an action leads to a forward movement
- Find the shortest / cheapest / fastest path to a goal: Reward -1 for each step that does not end in the goal

Simplifying assumptions

Simplifying assumptions

We assume for now:

Simplifying assumptions

We assume for now:

- Discrete time (steps over time)

Simplifying assumptions

We assume for now:

- Discrete time (steps over time)
- Finite number of possible actions a_i

$$a_i \in a_1, a_2, a_3, \ldots, a_n$$

Simplifying assumptions

We assume for now:

- Discrete time (steps over time)
- Finite number of possible actions a_i

$$a_i \in a_1, a_2, a_3, \ldots, a_n$$

- Finite number of states s_j

$$s_j \in s_1, s_2, s_3, \ldots, s_m$$

Simplifying assumptions

We assume for now:

- Discrete time (steps over time)
- Finite number of possible actions a_i

  $a_i \in a_1, a_2, a_3, \dots, a_n$

- Finite number of states s_j

  $s_j \in s_1, s_2, s_3, \dots, s_m$

- The context is a constant (stationary) MDP (*Markov Decision Process*), where reward and new state s' only depend on s, a, and (random) *noise*

Simplifying assumptions

We assume for now:

- Discrete time (steps over time)
- Finite number of possible actions a_i

$$a_i \in a_1, a_2, a_3, \dots, a_n$$

- Finite number of states s_j

$$s_j \in s_1, s_2, s_3, \dots, s_m$$

- The context is a constant (stationary) MDP (*Markov Decision Process*), where reward and new state *s'* only depend on *s*, *a*, and (random) *noise*
- Environment is observable

The agent's internal representation

The agent's internal representation

- An agent's *policy* π is the "rule" after which the agent chooses its action a in a given state s

$$\pi(s) \longmapsto a$$

The agent's internal representation

- An agent's *policy* π is the "rule" after which the agent chooses its action a in a given state s

$$\pi(s) \longmapsto a$$

- An agent's *utility function* U describes the expected future reward given s, when following policy π

$$U^{\pi}(s) \longmapsto \mathbb{R}$$

Grid World: A state's value

A state's value depends on the chosen policy

Grid World: A state's value

A state's value depends on the chosen policy

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

U with optimal policy

Grid World: A state's value

A state's value depends on the chosen policy

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

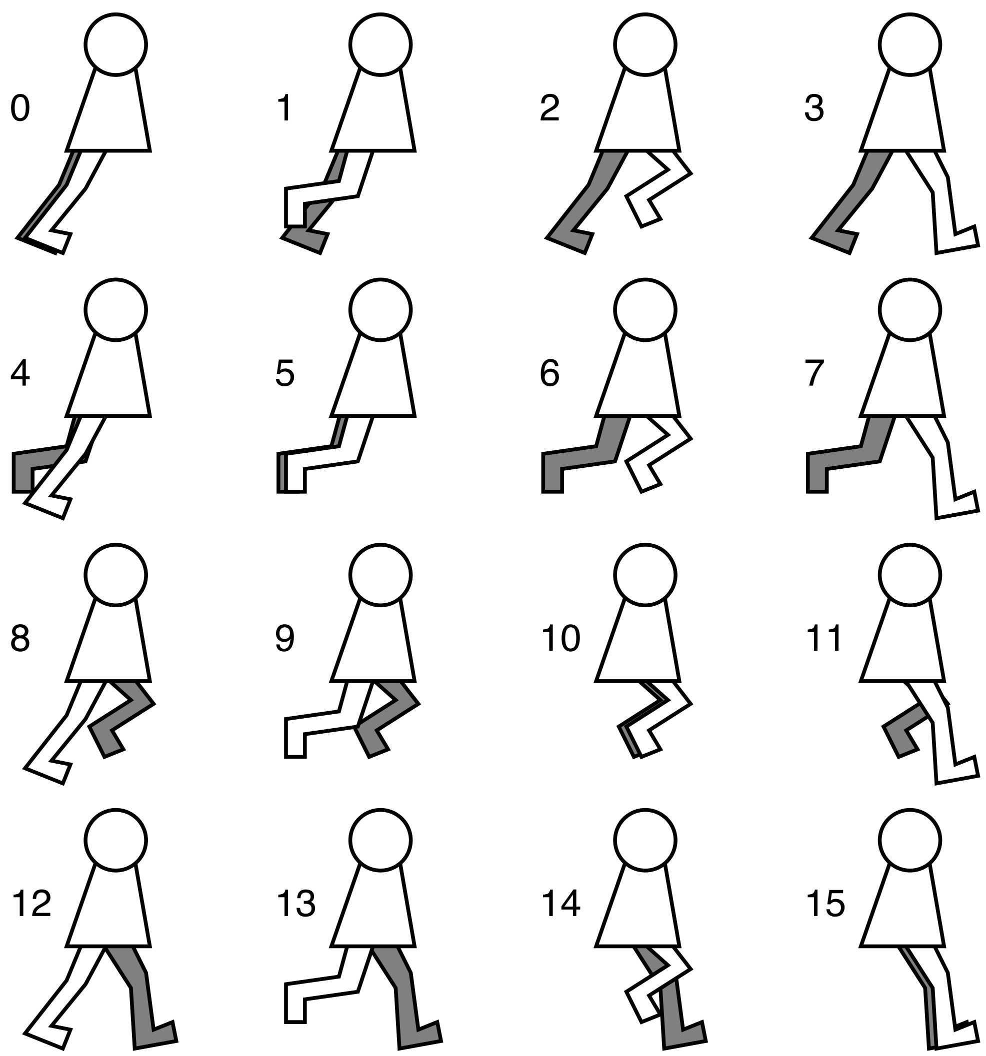
U with optimal policy

0	-14	-20	-22
-14	-18	-22	-20
-20	-22	-18	-14
-22	-20	-14	0

U with random policy

Cartoon Walker

16 discrete states, some really bad, 4 discrete actions, only some making the walker walk



Action	Effect
0	Move right (white) leg up / down
1	Move right (white) leg backward / forward
2	Move left (grey) leg up / down
3	Move left (grey) leg backward / forward

Bayesian reinforcement learning

Bayesian reinforcement learning

A remark:

Bayesian reinforcement learning

A remark:

One form of reinforcement learning integrates Bayesian learning into the process to obtain the transition model, i.e., $P(\, s' \mid s, \pi(s))$

Bayesian reinforcement learning

A remark:

One form of reinforcement learning integrates Bayesian learning into the process to obtain the transition model, i.e., $P(s' \mid s, \pi(s))$

This means to assume a prior probability for each hypothesis on how the model might look like and then applying Bayes' rule to obtain the posterior.

Bayesian reinforcement learning

A remark:

One form of reinforcement learning integrates Bayesian learning into the process to obtain the transition model, i.e., $P(s' \mid s, \pi(s))$

This means to assume a prior probability for each hypothesis on how the model might look like and then applying Bayes' rule to obtain the posterior.

We are not going into details here!

Outline

- Reinforcement learning
  - Problem definition
    - Learning situation
    - Roll of the reward
    - Simplified assumptions
    - Central concepts and terms
  - Known (observable) environment
    - Bellman’s equation
    - Approaches to solutions
    - Outlook: unknown environments, Monte Carlo method and policy gradients
  - Unknown environment
    - Temporal-Difference learning
    - Q-Learning
    - Sarsa-Learning
  - Improvements
    - The usefulness of making mistakes
    - Eligibility Trace

Environment model

Environment model

- Where do we get in each step?

$$\delta(s, a) \longmapsto s'$$

Environment model

- Where do we get in each step?

$$\delta(s, a) \longmapsto s'$$

- What will the reward be?

$$r(s, a) \longmapsto \mathbb{R}$$

Environment model

- Where do we get in each step?

$$\delta(s, a) \longmapsto s'$$

- What will the reward be?

$$r(s, a) \longmapsto \mathbb{R}$$

Environment model

- Where do we get in each step?

$$\delta(s, a) \longmapsto s'$$

- What will the reward be?

$$r(s, a) \longmapsto \mathbb{R}$$

The utility values of different states obey *Bellman's equation,* given a fixed policy π:

$$U^{\pi}(s) = r(s, \pi(s)) + \gamma \cdot U^{\pi}(\delta(s, \pi(s)))$$

Solving the equation

There are two ways of solving (this "optimal" version of) *Bellman's equation*

$$U^{\pi}(s) = r(s, \pi(s)) + \gamma \cdot U^{\pi}(\delta(s, \pi(s)))$$

- Directly: $U^{\pi}(s) = r(s, \pi(s)) + \gamma \cdot \sum_{s'} P(s' \mid s, \pi(s))\, U^{\pi}(s')$

- Iteratively (*Value / utility iteration*), stop when equilibrium is reached, i.e., "nothing happens"

$$U^{\pi}_{k+1}(s) \longleftarrow r(s, \pi(s)) + \gamma \cdot U^{\pi}_{k}(\delta(s, \pi(s)))$$

Finding optimal policy and value function

Finding optimal policy and value function

How can we find an *optimal policy* π^*?

Finding optimal policy and value function

How can we find an *optimal policy* π^*?

That would be easy if we had the *optimal value / utility function* U^*:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\left(r(s, a) + \gamma \cdot U^*(\delta(s, a))\right)$$

Finding optimal policy and value function

How can we find an *optimal policy* π^*?

That would be easy if we had the *optimal value / utility function* U^*:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\big(\, r(s, a) + \gamma \cdot U^*(\delta(s, a))\big)$$

Apply to the "optimal version" of Bellman's equation

$$U^*(s) = \max_{a}\big(\, r(s, a) + \gamma \cdot U^*(\delta(s, a))\big)$$

Finding optimal policy and value function

How can we find an *optimal policy* π^*?

That would be easy if we had the *optimal value / utility function* U^*:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}(\, r(s, a) + \gamma \cdot U^*(\delta(s, a)))$$

Apply to the "optimal version" of Bellman's equation

$$U^*(s) = \max_{a}(\, r(s, a) + \gamma \cdot U^*(\delta(s, a)))$$

Tricky to solve ... but possible:

Combine policy and value iteration by switching in each iteration step

Policy iteration

Policy iteration

Policy iteration provides exactly this switch.

Policy iteration

Policy iteration provides exactly this switch.

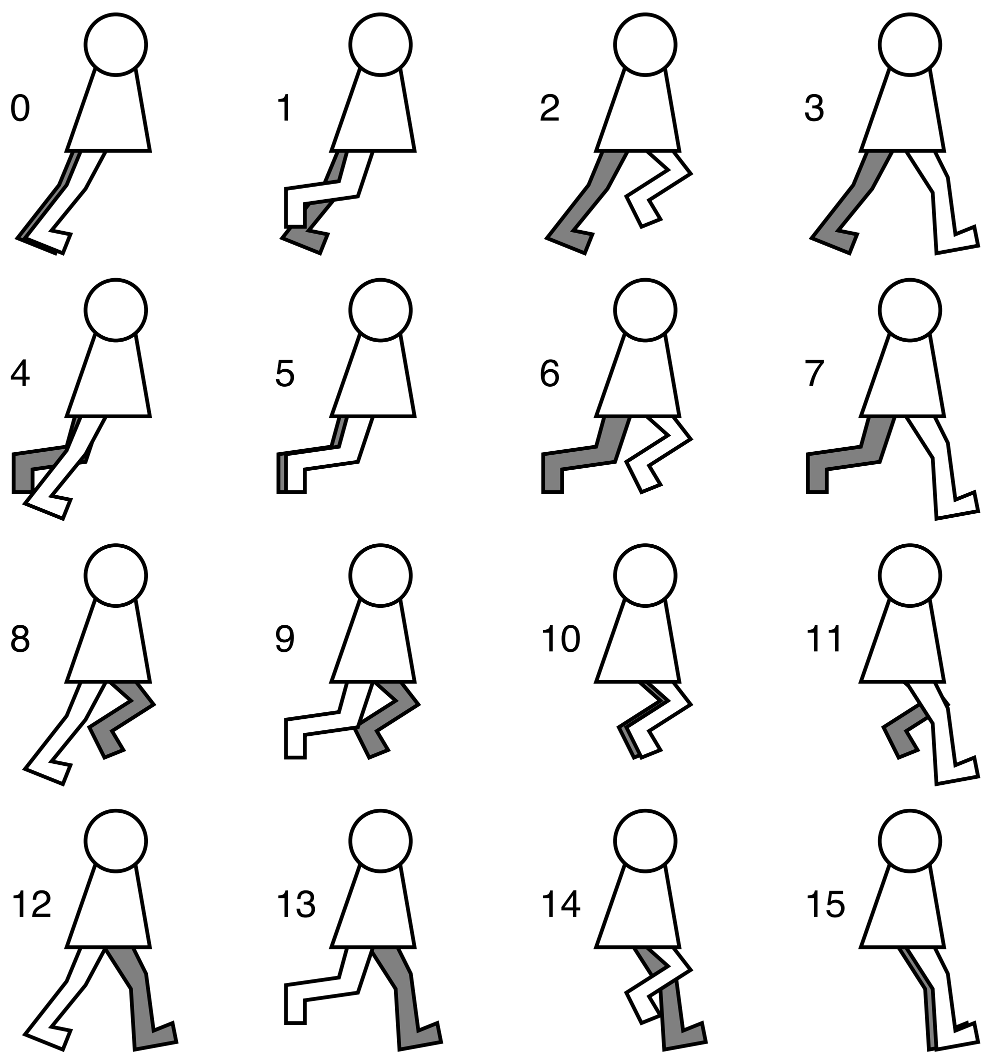
For each iteration step k:

$$\pi_k(s) = \underset{a}{\mathrm{argmax}}\left(r(s, a) + \gamma \cdot U_k(\delta(s, a))\right)$$

$$U_{k+1}(s) = r(s, \pi_k(s)) + \gamma \cdot U_k(\delta(s, \pi_k(s)))$$

Policy Iteration for Cartoon Walker

We cheat a bit, and use entirely known reward and transition functions…



Action	Effect
0	Move right (white) leg up / down
1	Move right (white) leg backward / forward
2	Move left (grey) leg up / down
3	Move left (grey) leg backward / forward

```
for s in range(len(policy)):
    policy[s] = argmax(
        lambda a: rew[s][a] + gamma * value[trans[s][a]],
        range(len(trans[s])))
```

```
for s in range(len(value)):
    a = policy[s]
    value[s] = rew[s][a] + gamma * value[trans[s][a]]
```

Monte Carlo approach

Monte Carlo approach

Usually the reward $r(s, a)$ and the state transition function $\delta(s, a)$ are unknown to the learning agent.

Monte Carlo approach

Usually the reward $r(s, a)$ and the state transition function $\delta(s, a)$ are unknown to the learning agent.

(What does that mean for learning to ride a bike? $\boxed{}$)

Monte Carlo approach

Usually the reward $r(s, a)$ and the state transition function $\delta(s, a)$ are unknown to the learning agent.

(What does that mean for learning to ride a bike?)

Monte Carlo approach

Usually the reward *r(s, a)* and the state transition function δ*(s, a)* are unknown to the learning agent.

(What does that mean for learning to ride a bike? $\boxed{}$)

Still, we can estimate U^* from *experience*, as a *Monte Carlo approach* will do:

- Start with a randomly chosen *s*
- Follow a policy π, store rewards and s_t for the step at time *t*
- When the goal is reached, update the $U^{\pi}(s)$ estimate for all visited states s_t with the future reward that was given when reaching the goal
- Start over with a randomly chosen *s* ...

Monte Carlo approach

Usually the reward $r(s, a)$ and the state transition function $\delta(s, a)$ are unknown to the learning agent.

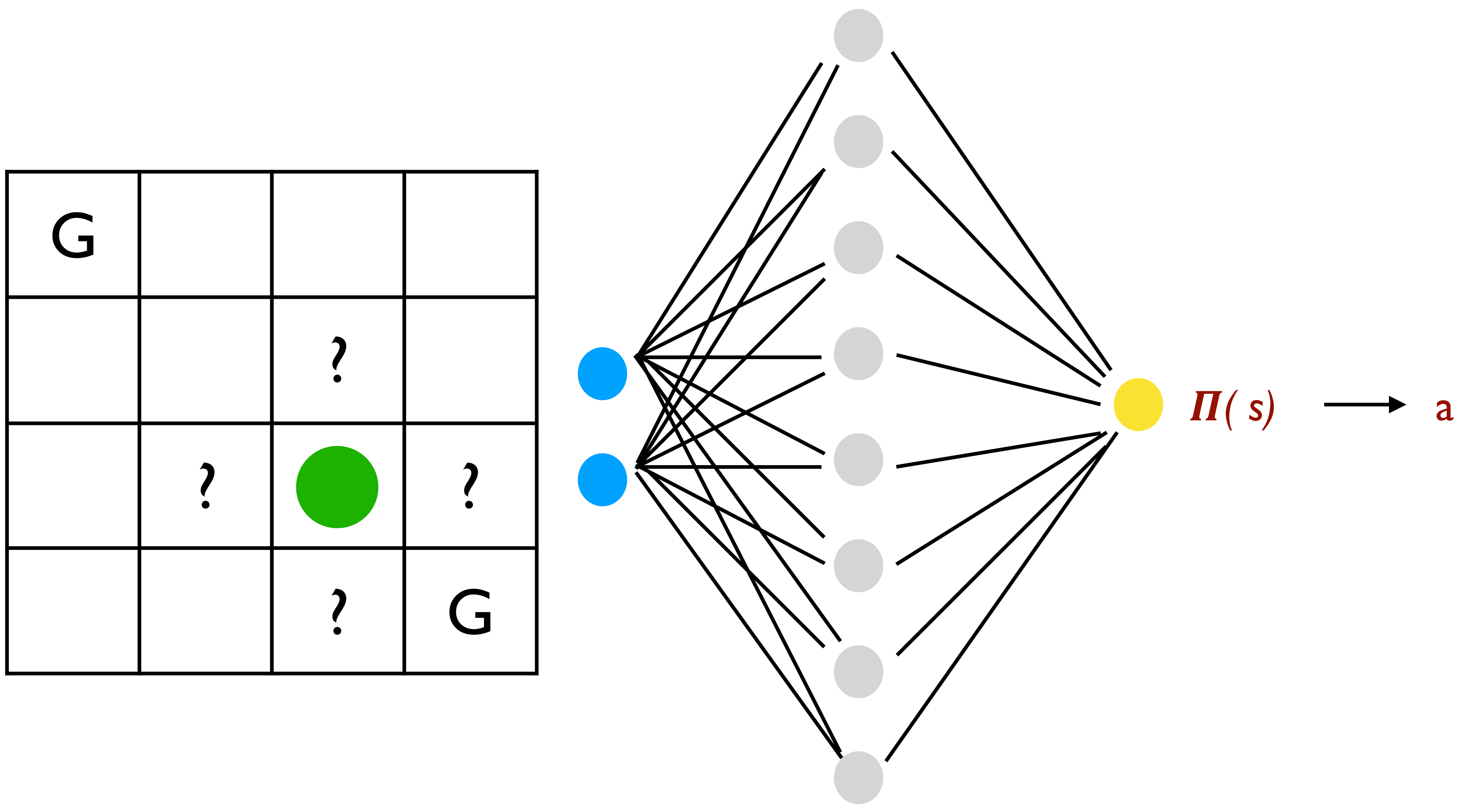
(What does that mean for learning to ride a bike? $\boxed{}$)

Still, we can estimate U^* from *experience*, as a *Monte Carlo approach* will do:

- Start with a randomly chosen s
- Follow a policy π, store rewards and s_t for the step at time t
- When the goal is reached, update the $U^\pi(s)$ estimate for all visited states s_t with the future reward that was given when reaching the goal
- Start over with a randomly chosen s ...

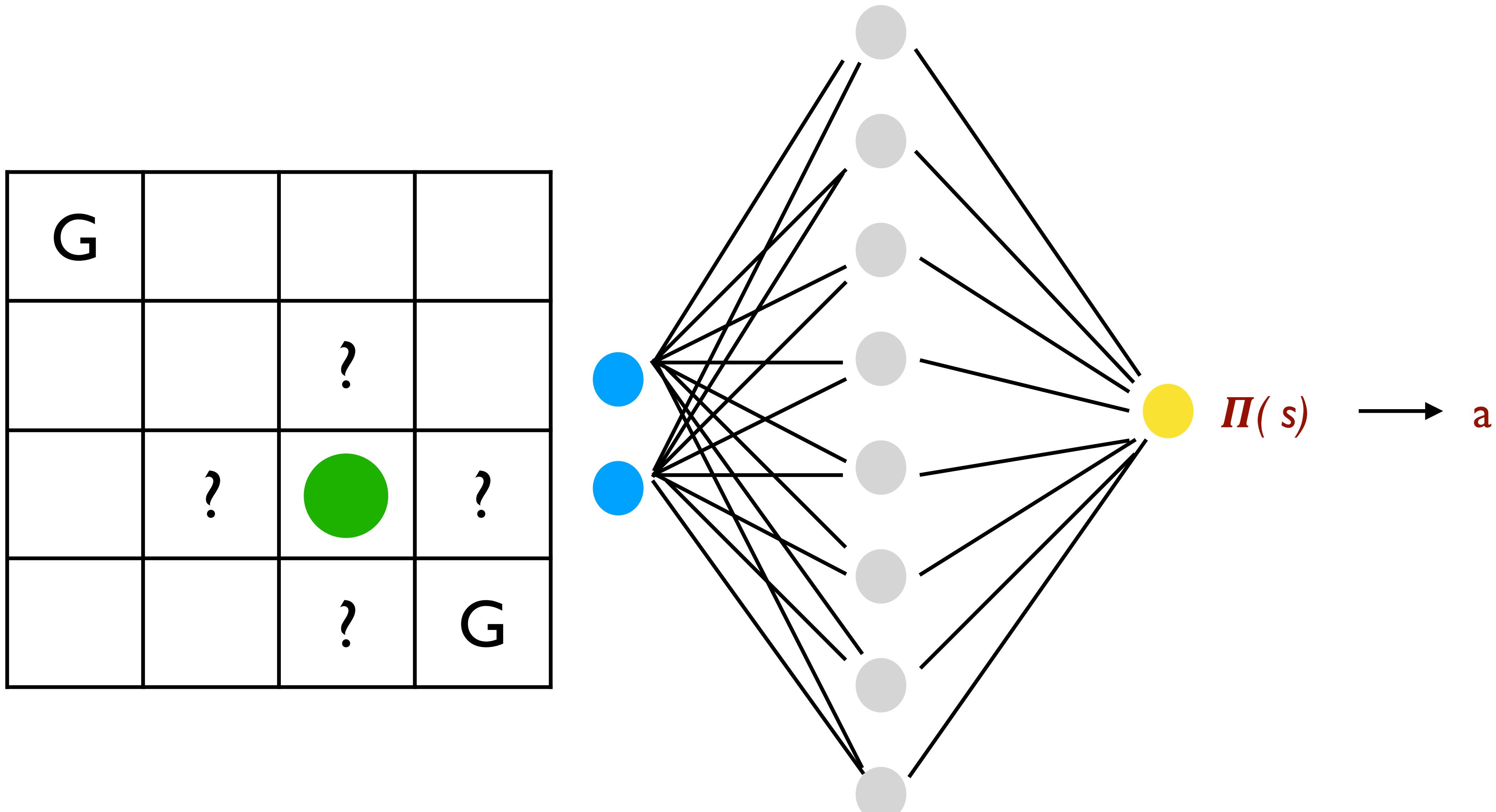
Converges slowly...

Policy gradients



Policy gradients

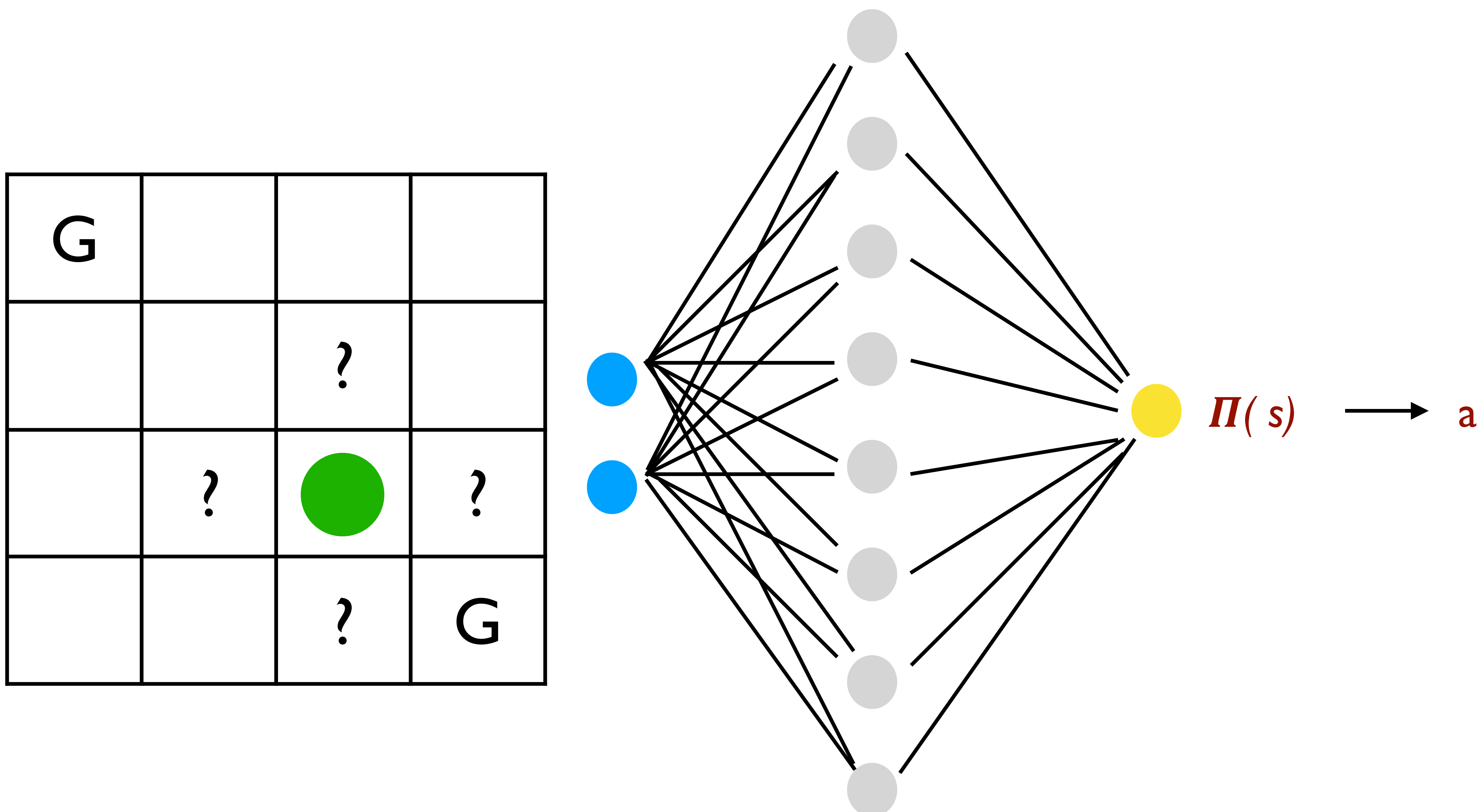
What if…



Policy gradients

What if…

… we take help of an ANN to learn a good policy?



Training the network

Training the network

If we had a “label” saying after a forward run that DOWN is the optimal thing to do for this state…

Training the network

If we had a "label" saying after a forward run that DOWN is the optimal thing to do for this state…

… we would compute the loss as:

- log P(y=DOWN | x)

… but we do not have this label, so we use the reward R we get from using our

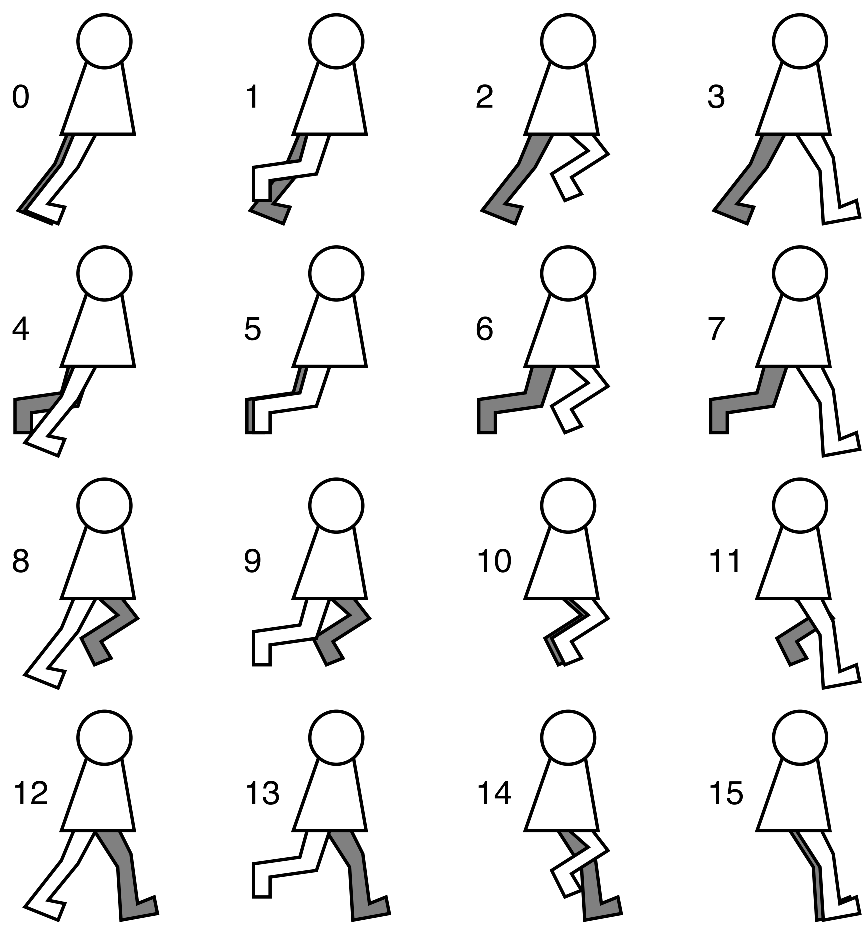
policy (the sampled action) to compute the loss:

Loss = - *R* log P(a) with R being r(s, a)

but that means that we have to save the gradients along our path through the state-action space

Policy Gradients for Cartoon Walker

Represent the walker's policy in a network with
- a single valued array (one input value) for the state
- one of four possible output "classes" (sampled from probability distribution)
- softmax activation
- and not too many hidden neurons ;-)



Action	Effect
0	Move right (white) leg up / down
1	Move right (white) leg backward / forward
2	Move left (grey) leg up / down
3	Move left (grey) leg backward / forward

Will need a lot more time and tweaking than the policy iteration!

Lab assignment 7

- The lab assignment is given as a package with instructions, code skeleton and some useful links also to hands-on material at https://github.com/ErikGartner/edan95-rlagent-handout

- Some hands-on experimenting material can be found at https://github.com/ageron/handson-ml