Learning in Neural Networks

Actor-Critic Methods (RL 5)

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Part 1: Introduction

Objectives:

Reinforcement Learning in Deep Artificial Neural Networks

REINFORCE with BASELINE algorithm

Actor-Critic algorithm

Eligibility traces for policy gradient

Model-based versus Model-free RL

Objectif: get ready to apply RL or read research papers in RL

$\rightarrow$ **final lecture on 'Foundations of RL'**

Reading for this week:

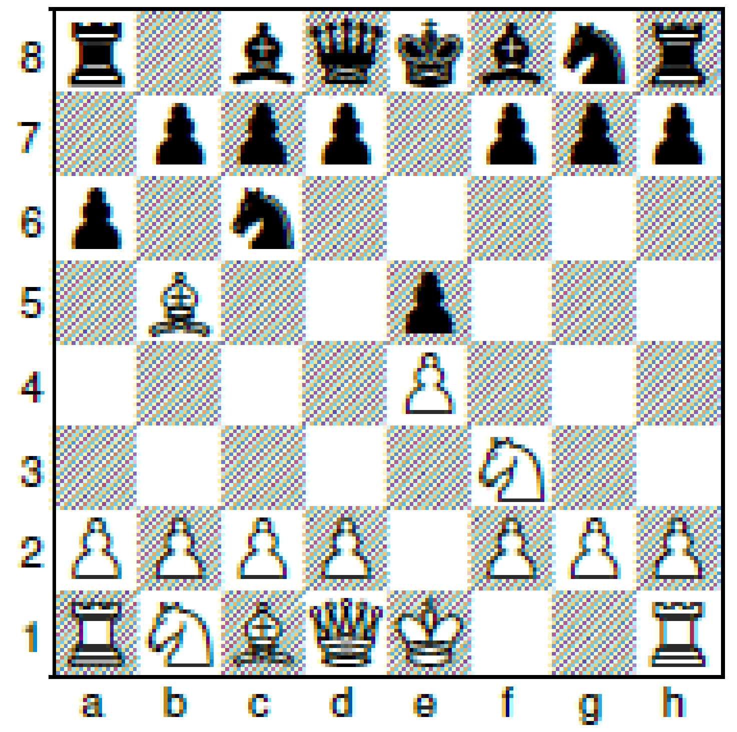
Sutton and Barto, Reinforcement Learning (MIT Press, 2nd edition 2018, also online)

Chapter: 13.5-13.8.

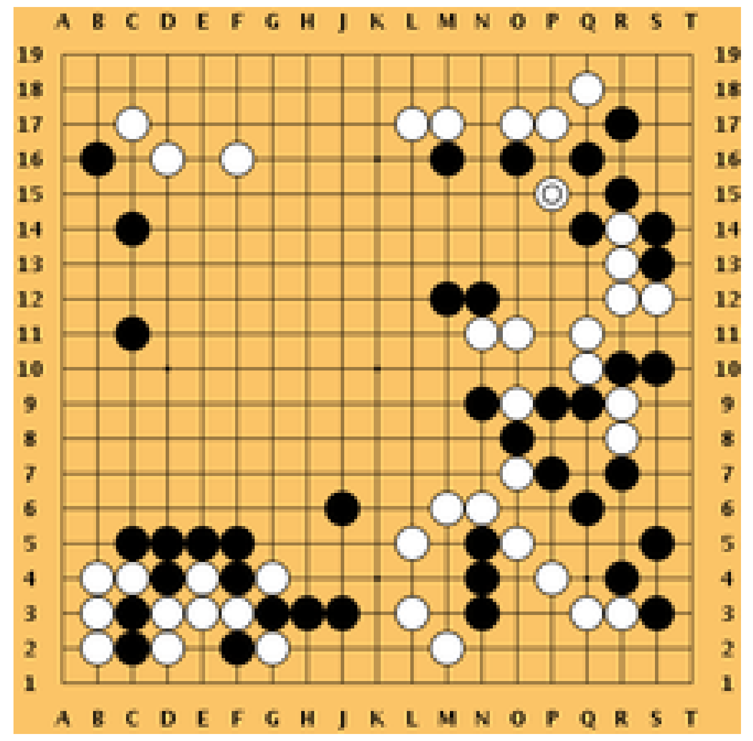
8.1 + 8.2

Deep reinforcement learning

Chess



Go



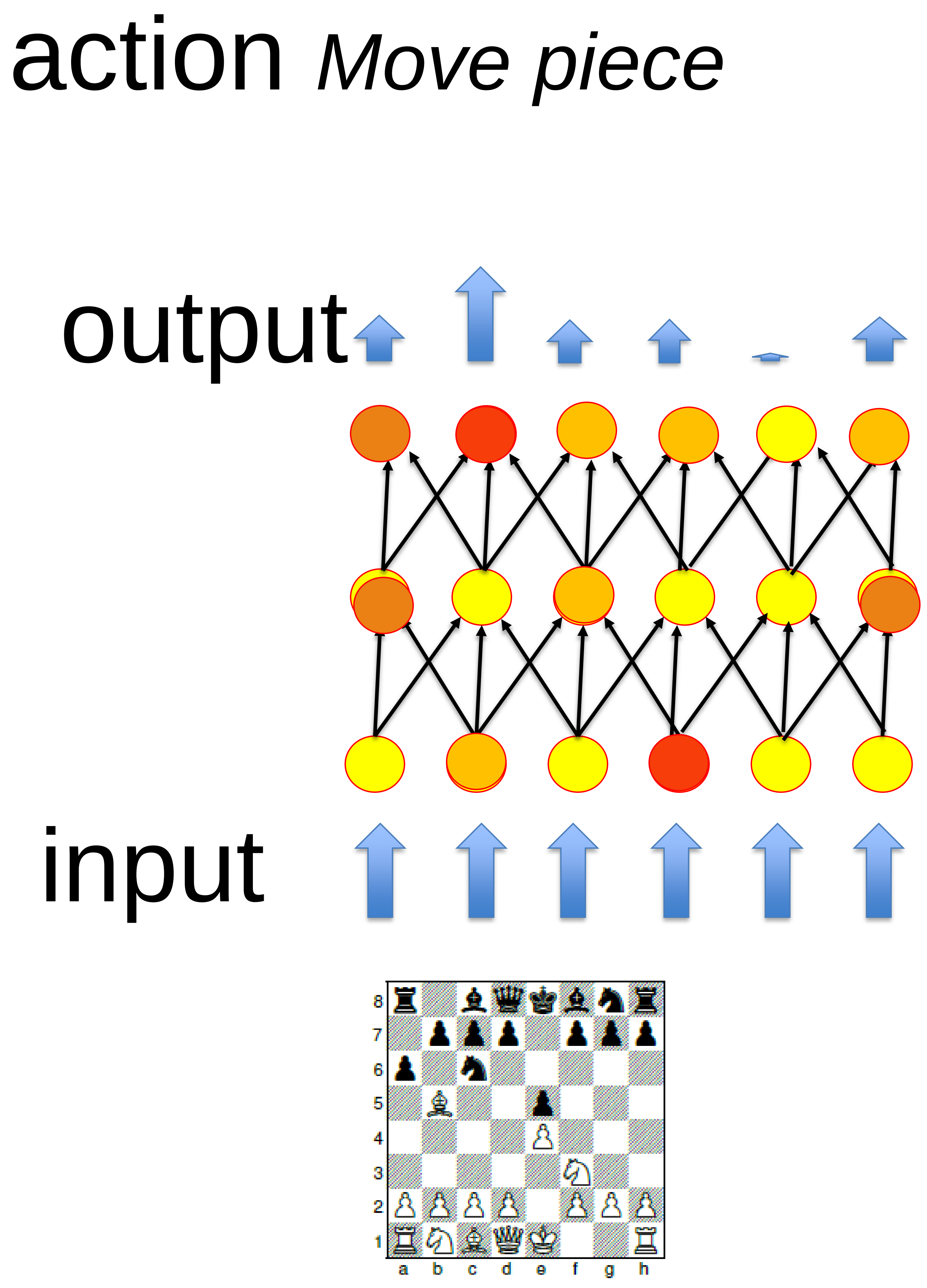
Artificial neural network (*AlphaZero*) discovers different strategies by playing against itself.

In Go, it beats Lee Sedol

(previous slide)
The success of deep artificial neural networks in Chess and Go has spurred a renewed interest in Reinforcement Learning.

Artificial Neural Networks for Reinforcement Learning

(Backprop = gradient descent rule in multilayer networks)



Neural network parameterizes actions in the output as a function of continuous state s.

One output per action.
Learn weights by playing against itself.

Two Methods:

- TD-learning
- Policy Gradient

can also be combined:

→ **actor-critic networks**

(previous slide)
Deep Reinforcement Learning (DeepRL) is reinforcement learning in a deep network. Suppose that each output unit of the network corresponds to one action (e.g. one type of move in chess). Parameters are the weights of the artificial neural network.

Actions are chosen, for example, by softmax on the output-values.

Weights are learned by playing against itself – doing gradient descent on an error function E (Loss function)

Two important classes of RL algorithms are **TD-learning and Policy Gradient.**
An actor-critic architecture combines the advantages of both.

Review: Q-values and V-Values

expected total discounted reward starting in s with action a_1: $Q(s, a_1)$

optimize by semigradient on Loss function

$$E(\boldsymbol{w}) = \frac{1}{2}[\underbrace{r_t + \gamma Q(s', a'}_{\text{target}}|\cancel{\boldsymbol{w}})\text{- } Q(s, a|\boldsymbol{w})]^2$$

ignore

take gradient

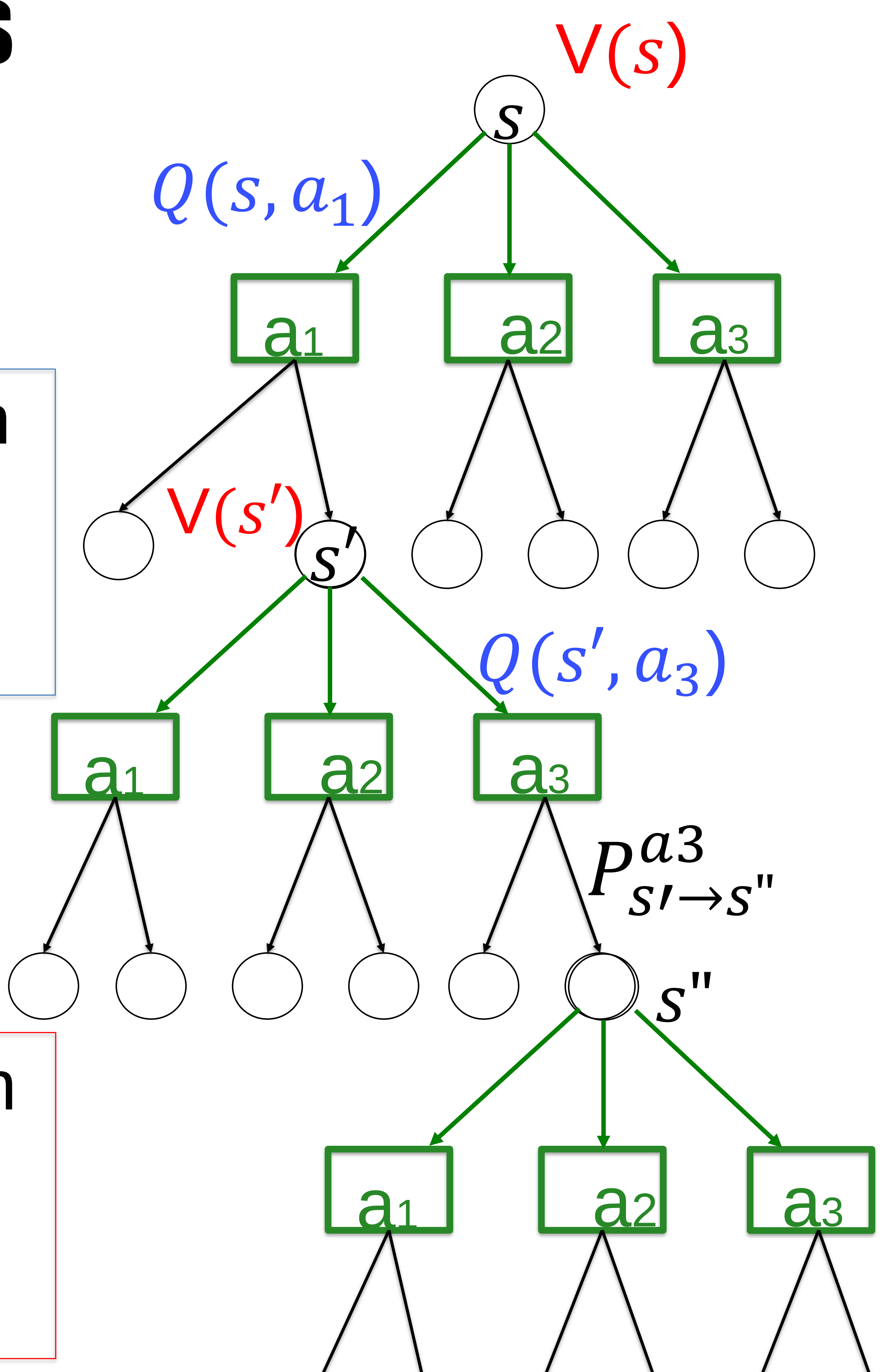
expected total discounted reward starting in s : $V(s)$

optimize by semigradient on Loss function

$$E(\boldsymbol{w}) = \frac{1}{2}[\underbrace{r_t + \gamma V(s'}_{\text{target}}|\cancel{\boldsymbol{w}})\text{- } V(s|\boldsymbol{w})]^2$$

ignore

take gradient



(previous slide)
The consistency condition of TD learning, can be formulated by an error function:
Either for Q-values

$$E = 0.5 \, [\, r + \gamma \, Q(s',a') - Q(s,a) \,]^2$$

or for V-values

$$E = 0.5 \, [\, r + \gamma \, V(s',a') - V(s,a) \,]^2$$

This error function will depend on the weights w.
We can change the weights by semi-gradient descent on the error function. This leads to the Backpropagation algorithm of ‘Deep learning’

Aims for today:

- Understand the principles of Deep Reinforcement-learning in Artificial Neural Networks.

- Understand how the Actor-Critic combines Policy Gradient methods and TD methods

- Understand why eligibility traces arise in policy gradient

- Understand Difference between Model-based and Model-free RL

Reinforcement Learning Lecture 5
Policy Gradient and Actor-Critic Methods

Wulfram Gerstner

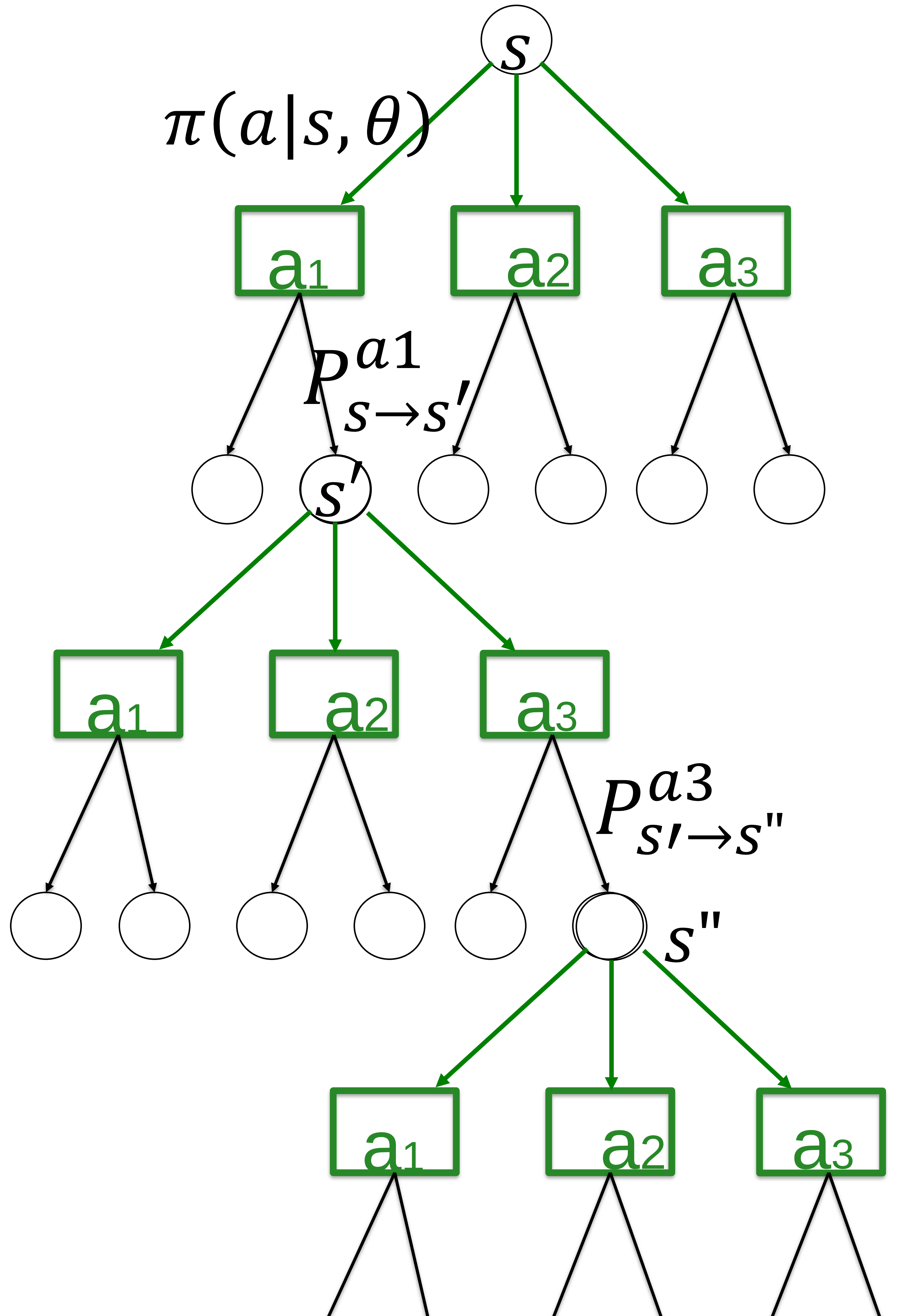
EPFL, Lausanne, Switzerland

Part 2: From policy gradient to Deep Reinforcement Learning

Review Policy Gradient

Aim:
update the parameters θ
of the policy π(*a*|*s*,θ)

Implementation:
- play episode from start to end;
- record rewards in each step;
- update the parameters θ



(previous slide and next slide)
Policy gradient methods are and alternative to TD methods.
We consider a single episode that started in state s_t with action a_t and ends after several steps in the terminal state s_{end}
The result of the calculation gives an update rule for each of the parameters.
The update of the parameter θ_j contains several terms.
(i) the first term is proportional to the total accumulated (discounted) reward, also called return $R_{s_t \to s_{end}}^{a_t}$
(ii) the second term is proportional to gamma times the total accumulated (discounted) reward but starting in state s_{t+1}
(iii) the third term is proportional to gamma-squared times the total accumulated (discounted) reward but starting in state s_{t+2}
(iv) …
We can think of this update as one update step for one episode. Analogous to the terminology used by Sutton and Barto, we call this the Monte-Carlo update for one episode.
The log-likelihood trick was explained earlier. Since this is a sampling based approach (1 episode=1 sample) each of the terms is proportional to ln π,

Review Policy Gradient: REINFORCE (episodic)

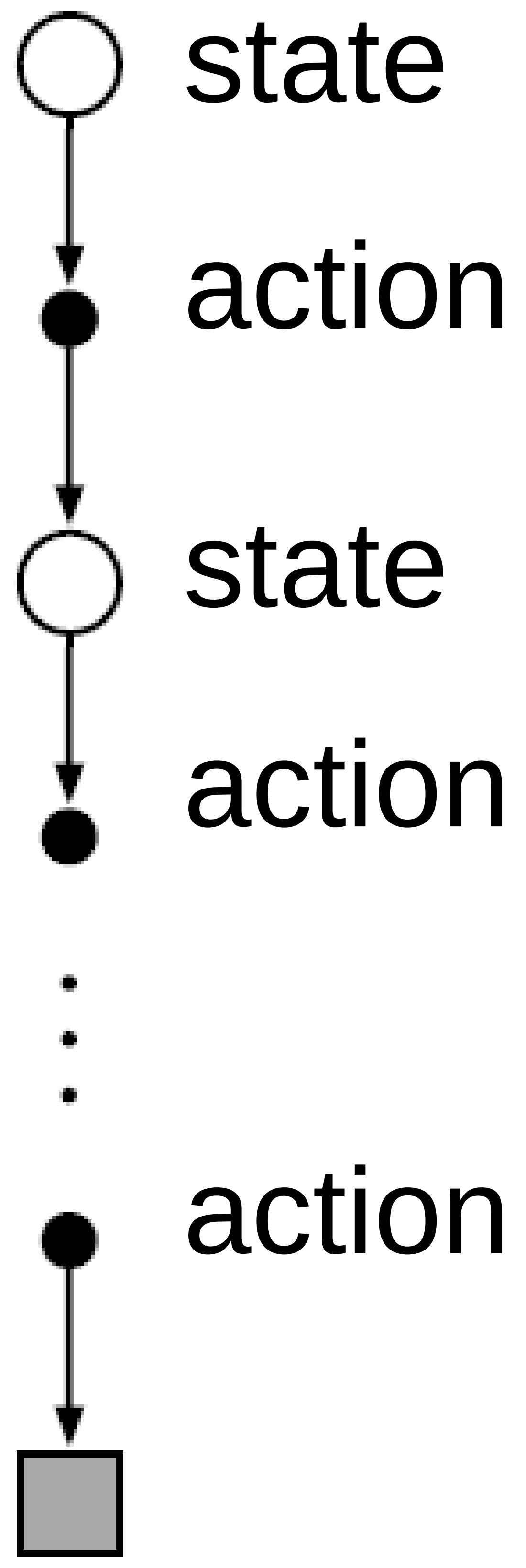
Calculation yields several terms of the form

Total accumulated discounted reward collected in one episode starting at s_t, a_t

$$\Delta\theta_j \propto \left[R^{a_t}_{s_t \to s_{end}}\right] \frac{d}{d\theta_j} \ln[\pi(a_t|s_t,\theta)]$$

$$+\gamma\left[R^{a_{t+1}}_{s_{t+1} \to s_{end}}\right] \frac{d}{d\theta_j} \ln[\pi(a_{t+1}|s_{t+1},\theta)]$$

$$+ \ldots$$



end of trial

EPISODIC: fixed start state s_t , fixed terminal state s_{end}

(next slide)

The policy gradient algorithm is also called REINFORCE. It if formulated here for fixed starting state and assumes the existence of a terminal state. Note that updates are only implemented at the end of the trial (i.e., once the agent has arrived in the terminal state).

As discussed in a previous lecture and in the exercise session, subtracting the mean of a variable helps to stabilize the algorithm.

There are two different ways to do this.

(i) Subtract the mean return (=value V) in a multistep-horizon algorithm. This is what we consider here in this section. NOW!

(ii) Subtract mean expected reward PER TIME STEP (related to the delta-error of TD learning) in a multi-step horizon algorithm.
This is what we will consider in section 3 under the term Actor-Critic.

Subtract a baseline

we derived this online gradient rule for multi-step horizon

$$\Delta\theta_j \propto [R_{s_t \to s_{end}}^{a_t}] \frac{d}{d\theta_j} \ln[\pi(a_t|s_t,\theta)] +$$
$$+\gamma[R_{s_{t+1} \to s_{end}}^{a_t}] \frac{d}{d\theta_j} \ln[\pi(a_{t+1}|s_{t+1},\theta)] + \ldots$$

But then this rule is also an online gradient rule

$$\Delta\theta_j \propto [R_{s_t \to s_{end}}^{a_t} - \boldsymbol{b(s_t)}] \frac{d}{d\theta_j} \ln[\pi(a_t|s_t,\theta)] +$$
$$+\gamma[R_{s_{t+1} \to s_{end}}^{a_t} - \boldsymbol{b(s_{t+1})}] \frac{d}{d\theta_j} \ln[\pi(a_{t+1}|s_{t+1},\theta)] + \ldots$$

with the **same solution (in expectation)**

because a baseline shift drops out if we take the gradient

(previous slide)
Please remember that the full update rule for the parameter θ_j in a multi-step episode contains several terms of this form; here only the first two of these terms are shown.

Similar to the case of the one-step horizon, we can subtract a bias b from the return $R^{a_t}_{s_t \to s_{end}}$ without changing the location of the maximum of the total expected return.

Moreover, this bias $b(s_t)$ can itself depend on the state s_t.
Thus the update rule now has terms

$$\begin{aligned}
\Delta\theta_j \;\propto\; & [R^{a_t}_{s_t \to s_{end}} - b(s_t)] \frac{d}{d\theta_j} \ln[\pi(a_t|s_t,\theta)] \\
& + \gamma [R^{a_{t+1}}_{s_{t+1} \to s_{end}} - b(s_{t+1})] \frac{d}{d\theta_j} \ln[\pi(a_{t+1}|s_{t+1},\theta)] \\
& + \gamma^2 [R^{a_{t+2}}_{s_{t+2} \to s_{end}} - b(s_{t+2})] \frac{d}{d\theta_j} \ln[\pi(a_{t+2}|s_{t+2},\theta)] \\
& + \dots
\end{aligned}$$

Subtract a reward baseline

Total accumulated discounted reward
collected in one episode starting at s_t, a_t

$$\Delta\theta_j \propto [R_{s_t \to s_{end}}^{a_t} - b(s_t)] \frac{d}{d\theta_j} \ln[\pi(a_t|s_t, \theta)] + \ldots$$

- The bias b can depend on state s
- Good choice is b ='mean of $[R_{s_t \to s_{end}}^{a_t}]$'

  → take $b(s_t) = V(s_t)$

  → learn value function *V(s)*

(previous slide
Is there a choice of the bias $b(s_t)$ that is particularly good?

One attractive choice is to take the bias equal to the expectation (or empirical mean). The logic is that if you take an action that gives more accumulated discounted reward than your empirical mean in the past, then this action was good and should be reinforced.
If you take an action that gives less accumulated discounted reward than your empirical mean in the past, then this action was not good and should be weakened.

But what is the expected discounted accumulated reward? This is, by definition, exactly the value of the state. Hence a good choice is to subtract the V-value.

And here is where finally the idea of Bellman equation and TD learning comes in through the backdoor: we can learn the V-value, and then use it as a bias in policy gradient.

Learning two Neural Networks: actor and value

Actions:

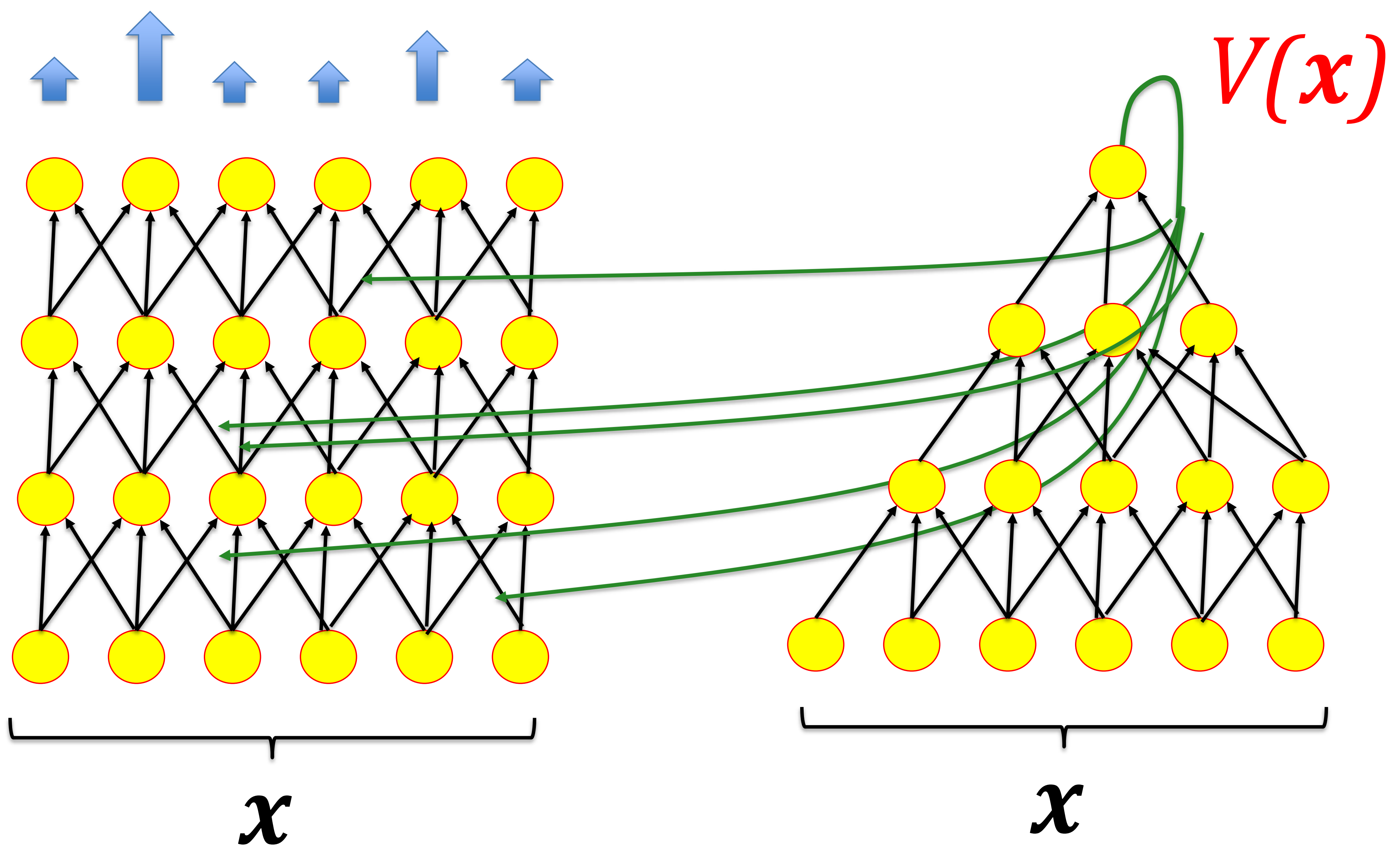
-Learned by
  Policy gradient

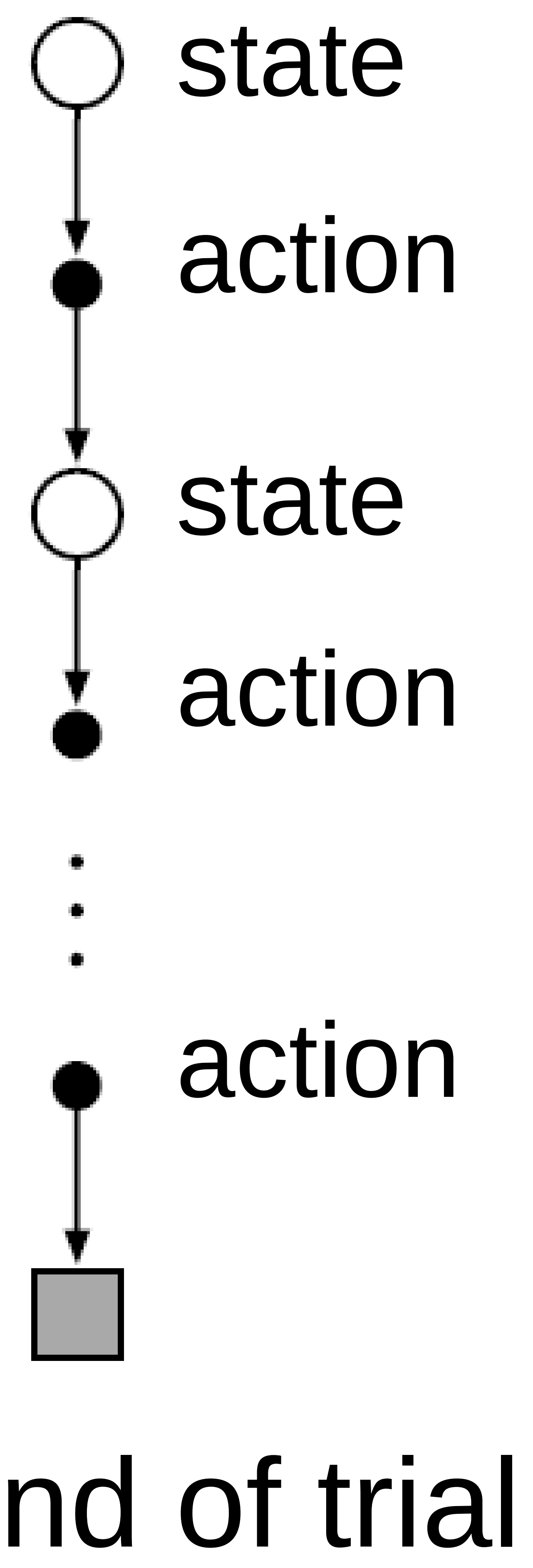
- Uses $V(x)$ as baseline

Value function:

- Estimated by Monte-Carlo

-provides baseline $b{=}V(x)$
  for action learning





x = states from episode:

$s_t, s_{t+1}, s_{t+2},$

(previous slide)
In the latter case we have two networks:

The actor network learns a first set of parameters, called θ in the algorithm of Sutton and Barto.
The value network learns a second set of parameters, with the label w .

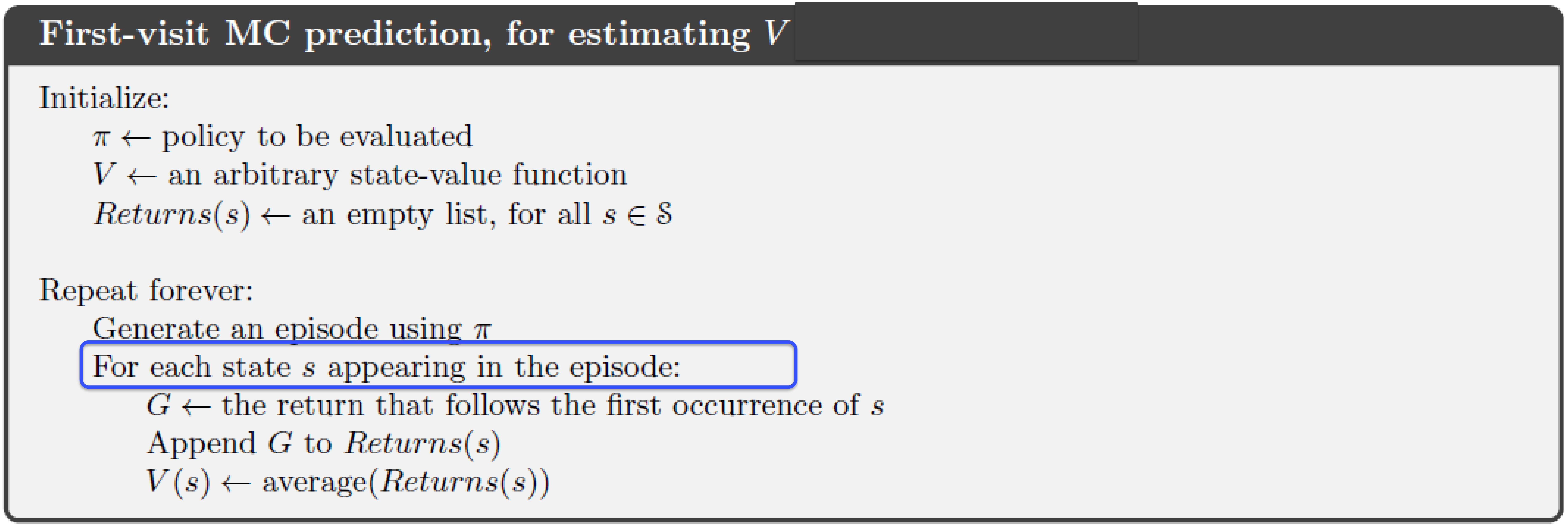
The value $b(x = s_{t+n}) = V(\boldsymbol{x})$ is the **estimated** total accumulated discounted reward of an episode starting at $x = s_{t+n}$

The weights of the network implementing V(x) can be learned by Monte-Carlo sampling the return: you go from state s until the end, accumulate rewards, and calculate the average over all episodes that have started from (or transited through) the same state s. (See Backup-diagrams and Monte-Carlo of earlier lecture).

The total accumulated discounted ACTUAL reward in ONE episode is $R^{a_{t+n}}_{s_{t+n} \to s_{end}}$
What matters is the difference $[R^{a_t}_{s_t \to s_{end}} - V(s_t)]$

REVIEW: Monte-Carlo Estimation of V-values (tabular)

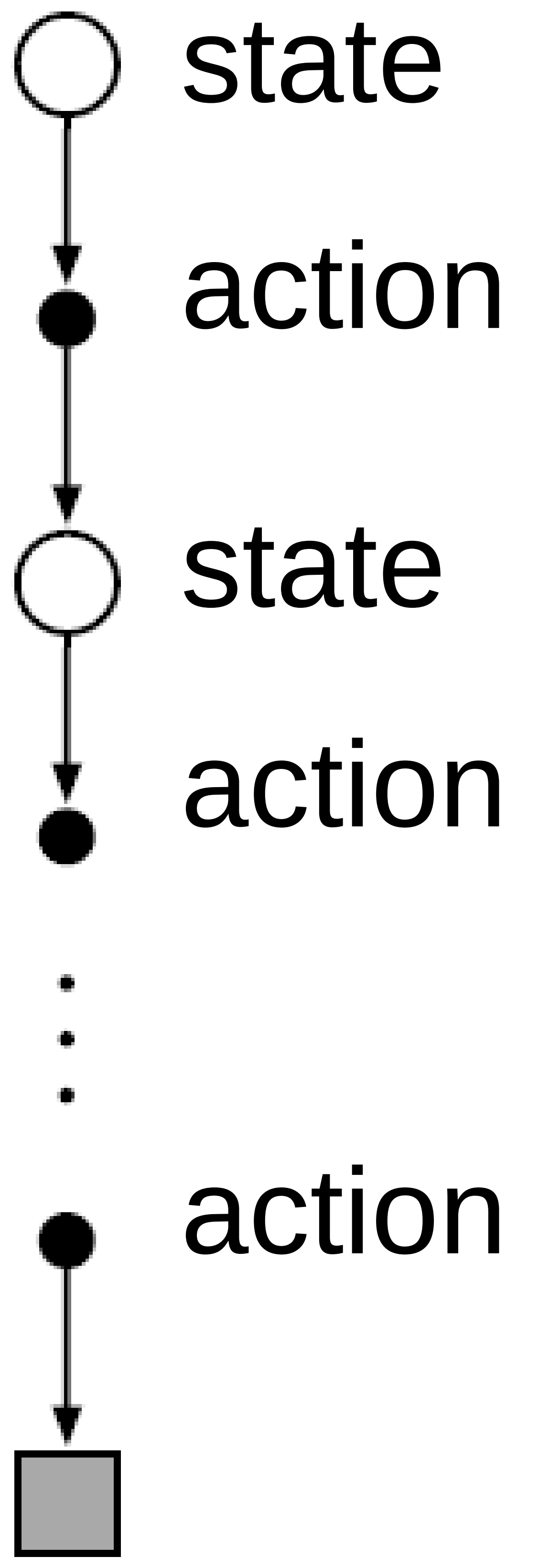
$$Return(s) = r_t + \gamma\, r_{t+1} + \gamma^2 r_{t+2} + \gamma^3\, r_{t+3}$$



First-visit MC prediction, for estimating V

```
Initialize:
    $\pi \leftarrow$ policy to be evaluated
    $V \leftarrow$ an arbitrary state-value function
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
    Generate an episode using $\pi$
    For each state $s$ appearing in the episode:
        $G \leftarrow$ the return that follows the first occurrence of $s$
        Append $G$ to $Returns(s)$
        $V(s) \leftarrow$ average$(Returns(s))$
```



end of trial

single episode starting in state s0 also allows to update V(s) of children states

(previous slide, Review). We can use Monte-Carlo estimates for V-values
In this (version of the) algorithm you first open V-estimators for all states.

For each state s that you encounter, you observe the sum of (discounted) rewards that you accumulate until the end of the episode. The total accumulated discounted reward starting from s is the ‘Return(s)’

After many episode you estimate the V-values V(s) as the average over the Returns(s).

Note that the above estimations are done in parallel for all states s that you encounter on your path. This includes ‘children states’.

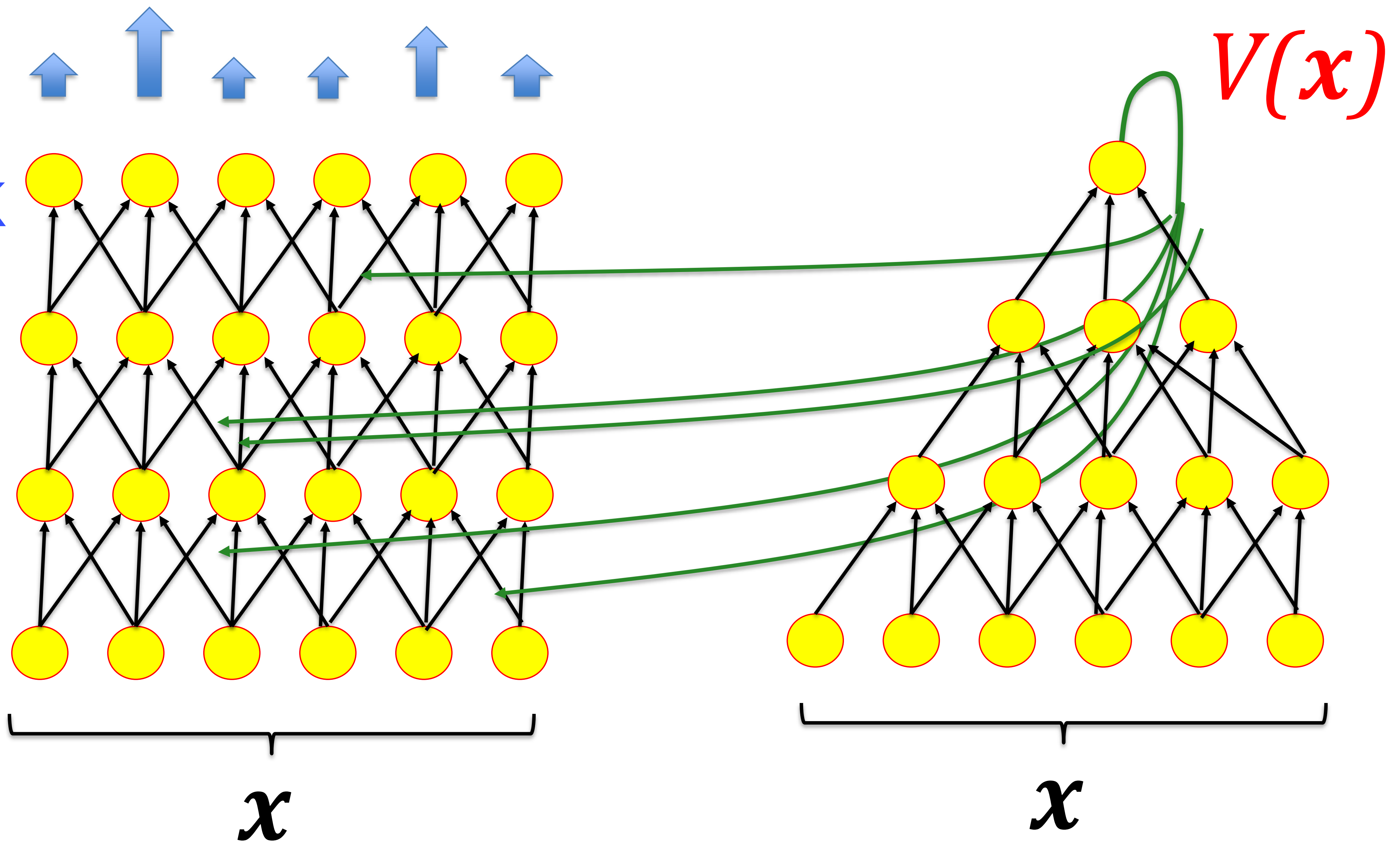
Also note that the Backup diagram is much deeper than that of TD-learning, since you always continue until the end of the episode before you can update V-values of states that have been encountered many steps before.

Learning two Neural Networks: actor and value

Actions:
-Learned by
Policy gradient
- Uses $V(\boldsymbol{x})$ as baseline

Value function:
- Estimated by Monte-Carlo
-provides baseline $b = V(\boldsymbol{x})$
for action learning

Parameters
are the network
weights θ



Parameters
are the **weights w**

$\boldsymbol{x}$ = states from
episode:
$s_t, s_{t+1}, s_{t+2},$

(previous slide)
In the latter case we have two networks:

The actor network learns a first set of parameters, called θ in the algorithm of Sutton and Barto.
The value network learns a second set of parameters, with the label w .

The value $b(x = s_{t+n}) = V(\boldsymbol{x})$ is the estimated total accumulated discounted reward of an episode starting at $x = s_{t+n}$

The total accumulated discounted ACTUAL reward in ONE episode is $R^{a_{t+n}}_{s_{t+n} \to s_{end}}$

‘REINFORCE’ with baseline

From book:
Sutton and Barto, 2018

REINFORCE with Baseline (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

```
Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, r_1 \ldots, S_{T-1}, A_{T-1}, r_T$ following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$                    $(G_t)$
        $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \gamma^t \delta \nabla \hat{v}(S_t,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$
```

(previous slide)
Algorithm in pseudocode taken from the book of Sutton and Barto.
For the actor, the algorithm evaluates terms of the form

$$[R_{s_{t+n}\to s_{end}}^{a_{t+n}} - b(s_{t+n})]\,\frac{d}{d\theta_j}\ln[\pi(a_{t+n}|s_{t+n},\theta)]$$

Where the return is $G = R_{s_{t+n}\to s_{end}}^{a_{t+n}}$

And the bias estimate is $\mathrm{v}(s_{t+n}) = b(s_{t+n})$

The terminal state in their notation occurs at time T and
the initial state has index 0.

For the value function, they use Monte-Carlo estimation of the total accumulated
reward in one episode (see previous slide).

Why subtract the mean?

Subtracting the expectation provides estimates
  that have (normally) smaller variance (look less noisy)

Note: in multi-step RL, the minimal variance is not exactly at
          bias=expection.
Reason: correlations

(previous slide)
Why is it useful to subtract the mean?

Whatever the choice of baseline, the algorithm should eventually converge to the same set of parameters. However, since the algorithm is based on stochastic gradient descent or ascent (i.e., the online rule instead of the full batch rule), the algorithm makes **noisy steps** that only go on average in the right direction.

Subtracting a baseline that is close to the mean generally reduces the noise.
The example with a product of independent variables shows that by subtracting the mean of x, the noise is considerable reduced in each of the samples! (Last week)

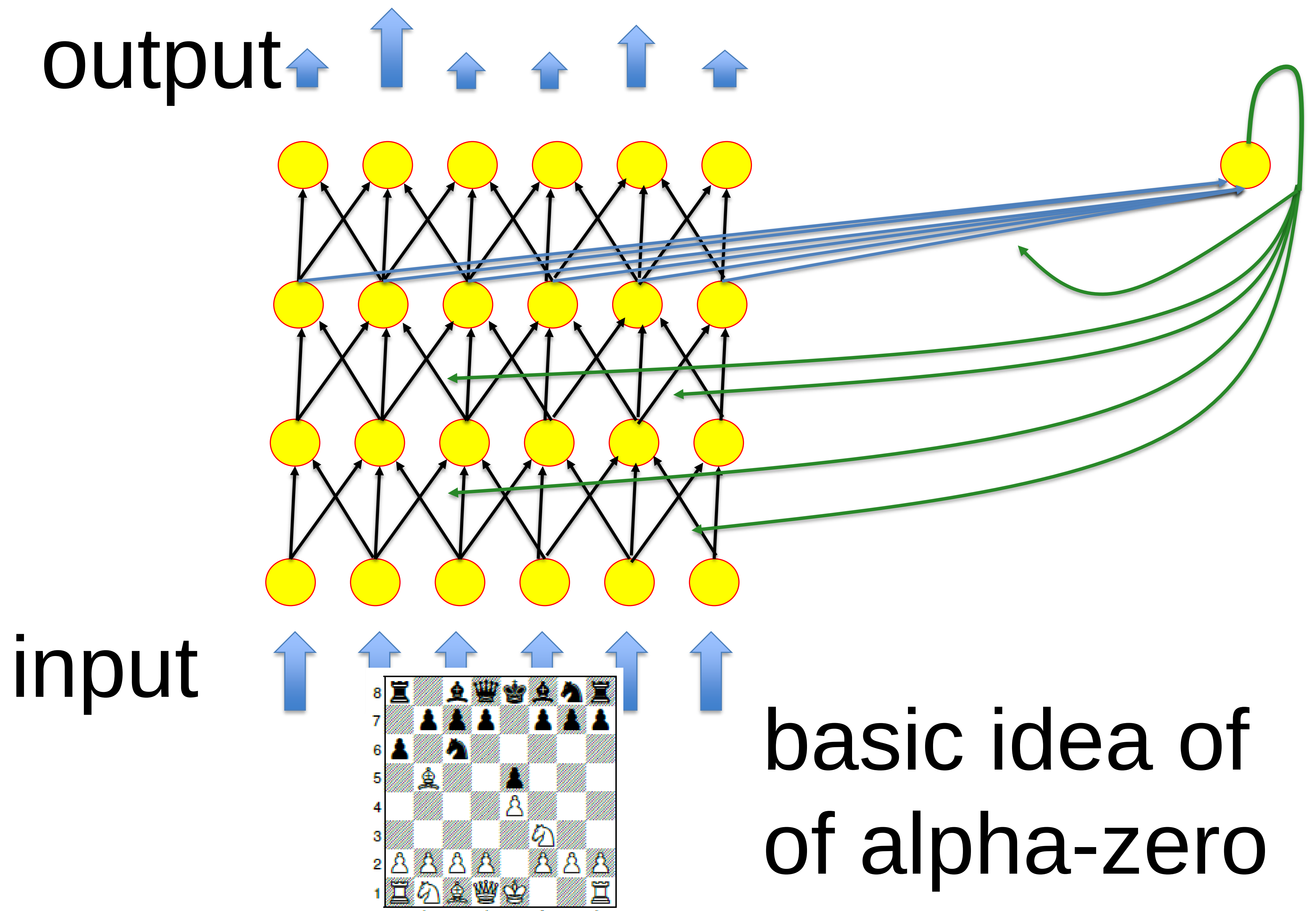
Unfortunately, in a multi-step reinforcement learning scenario, the minimal noise is not exactly the situation where one subtracts the mean because of correlations, but (at least we can assume that) it is close to it.

Deep reinforcement learning: exploit state representation

Network for choosing action:

Optimized by policy gradient

action: *Advance king*



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

$V(s)$

learning:

→ change connections

aims:

- learn value $V(s)$ of position
- learn action policy to win

Learning signal:

- η[actual Return - $V(s)$]

basic idea of of alpha-zero

(previous slide)
The value unit can either take directly the input (and hence forms a separate network) or it can also share a large fraction of the network with the policy gradient network (actor network).

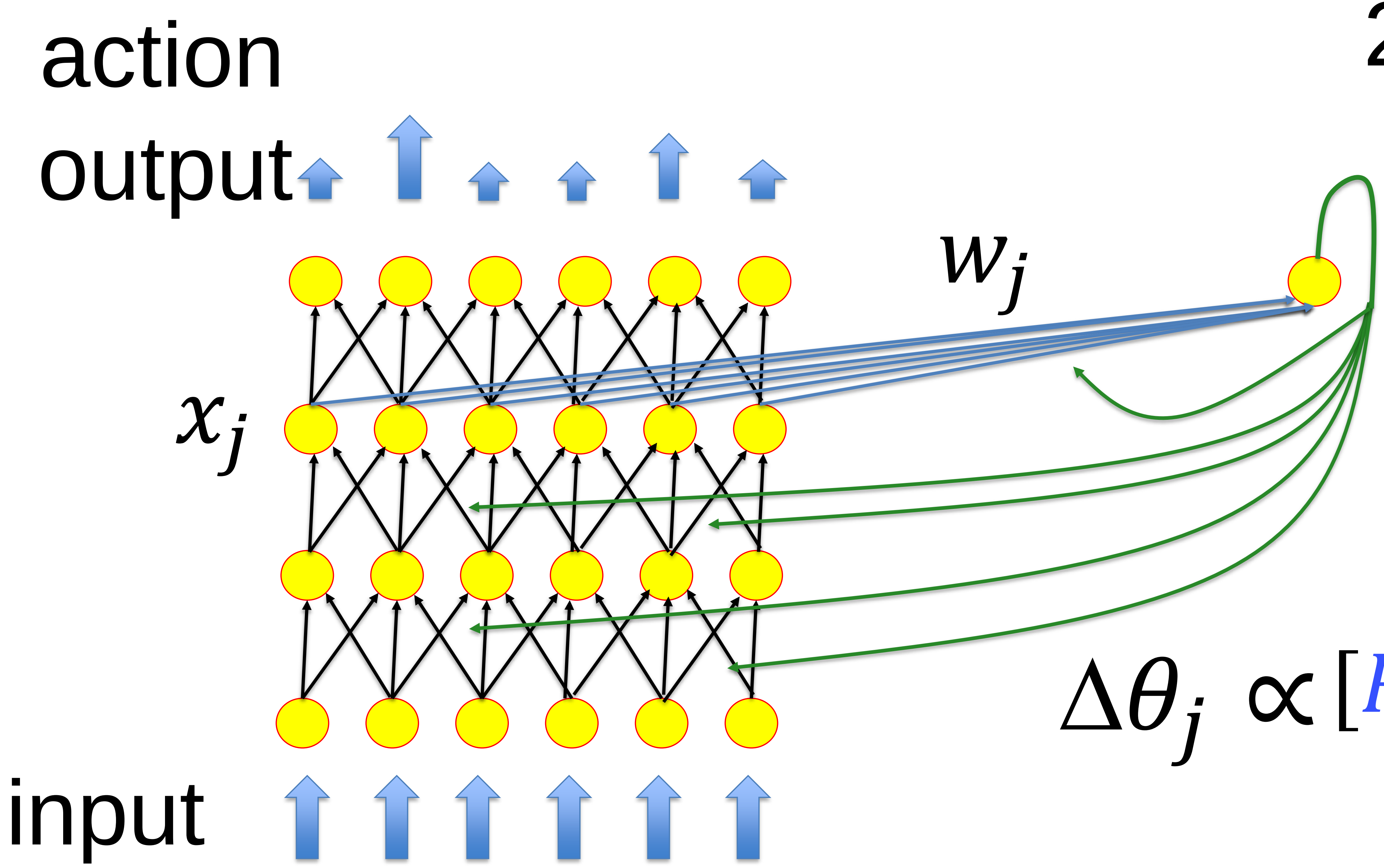
The actor network learns a first set of parameters, called θ in the algorithm of Sutton and Barto. The value unit learns a second set of parameters, with the label w_j for a connection from unit j to the value output.

The total accumulated discounted ACTUAL reward in ONE episode is $R^{a_{t+n}}_{s_{t+n} \to s_{end}}$
What matters is the difference $[R^{a_t}_{s_t \to s_{end}} - V(s_t)]$

Summary: Deep REINFORCE with baseline substraction

Network for choosing action:

Optimized by policy gradient



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

$$V(s_t) = \sum_j w_j x_j(t)$$

$$\Delta\theta_j \propto [R^{a_t}_{s_t \to s_{end}} - V(s_t)] \frac{d}{d\theta_j} \ln[\pi(a_t|s_t,\theta)] + \dots$$

$$\Delta w_j \propto [R^{a_t}_{s_t \to s_{end}} - V(s_t)] x_j(t) +$$

(previous slide)
Here the value unit receives input from the second-last layer. Units there have an activity x_j (j is the index of the unit) which represent the current input state in a compressed, recoded form (The network could for example be a convolutional network if the input consists of pixel images of outdoor scenes.

The actor network learns a first set of parameters, called θ in the algorithm of Sutton and Barto. The value unit learns a second set of parameters, with the label w_j for a connection from unit j to the value output.
The total accumulated discounted ACTUAL reward in ONE episode is $R^{a_{t+n}}_{s_{t+n}\to s_{end}}$
What matters is the difference $[R^{a_t}_{s_t\to s_{end}} - V(s_t)]$

Updates are:

$$\Delta\theta_j \propto [R^{a_t}_{s_t\to s_{end}} - V(s_t)]\,\frac{d}{d\theta_j}\ln[\pi(a_t|s_t,\theta)] + \ldots$$

$$\Delta w_j \propto [R^{a_t}_{s_t\to s_{end}} - V(s_t)]\,x_j$$

Reinforcement Learning Lecture 5
Policy Gradient and Actor-Critic Methods

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Part 3: Actor-Critic network

1. Introduction
2. From Policy gradient to Deep REINFORCE
3. **Actor-Critic network (‘Advantage Actor Critic’)**

(previous slide)
We continue with the idea of two different types of outputs:
A set of actions a_k , and a value V.

However, for the estimation of the V-value we now use the ‘bootstrapping’ provided by TD algorithm (see previous weeks) rather than the simple Monte-Carlo estimation of the (discounted) accumulated rewards in a single episode.

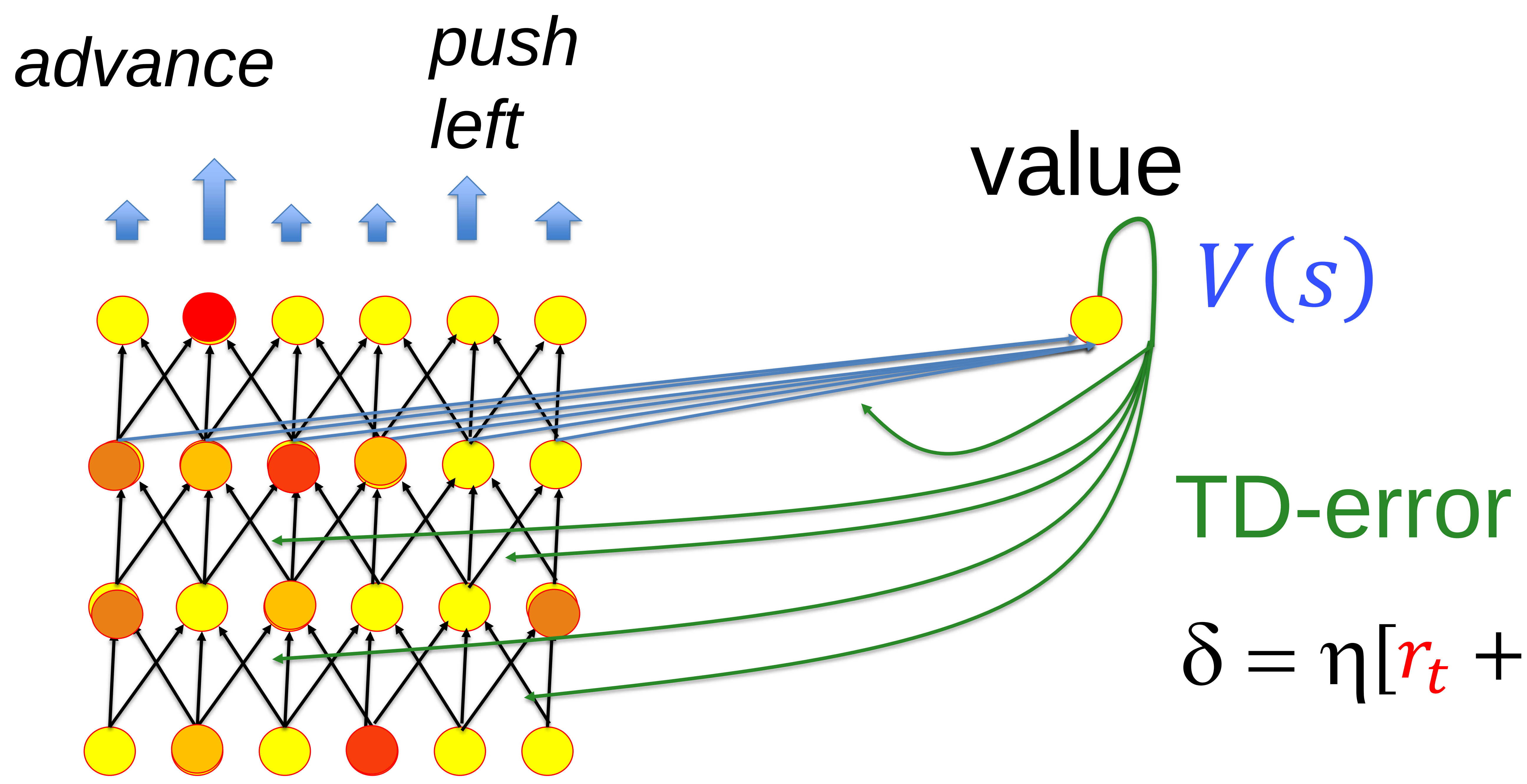
The networks structure remains the same as before:
An actor (action network) and a critic (value function).

Sutton and Barto reserve the term ‘actor-critic’ to the network where V-values are learned with a TD algorithm.
However, other people would also call the network that we saw previously in section 2 as an actor-critic network and the one that we study now is then called ‘advantage actor critic’ or AAC.

Actor-Critic = 'REINFORCE' with TD bootstrapping

- Estimate $V(s)$
- learn via TD error

actions



$$\delta = \eta[r_t + \gamma V(s_{t+1}) - V(s_t)]$$

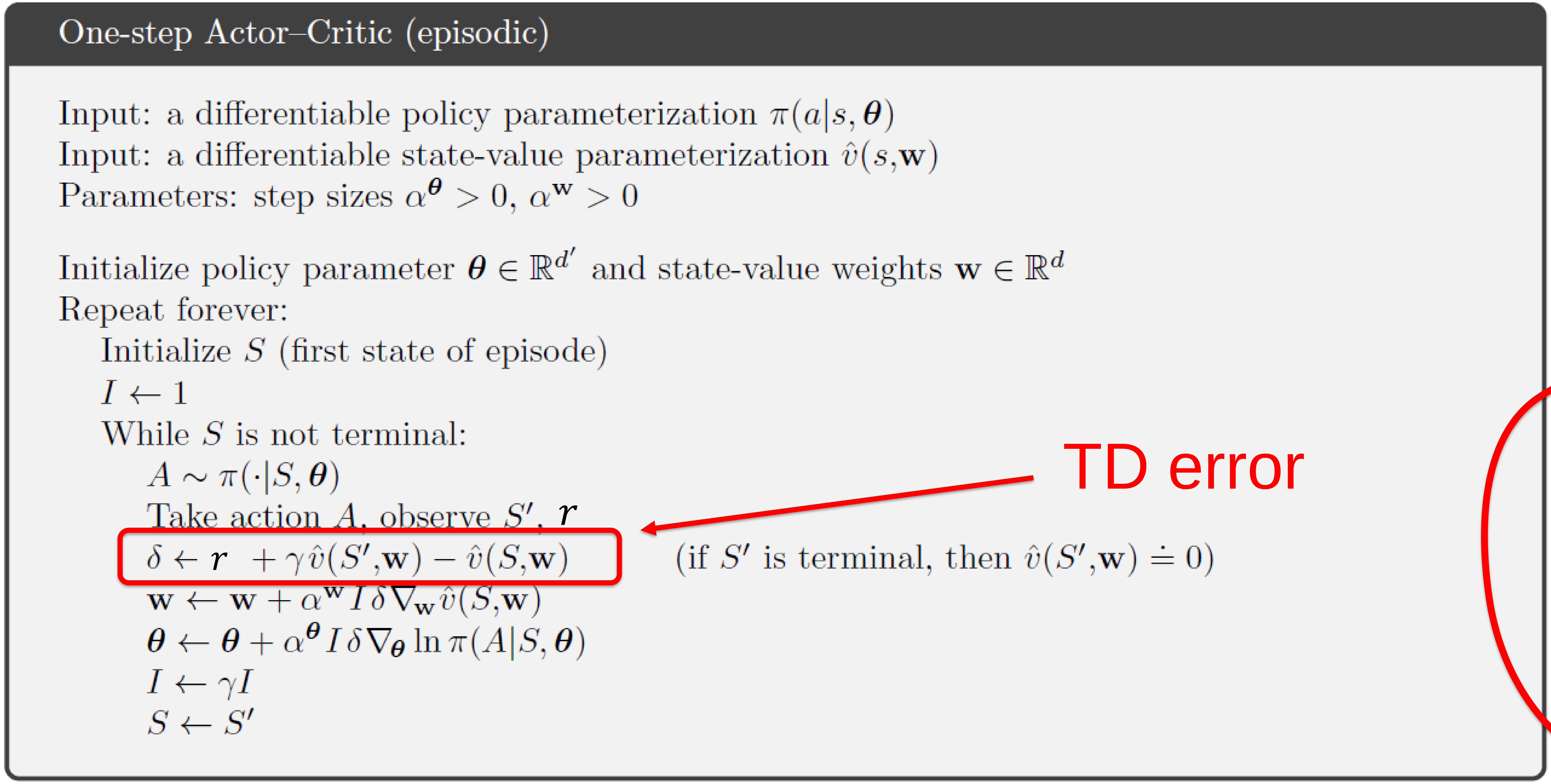
(previous slide)
Bottom right: Recall from the TD algorithms that the updates of the weights are proportional to the TD error δ

In the actor-critic algorithm the TD error is now also used as the learning signal for the policy gradient:

TD error: The current reward r_t at time step t
is compared with the expected reward for this time step $[V(s_t) - \gamma V(\quad) s_t \quad]$

[Note the difference to the algorithm in section 6:
There the total accumulated discounted reward $R^{a_t}_{s_t \to s_{end}}$
was compared with $V(s_t)$]

Actor-Critic = 'REINFORCE' with TD bootstrapping

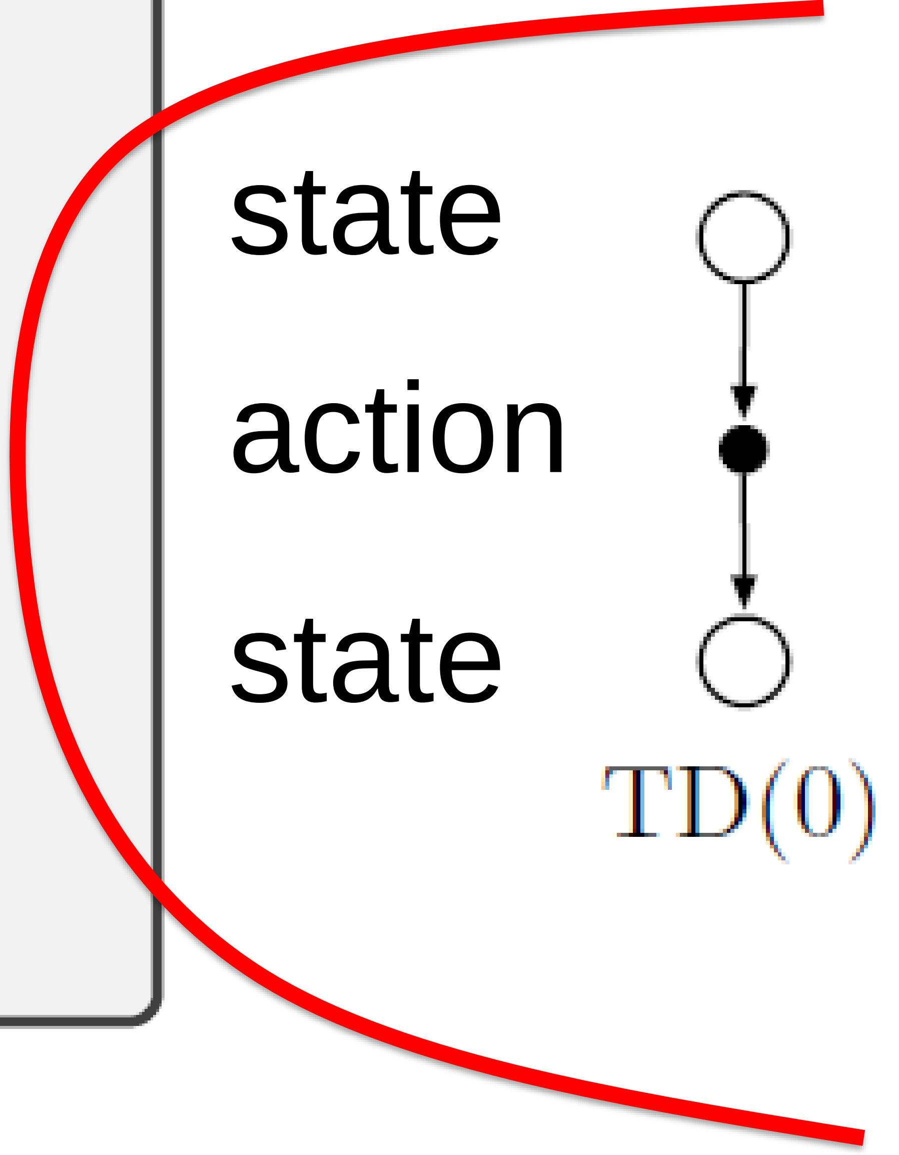


One-step Actor–Critic (episodic)

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$
Repeat forever:
    Initialize $S$ (first state of episode)
    $I \leftarrow 1$
    While $S$ is not terminal:
        $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
        Take action $A$, observe $S'$, $r$
        $\boxed{\delta \leftarrow r + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})}$        (if $S'$ is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \delta \nabla_{\mathbf{w}} \hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A|S,\boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$
```

TD error

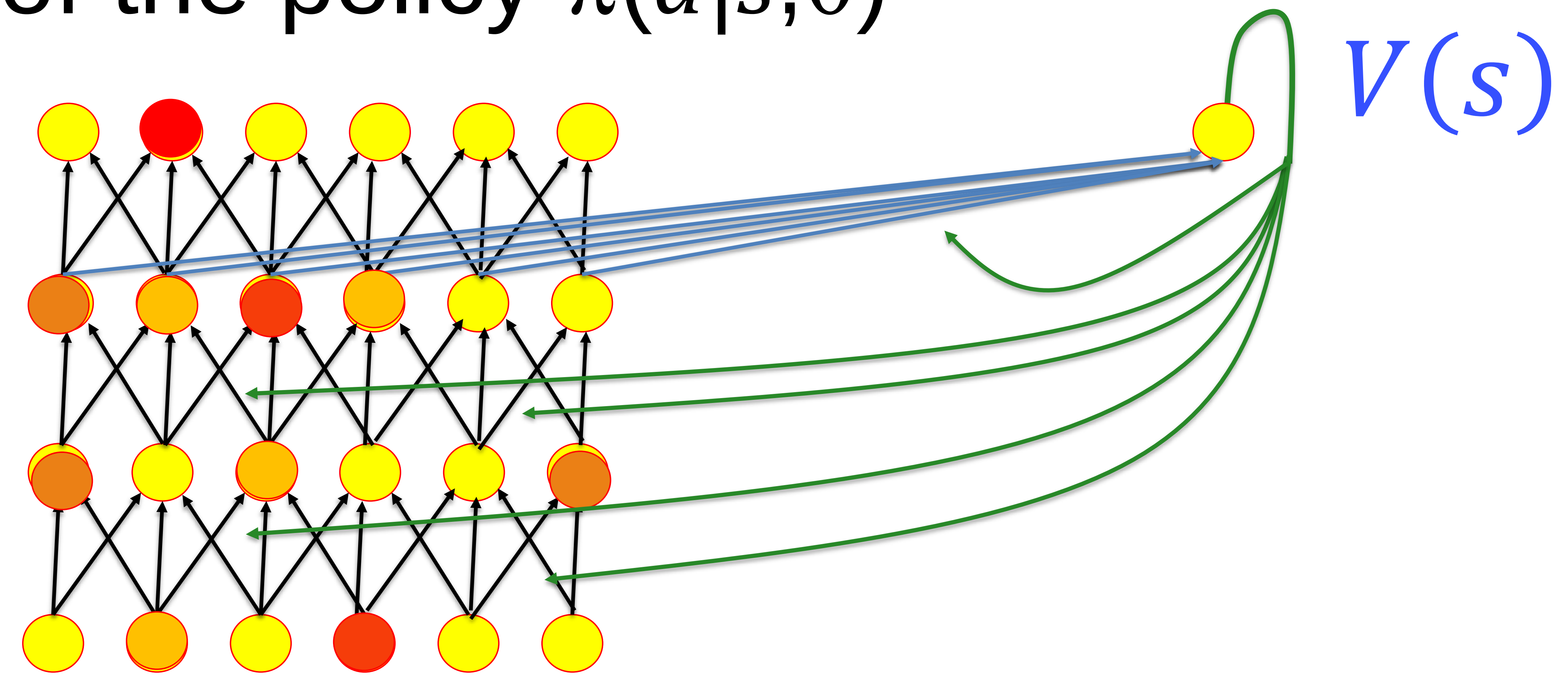


'online'

Previous slide.
Pseudocode of Algo in the notation of Sutton and Barto (2018)

Comparison: ACTOR-CRITIC (versus REINFORCE with baseline)

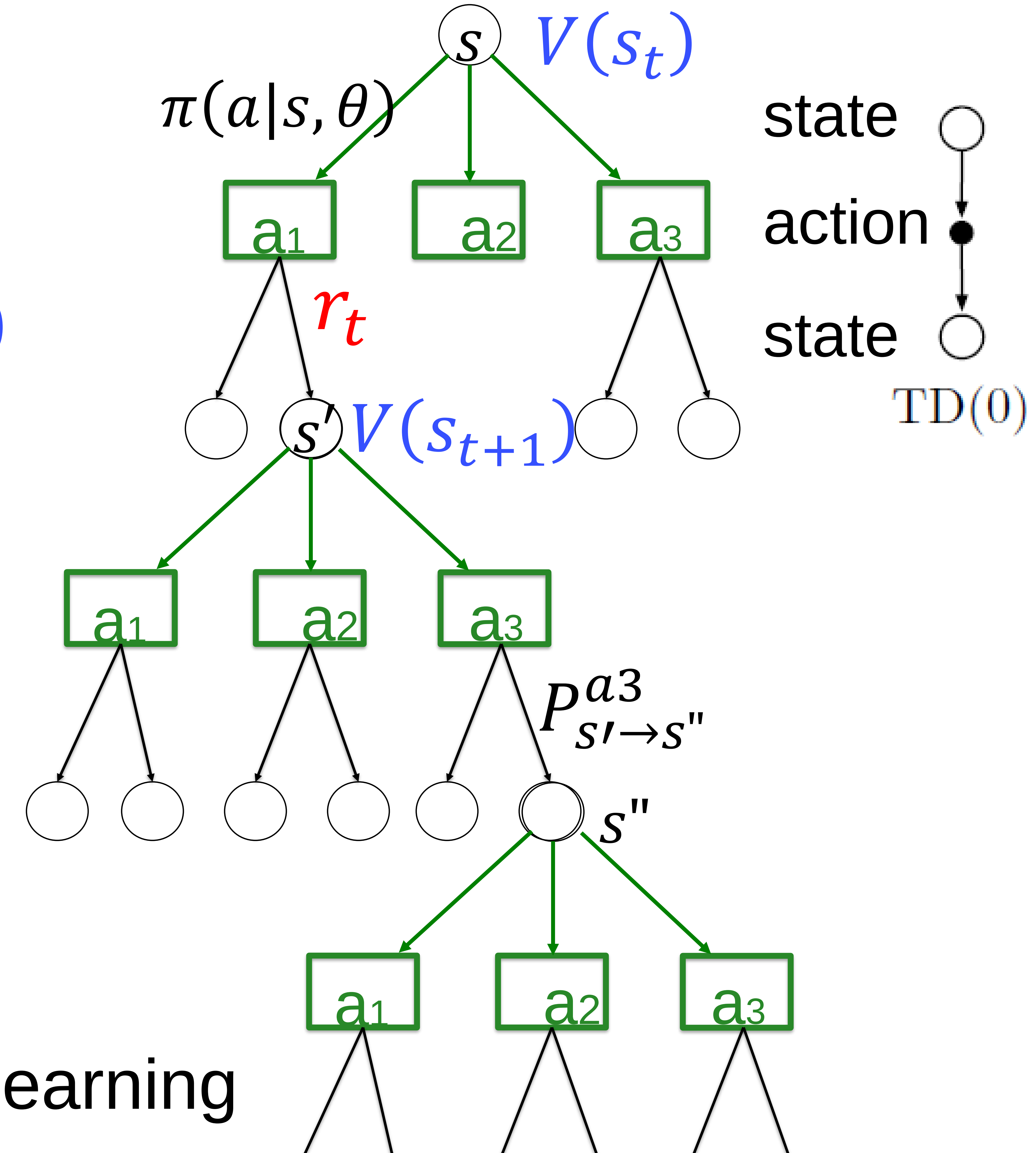
Aim of actor:
update the parameters θ
of the policy $\pi(a|s,\theta)$



update proportional toTD-error

$$\delta = \eta\,[r_t + \gamma V(s_{t+1}) - V(s_t)]$$

Aim of critic: estimate V using TD learning



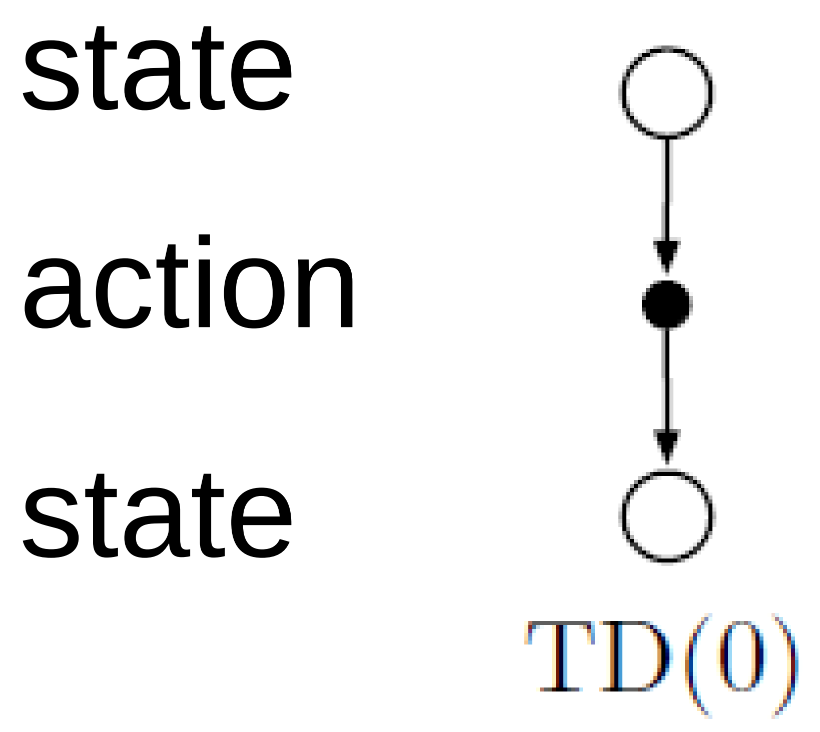
(previous slide)
For a comparison of ‘actor-critic’ with ‘REINFORCE with baseline’

In both algorithms (actor critic and REINFORCE with baseline), the actor learns actions via policy gradient.

In the actor-critic algorithm the critic learns the V-value via bootstrap TD-learning (see week 9).

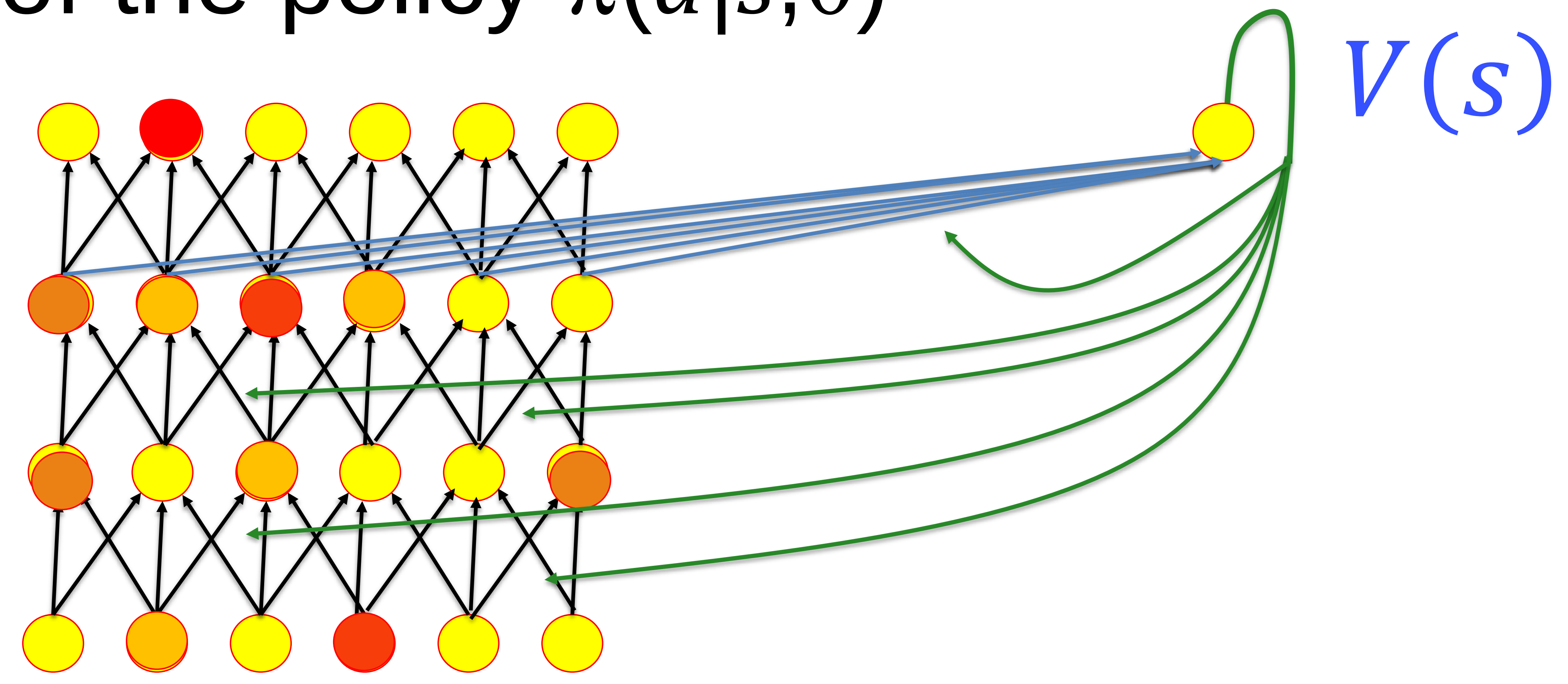
In the actor-critic algorithm the TD error is also used as the learning signal for the policy gradient.

The backup diagram of actor-critic is short:



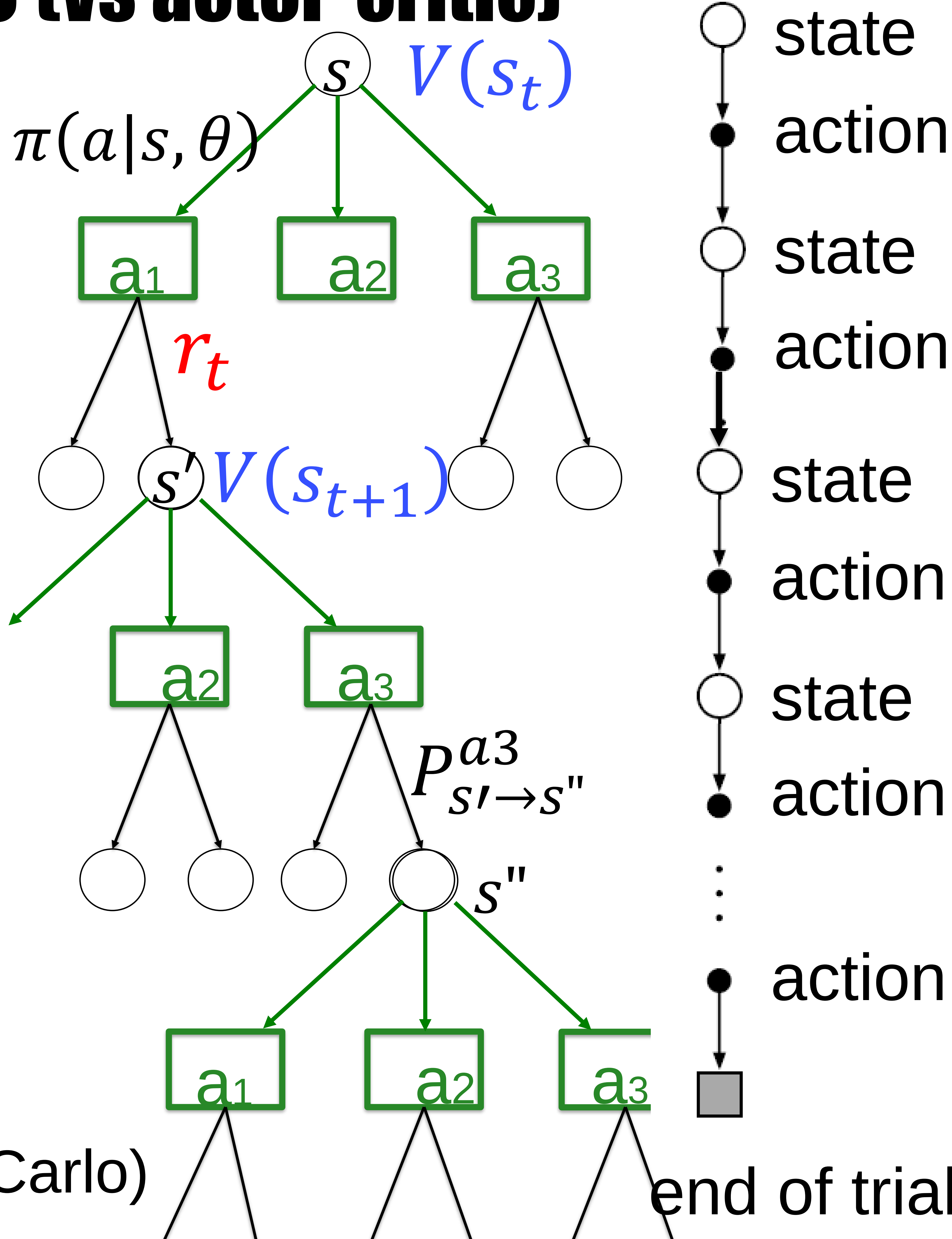
Comparsion: REINFORCE with baseline (vs actor-critic)

1. Aim of actor in REINFORCE
update the parameters θ
of the policy $\pi(a|s,\theta)$



2. update proportional to
RETURN-error: $[R^{a_t}_{s_t \to s_{end}} - V(s_t)]$

3. Aim of critic: estimate V (using Monte-Carlo)



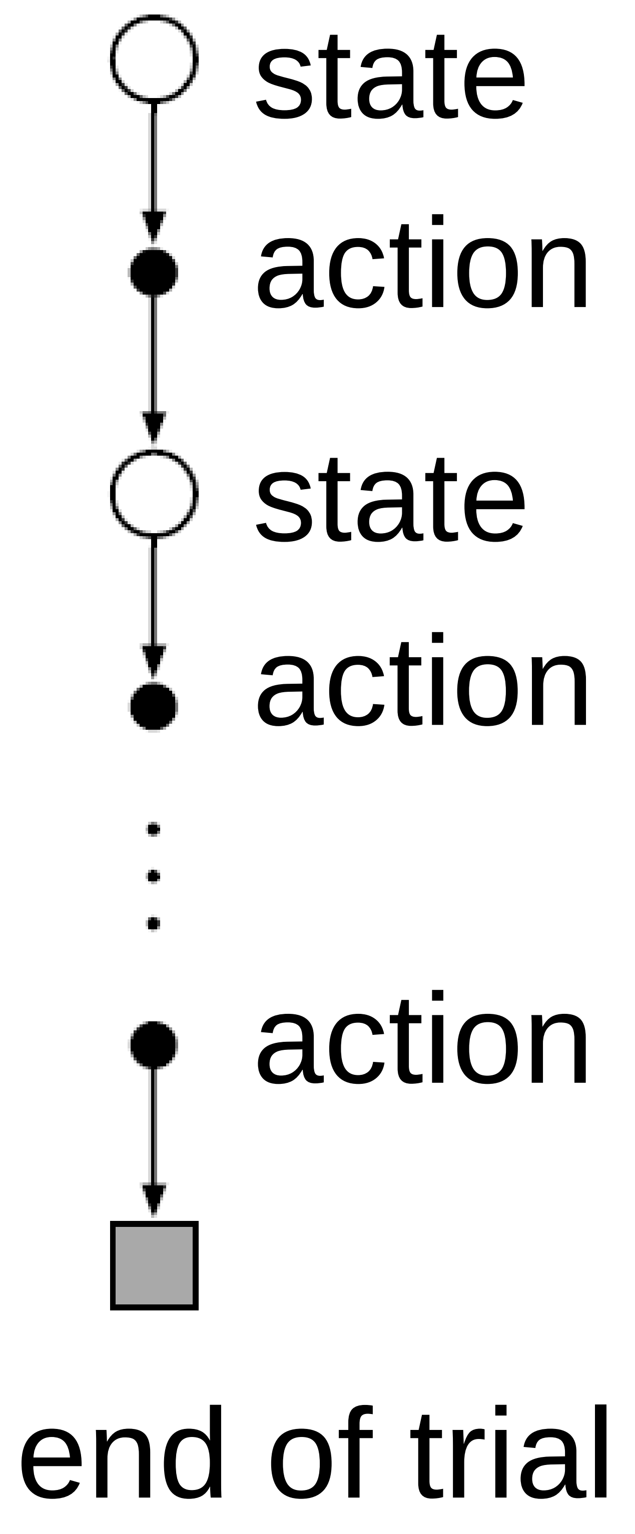
(previous slide)
We continue the comparison with REINFORCE with baseline.

In both algorithms (actor critic and REINFORCE with baseline), the actor learns actions via policy gradient.

In the REINFORCE algorithm the baseline estimator learns the V-value via Monte-Carlo sampling of full episodes.

In the REINFOCE algorithm the mismatch between actual return and estimated V-value ('RETURN error') is used as the learning signal for the policy gradient.

The Backup diagram is long:



Quiz: Policy Gradient and Deep RL

Your friend claims the following. Is he right?

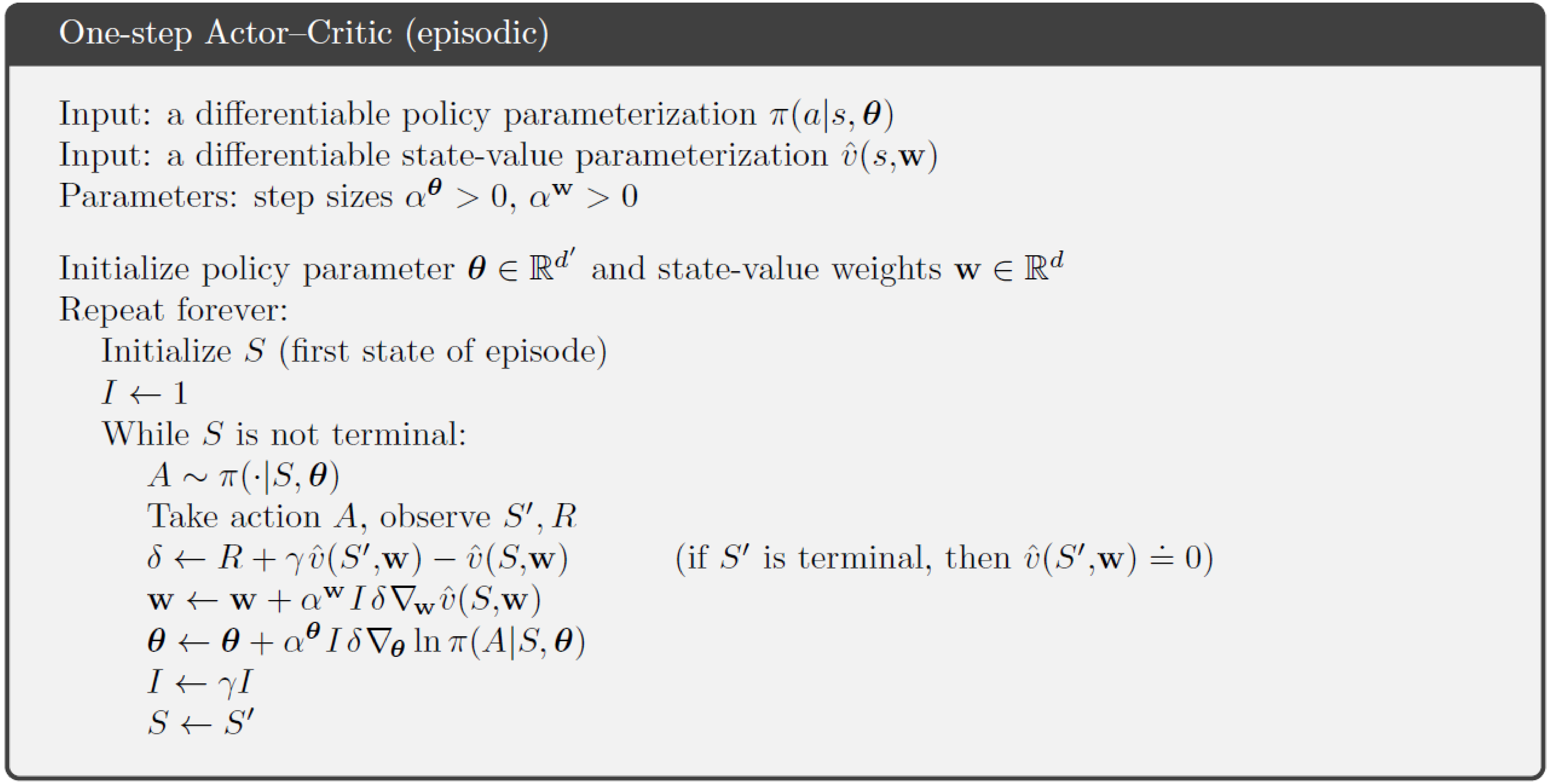
[ ] Even some policy gradient algorithms use V-values

[ ] V-values for policy gradient can be calculated in a separate deep network (but some parameters can be shared with the actor networ

[ ] The actor-critic network has basically the same architecture as deep REINFORCE with baseline: in both architectures one of the units represents the V-values

[ ] While actor-critic uses ideas from TD learning, REINFORCE WITH BASELINE does not.

Previous slide.
This is a repetition of an earlier slide.



One-step Actor–Critic (episodic)

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$
Repeat forever:
    Initialize $S$ (first state of episode)
    $I \leftarrow 1$
    While $S$ is not terminal:
        $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
        Take action $A$, observe $S', R$
        $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$        (if $S'$ is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \delta \nabla_{\mathbf{w}} \hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A|S,\boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$
```

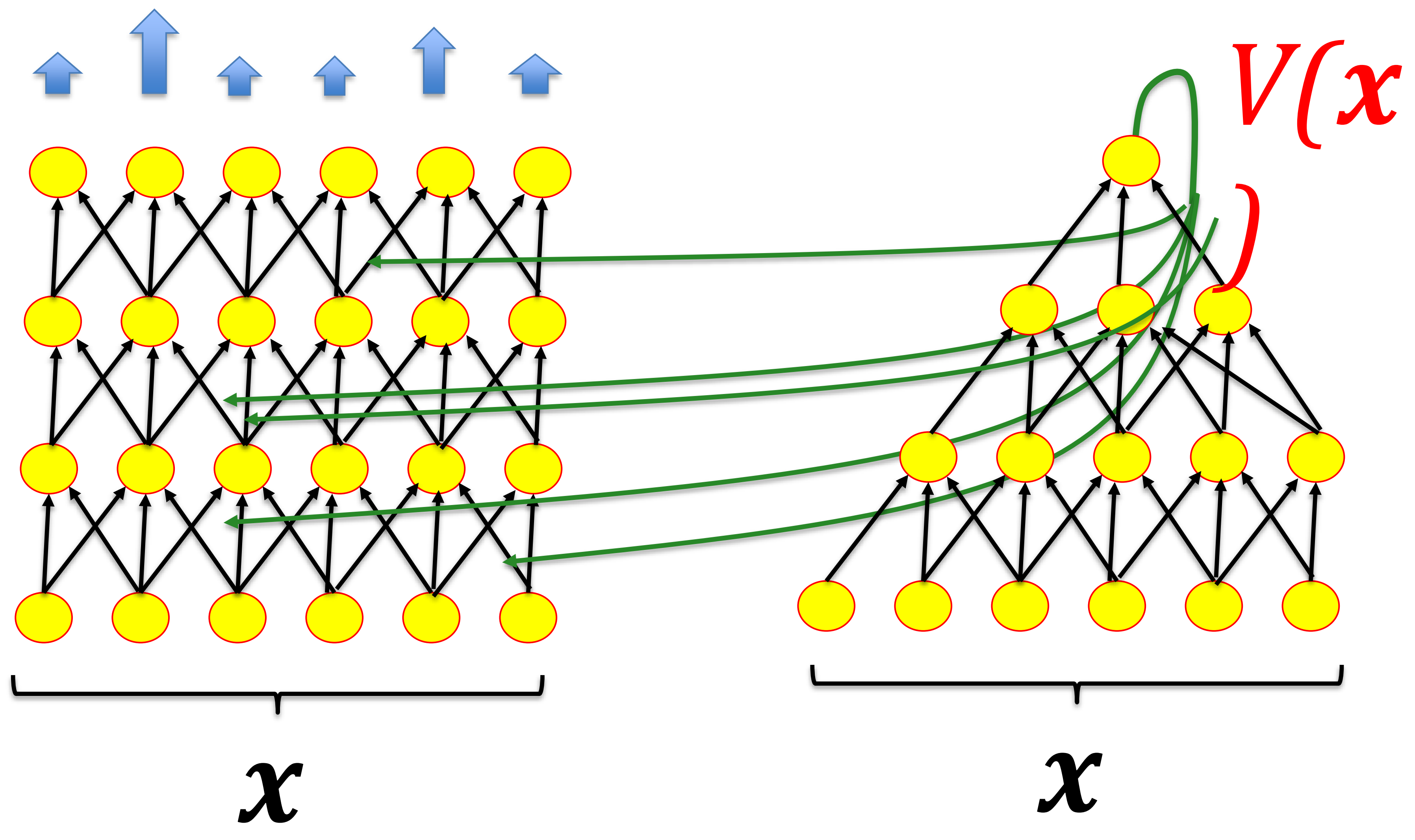
Teaching monitoring – monitoring of understanding

[ ] 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.

Terminology for Actor-Critic

Sutton and Barto and this class:		Some others
REINFORCE WITH BASELINE	→	Actor-Critic Architecture (in general sense)
Actor-Critic Algorithm (in narrow sense)	→	**Advantage Actor-Critic Algorithm**



x = states

Reinforcement Learning Lecture 5
Policy Gradient and Actor-Critic Methods

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Part 4: Eligibility traces for policy gradient

(previous slide)

Two weeks ago we discussed eligibility traces.

It turns out that policy gradient algorithm have an intimate link with eligibility traces. In fact, eligibility traces arise naturally for policy gradient algorithms.

In standard policy gradient (e.g., REINFORCE with baseline) we need to run each time until the end of the episode before we have the information necessary to update the weights.

The advantage of eligibility traces is that updates can be done truly online (similar to the actor critic with bootstrapping).

Actor-critic with eligibility traces

- Online algorithm
- Actor learns by policy gradient
- Critic learns by TD-learning

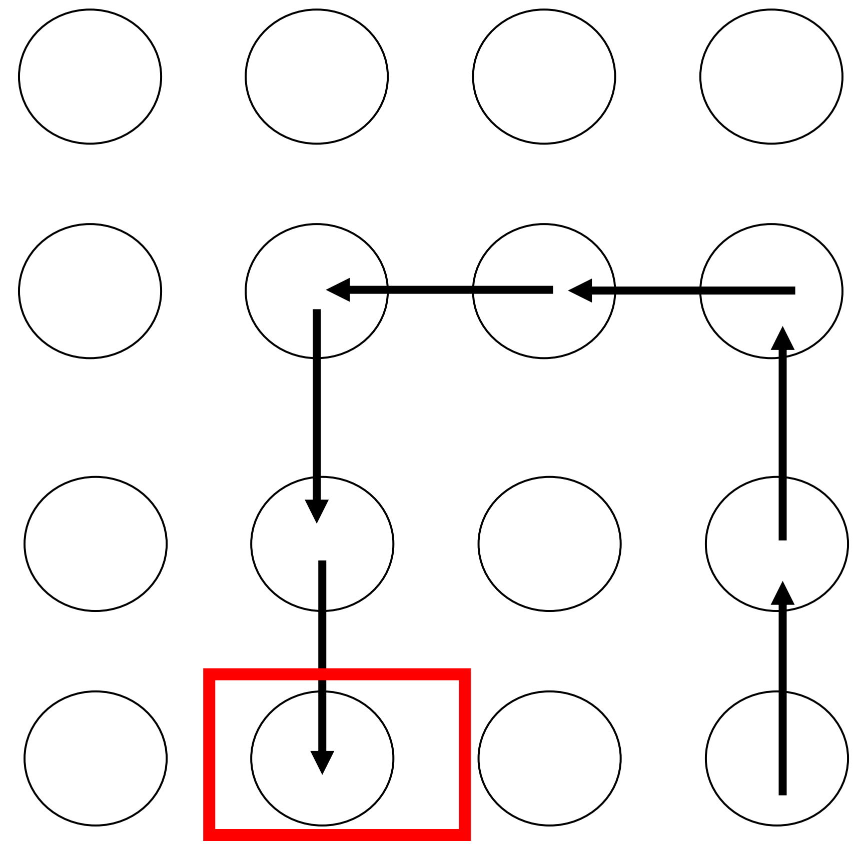
- For each parameter, one *eligibility trace*
- Update *eligibility traces* while moving
- Update weights proportional to TD-delta and *eligibility trace*

(previous slide)
The idea of policy gradient is combined with the notion of eligibility traces that we had seen two weeks ago.

The result is an algorithm that is truly online: you do not have to wait until the end of an episode to start with the updates.

Review: Eligibility Traces

Tabular: parameters *Q(s,a)*
eligibility trace *e(s,a)*

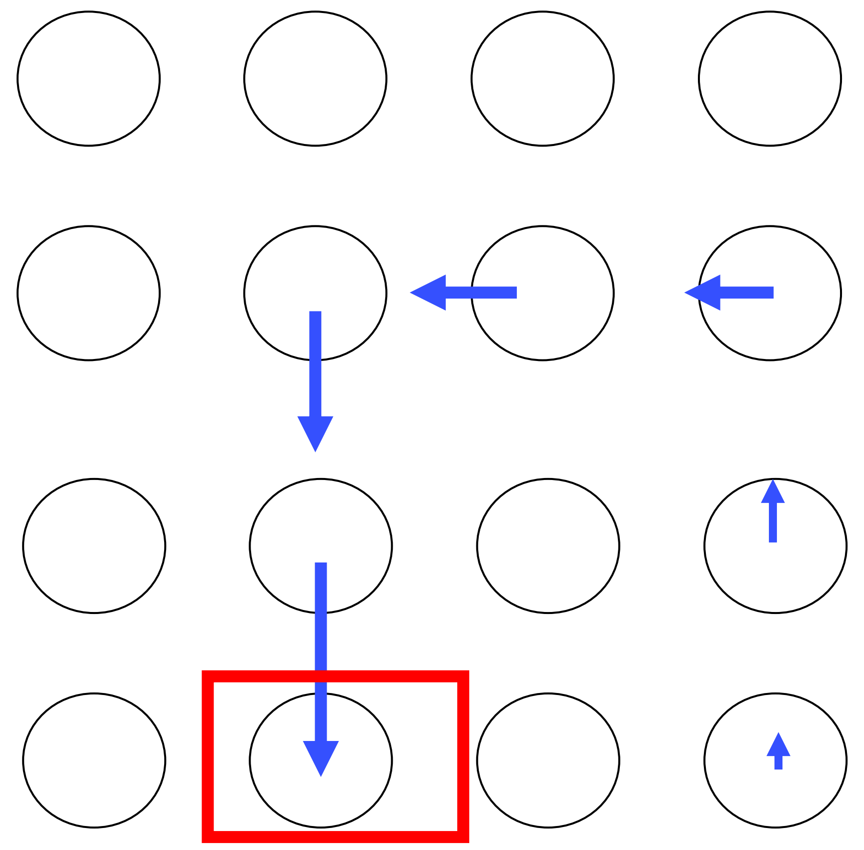


Idea:

- keep memory of previous state-action pairs
- memory decays over time
- Update an eligibility trace for state-action pair

$e(s,a) \quad \leftarrow \lambda\, e(s,a)$ decay of **all** traces

$e(s,a) \quad \leftarrow \quad e(s,a) + 1$ if action a chosen in state s



- update **all** Q-values:

$$\Delta Q(s,a) = \eta \; [\underbrace{r - (Q(s,a) - \gamma\, Q(s',a'))}_{\text{TD-delta}}]\; e(s,a)$$

Here: tabular SARSA with eligibility trace

One eligibility trace per parameter

(previous slide)
This the SARSA algorithm with eligibility traces that we had seen two weeks ago.
We had derived this algo for a tabular Q-learning model as well as for a network with basis functions and linear read-out units for the Q-values Q(s,a).

In the latter case it was not the Q value itself that had an eligibility trace, but the weights (parameters) that contributed to that Q-value.

We now use the same idea.

Eligibility Traces for actor-critic

Idea:

- keep memory of previous 'candidate updates'
- memory decays over time
- Update an **eligibility trace for each parameter**

$z_k \leftarrow z_k \ \lambda$ decay of **all** traces

$z_k \leftarrow z_k + \frac{d}{d\theta_k} \ln[\pi(a|s,\theta_k)]$ increase of **all** traces

- update **all** parameters of 'actor' network:

$$\Delta\theta_k = \eta \ [\underbrace{r - (V(s_t) - \gamma V(s_{t+1}))}_{\text{TD-delta}}] \ z_k$$

Here: policy gradient with eligibility trace (actor network)

(previous slide)
Eligibility traces can be generalized to deep networks.
Here we focus on the actor network.
For each parameter θ_k of the network we have a shadow parameter z_k : the eligibility trace.
Eligibility traces decay at each time step ($\lambda < 1$) and are updated proportional to the derivative of the log-policy. Interpretation:
The update of the eligibility trace can be seen as a 'candidate parameter update' – but it is not yet the 'real' update of the actual parameters.

The update of the actual parameters w_k of the actor network are proportional to the eligibility trace z_k and the TD-error

$$\delta = [r_t + \gamma V(s_{t+1}) - V(s_t)]$$

$$= [r_t - [V(s_t) - \gamma V(s_{t+1})]]$$

Parameters are updated at each time step of the episode (as opposed to Monte-Carlo where one has to wait for the end of the episode). Hence 'true online'.

NETWORK for Algorithm in Pseudo-code by Sutton and Barto.

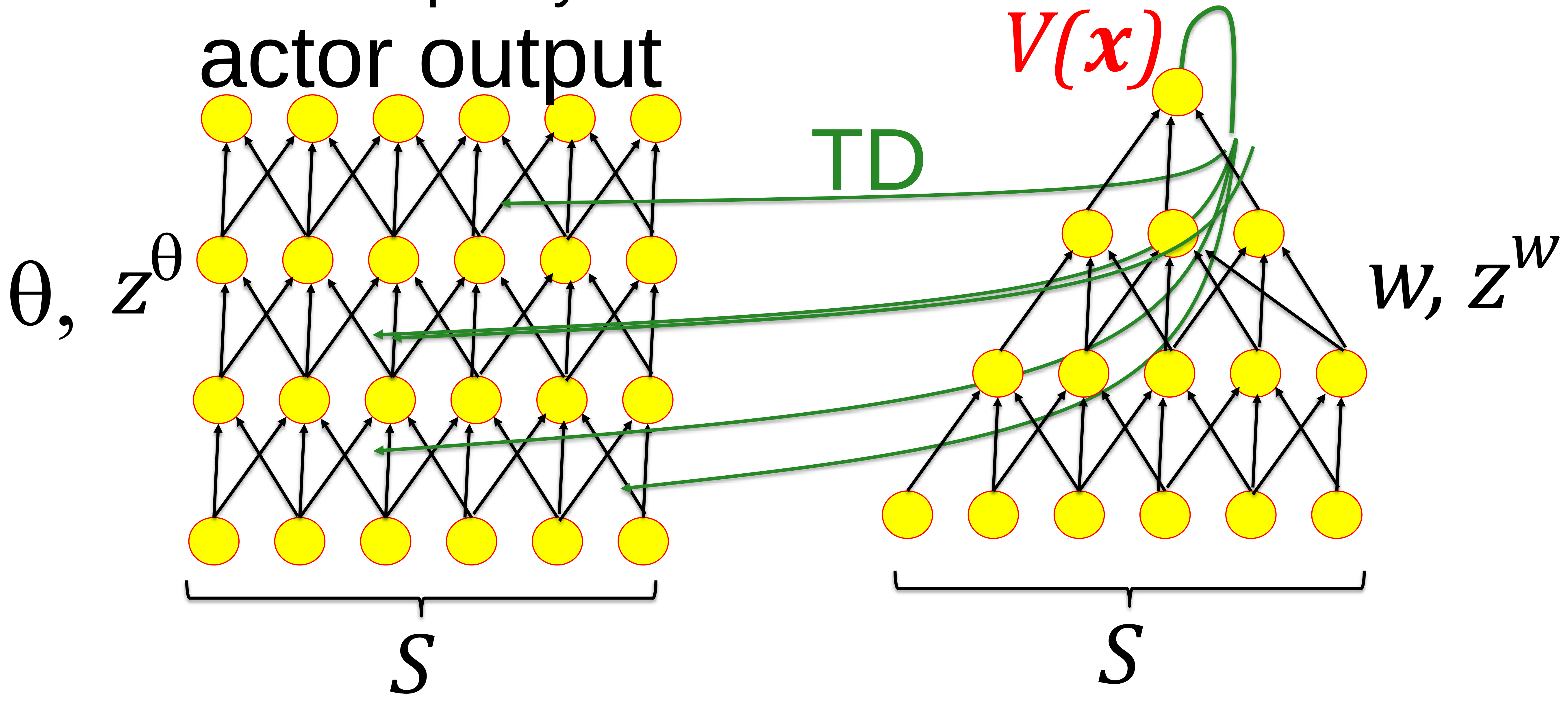
The actor network has parameters θ

Eligibility traces of actor have parameters z^{θ}.

The critic network has parameters w.

Eligibility traces of critic have parameters z^{w}.

Actor chooses actions with policy π



(previous slide) **Algorithm in Pseudo-code by Sutton and Barto.**

The actor network has parameters θ
While the critic network has parameters w.

The actor network is learned by policy gradient with eligibility traces.
The critic network by TD learning with eligibility traces.

Candidate updates are implemented as eligibility traces z.

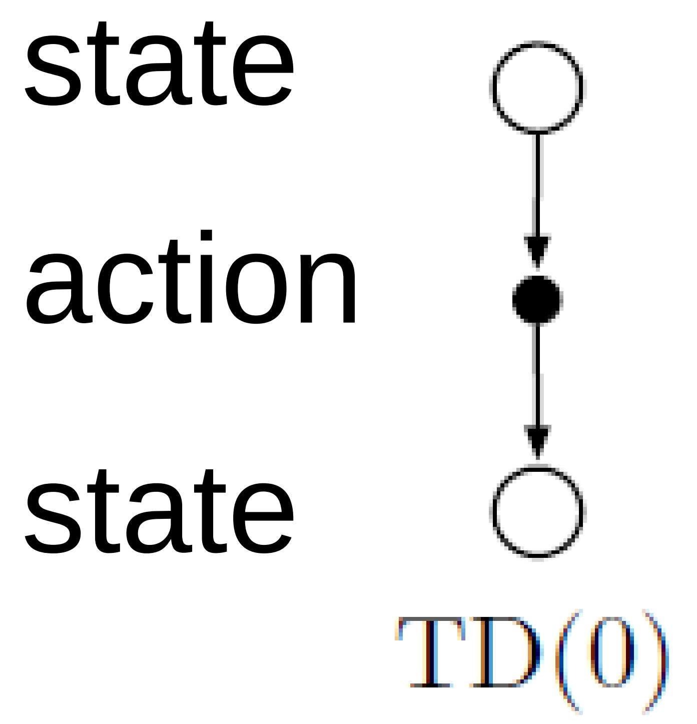
Actor-Critic with Eligibility traces (bootstrapping/TD trick)

Actor–Critic with Eligibility Traces (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: trace-decay rates $\lambda^{\boldsymbol{\theta}} \in [0,1]$, $\lambda^{\mathbf{w}} \in [0,1]$; step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize $S$ (first state of episode)
    $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ ($d'$-component eligibility trace vector)
    $\mathbf{z}^{\mathbf{w}} \leftarrow \mathbf{0}$ ($d$-component eligibility trace vector)
    $I \leftarrow 1$
    Loop while $S$ is not terminal (for each time step):
        $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
        Take action $A$, observe $S'$, $r$
        $\boxed{\delta \leftarrow r + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})}$        (if $S'$ is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{z}^{\mathbf{w}} \leftarrow \gamma \lambda^{\mathbf{w}} \mathbf{z}^{\mathbf{w}} + I \nabla \hat{v}(S,\mathbf{w})$
        $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \gamma \lambda^{\boldsymbol{\theta}} \mathbf{z}^{\boldsymbol{\theta}} + I \nabla \ln \pi(A|S,\boldsymbol{\theta})$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \mathbf{z}^{\mathbf{w}}$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \delta \mathbf{z}^{\boldsymbol{\theta}}$
        ~~$I \leftarrow \gamma I$~~
        $S \leftarrow S'$
```

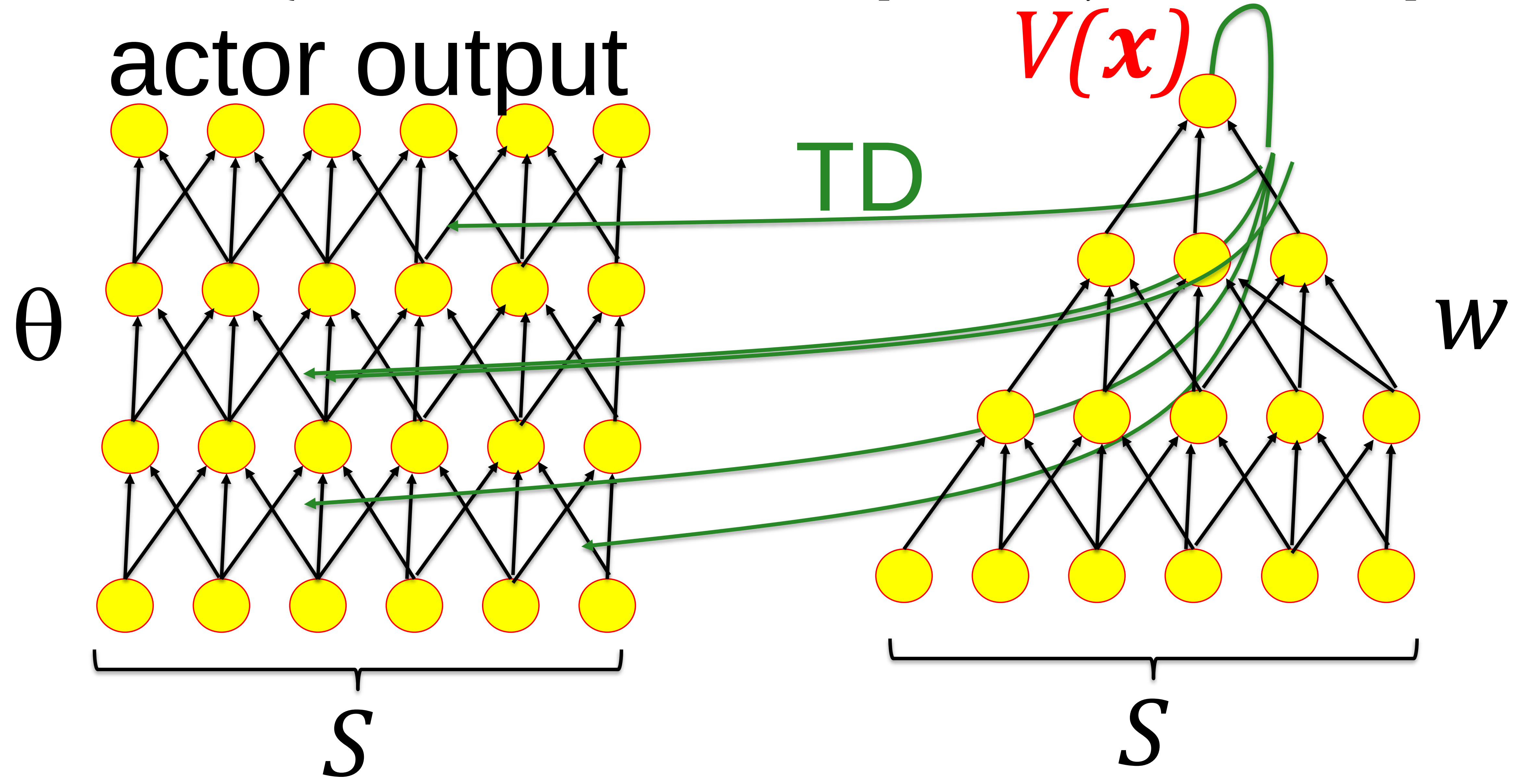
TD error

I suggest to cut this: use continuous and not episodic setting (see next section)



(previous slide) **Algorithm in Pseudo-code by Sutton and Barto.**
The actor network has parameters θ
While the critic network has parameters w

The actor network is learned by policy gradient with eligibility traces.
The critic network by TD learning with eligibility traces.
Note that Sutton and Barto include a discount factor γ (in the update of the eligibility trace) but in the exercises we will see that the discount factor can (to an excellent approximation) be absorbed into λ. This algo is for 'episodic', i.e. problem with terminal states (which causes the step 'I $\leftarrow \gamma I$ '; we drop this step later)



Quiz: Policy Gradient and Reinforcement learning

[ ] While actor-critic uses ideas from TD learning, REINFORCE with baseline uses Monte-Carlo estimates of V-values

[ ] Eligibility traces are ‘shadow’ variables for each parameter

[ ] Eligibility traces appear **naturally** in policy gradient algos.

(your comments)

The proof of the last item is what we will sketch now – proof in the exercises.

Introduction to Reinforcement Learning

Two types of algo with different philosophy

	TD learning	Policy Gradient
tabular	Bellman eq., Q/V values, Bootstrap	Direct action modeling Not Bootstrap
temporal smoothing	eligibility traces /n-step SARSA(λ), Q(λ)	eligibility traces (natural) REINFORCE (λ) w. baseline
continuous/ function approx	yes, ad hoc, heuristic semi-gradient	yes, natural, but slow

Actor-critic with eligibility traces and TD-updates

Previous page

Summary: Reinforcement Learning has two major types of algorithms, i.e. TD-learning and Policy Gradient.

The advantage of TD learning is the bootstrapping effect. Combined with ad-hoc eligibility traces it yields powerful algorithms for the tabular setting. Excellent example are SARSA(λ) and Q(λ)

The advantage of policy gradient are two-fold:

(i) since we optimize directly the action outcomes, effects of function approximation (such as smoothing, imprecise representations of different cases) are automatically taken into account

(ii) In the continuing setting, eligibility traces arise naturally.

By default (without baseline subtraction) the algorithm is slow.

The best way to add baseline subtraction is the actor-critic architecture.
It combines the best of both worlds, since the V-value for the base line uses TD learning.

Introduction to Reinforcement Learning

Two types of algo with different philosophy

	TD learning	Policy Gradient
tabular Lecture RL1+2	Bellman eq., Q/V values, Bootstrap	Direct action modeling Not Bootstrap Lecture RL4
temporal smoothing Lecture RL 3	eligibility traces /n-step SARSA(λ), Q(λ)	eligibility traces (natural) REINFORCE (λ) w. baseline
continuous/ function approx	yes, ad hoc, heuristic semi-gradient	yes, natural, but slow

Lecture RL5,
today

Actor-critic with eligibility traces
and TD-updates

Previous page

Summary:

During the first 7 weeks we covered in 5 lectures all the major types of algorithms.

Today is the end of the ‘Introduction’ part.

We are now ready to apply the algorithms in various applications –
or study specific topics such as the relation to biology and distributed hardware.
This will be done in the coming weeks.

At the end of the lecture today, you should be able to read current literature on Reinforcement Learning.

Reinforcement Learning Lecture 5
Policy Gradient and Actor-Critic Methods

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Part 5: Eligibility traces (Math)

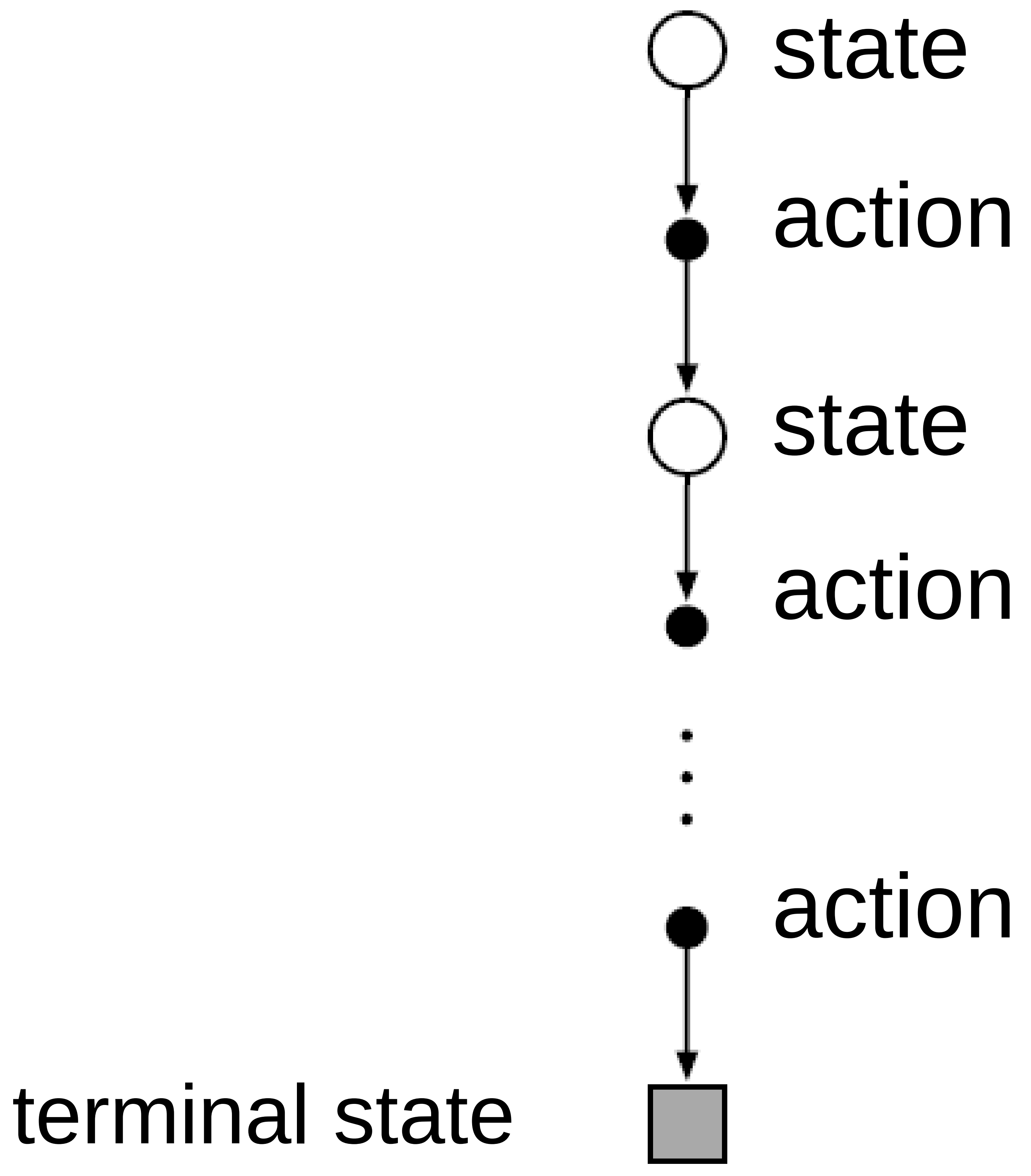
Summary: Eligibility traces from Policy Gradient

‘Policy Gradient Theorem’:
An algorithm with eligibility traces arises naturally in the frame work of policy gradient.

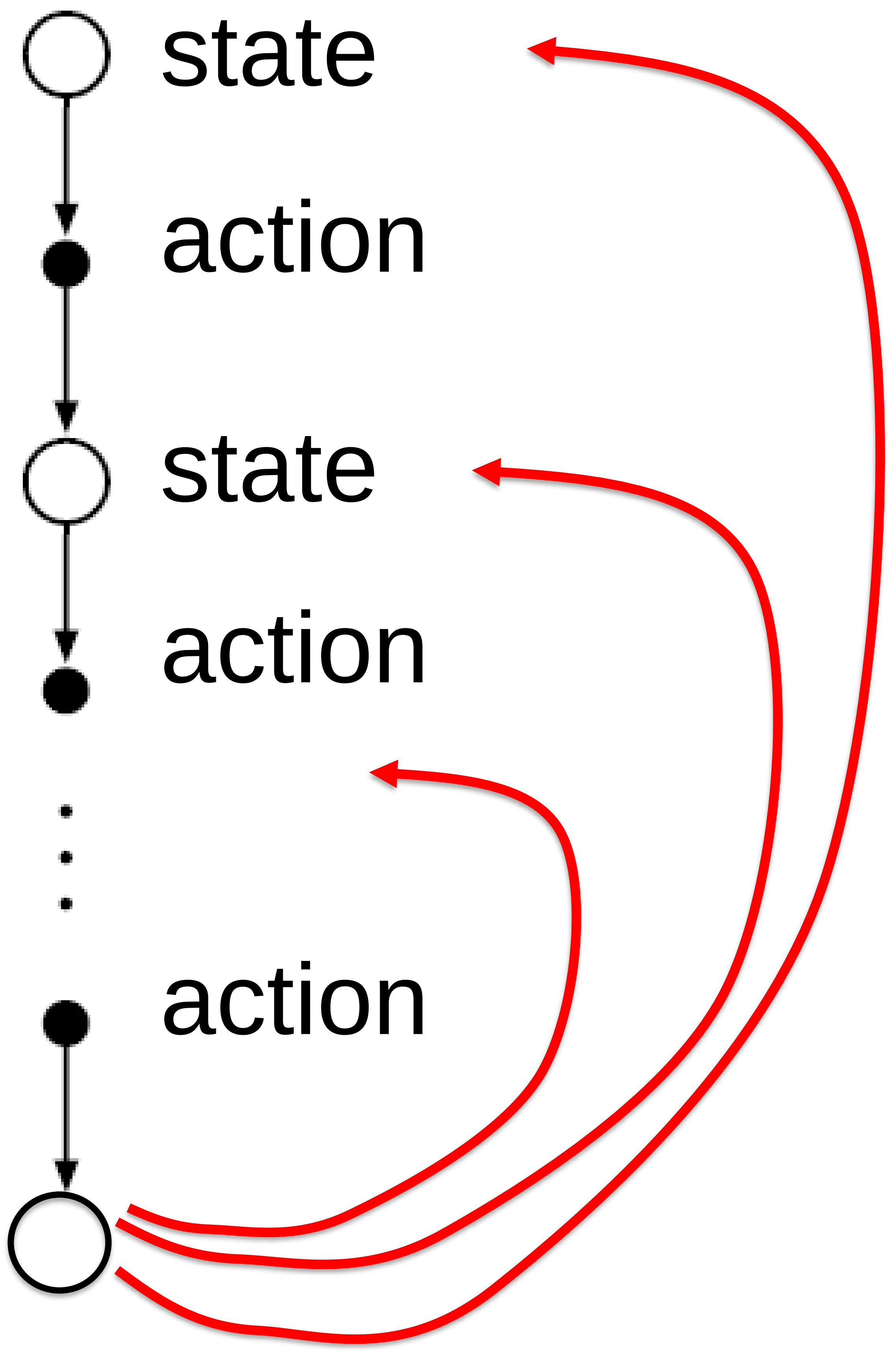
Proof: assumes setting of **‘continuing’** environment
(as opposed to episodic).

Aim: optimize returns from ALL states

Review: episodic setting versus infinite horizon continuing setting



end of episode



episode never ends: continuing setting

Previous slide.
Episodic setting means that we have a fixed start state and a terminal state. Once we have arrived at the terminal state we put the agent back to the start state.

Continuing setting means that every state is a possible start state. The environment allows loops. There is no strict terminal state since we can always imagine that the agent is from the nominal ‘terminal state’ automatically moved on to a (probabilistically selected) state inside the environment.

In both cases rewards can be collected at any state in the environment.

Review: Policy Gradient over multiple time steps (episodic)

Optimize RETURN from starting state s_t:

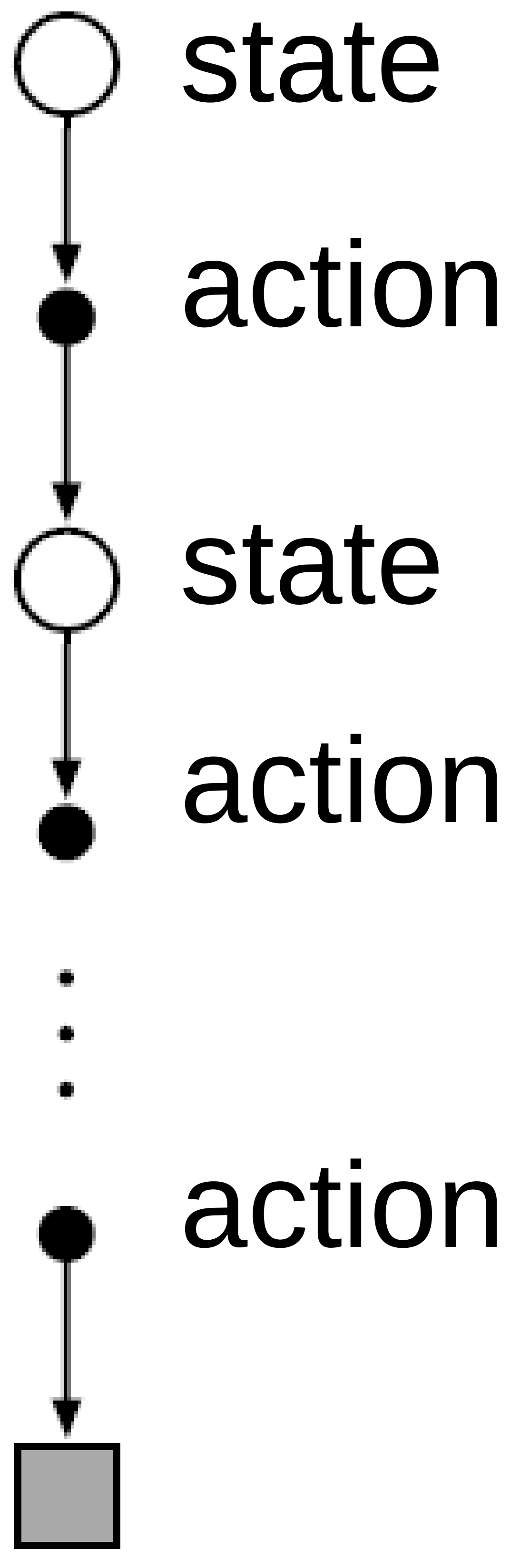
Calculation yields several terms of the form

Total accumulated discounted reward collected in one episode starting at s_t, a_t

$$\Delta\theta_j \propto \left[R^{a_t}_{s_t \to s_{end}}\right]\frac{d}{d\theta_j}\ln[\pi(a_t|s_t,\theta)]$$

$$+\gamma\left[R^{a_{t+1}}_{s_{t+1} \to s_{end}}\right]\frac{d}{d\theta_j}\ln[\pi(a_{t+1}|s_{t+1},\theta)$$

+ …



end of episode

Previous slide.
This is a repetition of an earlier slide.

NOW: Policy Gradient over multiple time steps (infinite horizon)

Optimize EXPECTED average RETURN from ALL STATES:

$\frac{1}{K}$E[Return from starting state s_t

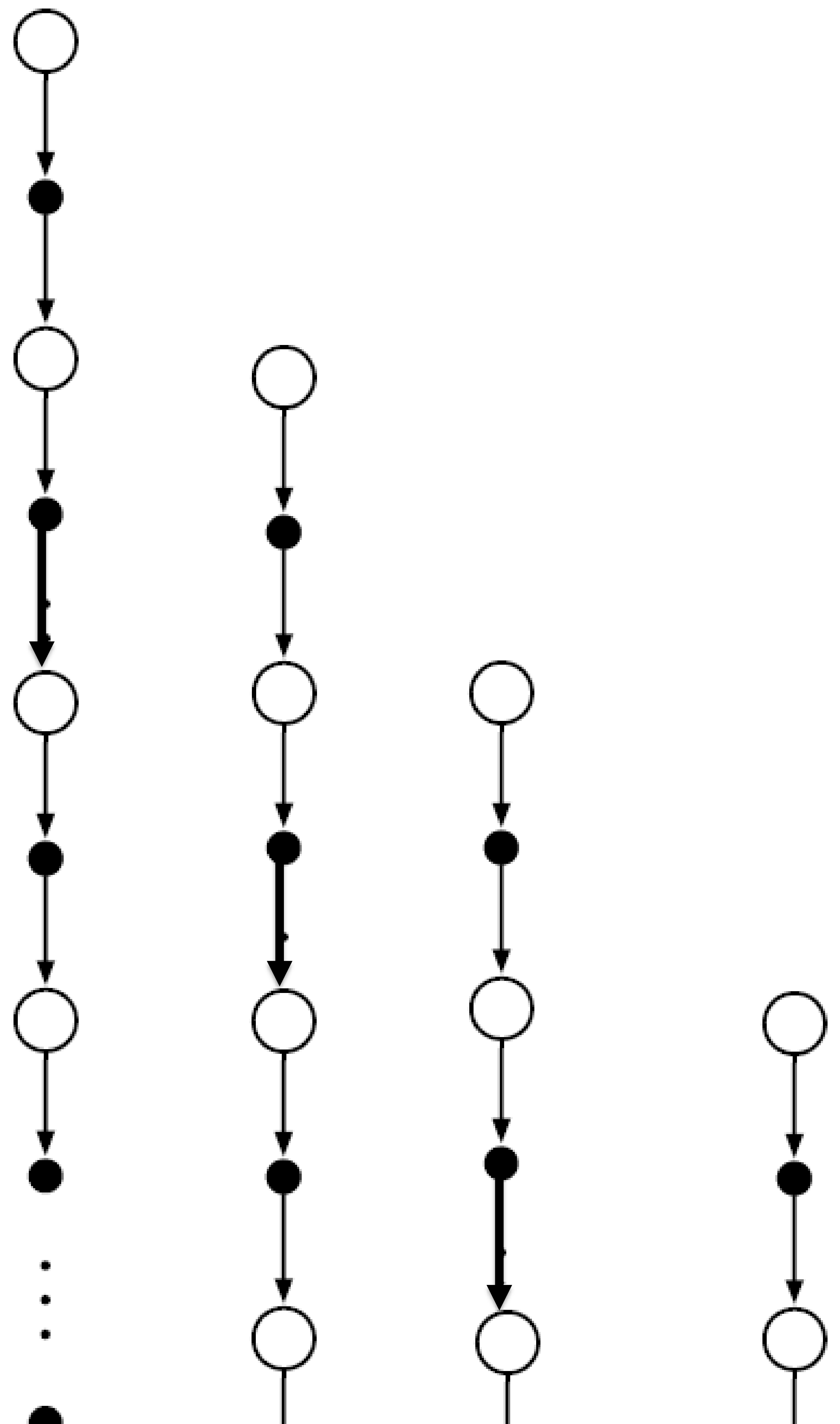
+ Return from next state s_{t+1}

+ Return from next state s_{t+2}

+ Return from next state s_{t+3}

… Return from s_{t+K}]

(and take limit $K \rightarrow \infty$)



Previous slide.
The Return from the first state is the sum over all future (discounted) rewards
starting at s_t
The Return from the next state is the sum over all future (discounted) reward
starting at s_{t+1}
… and so forth.

But the aim is to optimize the Expected Returns from ALL STATES.

This will give many terms

5. Policy Gradient over multiple time steps (Preparation for Exercise)

Step 1: Rewrite $R^{a_t}_{s_t \to s_{end}} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3}$

Step 2a: use log-policy update formula (*) for state s_t

Step 2b: Use same update formula, but for states s_{t+1}, s_{t+2}

Step 3: Sum results from step 2 and reorder terms according to r_{t+n}

$$(*)\ \Delta\theta_j \propto [R^{a_t}_{s_t \to s_{end}}] \frac{d}{d\theta_j} \ln[\pi(a_t|s_t, \theta)]$$

$$+\gamma [R^{a_{t+1}}_{s_{t+1} \to s_{end}}] \frac{d}{d\theta_j} \ln[\pi(a_{t+1}|s_{t+1}, \theta)]$$

$+ \ldots$

Blackboard

Previous slide.
This is the start of the calculations for the blackboard part.
It will also appear in the exercise.

5. Policy Gradient for eligibility traces (Exercise)

Step 4: Introduce ‘shadow variables’ for eligibility trace

$z_k \leftarrow z_k \ \lambda$ decay of **all** traces

$z_k \leftarrow z_k + \frac{d}{d\theta_k} \ln[\pi(a|s, \theta_k)]$ update of **all** traces

Step 5: Rewrite update rule for parameters with eligibility trace

$$\Delta\theta_k = \eta \ r_t \ z_k$$

Previous slide.
This is a repetition of the exercise

5. Algo of Policy Gradient with eligbility traces (Exercise)

Run trial. At each time step, observe state, action, reward

1) Update eligibility trace for all parameters k

$z_k \leftarrow z_k \;\gamma$ decay of **all** traces

$z_k \leftarrow z_k + \frac{d}{d\theta_k}\ln[\pi(a|s,\theta_k)]$ increase of **all** traces

2) update all parameters k

$$\Delta\theta_k = \eta \; r_t \; z_k$$

Note: I have set $\lambda = \gamma$ as a result of the calculation

Previous slide.

And these two updates can now be mapped to the algorithm of Sutton and Barto that we saw a few slides before.

Conclusion:
(i) eligibility traces are a compact form for rewriting a policy gradient algorithm.

(ii) The derivations show that the decay factor of policy gradient must be equal to the discount factor of the Return, hence $\gamma=\lambda$

For an episodic setting boundary effects would appear – but since we are here in the infinite-horizon continuing setting, we have nor boundary effects. Indeed, there are minor differences at the 'bounderies' that is, the beginning and end of each episode – but these do not matter if we think of the environment as being very large – potentially infinitely large because of loops.

Summary: Eligibility traces from Policy Gradient

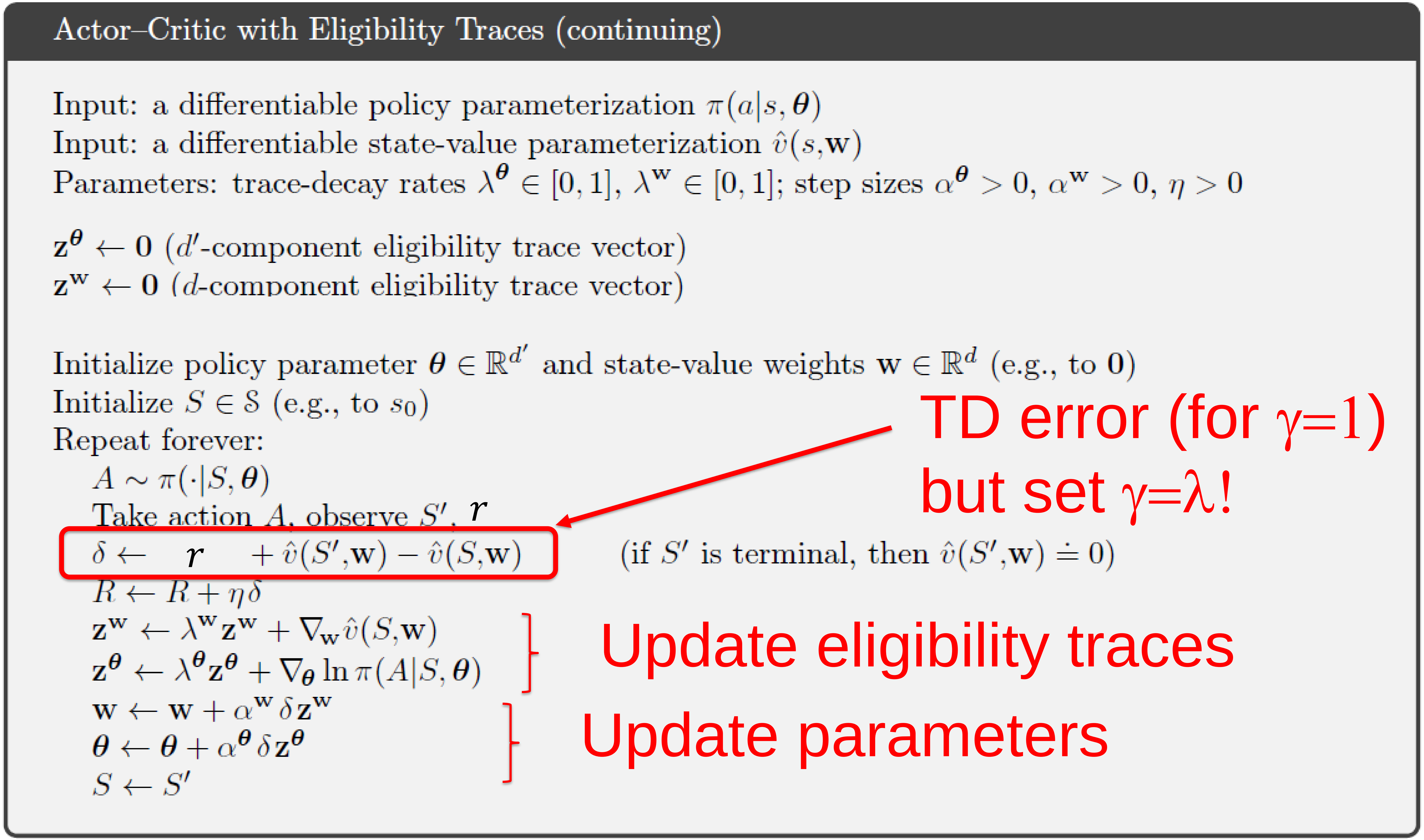
‘Policy Gradient Theorem’:
An algorithm with eligibility traces arises **naturally** in the framework of policy gradient.
Decay rate of eligibility traces = Discount Factor γ

Proof: - assumes setting of **‘continuing’** environment
(as opposed to episodic)
- we optimize the expected average
return from **ALL** states

Note: Algorithm for ‘continuing’ setting is the better one
(better than the one for episodic setting)

Previous slide. Your notes

Actor Critic with Eligibility Traces (continuing setting)



Actor–Critic with Eligibility Traces (continuing)

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w})$
Parameters: trace-decay rates $\lambda^{\boldsymbol{\theta}} \in [0,1]$, $\lambda^{\mathbf{w}} \in [0,1]$; step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$, $\eta > 0$

$\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ ($d'$-component eligibility trace vector)
$\mathbf{z}^{\mathbf{w}} \leftarrow \mathbf{0}$ ($d$-component eligibility trace vector)

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Initialize $S \in \mathcal{S}$ (e.g., to $s_0$)
Repeat forever:
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action $A$, observe $S'$, $r$
    $\boxed{\delta \leftarrow r + \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})}$        (if $S'$ is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\bar{R} \leftarrow \bar{R} + \eta\delta$
    $\mathbf{z}^{\mathbf{w}} \leftarrow \lambda^{\mathbf{w}}\mathbf{z}^{\mathbf{w}} + \nabla_{\mathbf{w}}\hat{v}(S,\mathbf{w})$
    $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \lambda^{\boldsymbol{\theta}}\mathbf{z}^{\boldsymbol{\theta}} + \nabla_{\boldsymbol{\theta}} \ln \pi(A|S,\boldsymbol{\theta})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\mathbf{z}^{\mathbf{w}}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\delta\mathbf{z}^{\boldsymbol{\theta}}$
    $S \leftarrow S'$
```

TD error (for $\gamma=1$) but set $\gamma=\lambda$!

Update eligibility traces

Update parameters

Adapted from Sutton&Barton 2018

Notes:

1) Here Sutton and Barto have suppressed the factor γ. (by setting $\gamma=1$).
   They do not identify $\gamma=\lambda$.

2) The starting point of the derivation of Sutton and Barto is to optimize the average reward (I optimized average return with discount factor γ.

3) However, I would set $\gamma=\lambda$ and add the λ in the update for the TD error:

$$\boxed{\delta \leftarrow \eta\, [\, r_t + \lambda V(S', w) - V(S, w)]}$$

The reason is
(i) that the critic learns V-values with TD-learning and should use the same discounting $\gamma=\lambda$ as the actor.
(ii) The baseline subtraction for an actor that uses discounted cumulative rewards for returns should match the consistency condition of the Bellman equation.

Reinforcement Learning Lecture 5
Policy Gradient and Actor-Critic Methods

Wulfram Gerstner

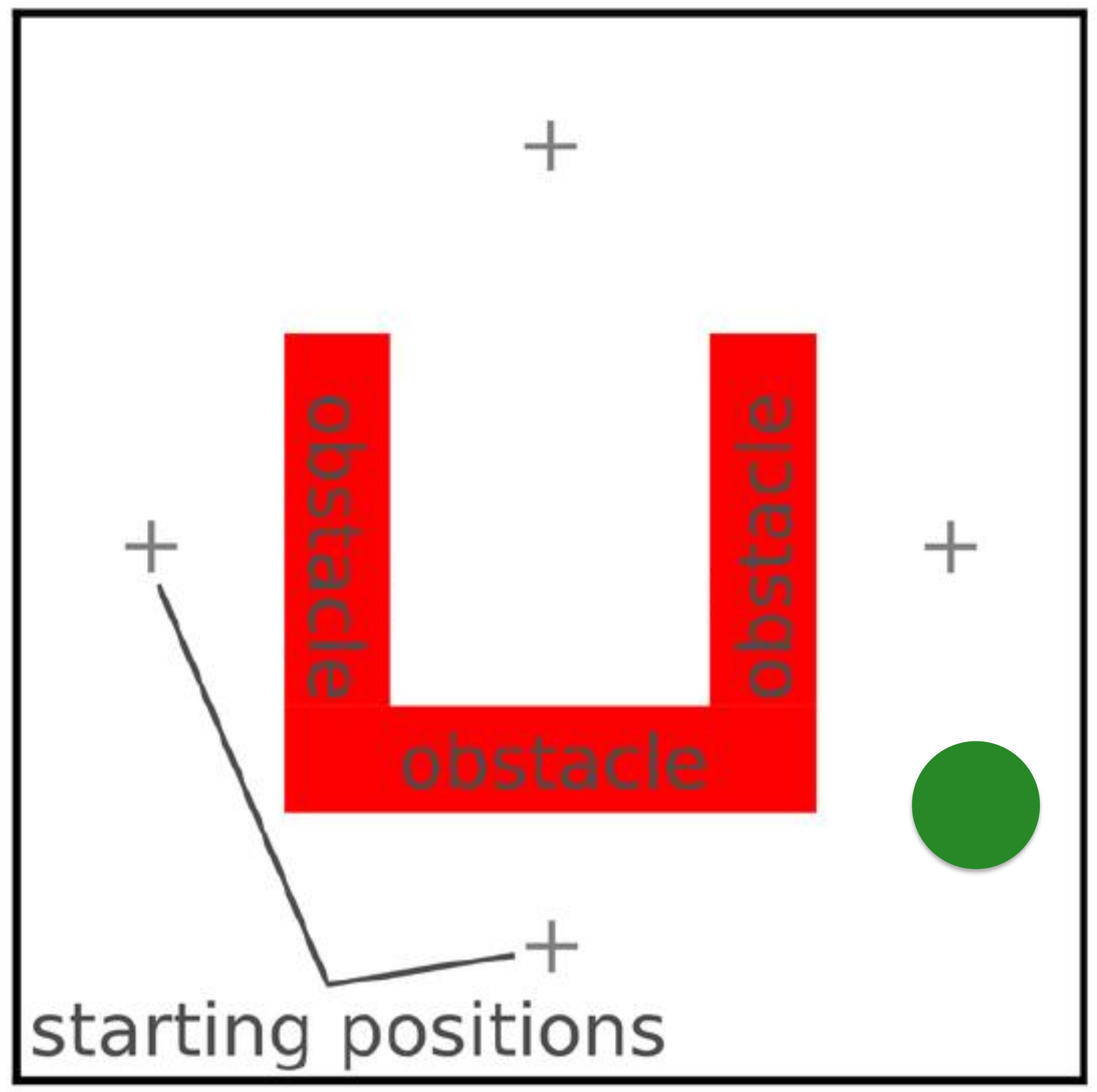
EPFL, Lausanne, Switzerland

Previous slide.
Final point: Model-based versus Model-free

7. Model-based versus Model-free

What happens in RL when you shift the goal after learning?

A



Previous slide.

Final point: are we looking at the right type of RL algorithm?

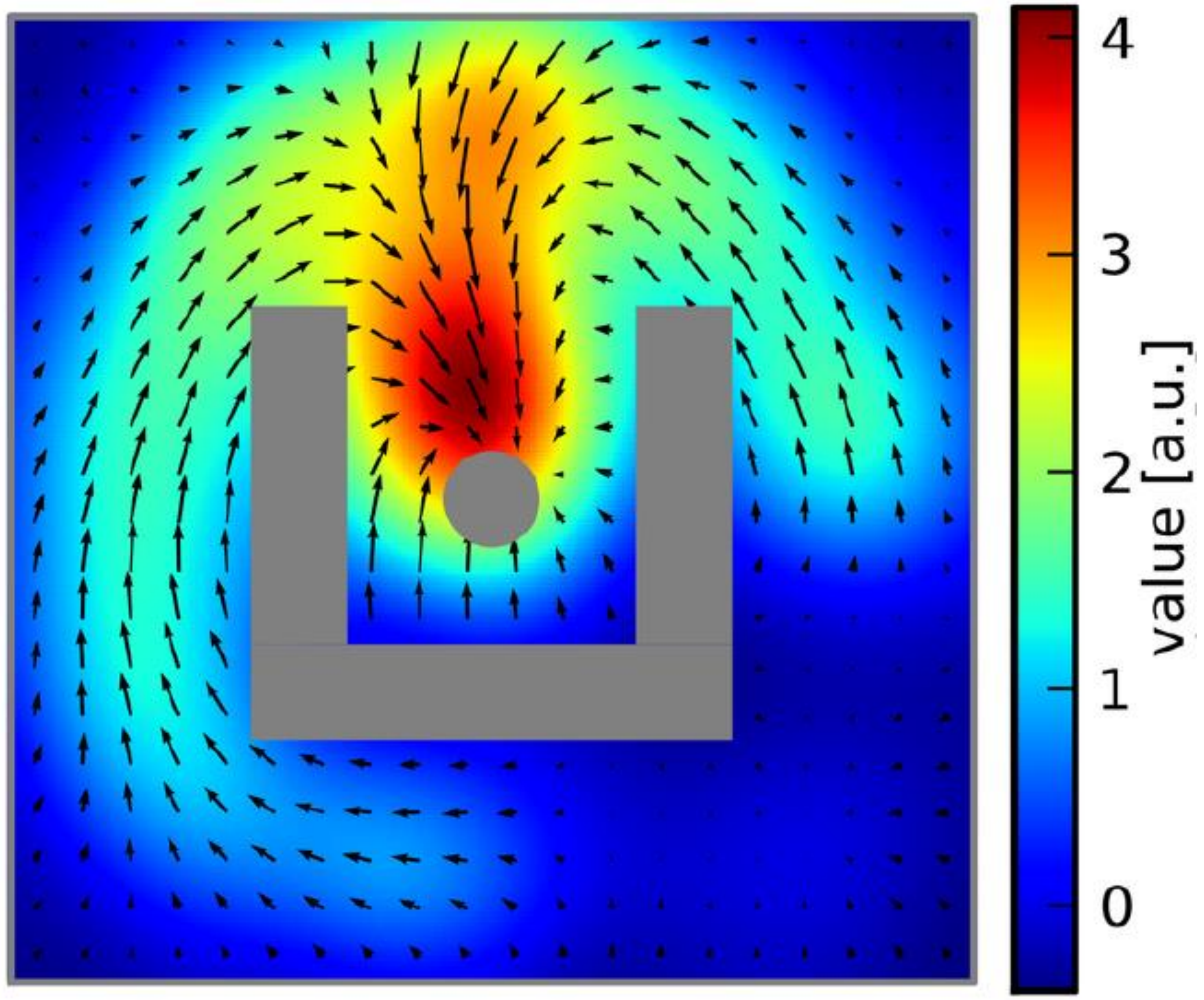
Imagine that the target location is shifted in the SAME environment.

7. Model-based versus Model-free Reinforcement Learning

What happens in RL when you shift the goal after learning?

→ The value function has to be re-learned from scratch.

agent learns ‘arrows’, but not the lay-out of the environment: Standard RL is ‘model-free’



Previous slide.
After a shift, the value function has to be relearned from scratch, because the RL algorithm does not build a model of the world. We just learn ‘arrows’: what is the next step (optimal next action), given the current state?

7. Model-based versus Model-free Reinforcement Learning

Definition:
Reinforcement learning is model-free, if the agent does not learn a model of the environment.

Note: of course, the learned actions are always implemented by some model, e.g., actor-critic. Nevertheless, the term model-free is standard in the field.

Previous slide.
All standard RL algorithms that we have seen so far are ‘model free’.

7. Model-based versus Model-free Reinforcement Learning

Definition:
Reinforcement learning is model-based, if the agent does also learn a model of the environment.

Examples: Model of the environment

- state s1 is a neighbor of state s17.
- if I take action a5 in state s7, I will go to s8.
- The distance from s5 to s15 is 10m.
- etc

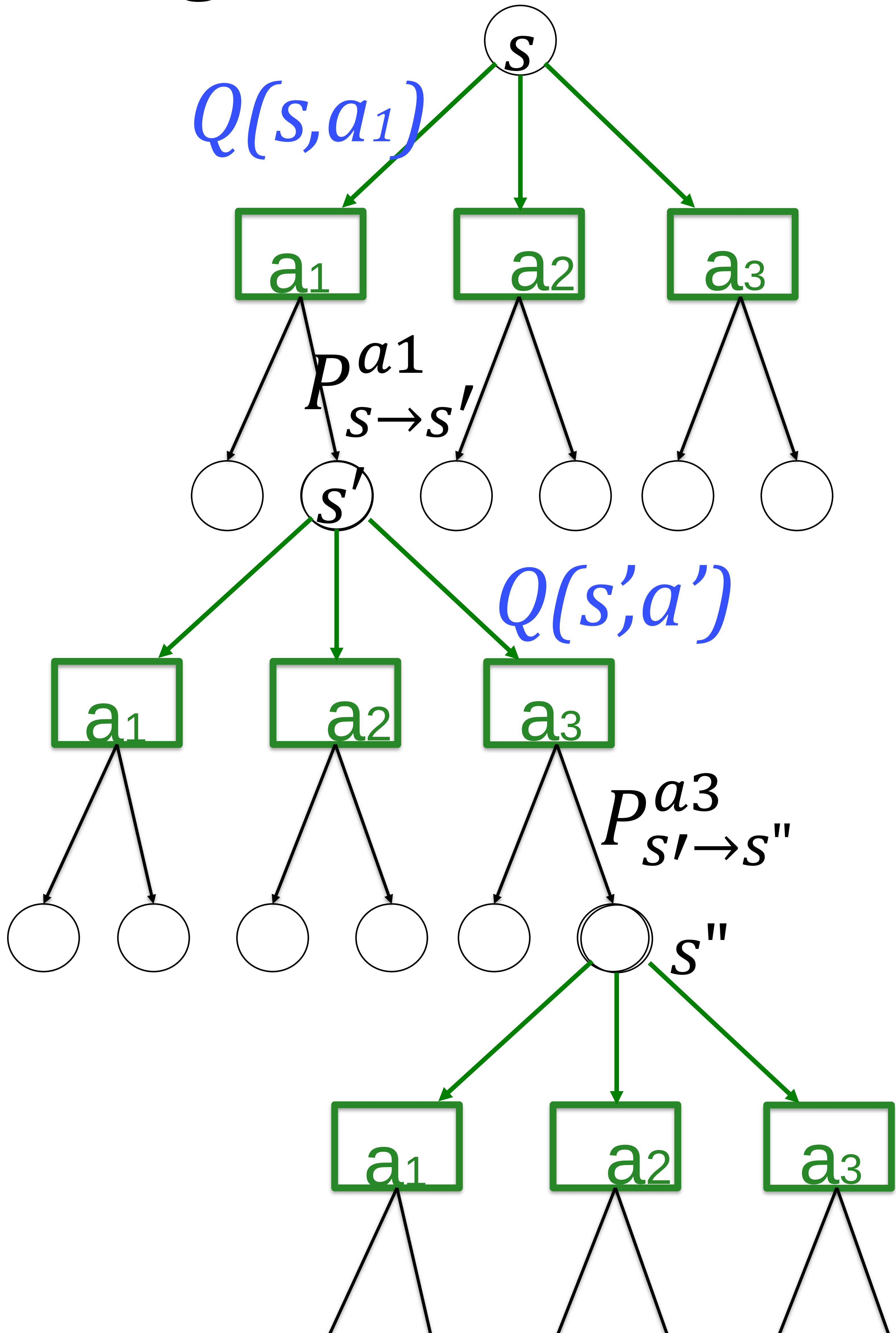
Previous slide.
Examples of knowledge of the environment, that would be typical for model based algorithm

7. Model-based versus Model-free Q-learning

Model-free:
the agent learns directly and only
the Q-values/V-values/policy

Model-based:
the agent learns the Q-values/policy
and also the transition probabilities

$$P^{a1}_{s \to s'}$$



Previous slide.
Let us go back to our ‘tree’. If the algorithm knows the transition probabilities, or builds up an estimation of the transition probabilities, then this means that it is a model-based algorithm.

If the algo does not contain elements that explicitly enable an estimation of transition probabilities, then the algorithm is model-free.

Note that knowledge of the physics of a robot (e.g., the laws of physics that govern the dynamics of movement) or knowledge of the rules of a game (e.g., allowed moves in chess) is an example of explicit knowledge of transition probabilities, and hence, if this knowledge is available to the algo, then the algo is called ‘model-based’.

7. Model-based versus Model-free Reinforcement Learning

Advantages of Model-based RL:

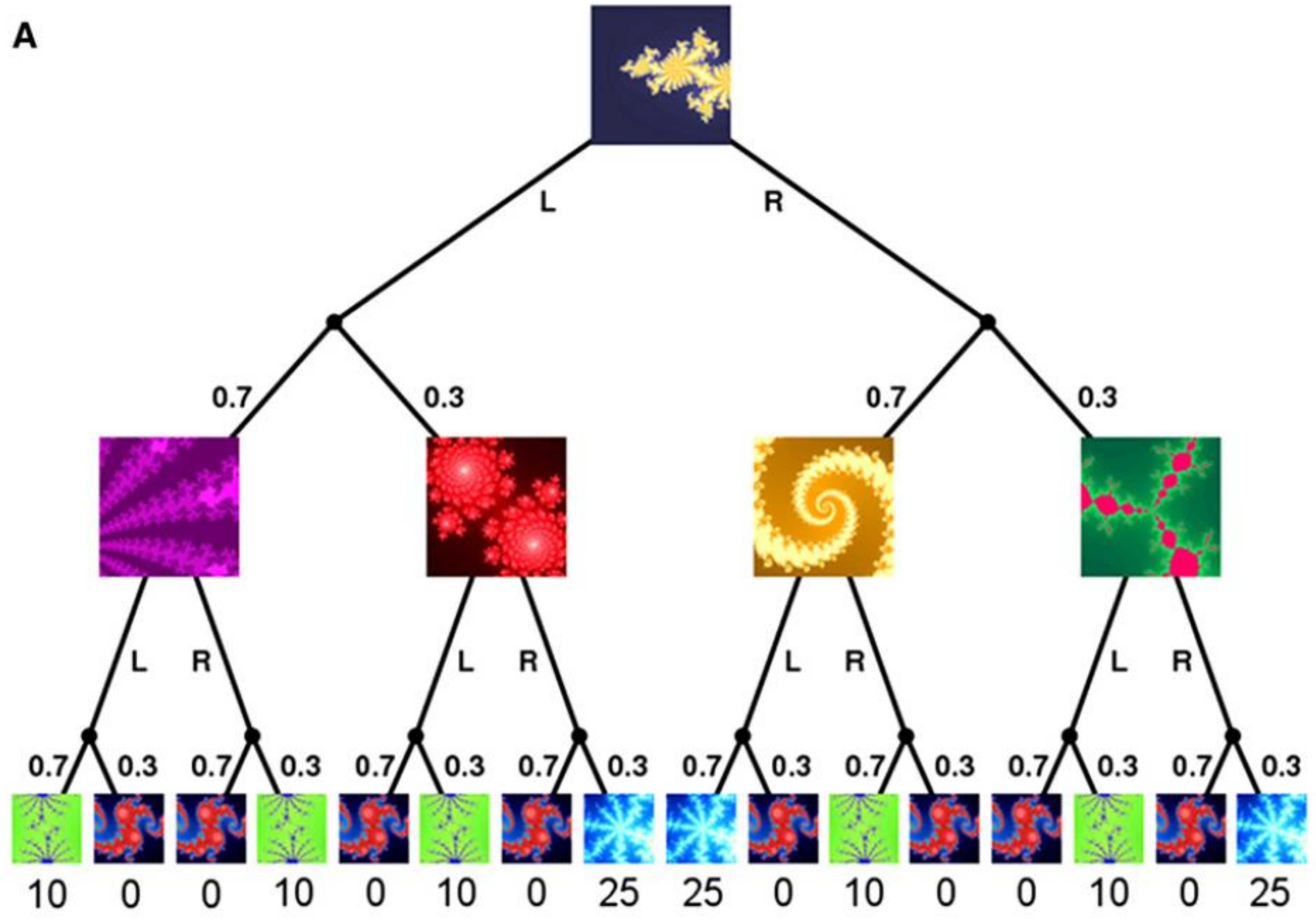
- the agent can readapt if the reward-scheme changes
- the agent can explore potential future paths in its ‘mind’
  → agent can plan an action path
- the agent can update Q-values in the background
  → dream about action sequences
  (run them in the model, not in reality)

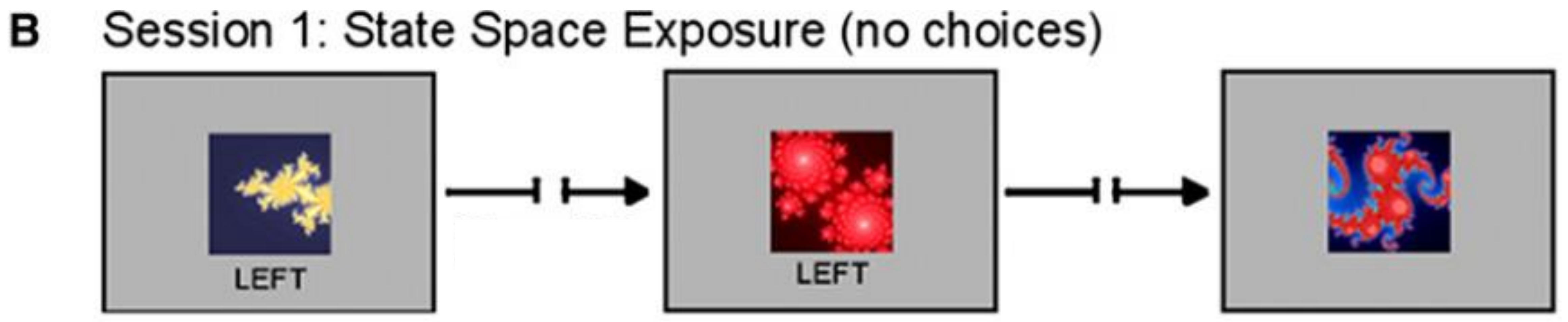
Note: Implementations of Chess and Go are ‘model-based’, because the agent knows the rules of the game and can therefore plan an action path. It does not even have to learn the ‘model’ (since the rules are given/known).

next slide.

Many modern applications of RL have a model-based component, because you need to play a smaller number of ‘real’ action sequences …
And computer power necessary for running things in the background is cheaper than acting ‘in real’.

7. Model-based learning





Model based:
You know what state to expect given current state and action choice. ‘state prediction’.
Experiment with human participants

Gläscher et al. 2010

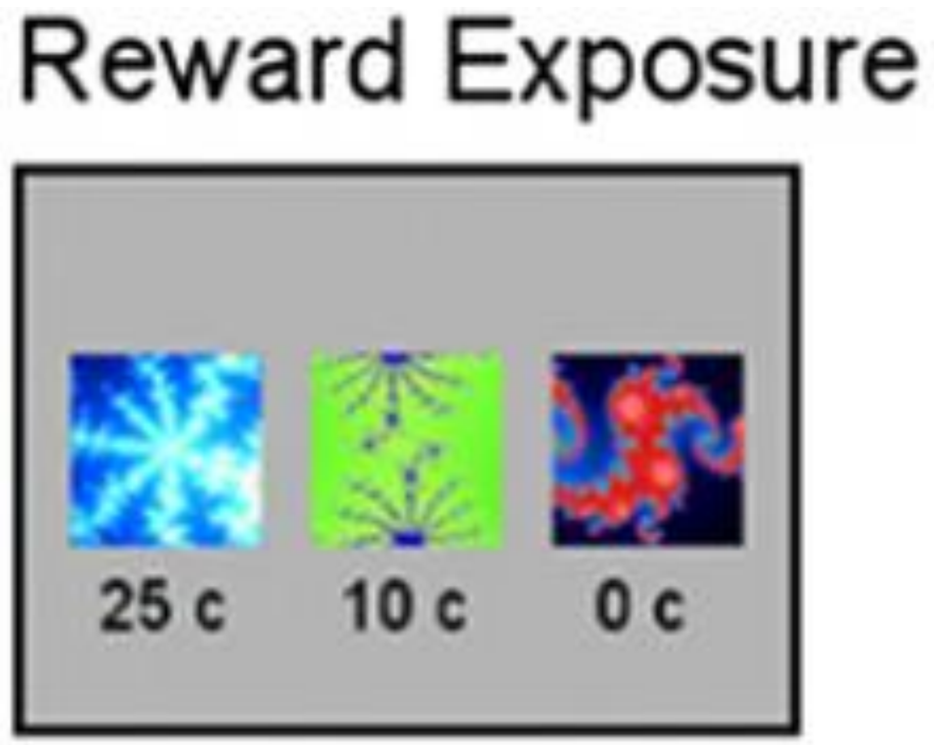
State and Reward Prediction Task (previous slide)

(A) A specific experimental task for human participants was a sequential two-choice Markov decision task in which all decision states are represented by fractal images. The task is to move through a binary decision tree. Each trial begins in the same state. Subjects can choose between a left (L) or right (R) button press. With a certain probability (0.7/0.3) they reach one of two subsequent states in which they can choose again between a left or right action. Finally, they reach one of three outcome states associated with different monetary rewards (0, 10cent, and 25cent).

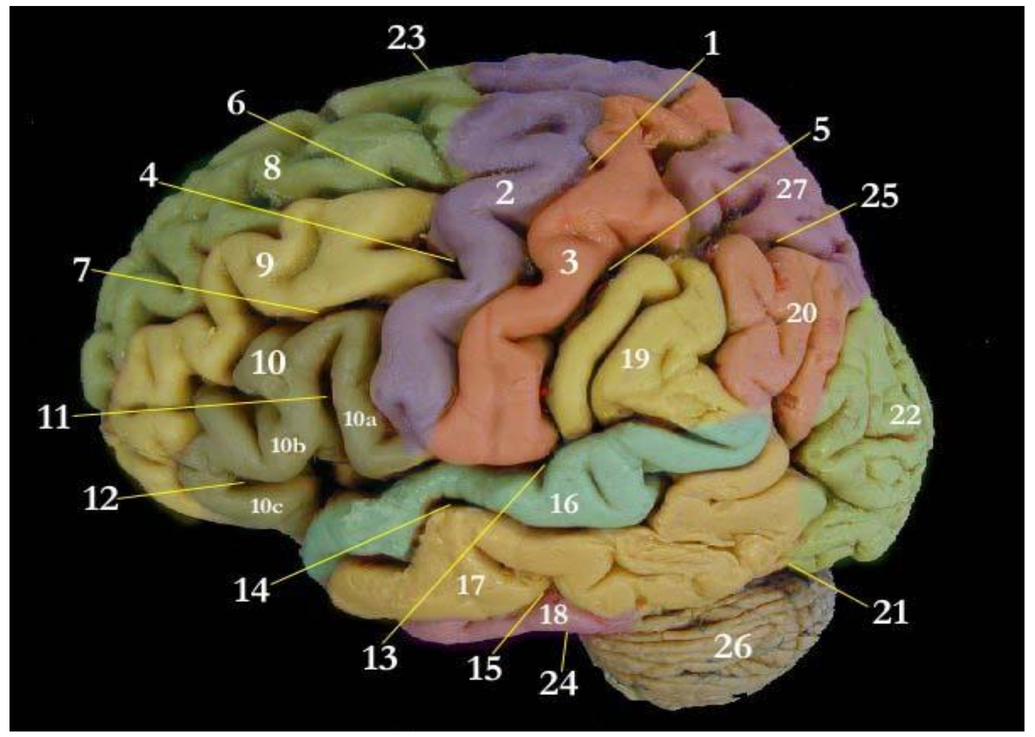
(B) Importantly, to encourage model-based behavior human participants first watched artificially generated sequences that enabled them to learn which actions caused which transition, so that they could estimate the transition probabilities.

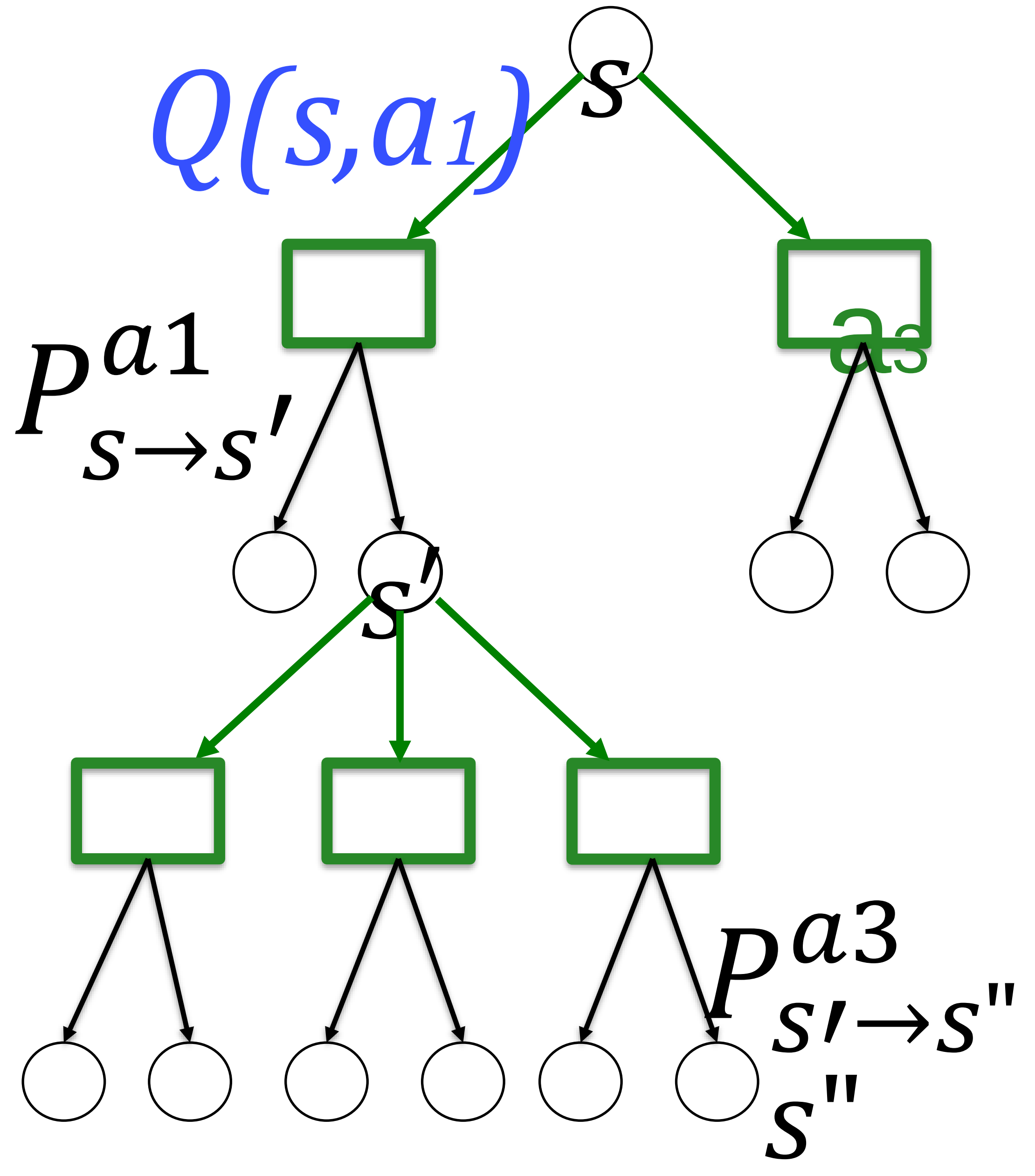
Gläscher et al. 2010

7. Model-based Reinforcement learning



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





Gläscher et al. 2010

Model based RL allows to think about consequences of actions:
Where will I get a reward?

You just need to play the probabilities forward over the model graph: you simulate an experience (in your mind!) before taking the real actions.

Gläscher et al. 2010

Summary: Model-based versus Model-free

- learns model of environment ‘transition matrix’	- does not
- knows ‘rules’ of game	- does not
- planning ahead is possible	- cannot plan ahead
- can update Bellman equation in ‘background’ without action	- cannot
- can simulate action sequences (without taking actions)	- cannot
- no need to use elgibility traces	- Eligibility traces and V-values keep memory of past
- is not online/causal	- completely online, causal, forward in time.

Model-based RL is more powerful than Model-free RL.

The overhead for memory/algorithmic complexity is often acceptable so that in computer-applications model-based is today preferred.

Updating knowledge in the background (without playing the actions) is also called ‘off-line’ update.

Learning outcomes and Conclusions:

- policy gradient algorithms
  → updates of parameter propto $[Return \quad]\frac{d}{d\theta_j}\ln[\pi]$
- why subtract the mean reward/mean Return?
  → reduces noise of the online stochastic gradient
- actor-critic framework (‘advantage actor critic’)
  → combines TD with policy gradient
- eligibility traces as ‘candidate parameter updates’
  → true online algorithm, no need to wait for end of episode
- Differences of model-based vs model-free RL
  → play out consequences in your mind by running the state transition model wait

The end of today's Lecture!
… and also the end of 8 weeks of
'Foundations of Learning in Neural Networks'

→ We are now ready to look at specific applications

Thanks!

The END