

Artificial Neural Networks

Deep Reinforcement Learning

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Part 1: Deep Q-learning

Objectives:

Deep Q-learning

Actor-Critic architectures

Eligibility traces for policy gradient

3-factor learning rules

Actor-Critic in the Brain

Model-based versus Model-free RL

Reading for this week:

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

Chapter 13; 15; 16.1, 16.4+16.5

Further reading:

- *Mnih et al. 2015, Nature, 518:529-533*
- *Chris Yoon, post: understanding actor-critic*
<https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f>
- *Gerstner et al. 2018, Frontiers in neural circuits 12: <https://doi.org/10.3389/fncir.2018.00053>*

Learning outcomes and Conclusions:

- **policy gradient algorithms**

 - updates of parameter propto $[R - b] \frac{d}{d\theta_j} \ln[\pi]$

- **why subtract the mean reward?**

 - reduces noise of the online stochastic gradient

- **actor-critic framework**

 - 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

Learning outcome for Relation to Biology:

- **three-factor learning rules can be implemented by the brain**
 - weight changes need presynaptic factor, postsynaptic factor and a neuromodulator (3rd factor)
 - actor-critic and other policy gradient methods give rise to very similar three-factor rules
- **eligibility traces as ‘candidate parameter updates’**
 - set by joint activation of pre- and postsynaptic factor
 - decays over time
 - transformed in weight update if dopamine signal comes
- **the dopamine signal has signature of the TD error**
 - responds to reward minus expected reward
 - responds to unexpected events that predict reward

Reading for this week (relation to biology):

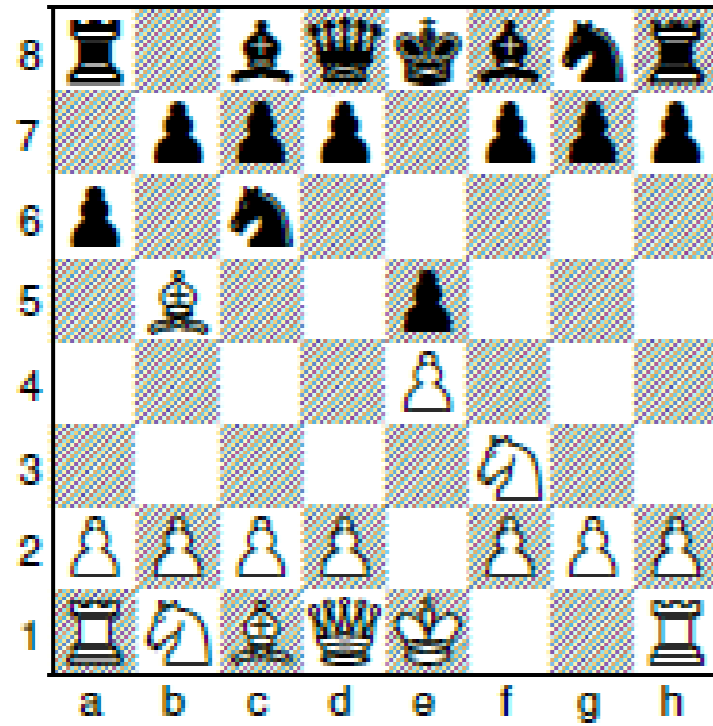
Background reading (relation to biology):

- (1) Schultz, Dayan, Montague (1997), A neural substrate of prediction and reward, *Science* 275:1593-1599
- (2) Schultz (2002), Getting formal with Dopamine and Reward, *NEURON* 36:241-263
- (3) Foster, Morris, Dayan, (2000), A model of hippocampally dependent navigation. *Hippocampus*, 10: 1-16
- (4) Crow, (1968), Cortical Synapses and Reinforcement: a hypothesis. *NATURE* 219:736-737
- (5) Fremaux, Sprekeler, Gerstner (2013) Reinforcement learning using a continuous-time actor-critic framework with spiking neurons *PLoS Computational Biol.* doi:10.1371/journal.pcbi.1003024
- (6) Fremaux, Gerstner (2016) Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules *Frontiers in neural circuits* 9, 85
- (7) J Gläscher, N Daw, P Dayan, JP O'Doherty (2010) States versus rewards: dissociable neural prediction error signals underlying model-based and model-free reinforcement learning, *Neuron* 66 (4), 585-595
- (8) Gerstner et al. (2018) *Frontiers in neural circuits* 12: <https://doi.org/10.3389/fncir.2018.00053>

(your comments)

Deep reinforcement learning

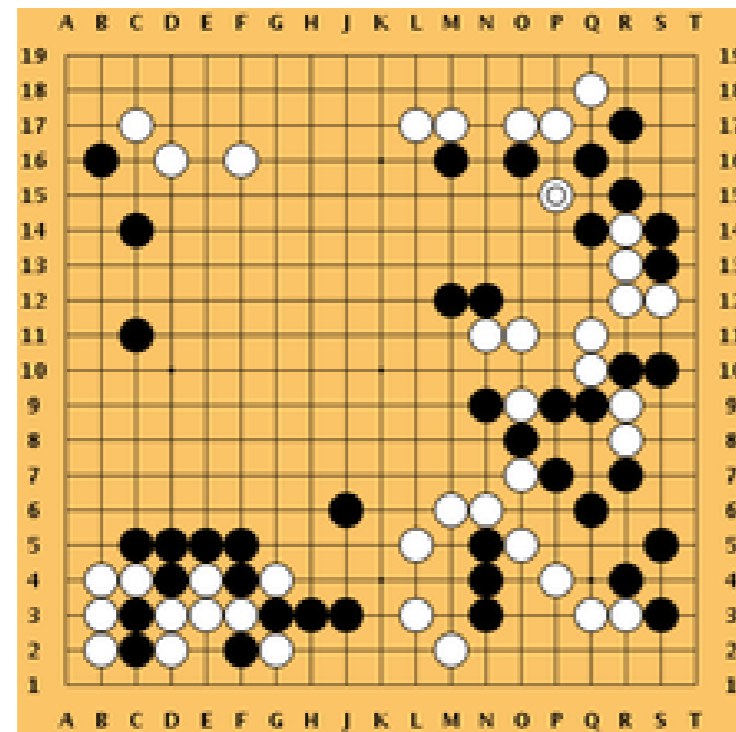
Chess



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

In Go, it beats Lee Sedol

Go



(previous slide)

In a previous week we have seen that we can model Q-values in continuous state space as a function of the state s , and parameterized with weights w .

But in fact, a model of Q-values also works when the input space is discrete, such as it is in chess. Suppose that each output corresponds to one action (e.g. one type of move in chess).

We can use a neural network where the output are the Q-values of the different actions while the input represents the current state s .

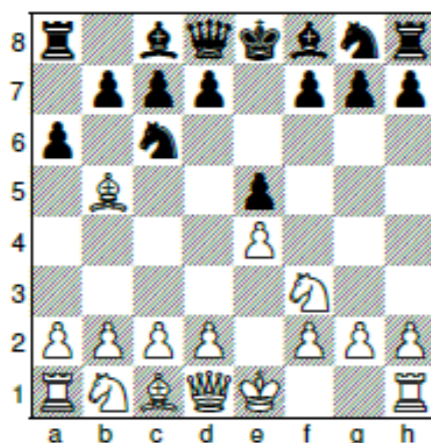
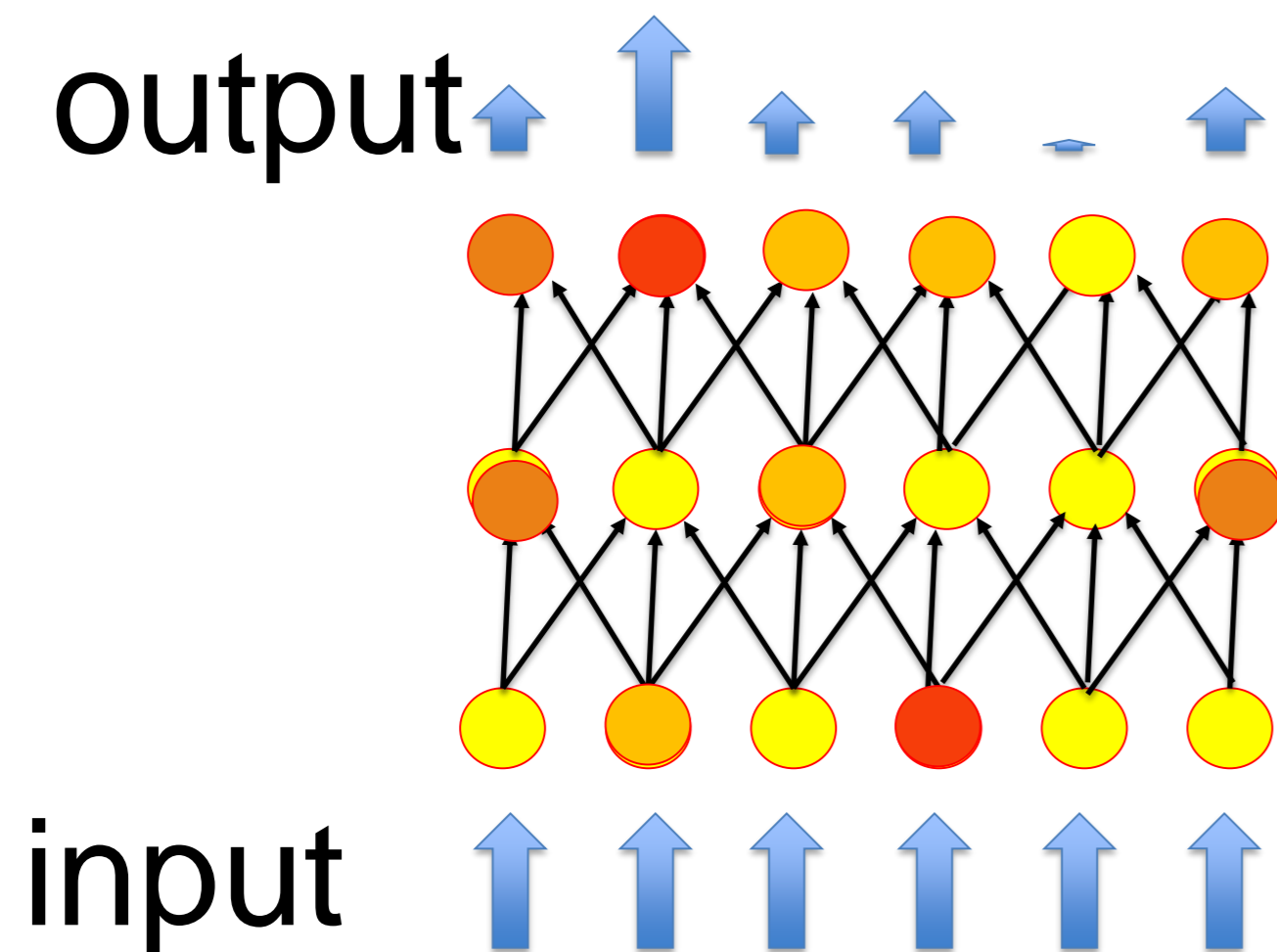
Thus, an output unit n represents $Q(a_n, s)$.

Backprop for Deep Q-learning

(Backprop = gradient descent rule in multilayer networks)

action and Q-values:

Move piece



Outputs are Q-values

→ actions are easy to choose

(e.g., softmax, epsilon-greedy)

Neural network parameterizes Q-values as a function of continuous state s .

One output for each action a .

Learn weights by playing against itself.

Error function for SARSA

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

(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 Q-values in the output.

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

The consistency condition of TD learning, can be formulated by an error function:

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

This error function will depend on the weights w (since $Q(s,a)$ depends on w).

We can change the weights by gradient descent on the error function. This leads to the Backpropagation algorithm of 'Deep learning'

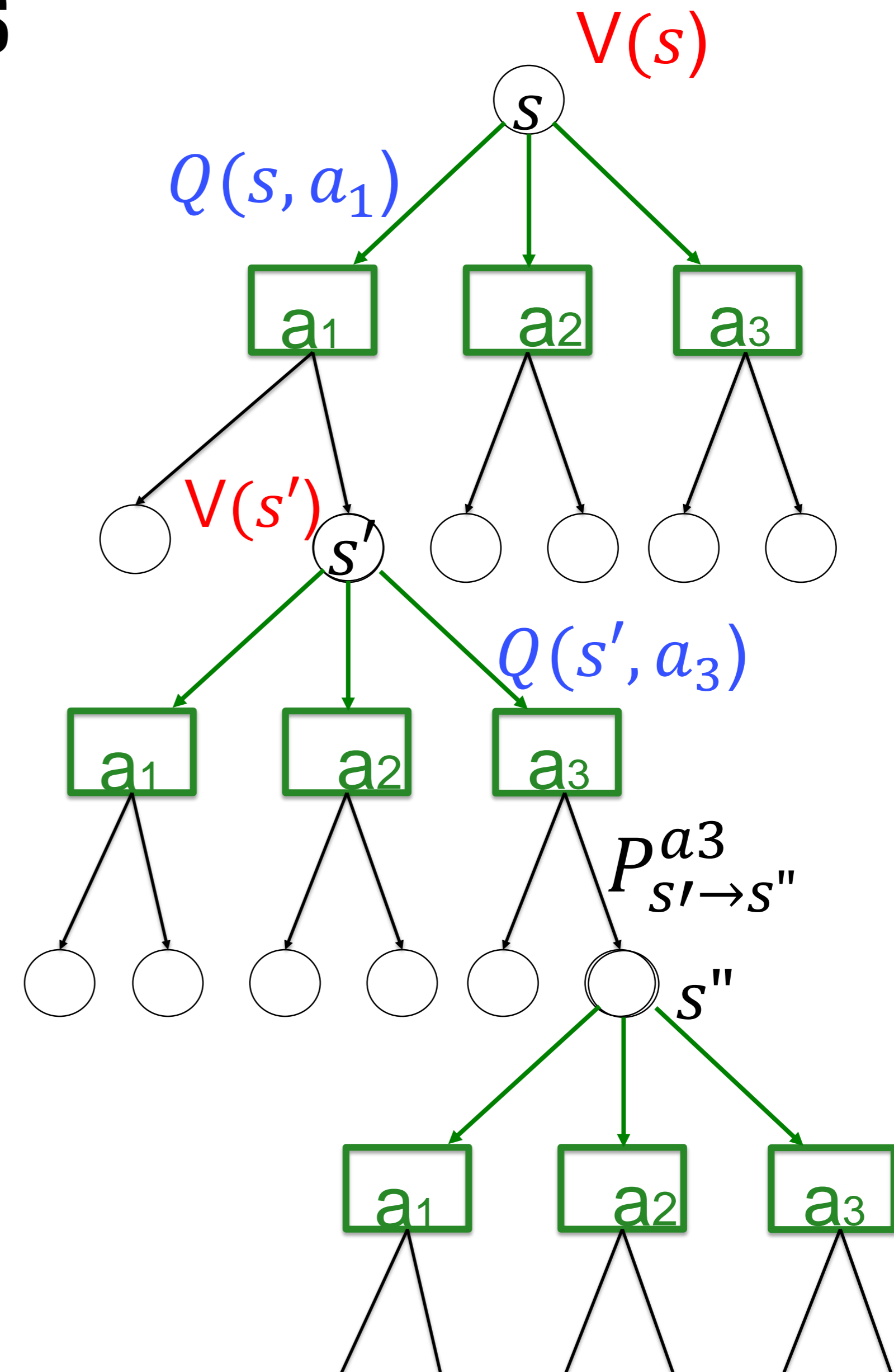
Review Q-values and V-Values

expected total discounted reward starting in s with action a_1

$$Q(s, a_1)$$

expected total discounted reward starting in s

$$V(s)$$



Review Q-values and V-values

Consistency condition of Bellman Eq.

$$Q(s, a) = \sum_{s'} P_{s \rightarrow s'}^a \left[R_{s \rightarrow s'}^a + \gamma \sum_{a'} \pi(s', a') Q(s', a') \right]$$

On-line consistency condition
(should hold on average)

$$Q(s, a) = r_t + \gamma Q(s', a')$$

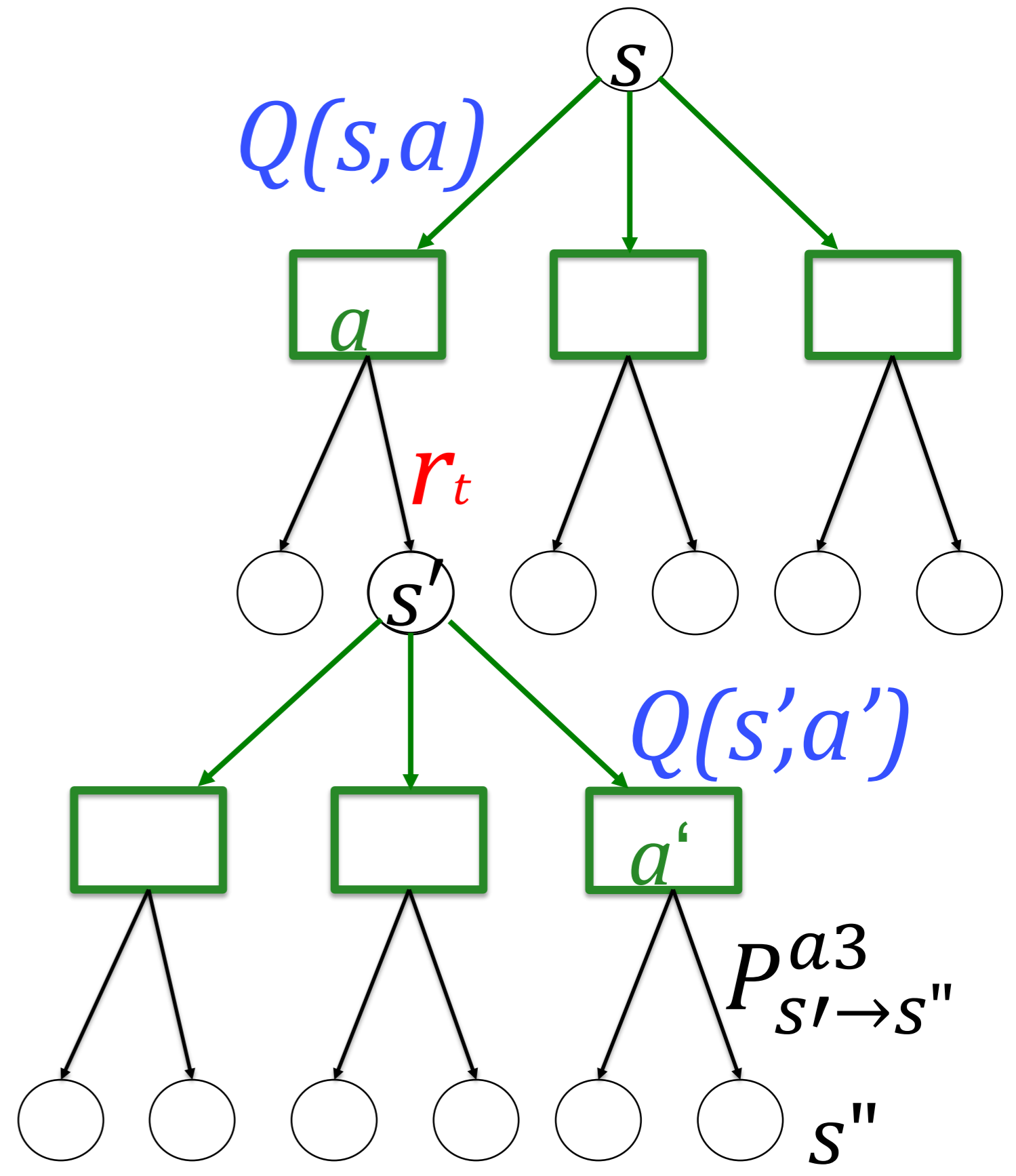
yields (online) Error function of SARSA

target

$$E(\mathbf{w}) = \frac{1}{2} [r_t + \gamma Q(s', a' | \mathbf{w}) - Q(s, a | \mathbf{w})]^2$$

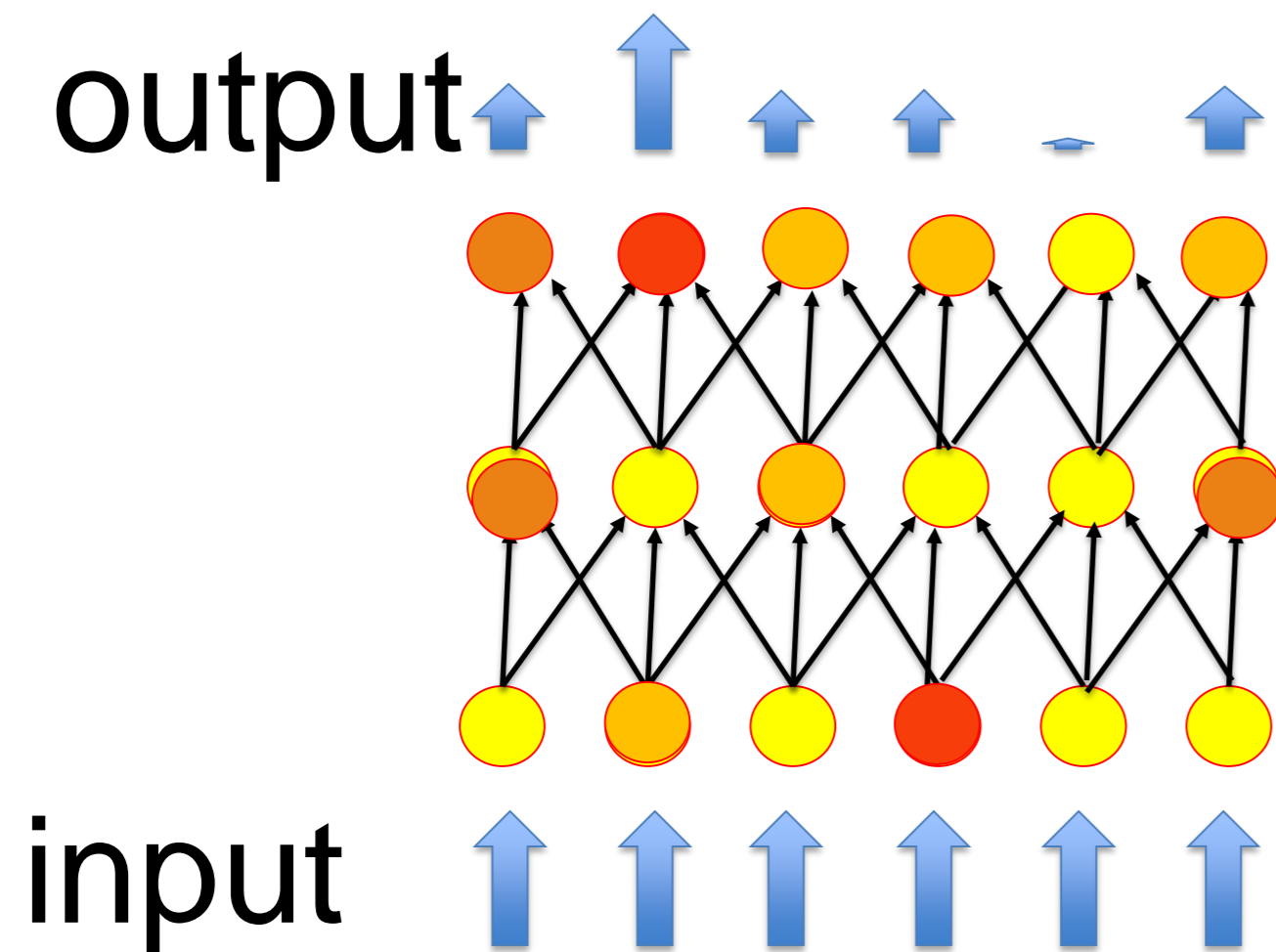
'semi-gradient' ignore

take gradient w.r.t \mathbf{w}



Deep Q-learning: semi-gradient on error function

output=Q-values



Input - states

Outputs are Q-values
→ actions are easy to choose
(e.g., softmax)

**Neural network parameterizes Q-values as a function of continuous state s .
One output for one action a .**

Error function for Q-learning

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

Example: Atari-video games (*Mnih et al. 2015*)

input = four frames from video; network = ConvNet; reward = score increase

(previous slide)

Deep Q-Learning uses the a deep network which transforms the state (encoded in the input units) into Q-values in the output.

Actions are chosen, for example, by softmax or epsilon-greedy on the Q-values in the output.

Weights are learned by taking the semi-gradient on the error function,

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

Recall that SARSA and Q-learning are TD algorithms. Recall also that the idea of the semi-gradient is to stabilize the target $r + \gamma \max_{a'} Q(s', a')$

When Mnih et al. applied DeepQ to video games they used a few additional tricks: to stabilize the target even further, they kept target Q-values and current Q-values in two separate networks; and they stored past transitions s, a, s', a' so that they could be replayed at any time (without actual action taking), so as to update Q-values.

Summary: Deep Q-learning

- Q-learning with continuous (or high-dim.) state space
- Q-values represented by output of deep ANN
- Action choice (=policy) depends on Q-values
- For training use semi-gradient with error function either SARSA (online, on-policy)

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

or Q-learning (off-policy)

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

- Further tricks (off-line updates, target stabilization)

(previous slide)

Deep Q-Learning is SARSA in a deep ANN.

Weights are learned by taking the semi-gradient on the error function,

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

Recall that SARSA and Q-learning are TD algorithms.

In the next part, we will go to Policy Gradient Methods; and in the third part we combine policy gradient methods with TD-learning!

Artificial Neural Networks

Deep Reinforcement Learning

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Part 2: From policy gradient to Deep Reinforcement Learning

1. Deep Q-Learning
2. From Policy gradient to Deep RL

(previous slide and next slide)

Policy gradient methods are an alternative to TD methods.

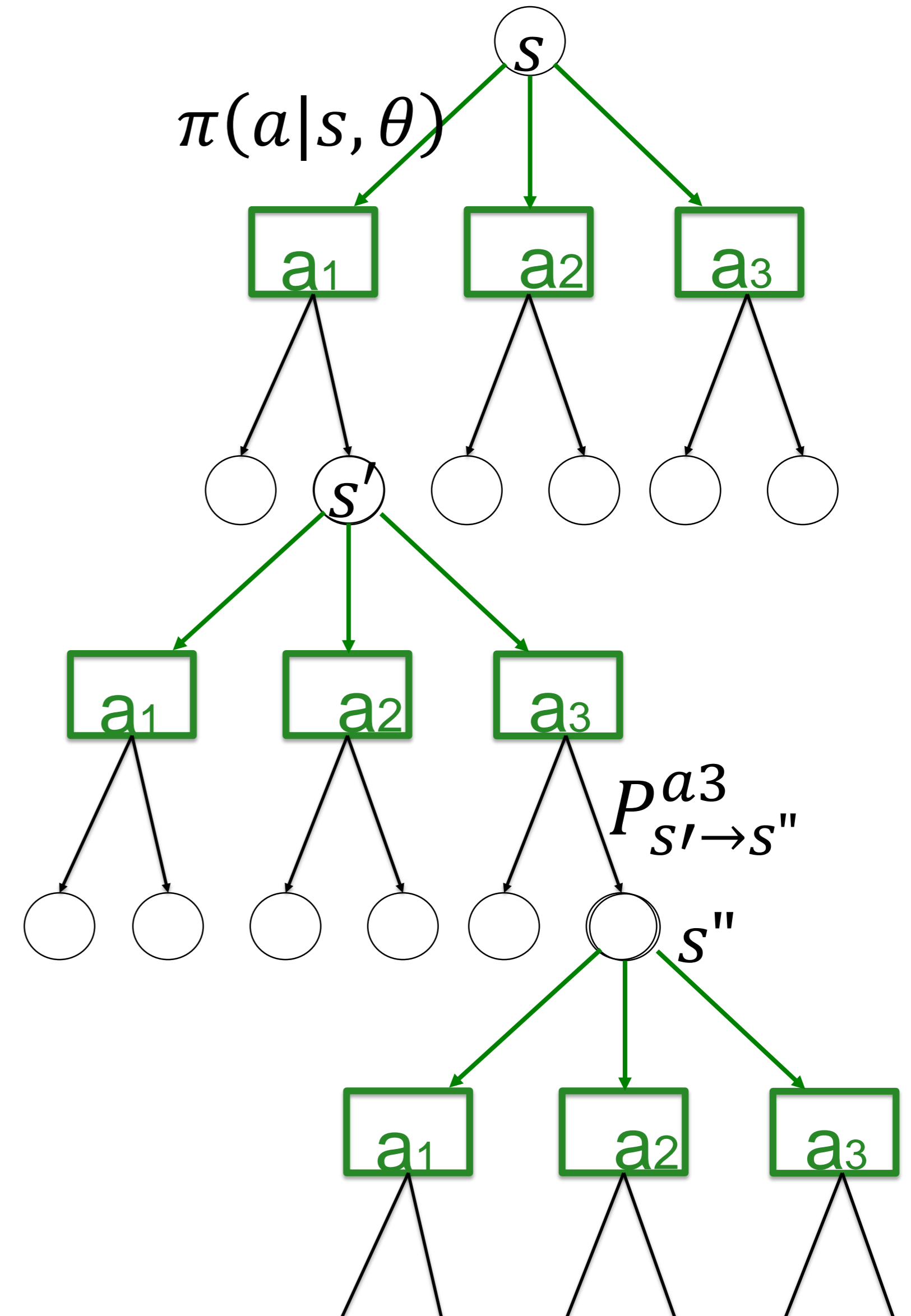
Review Policy Gradient

Aim:

update the parameters θ of the policy $\pi(a|s, \theta)$ so as to maximize return

Implementation:

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



(previous slide and next slide)

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 $R_{s_t \rightarrow s_{end}}^{a_t}$

(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 $\ln \pi$,

Review Policy Gradient: REINFORCE

Gradient calculation yields several terms of the form

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


$$\begin{aligned} \Delta\theta_j \propto & \left[R_{s_t \rightarrow s_{end}}^{a_t} \right] \frac{d}{d\theta_j} \ln[\pi(a_t | s_t, \theta)] \\ & + \gamma \left[R_{s_{t+1} \rightarrow s_{end}}^{a_{t+1}} \right] \frac{d}{d\theta_j} \ln[\pi(a_{t+1} | s_{t+1}, \theta)] \\ & + \dots \end{aligned}$$

‘Monte-Carlo update step’: single episode, from start to end

(previous slide and next slide)

The policy gradient algorithm is also called REINFORCE.

As discussed in a previous lecture (RL3) 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 (or the mean reward in a one step-horizon algorithm).

This is what we consider here in this section

(ii) Subtract mean expected reward PER TIME STEP (related to the delta-error) in a multi-step horizon algorithm.

This is what we will consider in section 3 under the term Actor-Critic.

Subtract a reward baseline (bias b)

we derived this online gradient rule for multi-step horizon

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

But then this rule is also an online gradient rule

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

with the same 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 of these terms is shown.

Similar to the case of the one-step horizon, we can subtract a bias b from the return $R_{s_t \rightarrow s_{end}}^{a_t}$ 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 (which takes place at the end of an episode) has terms

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

Subtract a reward baseline (bias b)

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

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

- The bias b can depend on state s
- Good choice is $b = \text{'mean of } [R_{s_t \rightarrow s_{end}}^{a_t}] \text{'}$
 - 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) of the return $R_{s_t \rightarrow s_{end}}^{a_t}$. 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 comes in through the backdoor: we can learn the V-value, and then use it as a bias in policy gradient.

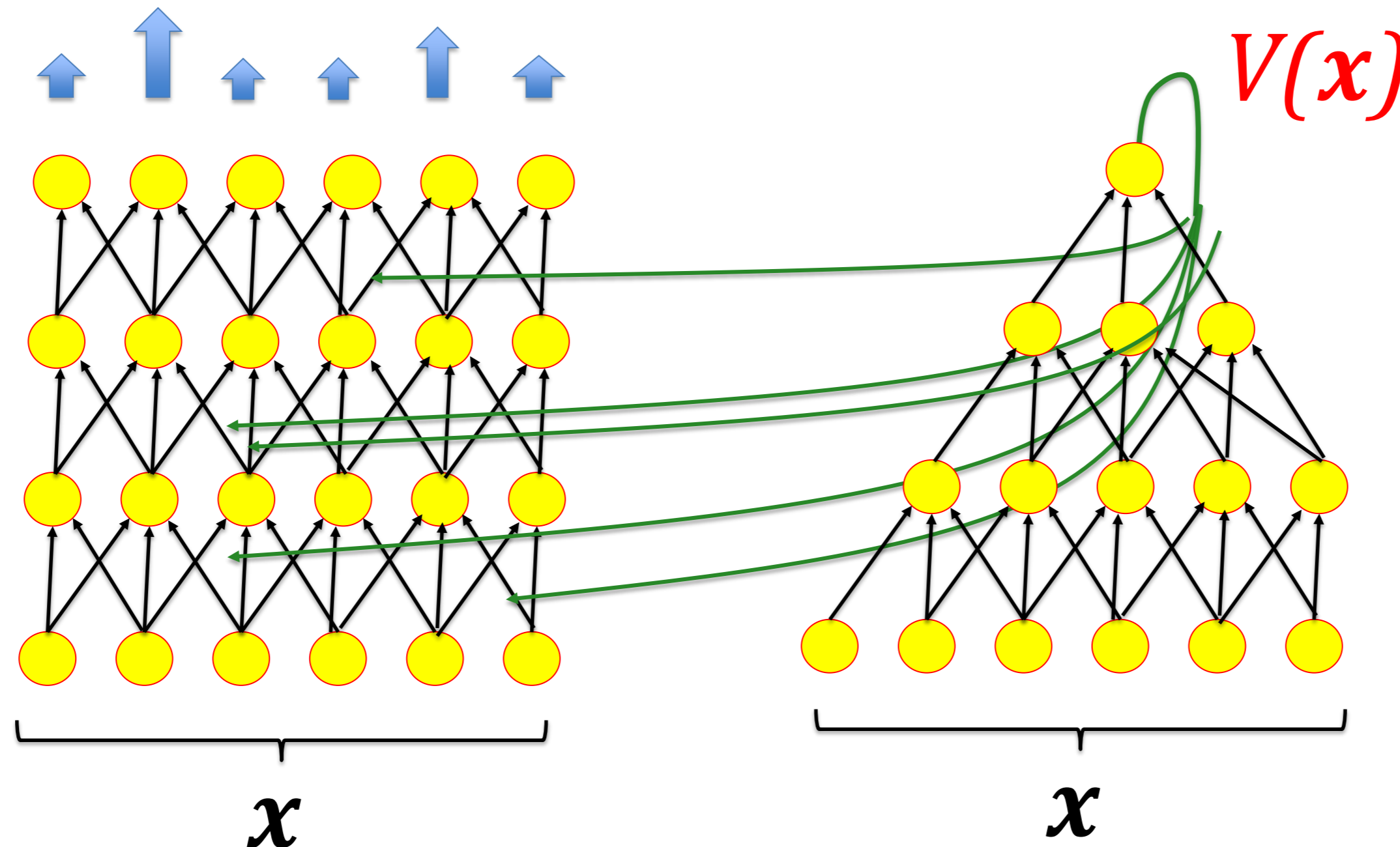
Learning two Neural Networks: actor and value

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)

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 $b(x = s_{t+n}) = V(x)$ 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 of the return $R_{s_t \rightarrow s_{end}}^{a_t}$: 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 $R_{s_{t+n} \rightarrow s_{end}}^{a_{t+n}}$ which is evaluated for a states s_{t+n} that have been visited in the episode.

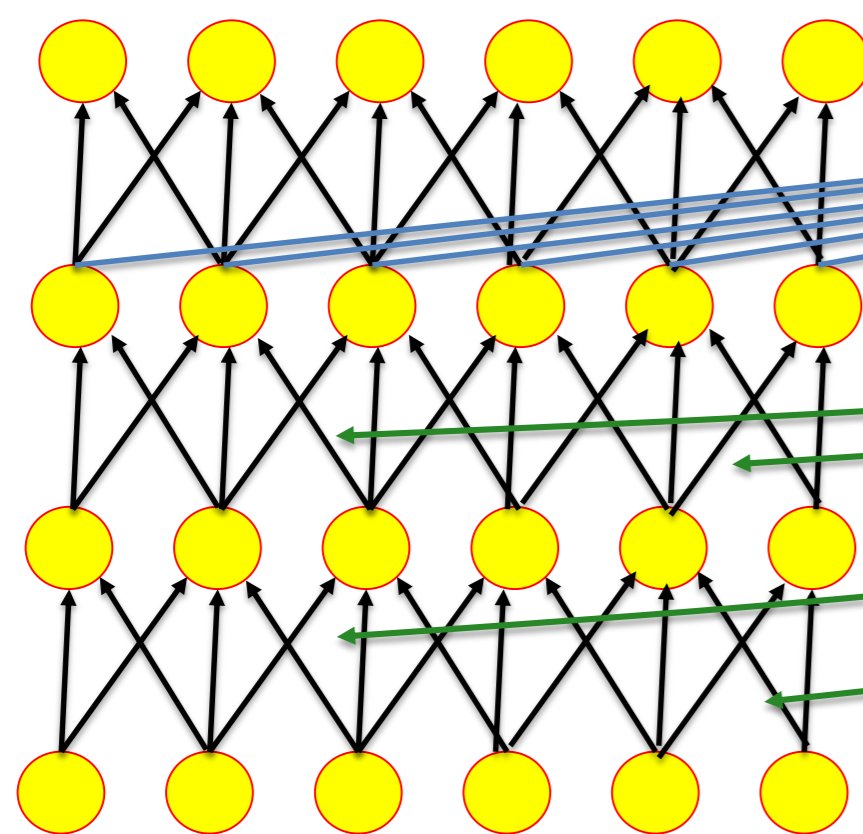
What matters in the learning rule is the difference $[R_{s_{t+n} \rightarrow s_{end}}^{a_{t+n}} - V(s_{t+n})]$

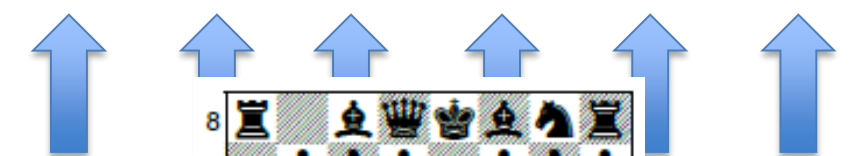
Deep reinforcement learning: alpha-zero

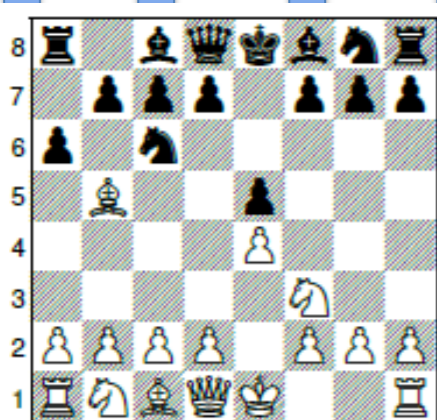
Network for choosing action:
Optimized by policy gradient

action: *Advance king*

output 



input 



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:

- $\eta [R_{s_{t+n} \rightarrow s_{end}}^{a_{t+n}} - V(s_{t+n})]$

(previous slide)

The value unit 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 $R_{s_{t+n} \rightarrow s_{end}}^{a_{t+n}}$

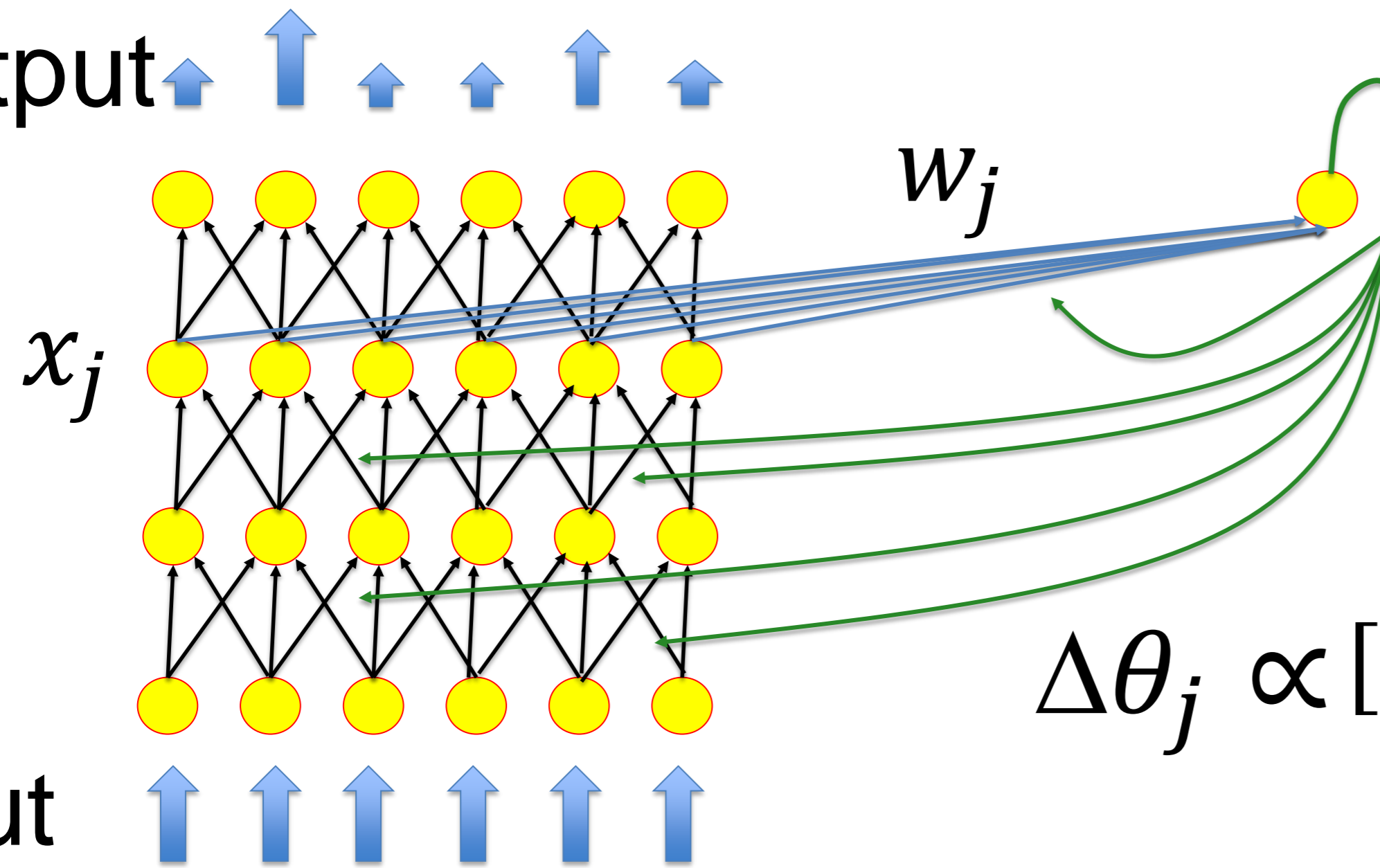
In the update rule we use $[R_{s_{t+n} \rightarrow s_{end}}^{a_{t+n}} - V(s_{t+n})]$

Summary: Deep REINFORCE with baseline subtraction

Network 1 for choosing action
Optimized by policy gradient

Network 2 for value of state:
optimized by Monte-Carlo
estimation of return

action
output



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

Update at end of episode:

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

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

(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 $R_{s_{t+n} \rightarrow s_{end}}^{a_{t+n}}$

What matters is the difference $[R_{s_t \rightarrow s_{end}}^{a_t} - V(s_t)]$

Updates are:
$$\Delta \theta_j \propto [R_{s_t \rightarrow s_{end}}^{a_t} - V(s_t)] \frac{d}{d\theta_j} \ln[\pi(a_t | s_t, \theta)] + \dots$$

$$\Delta w_j \propto [R_{s_t \rightarrow s_{end}}^{a_t} - V(s_t)] x_j$$

Artificial Neural Networks

Deep Reinforcement Learning

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Part 3: Actor-Critic network

1. Deep Q-Learning
2. From Policy gradient to Deep REINFORCE
3. Actor-Critic network (Advantage Actor Critic)

(previous slide)

We continue with the idea of two different type 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 earlier week) rather than the simple Monte-Carlo estimation of the (discounted) accumulated rewards in a single episode.

The networks structure remains the same as before:

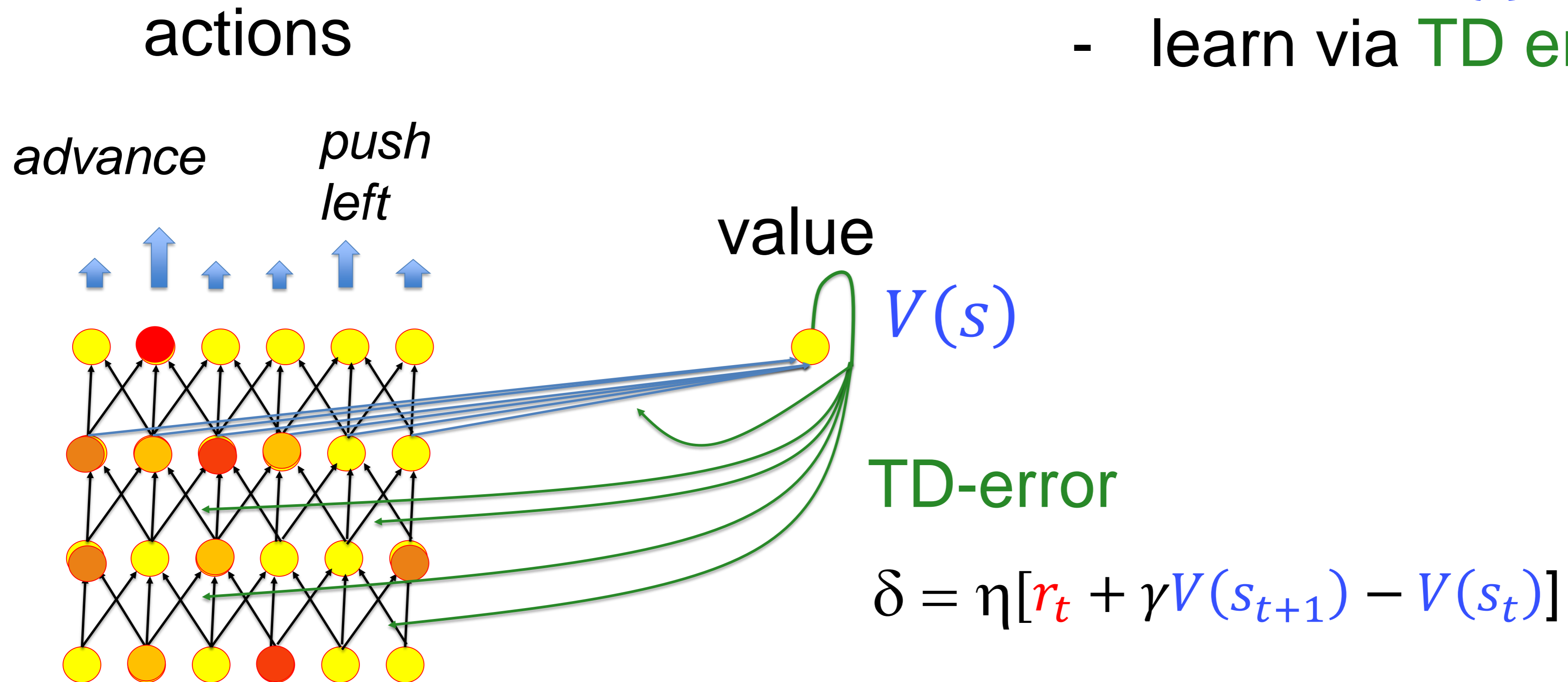
An actor (action network) and a critic (value function).

Sutton and Barto reserve the term ‘actor-critic’ to the network where V -values are learned with a TD algorithm (which is the one we discuss now). This is the ‘actor-critic network’ in the narrow sense.

However, other people would also call the network that we saw in the previous section as an actor-critic network (in the broad sense) and the one that we study now is then called ‘advantage actor critic’ or ‘TD-actor-critic’. The terminology is floating between different groups of researchers.

Actor-Critic = 'REINFORCE' with TD bootstrapping

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



Also called: - Advantage Actor-Critic

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

[Note the difference to the algorithm in section 6:

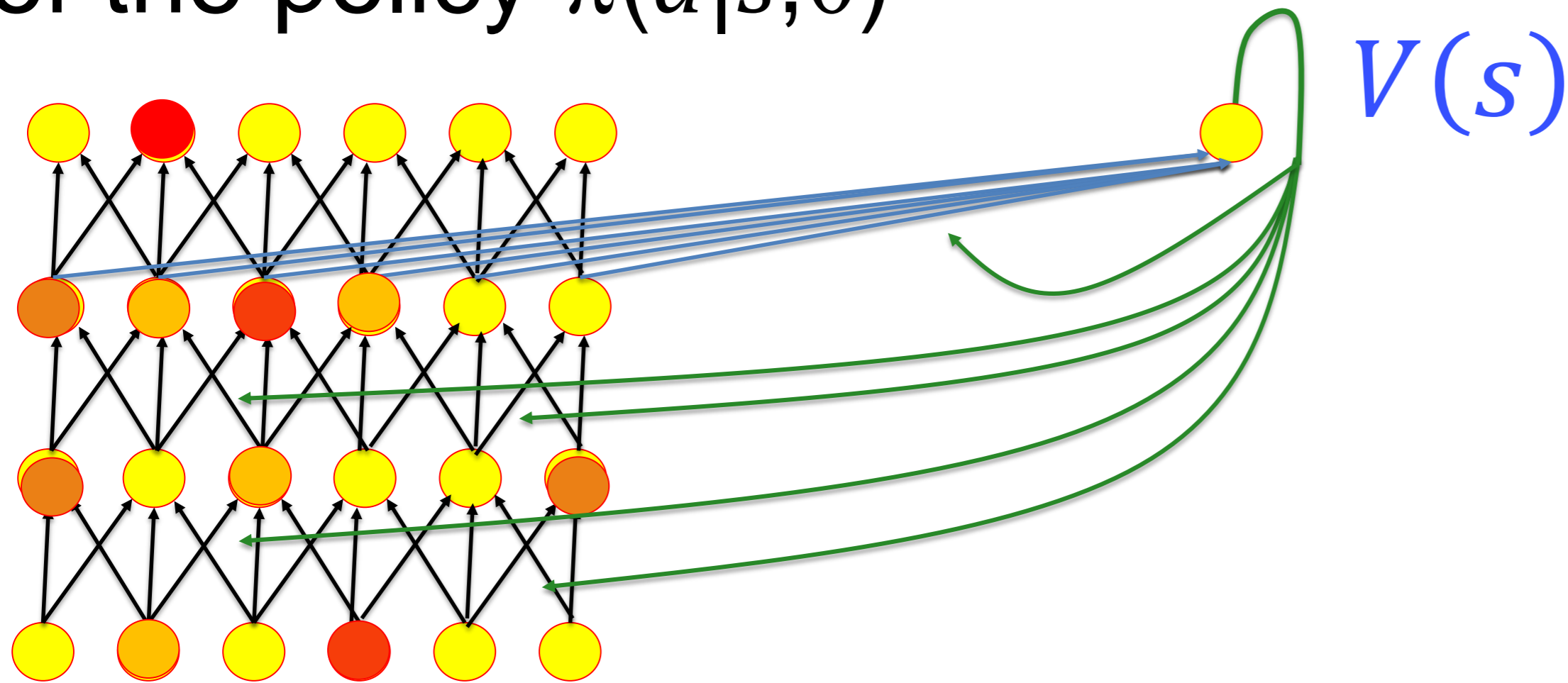
There the total accumulated discounted reward $R_{s_t \rightarrow s_{end}}^{a_t}$

was compared with $V(s_t)$]

Advantage ACTOR-CRITIC (versus REINFORCE with baseline)

1. Aim of actor in Actor-Critic:

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

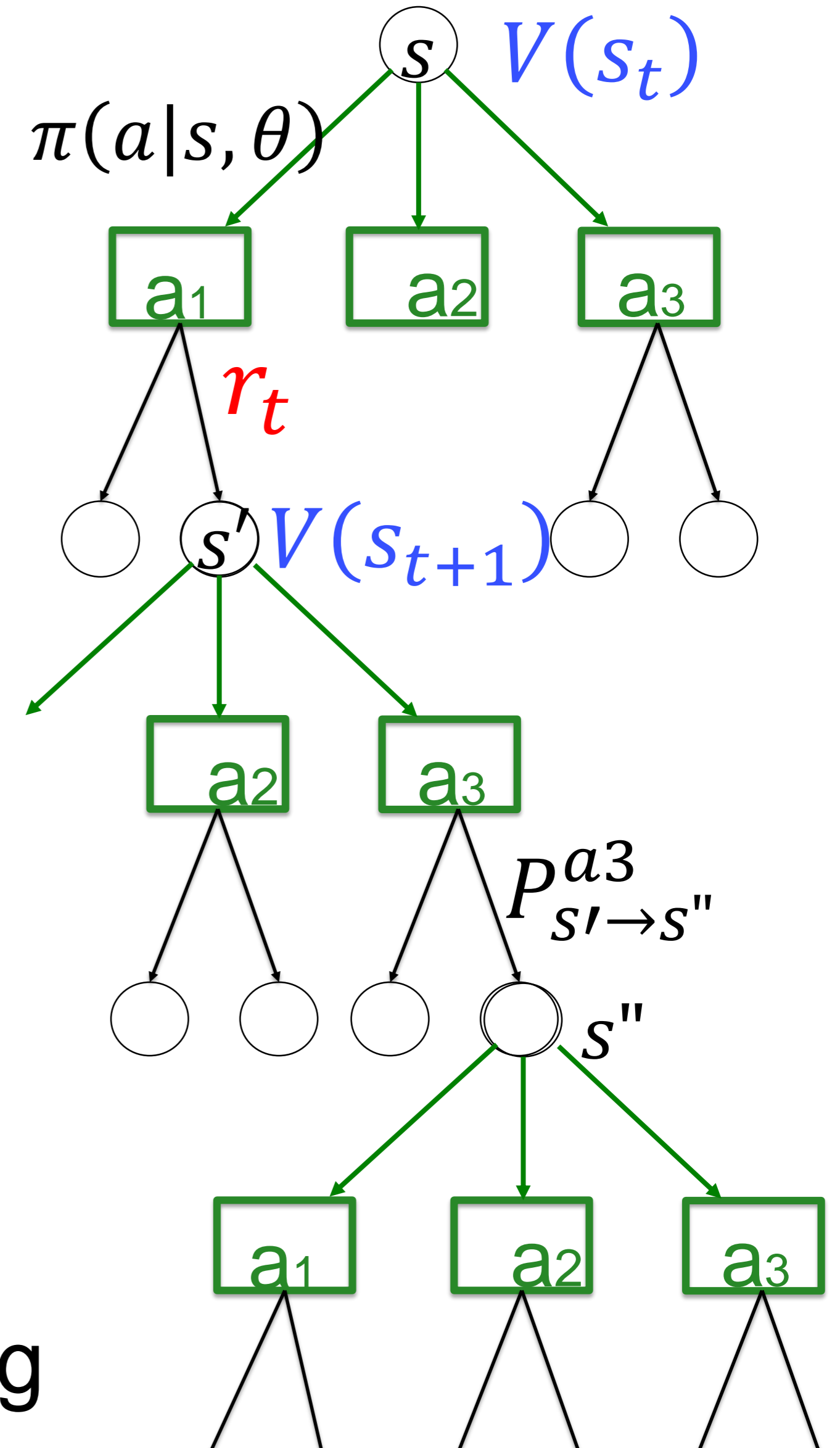


2. update at each step proportional to

TD-error

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

3. Aim of critic: estimate V using TD learning



(previous slide)

For a comparison of 'actor-critic' with 'REINFORCE with baseline', let us list:

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

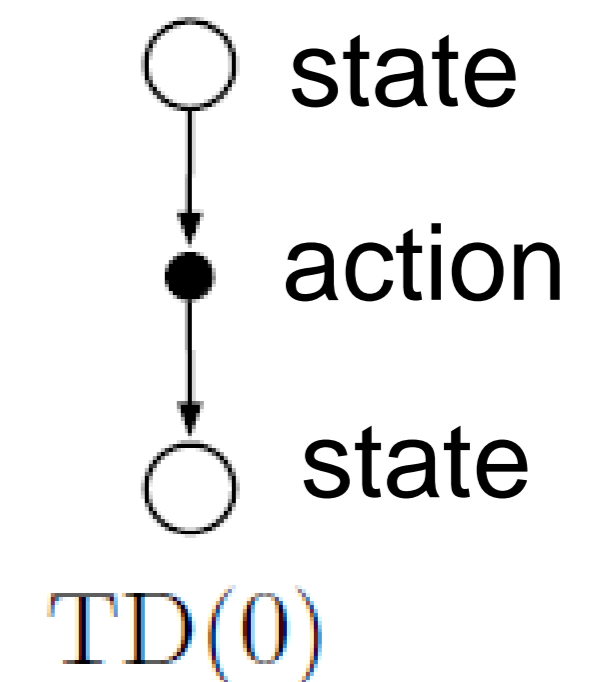
In the actor-critic algorithm the critic learns the V-value via bootstrap TD-learning (see earlier lecture). For the direct weights to the value units

$$\Delta w_j \propto [r_t + \gamma V(s_{t+1}) - V(s_t)] x_j$$

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

$$\Delta \theta_j \propto [r_t + \gamma V(s_{t+1}) - V(s_t)] \frac{d}{d\theta_j} \ln[\pi(a_t | s_t, \theta)] + \dots$$

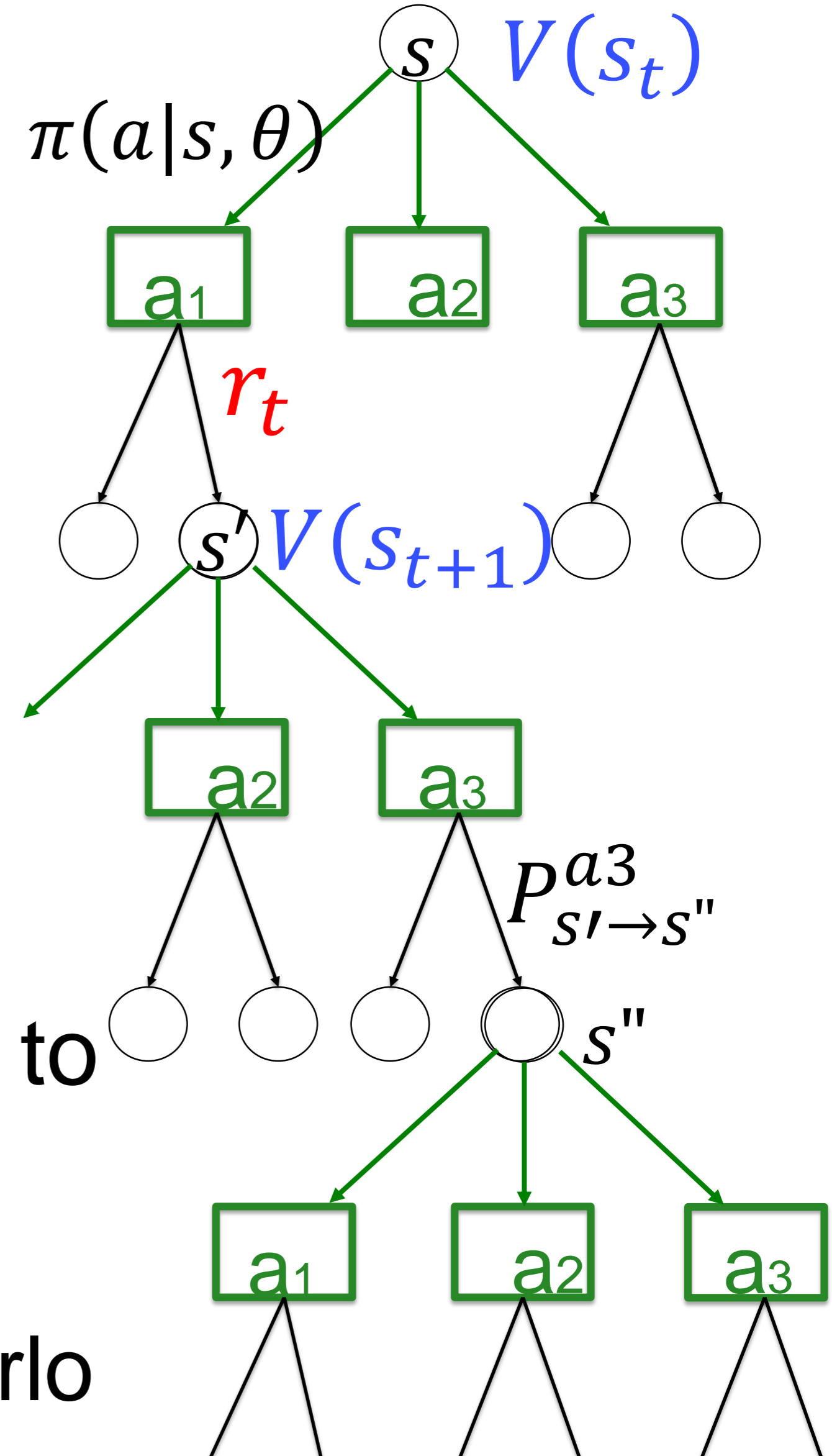
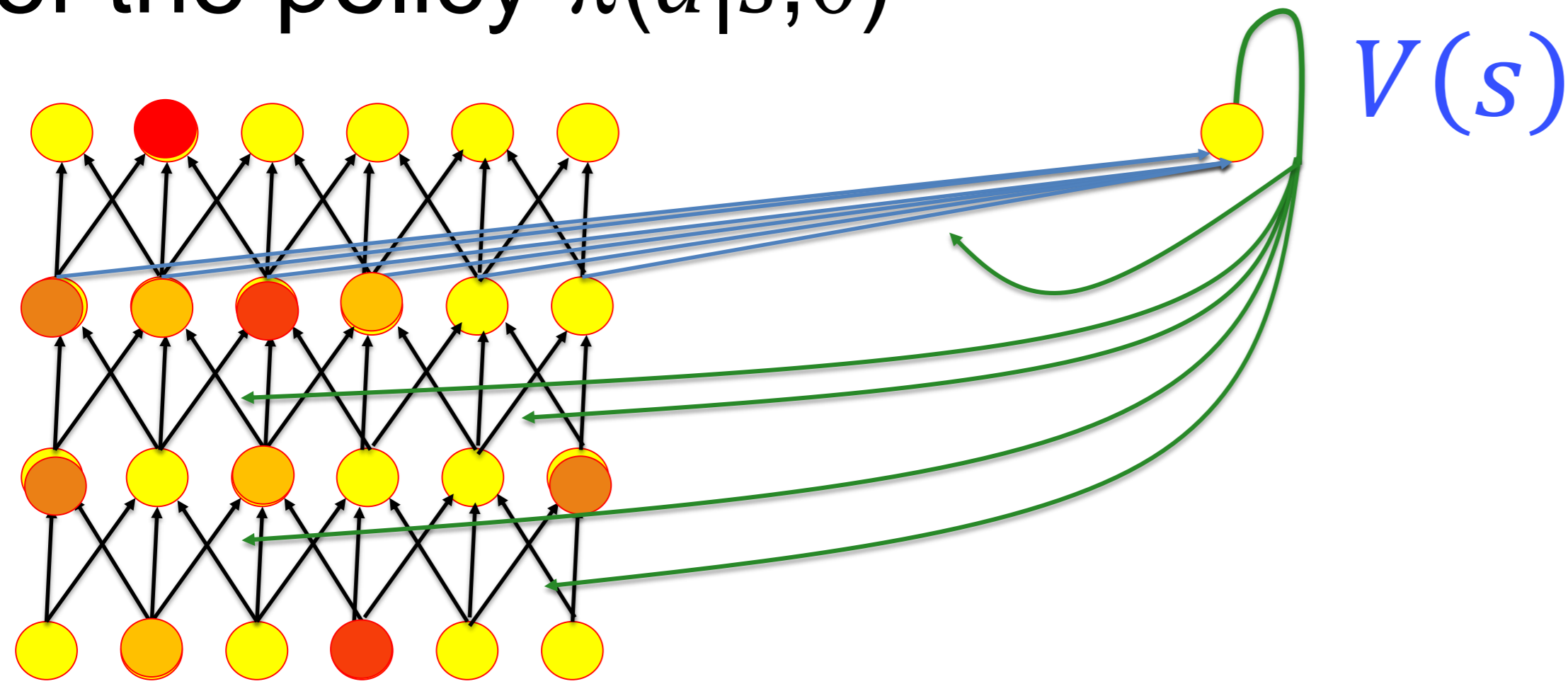
The backup diagram of the actor-critic is short. Actor-critic is meant here in the narrow sense of advantage actor critic (A2C).



REINFORCE with baseline (vs advantage actor-critic)

1. Aim of actor in REINFORCE

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



2. update at end of episode proportional to RETURN-error: $[R_{s_t \rightarrow s_{end}}^{a_t} - V(s_t)]$

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

(previous slide)

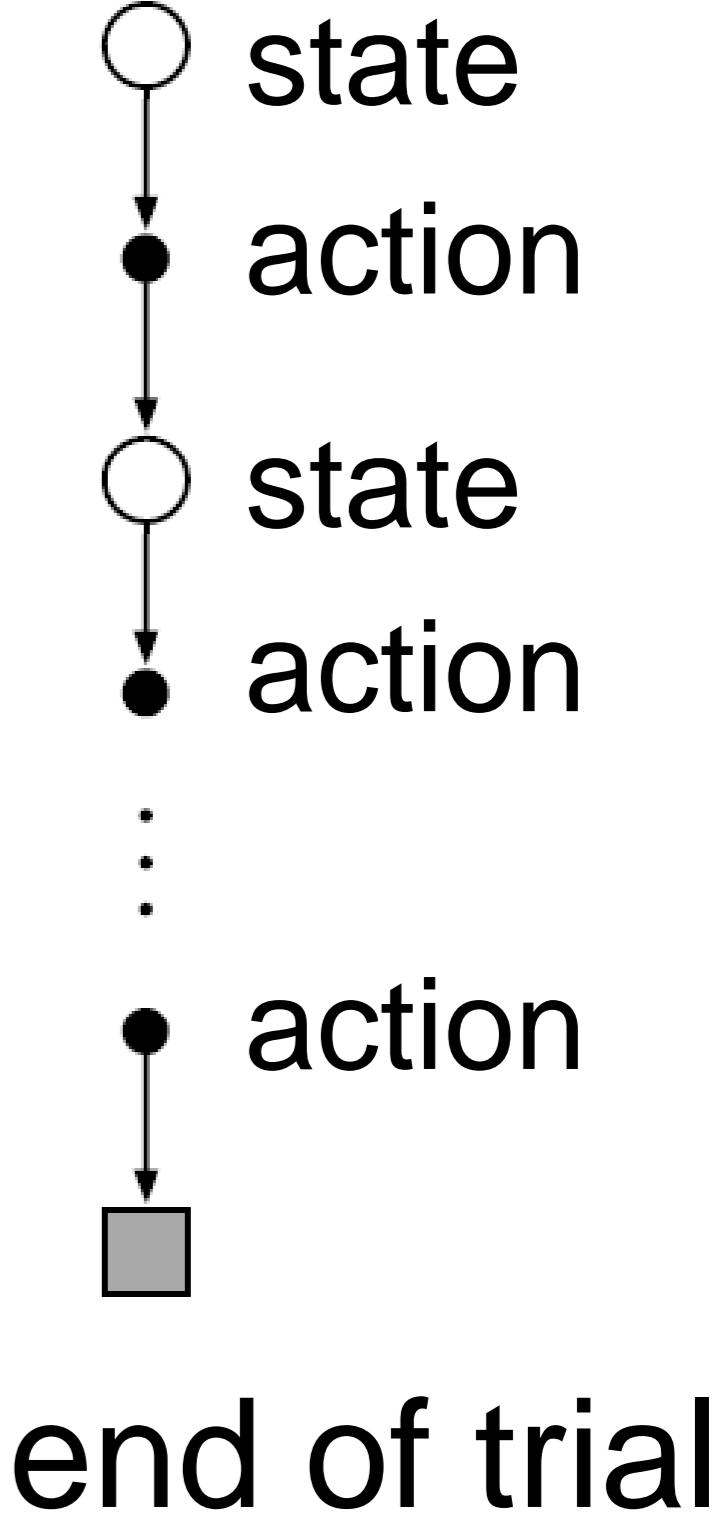
We continue the comparison with REINFORCE with baseline.

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 REINFORCE 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 she right?

Even some policy gradient algorithms use V-values

V-values for policy gradient can be calculated in a separate deep network

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

While advantage actor-critic uses ideas from TD learning, REINFORCE with baseline does not.

Your comments.

Artificial Neural Networks

Deep Reinforcement Learning

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Part 4: Eligibility traces for policy gradient

1. Deep Q-learning
2. From Policy gradient to Deep RL
3. Actor-Critic
4. **Eligibility traces for policy gradient and actor-critic**

(previous slide)

Some weeks ago we discussed eligibility traces in Reinforcement Learning.

It turns out that policy gradient algorithms have an intimate link to 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

- Actor learns by policy gradient
- Critic learns by TD-learning
- For each parameter, one *eligibility trace*
- Update *eligibility traces* while moving
- Update weights of actor and critic
 - proportional to **TD-delta** times *eligibility trace*

Combines several advantages:

- online, on-policy: like SARSA, or advantage actor-critic
- policy gradient for actor:
 - function approximator/continuous states
- rapid learning (like n-step or other eligibility trace algos)

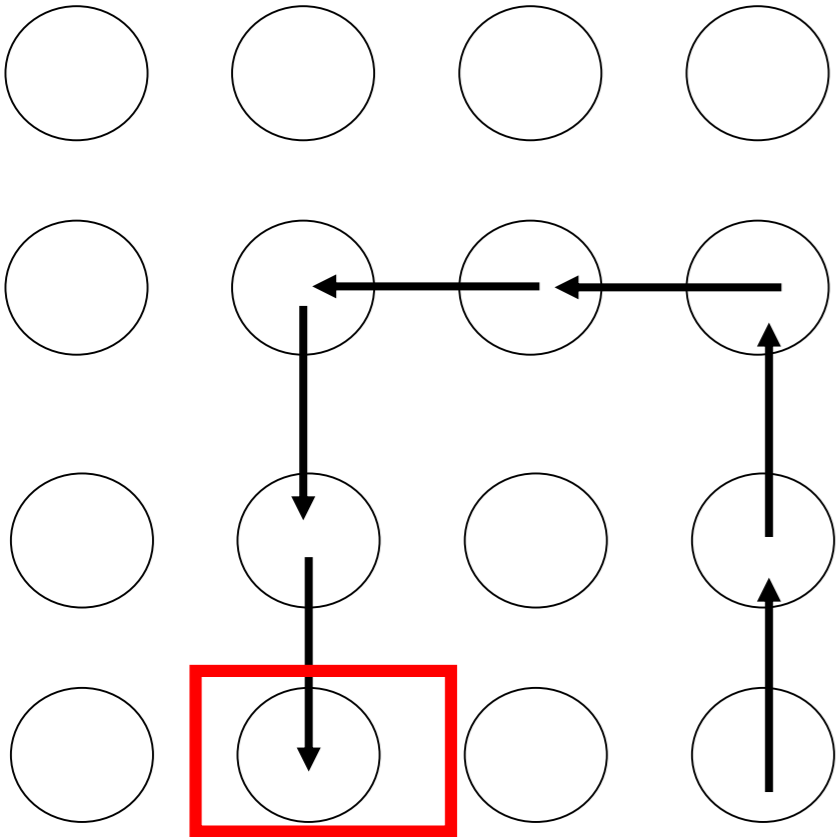
(previous slide)

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.

Yes, you avoid the disadvantage of standard TD(0) that has very propagation of information from the target back to earlier states.

Review: Eligibility Traces

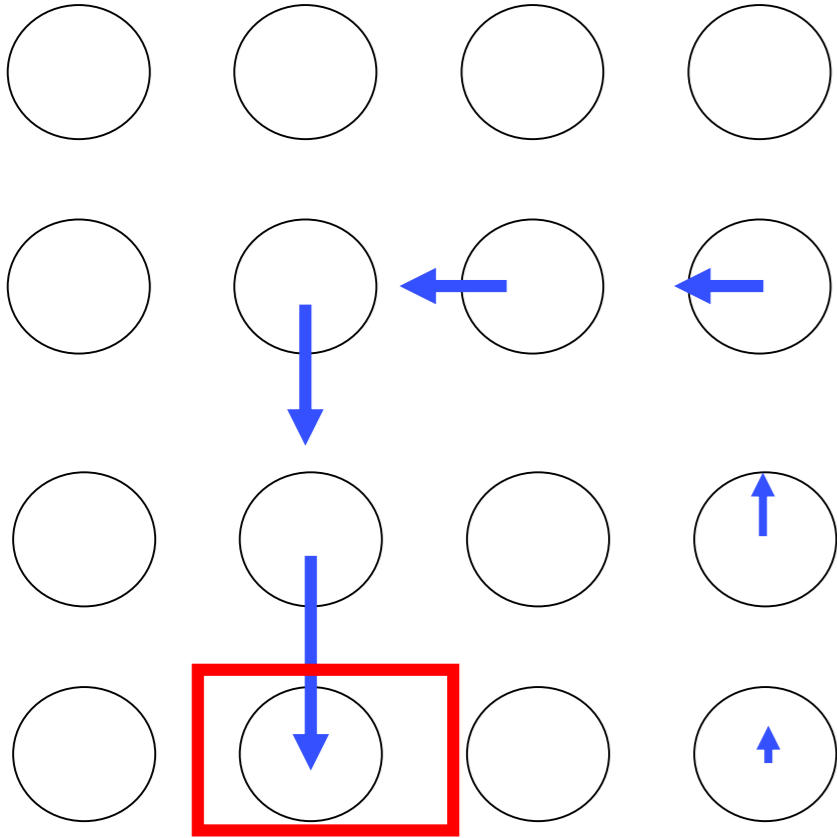


Idea:

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

$$e(s, a) \leftarrow \lambda e(s, a) \quad \text{decay of all traces}$$

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



- 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: SARSA with eligibility trace

(previous slide)

This is the SARSA algorithm with eligibility traces that we had seen some 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 in policy gradient

Idea:

- keep memory of previous ‘candidate updates’
- memory decays over time
- update an **eligibility trace for each parameter** θ_k

$$z_k \leftarrow z_k \lambda \quad \text{decay of **all** traces}$$

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

- update **all** parameters of ‘actor’ network:

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

Here: policy gradient with eligibility trace

(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

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

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’.

Actor-Critic with Eligibility traces: schematics

The actor network has parameters θ

Eligibility traces of actor have parameters z .

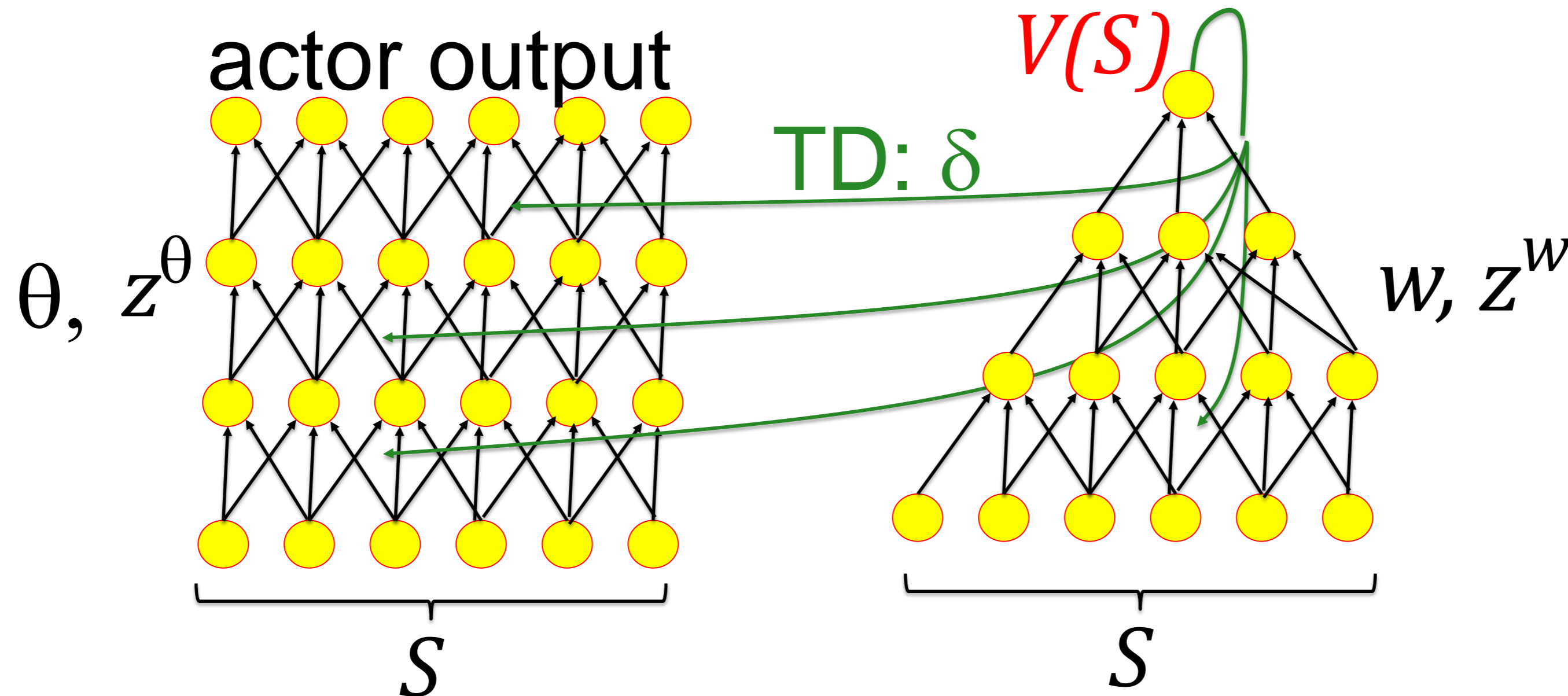
Actor chooses actions with policy π

The critic network has parameters w .

Eligibility traces of critic have parameters z .

The TD error δ is fed back to update the *weights*

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



(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

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

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$

Algorithm parameters: $\lambda^{\mathbf{w}} \in [0, 1]$, $\lambda^{\theta} \in [0, 1]$, $\alpha^{\mathbf{w}} > 0$, $\alpha^{\theta} > 0$

Initialize state-value weights $\mathbf{w} \in \mathbb{R}^d$ and policy parameter $\theta \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Initialize $S \in \mathcal{S}$ (e.g., to s_0)

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

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

Loop forever (for each time step):

$A \sim \pi(\cdot|S, \theta)$

Take action A , observe S', r

$\delta \leftarrow r + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$

$\mathbf{z}^{\mathbf{w}} \leftarrow \lambda^{\mathbf{w}} \mathbf{z}^{\mathbf{w}} + \nabla \hat{v}(S, \mathbf{w})$

$\mathbf{z}^{\theta} \leftarrow \lambda^{\theta} \mathbf{z}^{\theta} + \nabla \ln \pi(A|S, \theta)$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \mathbf{z}^{\mathbf{w}}$

$\theta \leftarrow \theta + \alpha^{\theta} \delta \mathbf{z}^{\theta}$

$S \leftarrow S'$

*Adapted from
Sutton and Barto*

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

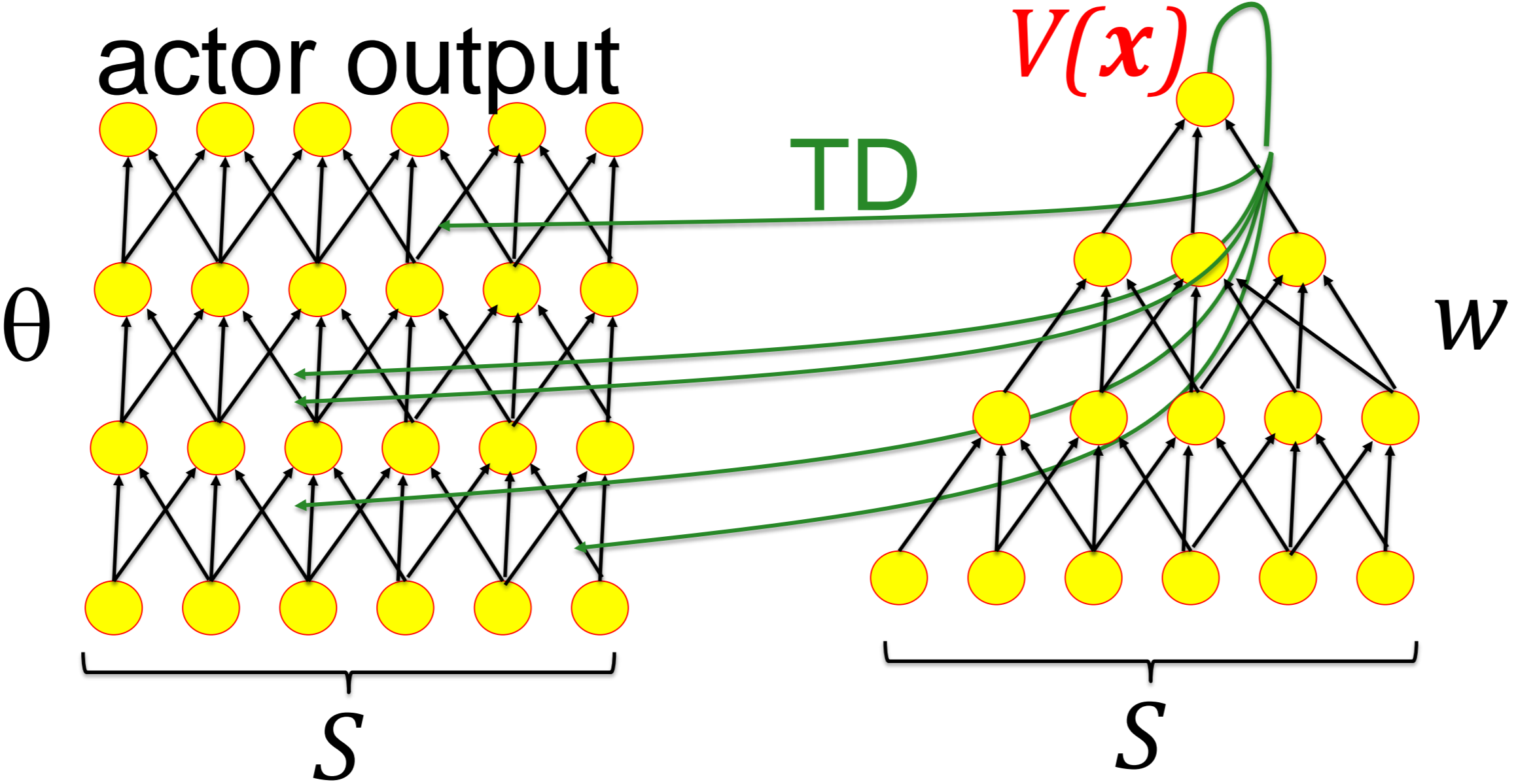
The actor network has parameters θ

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 but in the exercises we will see that the discount factor can (if $\lambda < \gamma$) be absorbed into an effective λ_0



Quiz: Policy Gradient and Reinforcement learning

Your friend claims the following. Is she right?

Eligibility traces are 'shadow' variables for each parameter

Actor-critic with eligibility is an online algorithm

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.

Actor-Critic with Eligibility traces

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

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$

Parameters: trace-decay rates $\lambda^{\theta} \in [0, 1]$, $\lambda^{\mathbf{w}} \in [0, 1]$; step sizes $\alpha^{\theta} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ 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)

$\mathbf{z}^{\theta} \leftarrow \mathbf{0}$ (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):

$A \sim \pi(\cdot|S, \theta)$

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 $\hat{v}(S', \mathbf{w}) \doteq 0$)

$\mathbf{z}^{\mathbf{w}} \leftarrow \gamma \lambda^{\mathbf{w}} \mathbf{z}^{\mathbf{w}} + I \nabla \hat{v}(S, \mathbf{w})$

$\mathbf{z}^{\theta} \leftarrow \gamma \lambda^{\theta} \mathbf{z}^{\theta} + I \nabla \ln \pi(A|S, \theta)$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \mathbf{z}^{\mathbf{w}}$

$\theta \leftarrow \theta + \alpha^{\theta} \delta \mathbf{z}^{\theta}$

$I \leftarrow \gamma I$

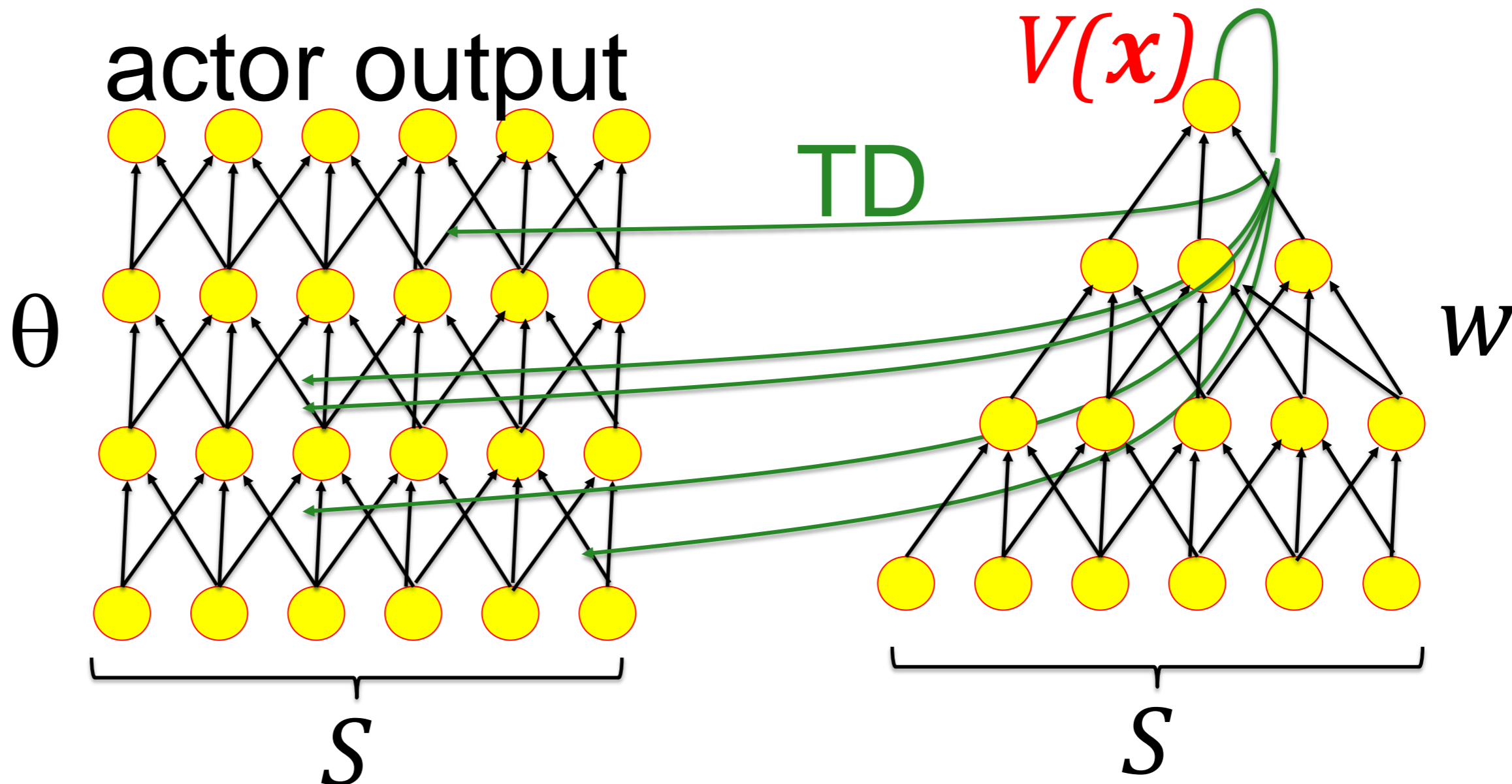
$S \leftarrow S'$

Sutton and Barto

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

Sutton and Barto make a difference between EPISODIC (one fixed start location one clear target where an episode ends) and CONTINUOUS RL scenario. This here is the episodic one.

There is an extra discount in a factor λ (for which I don't have an explanation except this one: if you redo the calculation of the exercise, but you know that you NEVER start in states 2 and 3, then the summations look differently). *Note that Sutton and Barto include a discount factor γ but in the exercises we will see that the discount factor can (if $\lambda < \gamma$) be absorbed into an effective λ_0*



Artificial Neural Networks

Deep Reinforcement Learning

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Part 4*: Eligibility traces appear naturally in policy gradient

1. Deep Q-learning
2. From Policy gradient to Deep RL
3. Actor-Critic
4. **Eligibility traces for policy gradient:**
→ How do they arise?

Previous slide.

In this section I want to show that eligibility traces arises naturally in the context of policy gradient when we optimize the expected return.

Sutton and Barto refer to this as switching from a forward view to a backward view. I have difficulties with this terminology, but let us just go through the calculation.

Review: Policy Gradient to optimize return starting at s_t, a_t

Gradient calculation yields several terms of the form

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

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

Previous slide.

This is a repetition of an earlier slide.

Policy Gradient over multiple time steps: toward eligibility traces

Step 1: Rewrite $R_{s_t \rightarrow s_{end}}^{a_t} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots$

Step 2: Use same update formula, but also for state s_{t+k}

Step 3: Reorder terms according to r_{t+n}

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

Previous slide.

This is an introduction to the main idea of one of the exercises.

Policy Gradient over multiple time steps: toward eligibility traces

$$R_{s_t \rightarrow s_{end}}^{a_t} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3}$$

Steps 1+3: Insert and reorder terms according to r_{t+n}

$$\begin{aligned} \Delta \theta_j &\propto [R_{s_t \rightarrow s_{end}}^{a_t}] \frac{d}{d\theta_j} \ln[\pi(a_t | s_t, \theta)] \\ &\quad [r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots] \frac{d}{d\theta_j} \ln[\pi(a_t | s_t, \theta)] \\ &+ \gamma [R_{s_{t+1} \rightarrow s_{end}}^{a_{t+1}}] \frac{d}{d\theta_j} \ln[\pi(a_{t+1} | s_{t+1}, \theta)] \\ &\quad [\gamma r_{t+1} + \gamma^2 r_{t+2} + \dots] \frac{d}{d\theta_j} \ln[\pi(a_{t+1} | s_{t+1}, \theta)] \\ &+ \gamma^2 [R_{s_{t+2} \rightarrow s_{end}}^{a_{t+2}}] \frac{d}{d\theta_j} \ln[\pi(a_{t+2} | s_{t+2}, \theta)] \\ &\quad [\gamma^2 r_{t+2} + \dots] \frac{d}{d\theta_j} \ln[\pi(a_{t+2} | s_{t+2}, \theta)] \end{aligned}$$

Steps 2 and 3: reorder terms according to r_{t+n}

Optimizing return from s_t, a_t : $R_{s_t \rightarrow s_{end}}^{a_t} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots$

$$\Delta\theta_j \propto [r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots] \frac{d}{d\theta_j} \ln[\pi(a_t | s_t, \theta)]$$

$$[\gamma r_{t+1} + \gamma^2 r_{t+2} + \dots] \frac{d}{d\theta_j} \ln[\pi(a_{t+1} | s_{t+1}, \theta)]$$

$$[\gamma^2 r_{t+2} + \dots] \frac{d}{d\theta_j} \ln[\pi(a_{t+2} | s_{t+2}, \theta)]$$

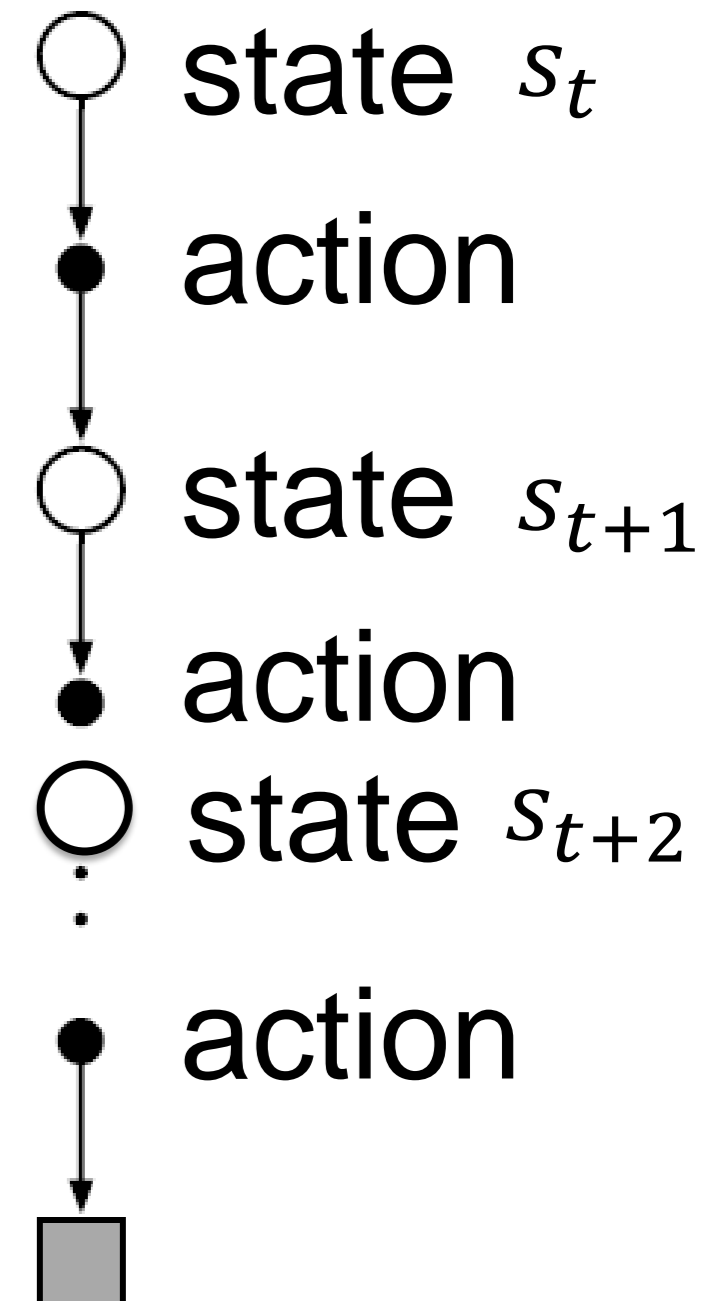
Optimizing return from s_{t+1}, a_{t+1} : $R_{s_{t+1} \rightarrow s_{end}}^{a_{t+1}} = r_{t+1} + \gamma r_{t+2} + \dots$

$$\Delta\theta_j \propto [r_{t+1} + \gamma r_{t+2} + \dots] \frac{d}{d\theta_j} \ln[\pi(a_{t+1} | s_{t+1}, \theta)]$$

$$[\gamma r_{t+2} + \dots] \frac{d}{d\theta_j} \ln[\pi(a_{t+2} | s_{t+2}, \theta)]$$

Optimizing return from s_{t+2}, a_{t+2} : $R_{s_{t+2} \rightarrow s_{end}}^{a_{t+2}} = r_{t+2} + \gamma r_{t+3} + \dots$

$$\Delta\theta_j \propto [r_{t+2} + \dots] \frac{d}{d\theta_j} \ln[\pi(a_{t+2} | s_{t+2}, \theta)]$$



end of trial

Policy Gradient over multiple time steps: toward eligibility traces

Step 4: Introduce 'shadow variables' for eligibility trace

$$z_k \leftarrow z_k \lambda \quad \text{decay of **all** traces}$$

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

Step 5: Rewrite update rule for parameters with eligibility trace

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

Step 6: subtract baseline: $\delta_t = [r_t + \gamma V(s_{t+1}) - V(s_t)]$

$$\Delta\theta_k = \eta \delta_t z_k$$

Previous slide.

This is a sketch of the exercise. After the reordering of the terms, we introduce eligibility traces which are adapted online.

Policy Gradient with eligibility traces: essentials

Maximizing discounted return with policy gradient gives naturally rise to eligibility traces! → easy to implement:

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

1) Update eligibility trace

$$z_k \leftarrow z_k \lambda \quad \text{decay of **all** traces}$$

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

2) update parameters

$$\Delta\theta_k = \eta \delta_t z_k$$

→ see pseudo-algorithm: actor-critic with eligibility trace

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: eligibility traces are a compact form for rewriting a policy gradient algorithm.

[There are minor differences at the 'boundaries' that is, the beginning and end of each episode – but these do not matter].

Actor-Critic with Eligibility traces

Actor-Critic with Eligibility Traces (continuing), for estimating $\pi_{\theta} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$

Algorithm parameters: $\lambda^{\mathbf{w}} \in [0, 1]$, $\lambda^{\theta} \in [0, 1]$, $\alpha^{\mathbf{w}} > 0$, $\alpha^{\theta} > 0$, $\alpha^{\bar{R}} > 0$

Initialize $\bar{R} \in \mathbb{R}$ (e.g., to 0)

Initialize state-value weights $\mathbf{w} \in \mathbb{R}^d$ and policy parameter $\theta \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Initialize $S \in \mathcal{S}$ (e.g., to s_0)

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

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

Loop forever (for each time step):

$A \sim \pi(\cdot|S, \theta)$

Take action A , observe S' , r

$\delta \leftarrow r - \bar{R} + \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$

$\bar{R} \leftarrow \bar{R} + \alpha^{\bar{R}} \delta$

$\mathbf{z}^{\mathbf{w}} \leftarrow \lambda^{\mathbf{w}} \mathbf{z}^{\mathbf{w}} + \nabla \hat{v}(S, \mathbf{w})$

$\mathbf{z}^{\theta} \leftarrow \lambda^{\theta} \mathbf{z}^{\theta} + \nabla \ln \pi(A|S, \theta)$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \mathbf{z}^{\mathbf{w}}$

$\theta \leftarrow \theta + \alpha^{\theta} \delta \mathbf{z}^{\theta}$

$S \leftarrow S'$

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: eligibility traces are a compact form for rewriting a policy gradient algorithm.

[There are minor differences at the ‘boundaries’ that is, the beginning and end of each episode, if we have an episodic RL scenario – but these do not matter].

In this version here, Sutton and Barto have another bias correction \bar{R} .

For some reason they have no γ in the TD-delta, which I normally add.

Summary: Actor-critic with eligibility traces

- Actor learns by policy gradient
 - eligibility trace appear naturally
- Critic learns by TD-learning
 - eligibility trace avoids problem of slow information transfer
 - rapid learning of critic
- Update eligibility traces while moving
 - online, on-policy
 - short back-up diagram
- For each parameter, one eligibility trace
 - shadow parameter: candidate weight update
- Update actual weights is proportional to TD-delta

→ use this algo!

Your comments.

Artificial Neural Networks

Deep Reinforcement Learning

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Part 5: Three-factors rules in Actor-Critic Learning

1. Deep Q-learning
2. From Policy gradient to Deep RL
3. Actor-Critic
4. Eligibility traces for policy gradient and actor-critic
5. **Three-factor rules in Actor-Critic Learning**

Review: Actor-Critic with TD signal and eligibility trace

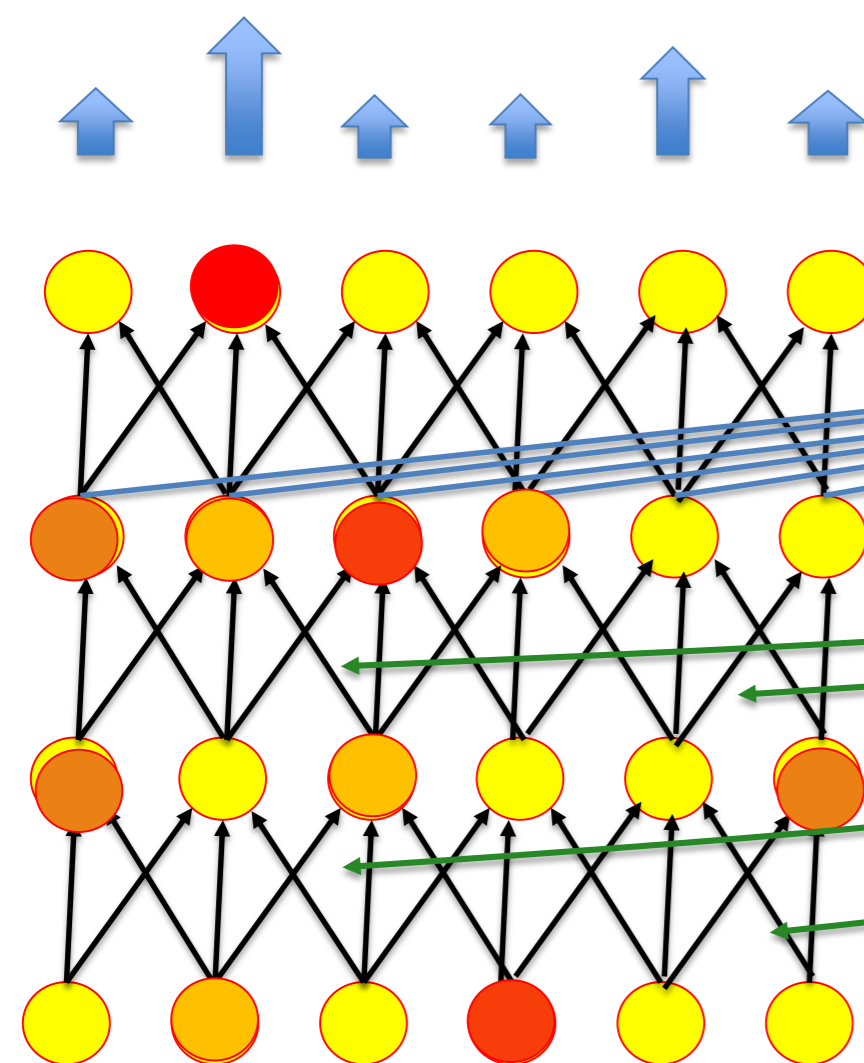
Actor

Critic

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

actions

value



$V(s)$ while acting:

- update eligibility traces z_{lk}
- calculate values $V(s)$

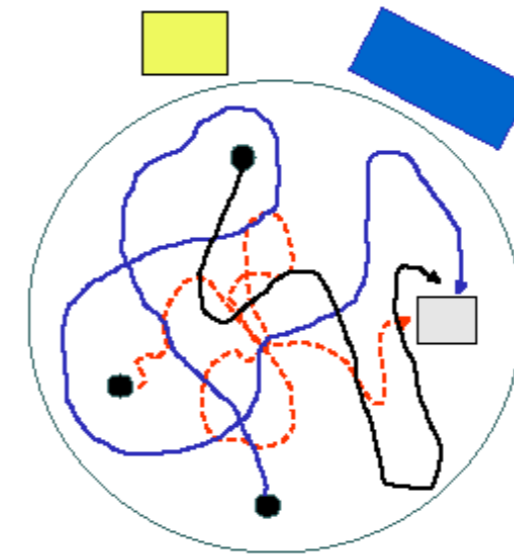
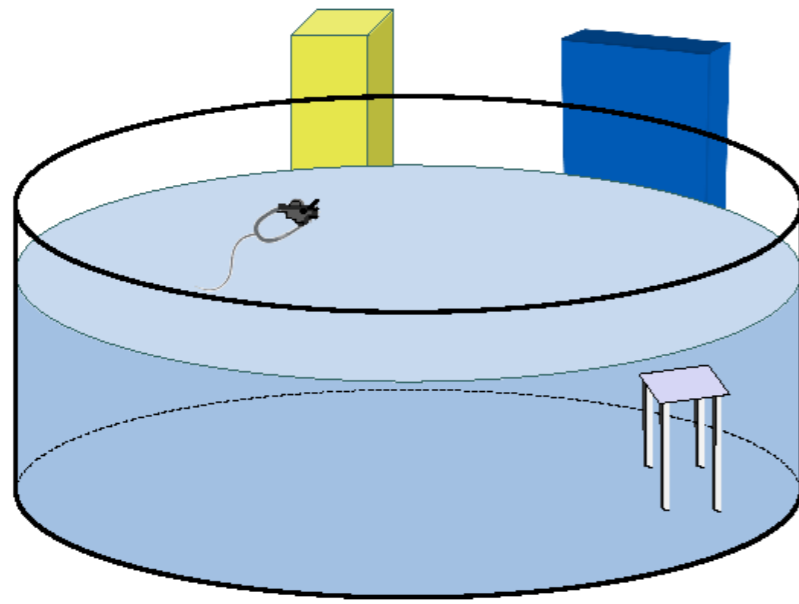
TD-error $\delta = \eta [r_t - [V(s) - \gamma V(s')]]$

- update weights

$$\Delta w_{lk} = \eta \delta_t z_{lk}$$

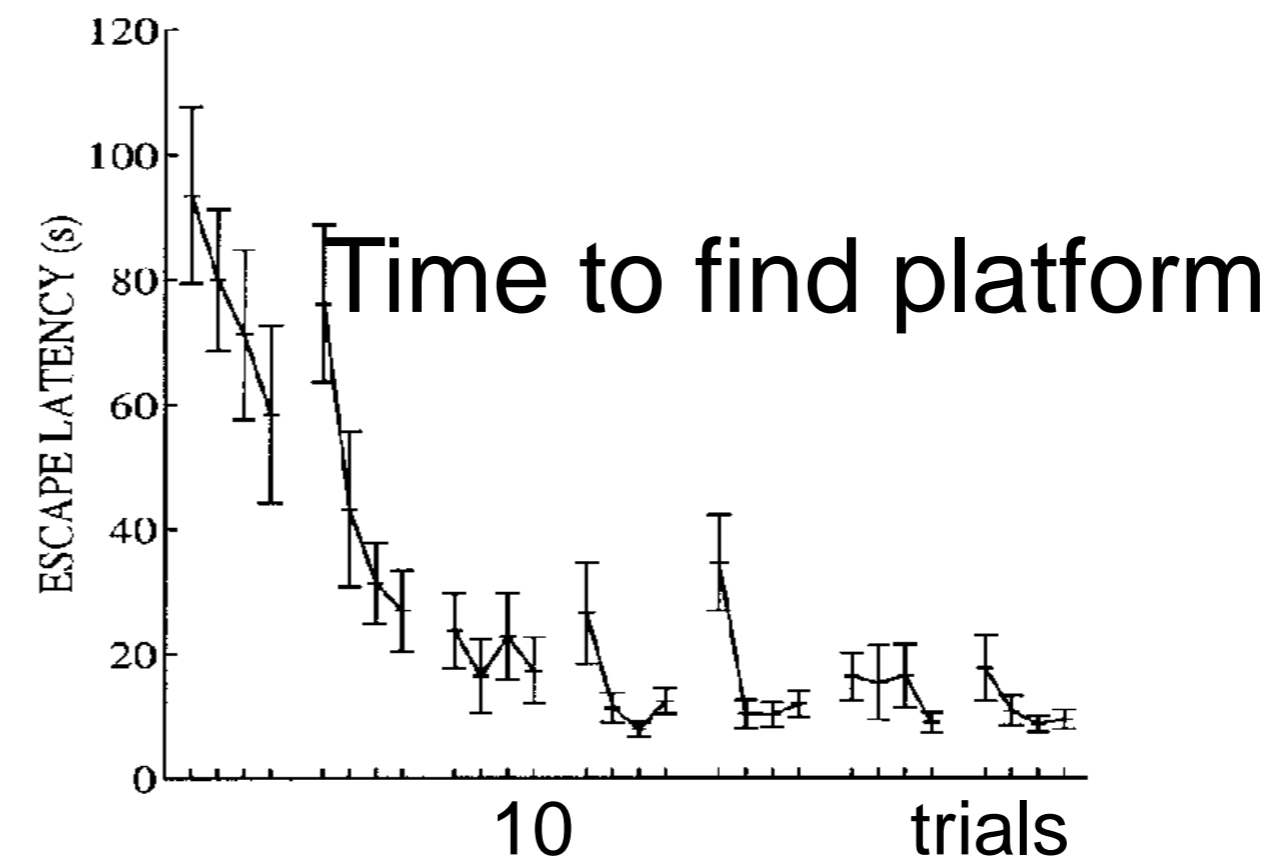
Animal conditioning: Learning to escape

Morris Water Maze



Rats learn to find
the hidden platform

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



Foster, Morris, Dayan 2000

Previous slide.

Behavioral experiment in the Morris Water Maze.

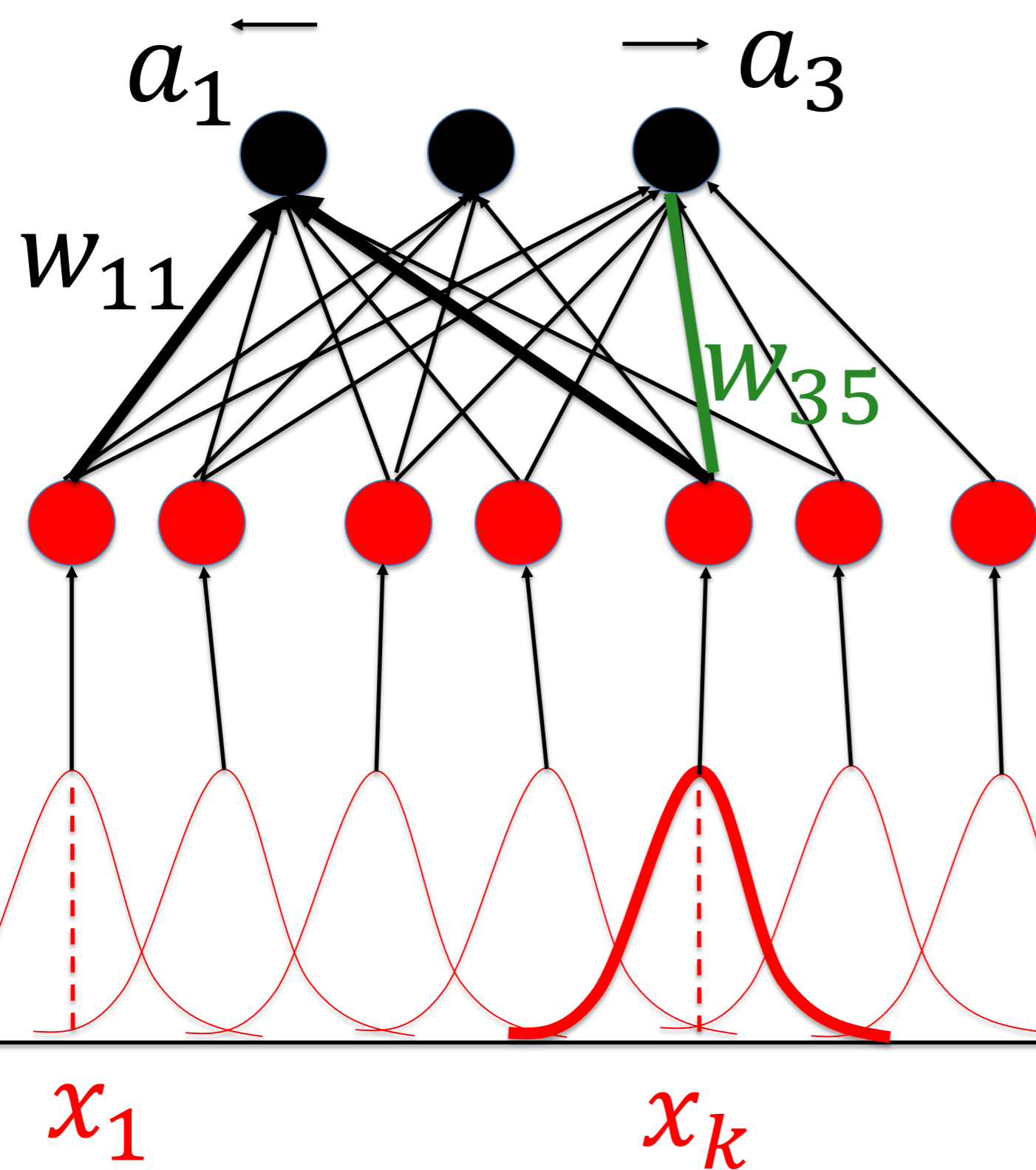
The water is milky so that the platform is not visible.

After a few trials the rat swims directly to the platform.

Platform is like a reward because the rat gets out of the cold water.

Linear activation model with softmax policy

left: $a_1=1$ stay: $a_2=1$ right: $a_3=1$



$$\pi(a_j = 1 | x, \theta) = \text{softmax} \left[\sum_k w_{jk} y_k \right]$$

$$y_k = f(x - x_k)$$

f = basis function

calculate derivative:

$$\Delta z_{35} = \frac{d}{dw_{35}} \ln[\pi(a_i = 1 | x)]$$

Previous slide.

I now want to show that reinforcement learning with policy gradient gives rise to three-factor learning rules.

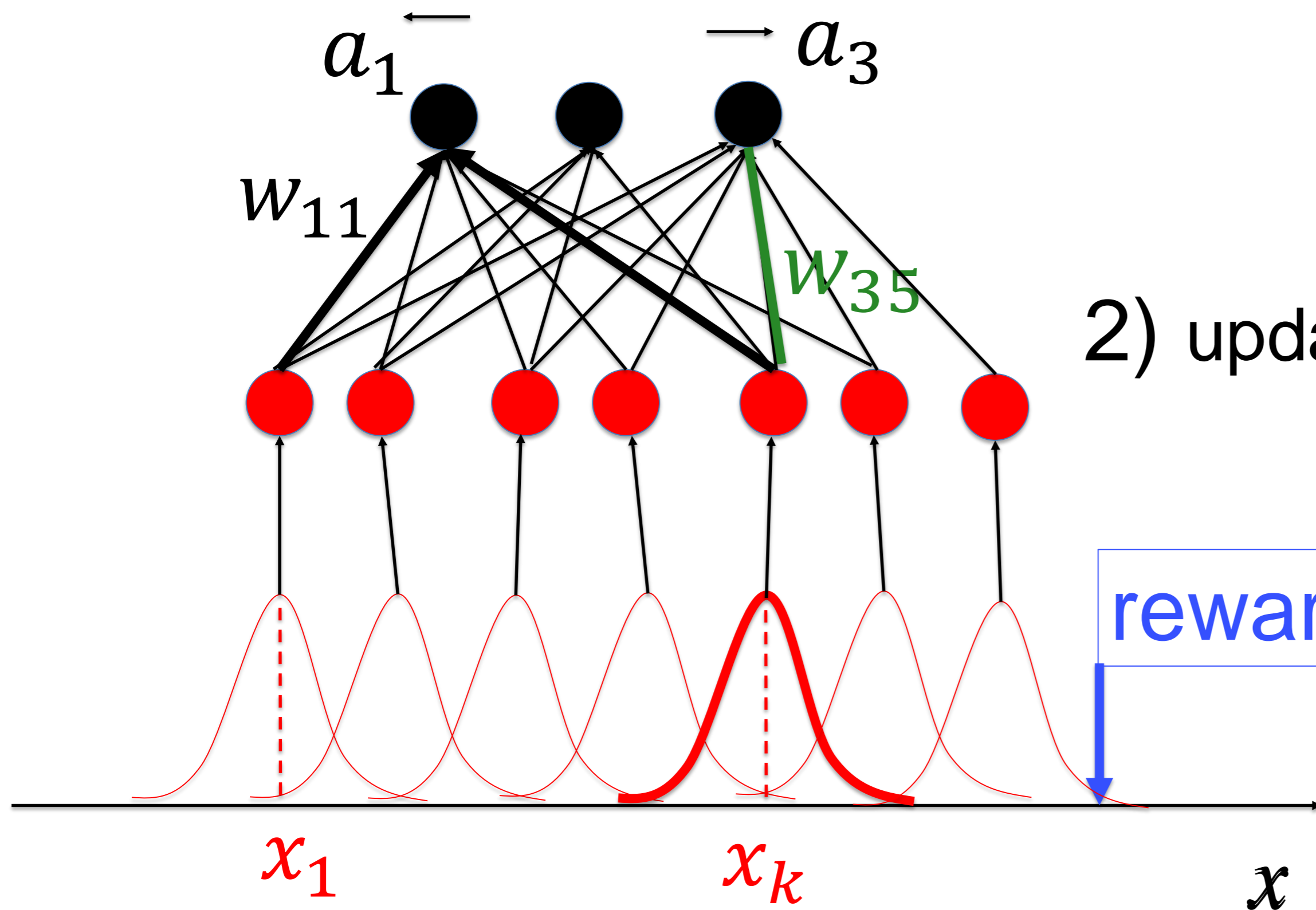
Suppose the agent moves on a linear track.

There are three possible actions: left, right, or stay.

The policy is given by the softmax function. The total drive of the action neurons is a linear function of the activity y of the hidden neurons which in turn depends on the input x . The activity of hidden neuron k is $f(x-x_k)$. The basis function f could for example be a Gaussian function with center at x_k .

Policy gradient model with softmax policy and eligibility trace

left: stay: right:



1) Update eligibility trace (for each weight)

$$z_{ik} \leftarrow z_{ik} \lambda$$

$$z_{ik} \leftarrow z_{ik} + \frac{d}{dw_{ik}} \ln[\pi(a_i|x)]$$

2) update weights

$$\Delta w_{ik} = \eta [r_t - b] z_{ik}$$

From calculation

Previous slide.

Now we apply the update rule resulting from policy gradient with eligibility traces descent (copy from earlier slide).

This is the in-class exercise (Exercise 1 of this week), explicitly rewritten with eligibility traces.

Example: Linear activation model with softmax policy

left: $a_1=1$ stay: $a_2=1$ right: $a_3=1$

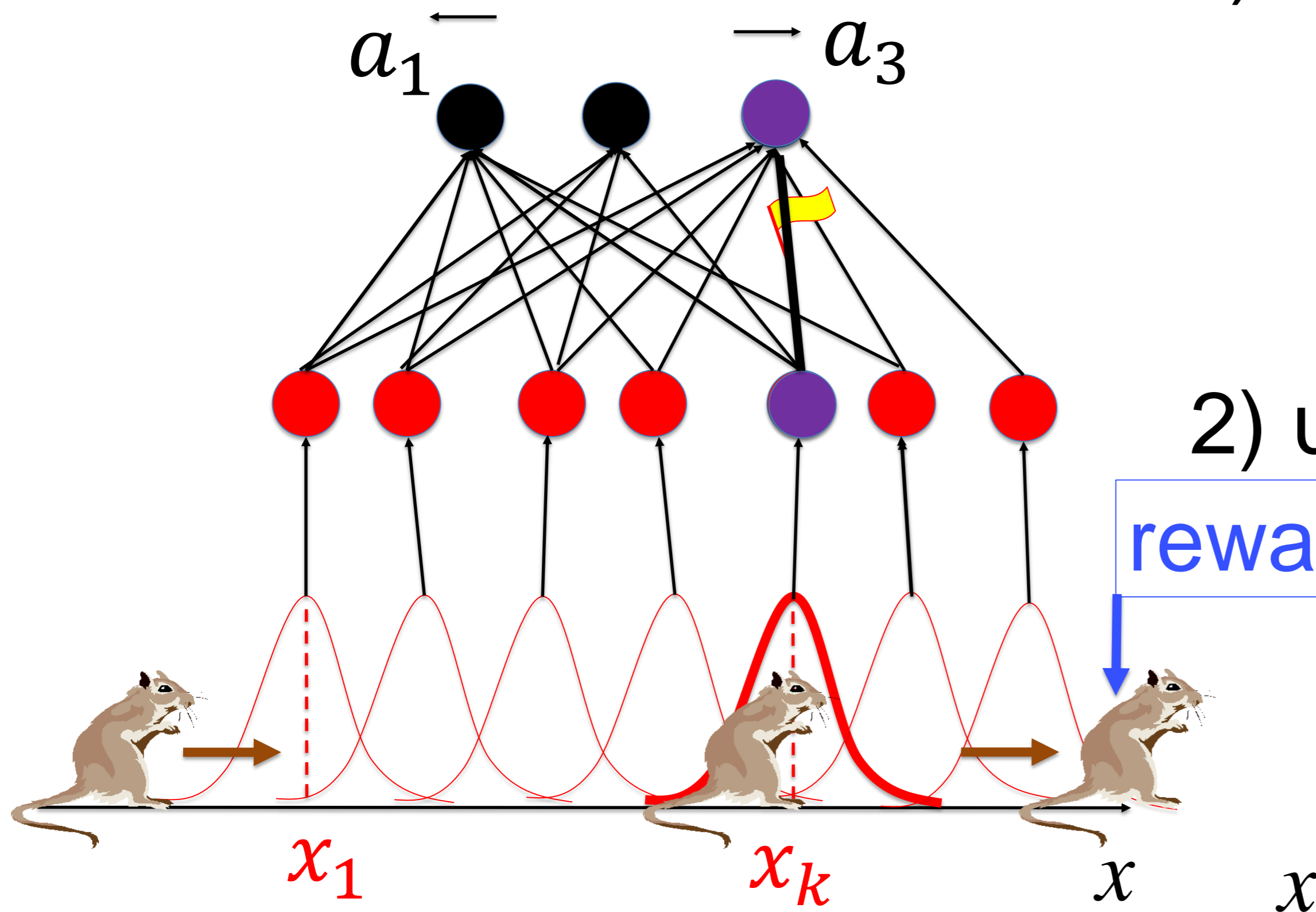
- 0) Choose action $a_i \in \{0,1\}$
- 1) Update eligibility trace

$$Z_{ik} \leftarrow Z_{ik} \lambda$$

$$Z_{ik} \leftarrow Z_{ik} + y_k(x)[a_i - \pi(a_i|x)]$$

- 2) update weights

$$\Delta w_{ik} = \eta [r_t - b] Z_{ik}$$



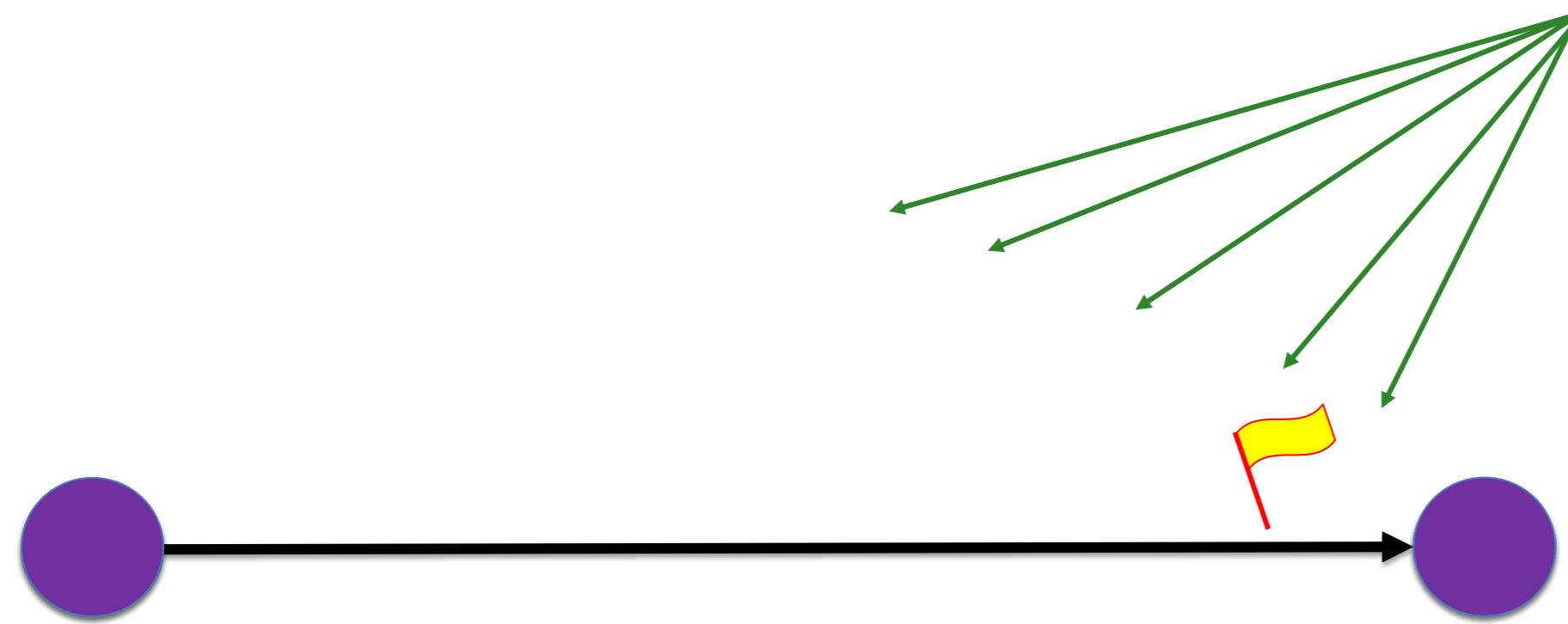
Previous slide.

This is the result of the in-class exercise (Exercise 1 of this week).

Importantly, the update of the eligibility trace is a local learning rule that depends on a presynaptic factor and a postsynaptic factor.

3-factor rule

- Eligibility trace is set by joint activity of presynaptic and postsynaptic neuron
 - Update proportional to $[\text{reward} - b]$ and eligibility trace
- success* = $[\text{reward} - b]$



presynaptic neuron:
index j
(represents state)

postsynaptic neuron:
index i
(represents action)

Previous slide.

Three factor rule needs

- Activity of presynaptic neuron
- Activity of postsynaptic neuron
- Broadcasted success signal: reward minus baseline

These three factor learning rules are important because they are completely asynchronous, local, and online and could therefore be implemented in biology or parallel hardware.

Summary: 3-factor rules from Policy gradient

- Policy gradient with one hidden layer and linear softmax readout yields a 3-factor rule
- Eligibility trace is set by joint activity of presynaptic and postsynaptic neuron
- Update happens proportional to reward and eligibility trace

- The presynaptic neuron represents the state
- The postsynaptic neuron the action
- True online rule
 - could be implemented in biology
 - can also be implemented in parallel asynchr. hardware

Previous slide.

Summary: A policy gradient algorithm in a network where the output layer has a linear drive with softmax output leads to a three-factor learning rule for the connections between neurons in the hidden layer and the output.

These three factor learning rules are important because they are completely asynchronous, local, and online and could therefore be implemented in biology or parallel hardware.

Artificial Neural Networks

Deep Reinforcement Learning

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Part 5*: Application: real brains and real tasks

1. Deep Q-learning
2. From Policy gradient to Deep RL
3. Actor-Critic
4. Eligibility traces for policy gradient and actor-critic
5. **Three-factor rules**
Application: real brains and real tasks

(your comments)

Reinforcement learning algorithms are a class of 'bio-inspired' algorithms. But have they something to do with the brain?

Neural Networks for action learning

Learning by reward:

- Learning skills
- Learning to find food

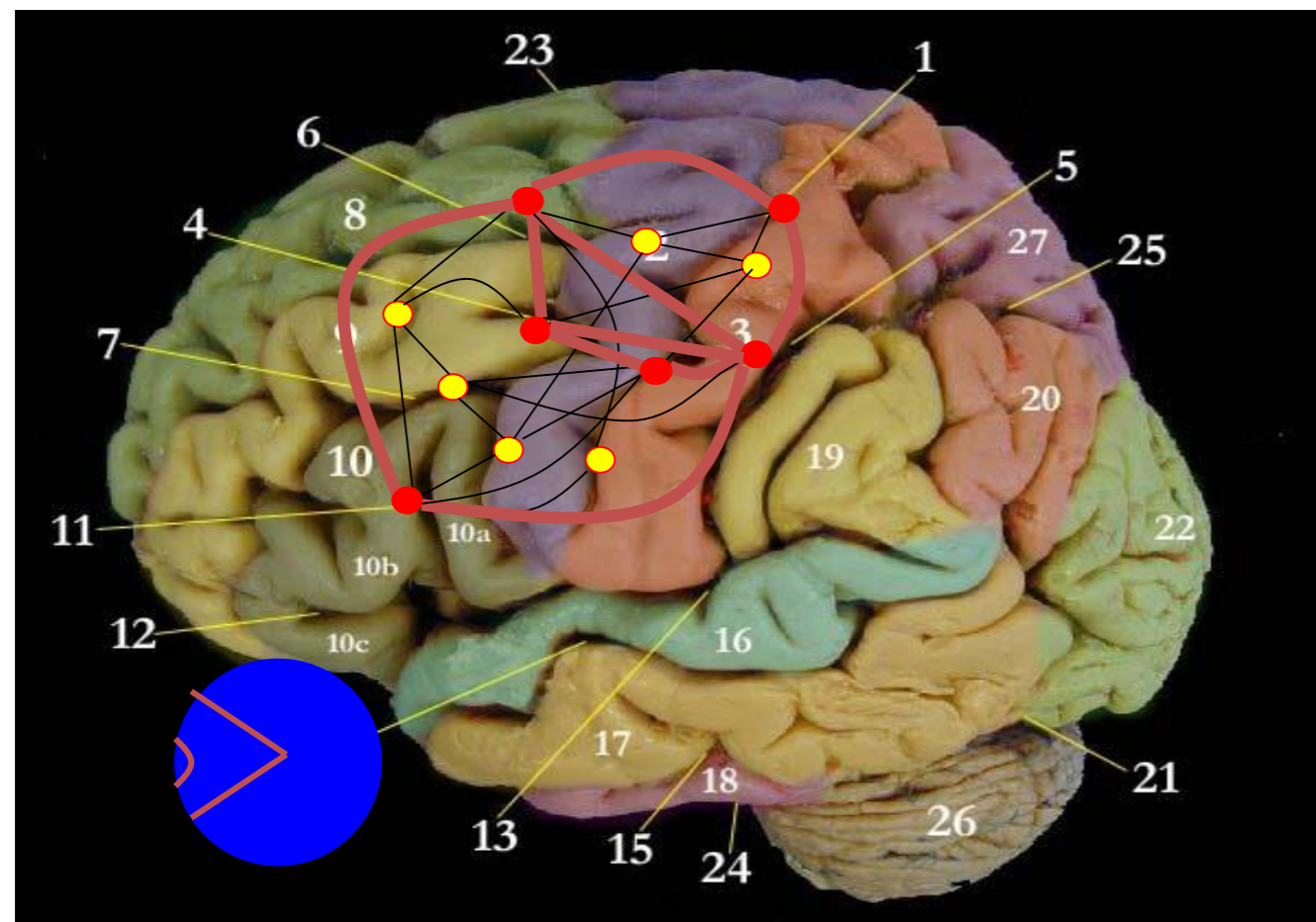
‘Reward for action’

- ‘goodie’ for dog
- ‘success’
- ‘compliment’
- ‘chocolate’

BUT:

Reward is rare:

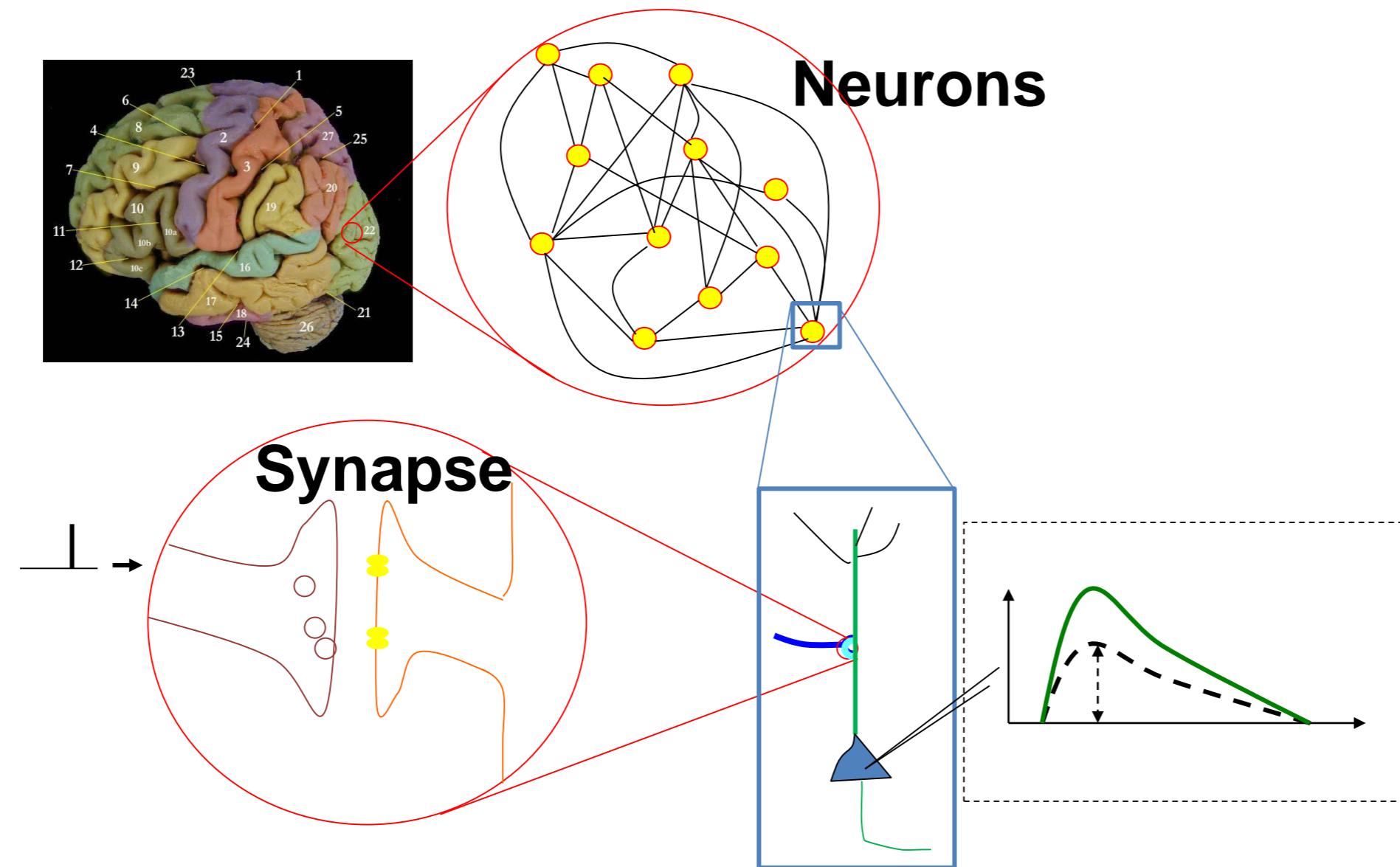
‘sparse feedback’ after a long action sequence



Actor-Critic in the brain?

Questions for this part:

- does the brain implement reinforcement learning algorithms?
- Can the brain implement an actor-critic structure?

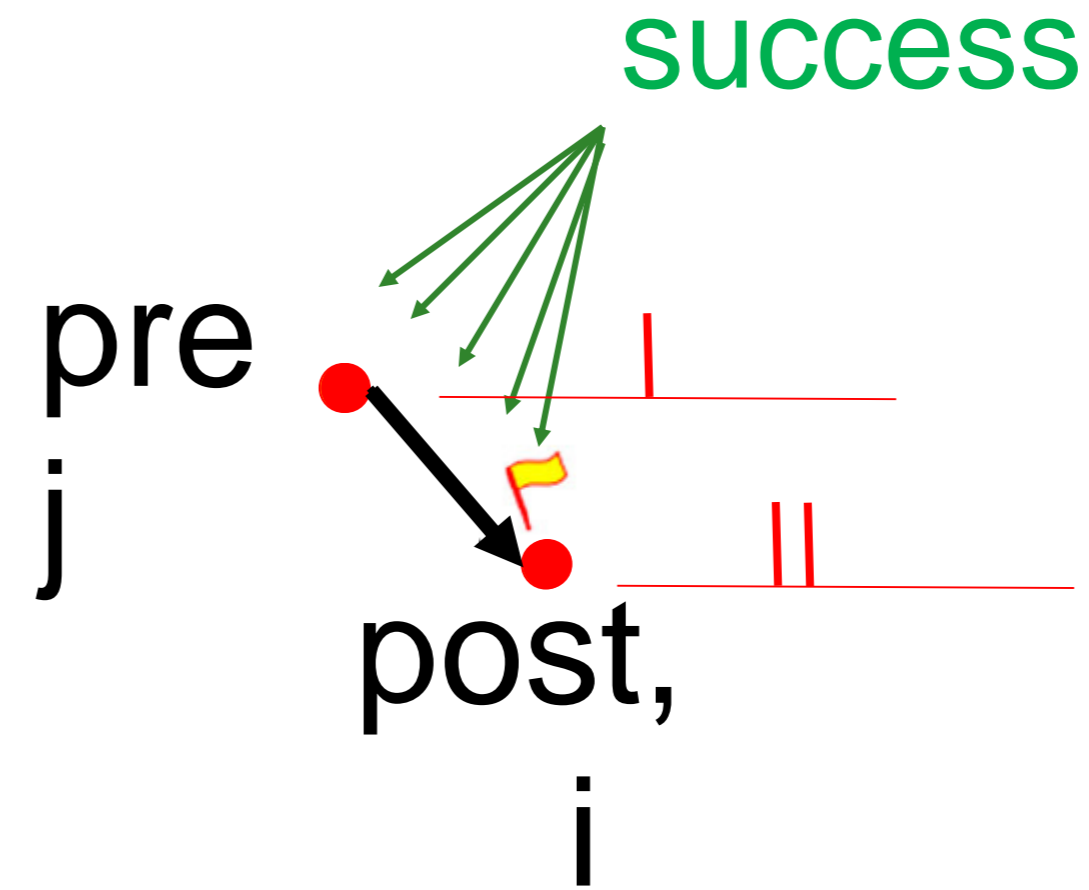


Synaptic Plasticity = Change in Connection Strength

Three-factor rules in the brain?

- does the brain implement reinforcement learning algorithms?
→ do synaptic connections follow three-factor rules?

$$\Delta w_{ij} = F(w_{ij}; \text{PRE}_j, \text{POST}_i, \text{3rd factor})$$



3rd factor: success

Candidate neuromodulator

- Dopamine

*Crow 1968; Barto 1985
Schultz et al. 1997; Waelti et al., 2001;
Reynolds and Wickens 2002;
Lisman et al. 2011*

Previous slide.

Reinforcement Learning includes a set of very powerful algorithms – as we have seen in previous lectures.

For today the big question is:

Is the structure of the brain suited to implement reinforcement learning algorithms?

If so which one? Q-learning or SARSA? Policy gradient?

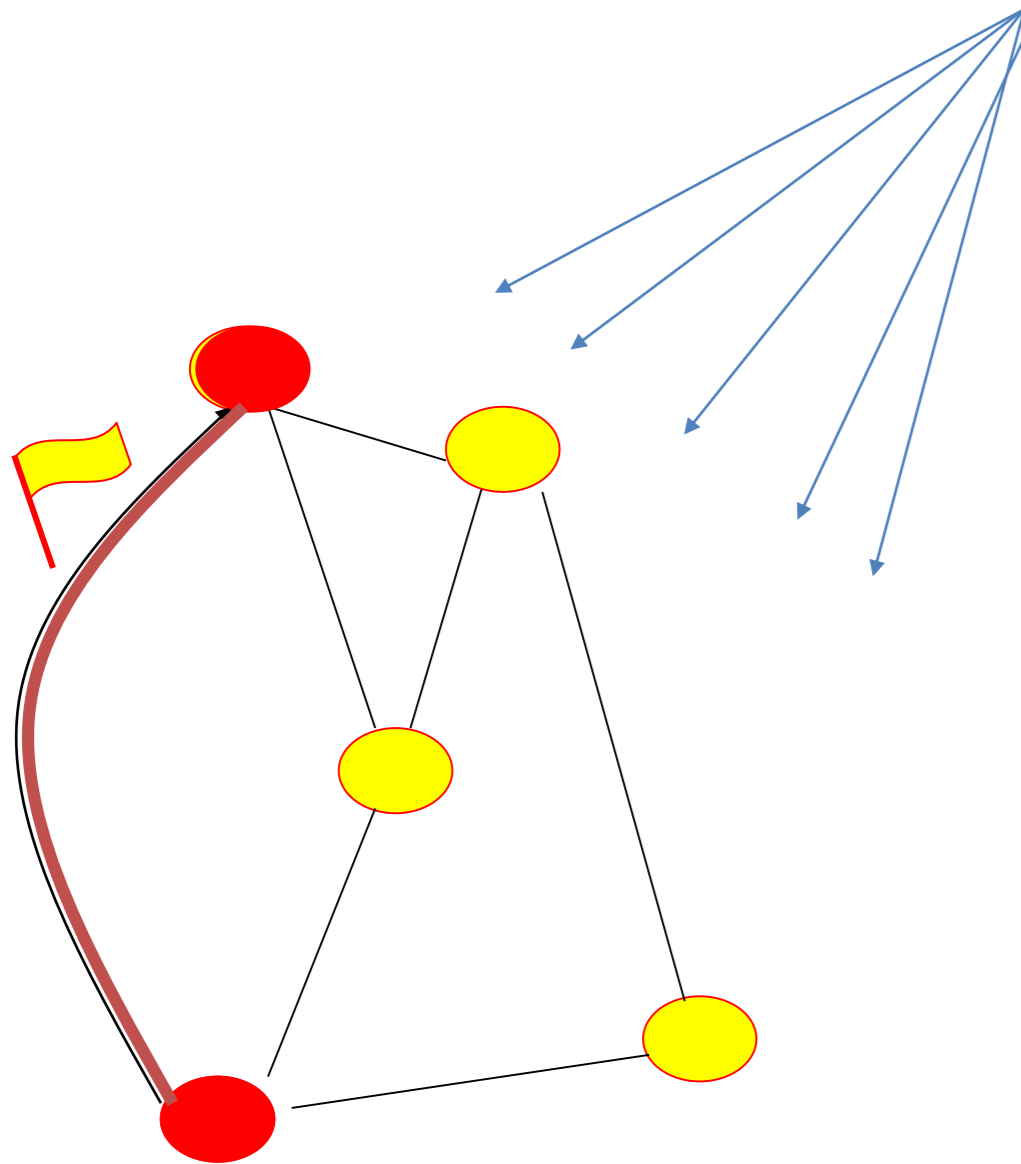
Is the brain architecture compatible with an actor-critic structure?

These are the questions we will address in the following.

And to do so, we have to first get a bit of background information on brain anatomy.

Recent experiments on Three-factor rules

Neuromodulators for reward, interestingness, surprise;
attention; novelty



Step 1: co-activation sets eligibility trace

Step 2: eligibility trace decays over time

Step 3: delayed neuro-Modulator:
eligibility trace translated into weight change

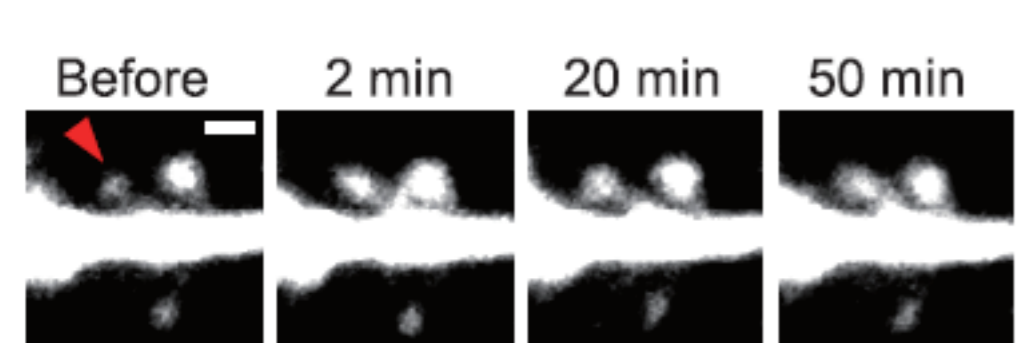
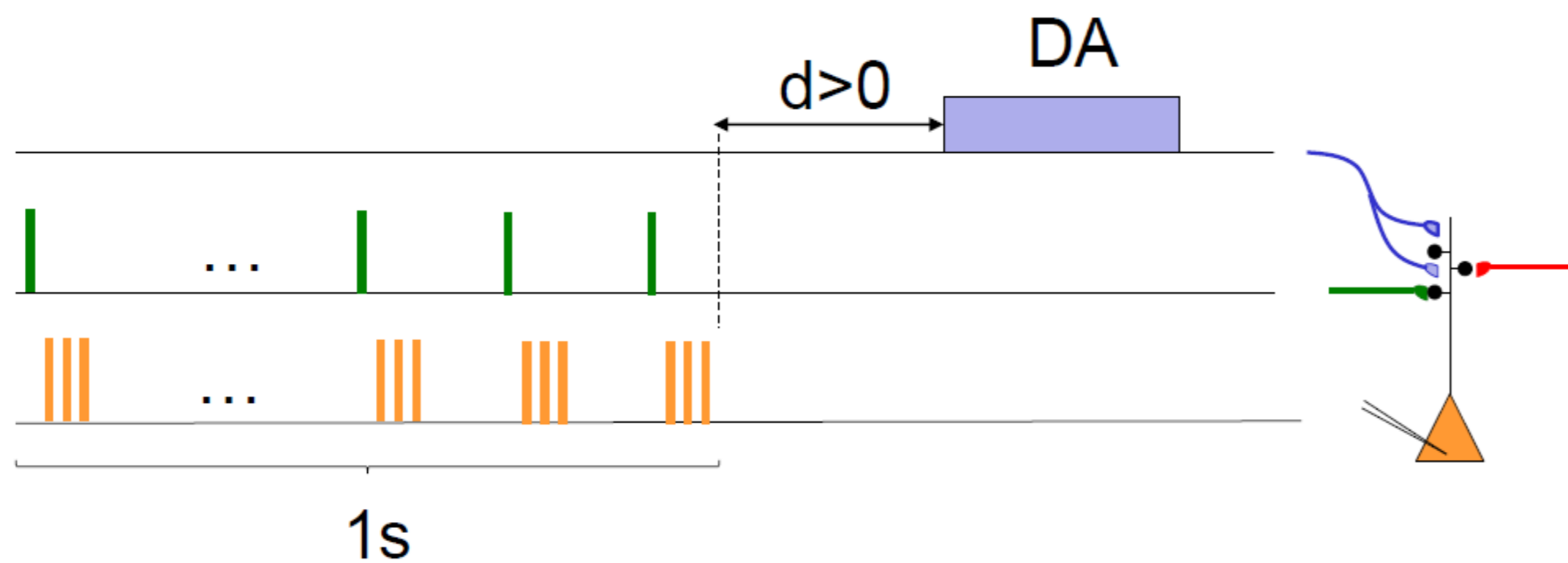
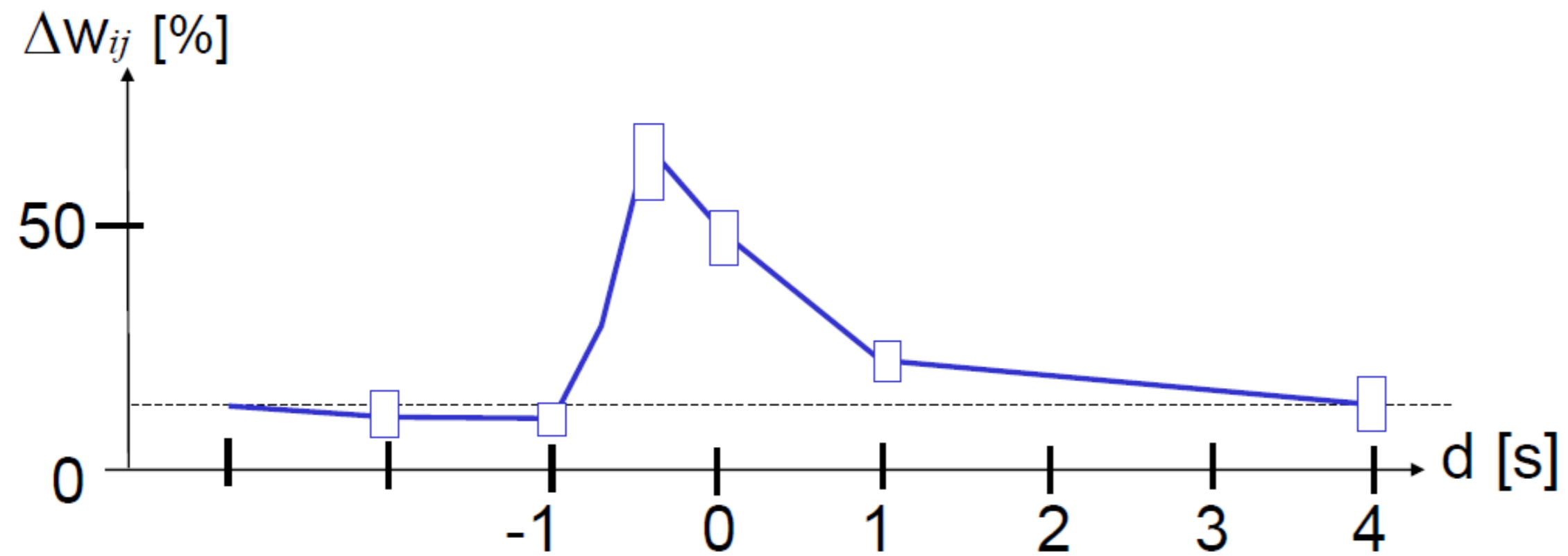
Previous slide.

three-factor learning rules are a theoretical concept.

But are there any experiments? Only quite recently, a few experimental results were published that directly address this question.

Three-factor rules in striatum: eligibility trace and delayed DA

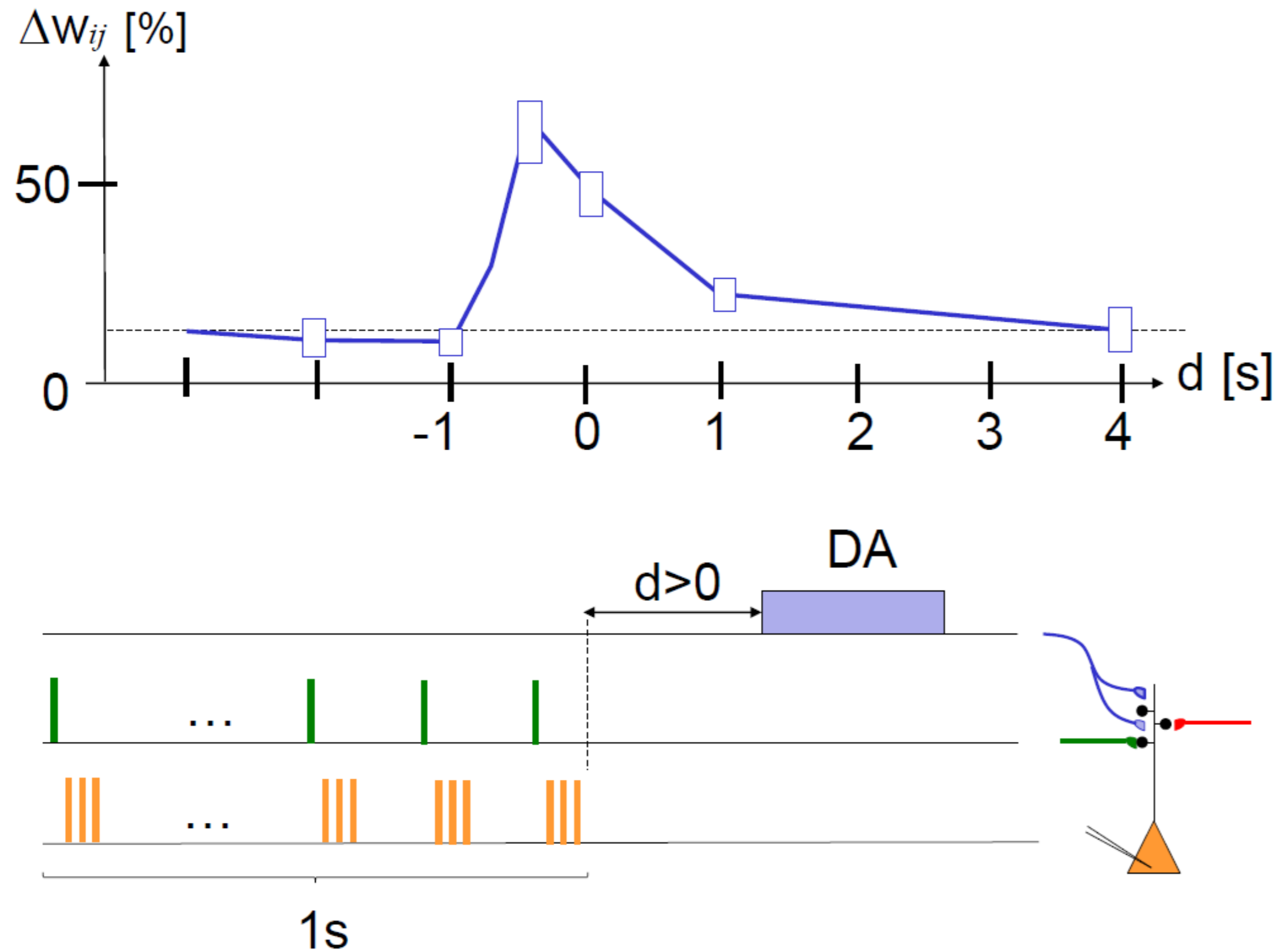
Yagishita et al. 2014



@457 nm, 30 Hz x 10

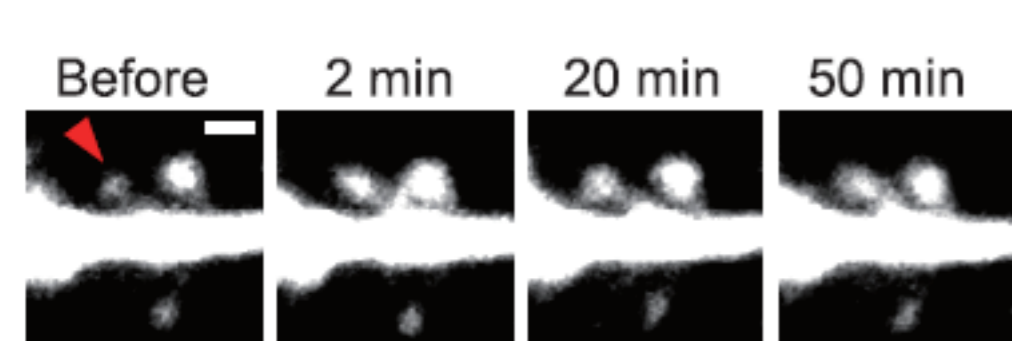
- Dopamine can come with a delay of 0 -1s
- Long-Term stability over at least 50 min.

5. Three-factor rules in striatum: eligibility trace and delayed Da



Yagishita et al. 2014

In striatum medial spiny cells, stimulation of presynaptic glutamatergic fibers (green) followed by three postsynaptic action potentials (STDP with pre-post-post-post at +10ms) repeated 10 times at 10Hz yields LTP if dopamine (DA) fibers are stimulated during the presentation ($d < 0$) or shortly afterward ($d = 0$ s or $d = 1$ s) but not if dopamine is given with a delay $d = 4$ s; redrawn after Fig. 1 of (Yagishita et al., 2014), with delay d defined as time since end of STDP protocol



@457 nm, 30 Hz x 10

- Dopamine can come with a delay of 0-1s
- Long-Term stability over at least 50 min.

Neuromodulators as Third factor

Three factors are needed for synaptic changes:

- Presynaptic factor = activity of presynaptic neuron
- Postsynaptic factor = activity of postsynaptic neuron
- Third factor = Neuromodulator such as dopamine

Presynaptic and postsynaptic factor '**select**' the synapse.

→ a small subset of synapses becomes 'eligible' for change.

The 'Third factor' is a nearly **global** signal

→ broadcast signal, potentially received by all synapses.

Synaptic change requires all three factors

Previous slide.

The third factor in a three-factor learning rule should be the global factor signaling success or reward. We said that the third factor could be a neuromodulator such as dopamine.

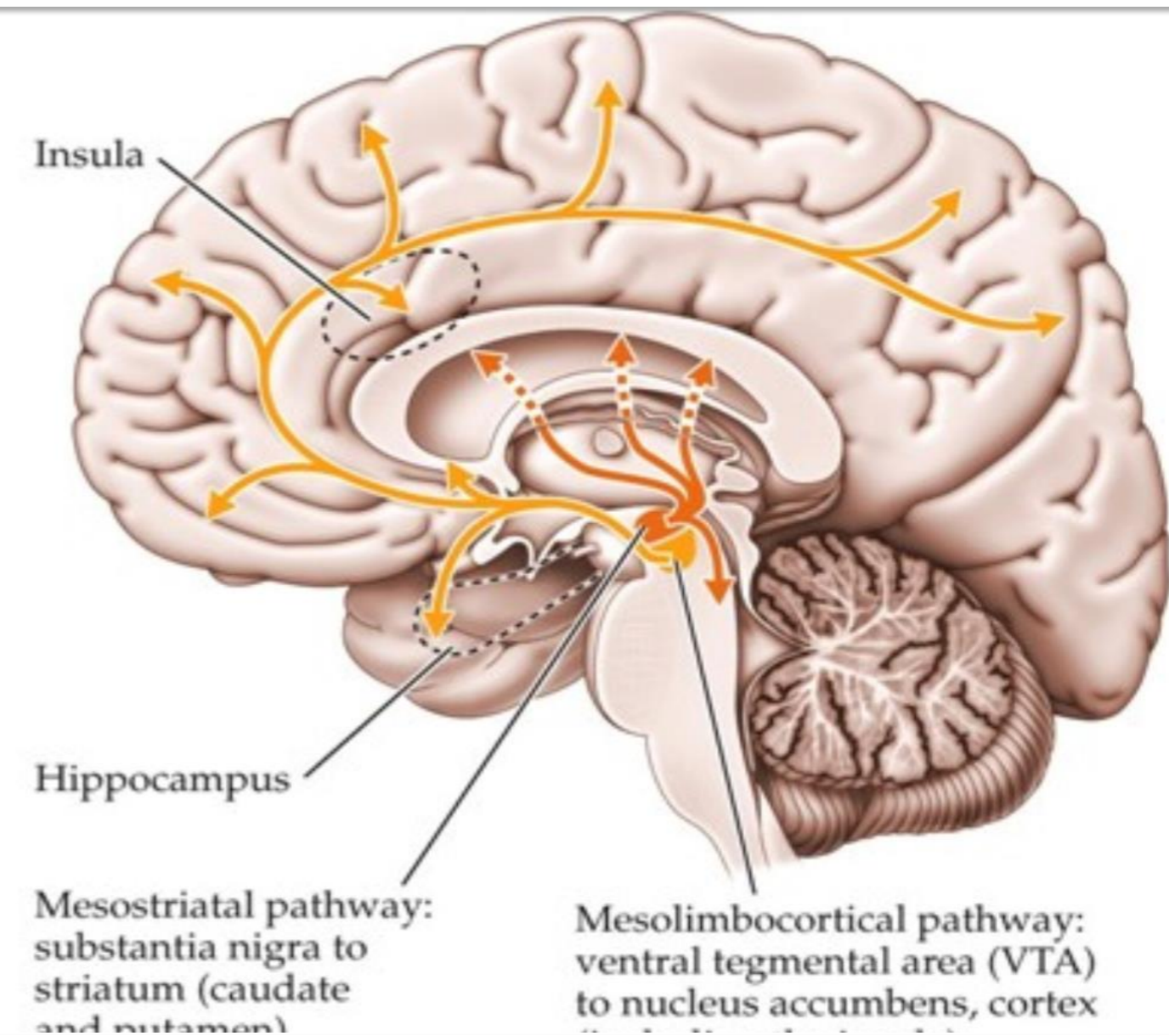
Review: Dopamine as 3rd factor broadcasts reward

Neuromodulator dopamine: - is nearly globally broadcasted
- signals reward minus expected reward

‘success signal’

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

Dopamine



Previous slide. Dopamine neurons send dopamine signals to many neurons and synapses in parallel in a broadcast like fashion.

Dopamine as Third factor

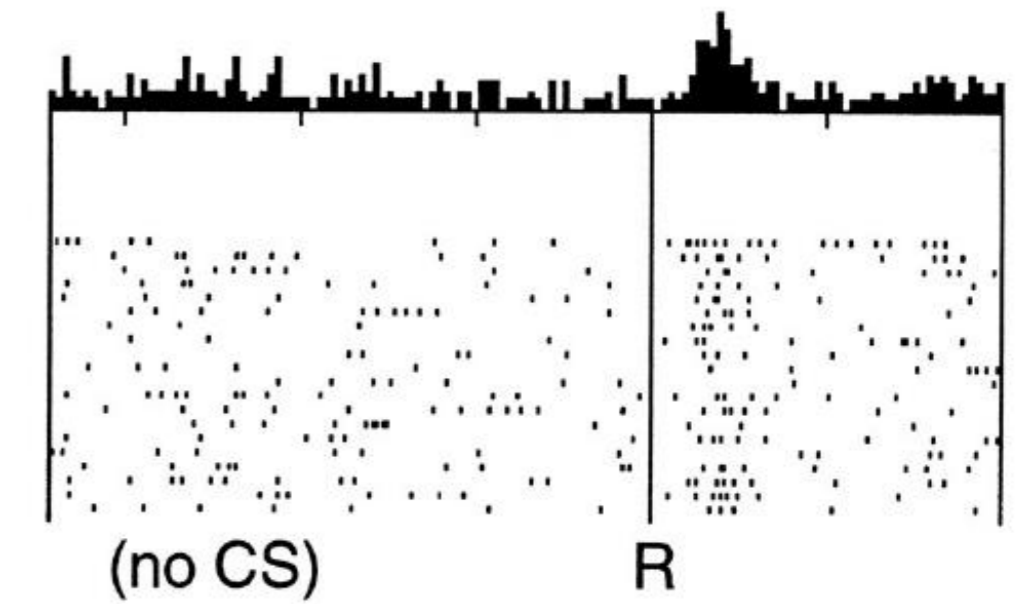
Conditioning:
red light \rightarrow 1s \rightarrow reward



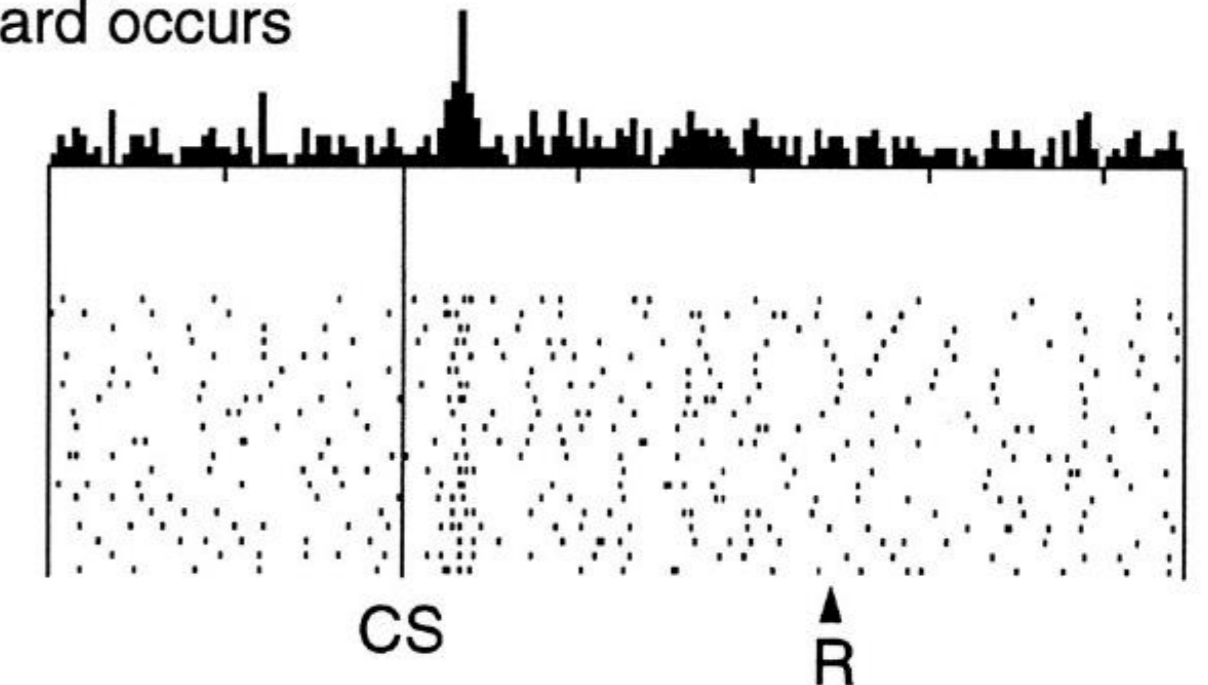
CS:
Conditioning
Stimulus

Sutton book, reprinted from W. Schultz

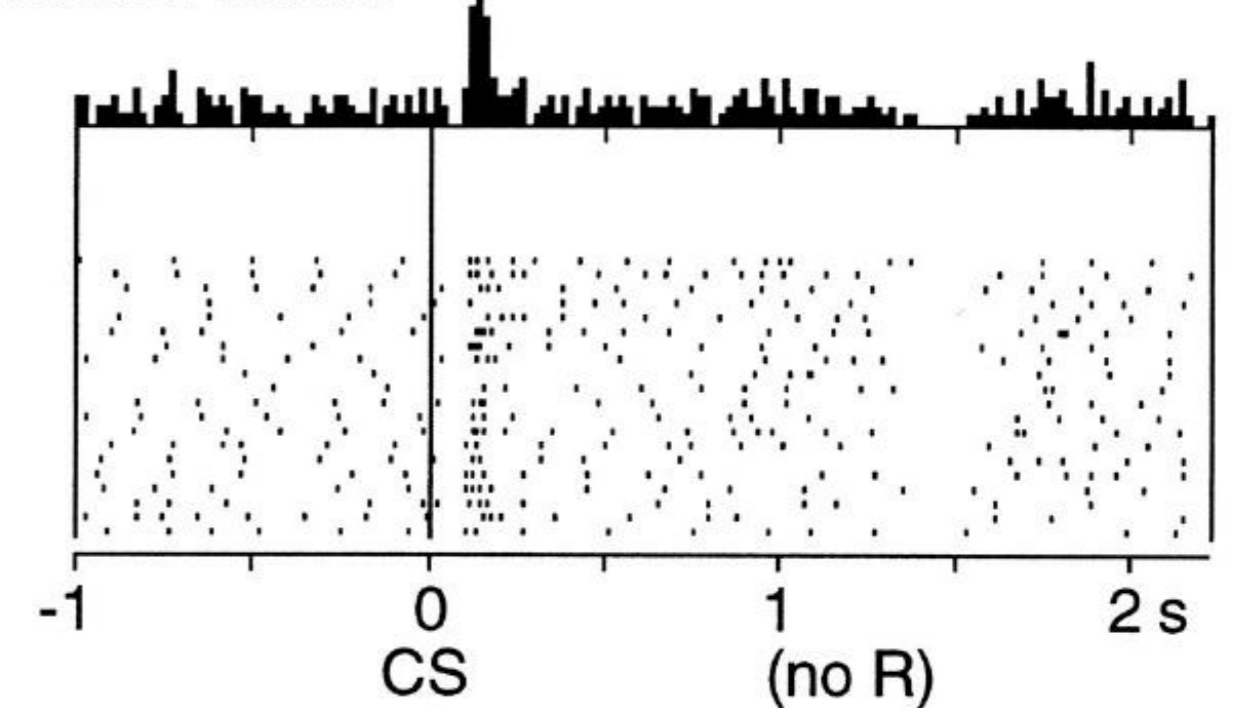
No prediction
Reward occurs



Reward predicted
Reward occurs



Reward predicted
No reward occurs



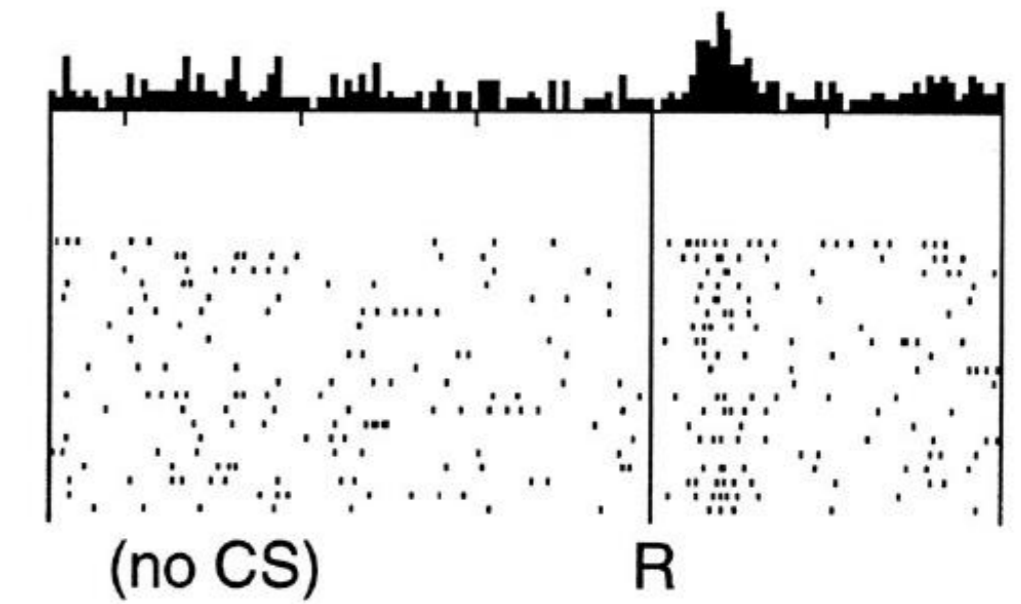
5. Dopamine as Third factor

This is now the famous experiment of W. Schultz.

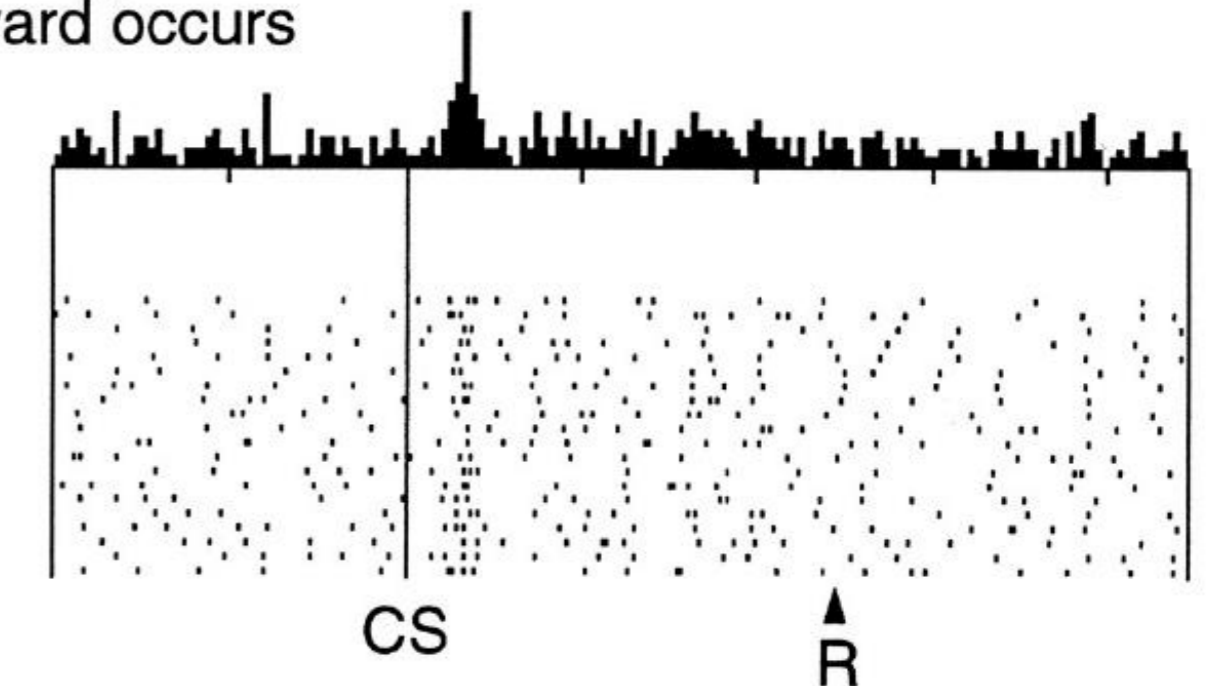
In reality the CS was not a red light, but that does not matter

Figure 15.3: The response of dopamine neurons drops below baseline shortly after the time when an expected reward fails to occur. Top: dopamine neurons are activated by the unpredicted delivery of a drop of apple juice. Middle: dopamine neurons respond to a conditioned stimulus (CS) that predicts reward and do not respond to the reward itself. Bottom: when the reward predicted by the CS fails to occur, the activity of dopamine neurons drops below baseline shortly after the time the reward is expected to occur. At the top of each of these panels is shown the average number of action potentials produced by monitored dopamine neurons within small time intervals around the indicated times. The raster plots below show the activity patterns of the individual dopamine neurons that were monitored; each dot represents an action potential. From Schultz, Dayan, and Montague, *A Neural Substrate of Prediction and Reward*, *Science*, vol. 275, issue 5306, pages 1593-1598, March 14, 1997. Reprinted with permission from AAAS.

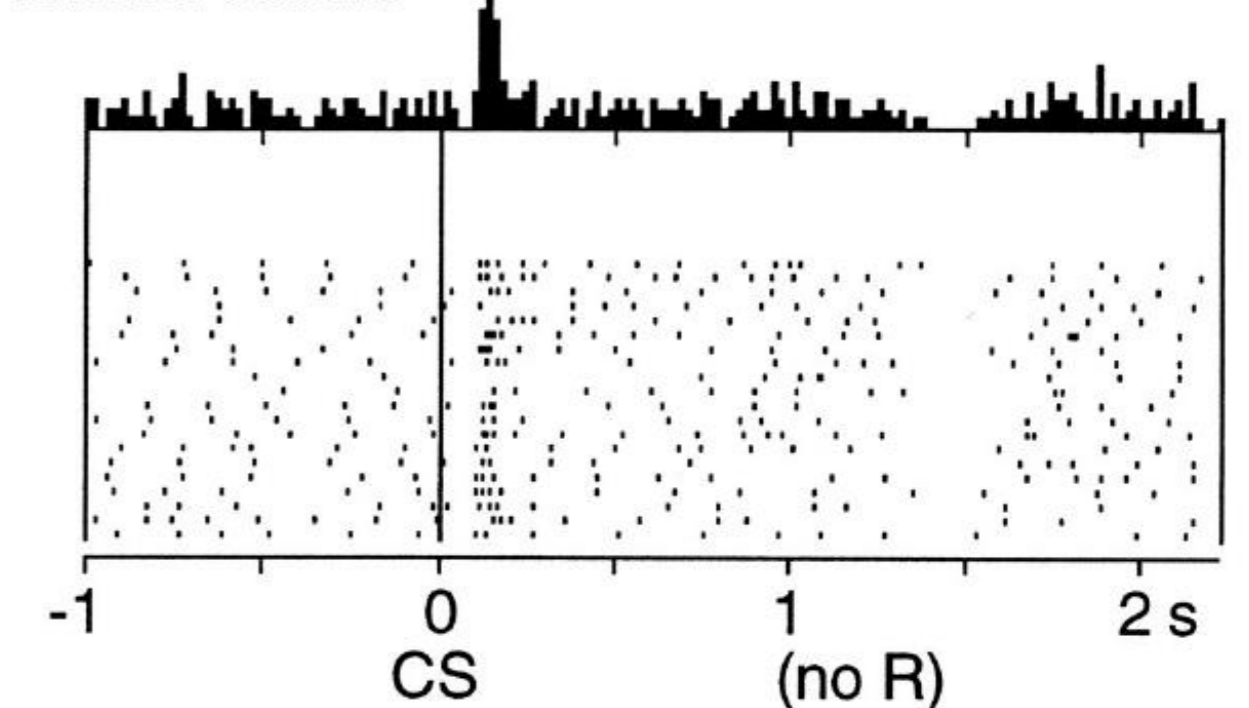
No prediction
Reward occurs



Reward predicted
Reward occurs



Reward predicted
No reward occurs



Summary: Dopamine as Third factor

- Dopamine signals 'reward minus expected reward'
- Dopamine signals an 'event that predicts a reward'
- Dopamine signals approximately the TD-error

$$DA(t) = [r(t) - \underbrace{(V(s) - \gamma V(s'))}_{\text{TD-delta}}]$$

Previous slide.

The paper of W. Schultz has related the dopamine signal to some basic aspects of Temporal difference Learning. The Dopamine signal is similar to the TD error.

Eligibility Traces with TD in Actor-Critic

Idea:

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

$$z_{ik} \leftarrow z_{ik} \lambda \quad \text{decay of **all** traces}$$

$$z_{ik} \leftarrow z_{ik} + \frac{d}{dw_{ik}} \ln[\pi(a|s, w_{ik})] \quad \text{update of **all** traces}$$

- update **all** parameters:

$$\Delta w_{ik} = \eta \underbrace{[r - (V(s) - \gamma V(s'))]}_{\text{TD-delta}} z_{ik}$$

→ policy gradient with eligibility trace and TD error

Previous slide.

Review of algorithm with actor-critic architecture and policy gradient with eligibility traces and TD.

Summary: Eligibility Traces with TD in Actor-Critic

Three-factor rules:

Presynaptic and postsynaptic factor 'select' the synapse.

→ a small subset of synapses becomes 'eligible' for change.

The 'Third factor' is a nearly global broadcast signal

→ potentially received by all synapses.

Synapses need all three factors for change

The 'Third factor' can be the Dopamine-like TD signal

→ Need actor-critic architecture to calculate $\gamma V(s') - V(s)$

→ Dopamine signals $[r_t + \gamma V(s') - V(s)]$

Previous slide.

The three factor rule, dopamine, TD signals, value functions now all fit together.

Artificial Neural Networks

Deep Reinforcement Learning

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Part 6: Application: Learning to find a reward

1. Deep Q-learning
2. From Policy gradient to Deep RL
3. Actor-Critic
4. Eligibility traces for policy gradient and actor-critic
5. Application: real brains and real tasks
6. **Application: learning to find a reward**

Previous slide.

We said that the three factor rule, dopamine, TD signals, value functions now all fit together. Let's apply this to the problem of navigation.

Coarse Brain Anatomy: hippocampus

Hippocampus

- Sits below/part of temporal cortex
- Involved in memory
- Involved in spatial memory

Spatial memory:

knowing where you are,

knowing how to navigate in an environment

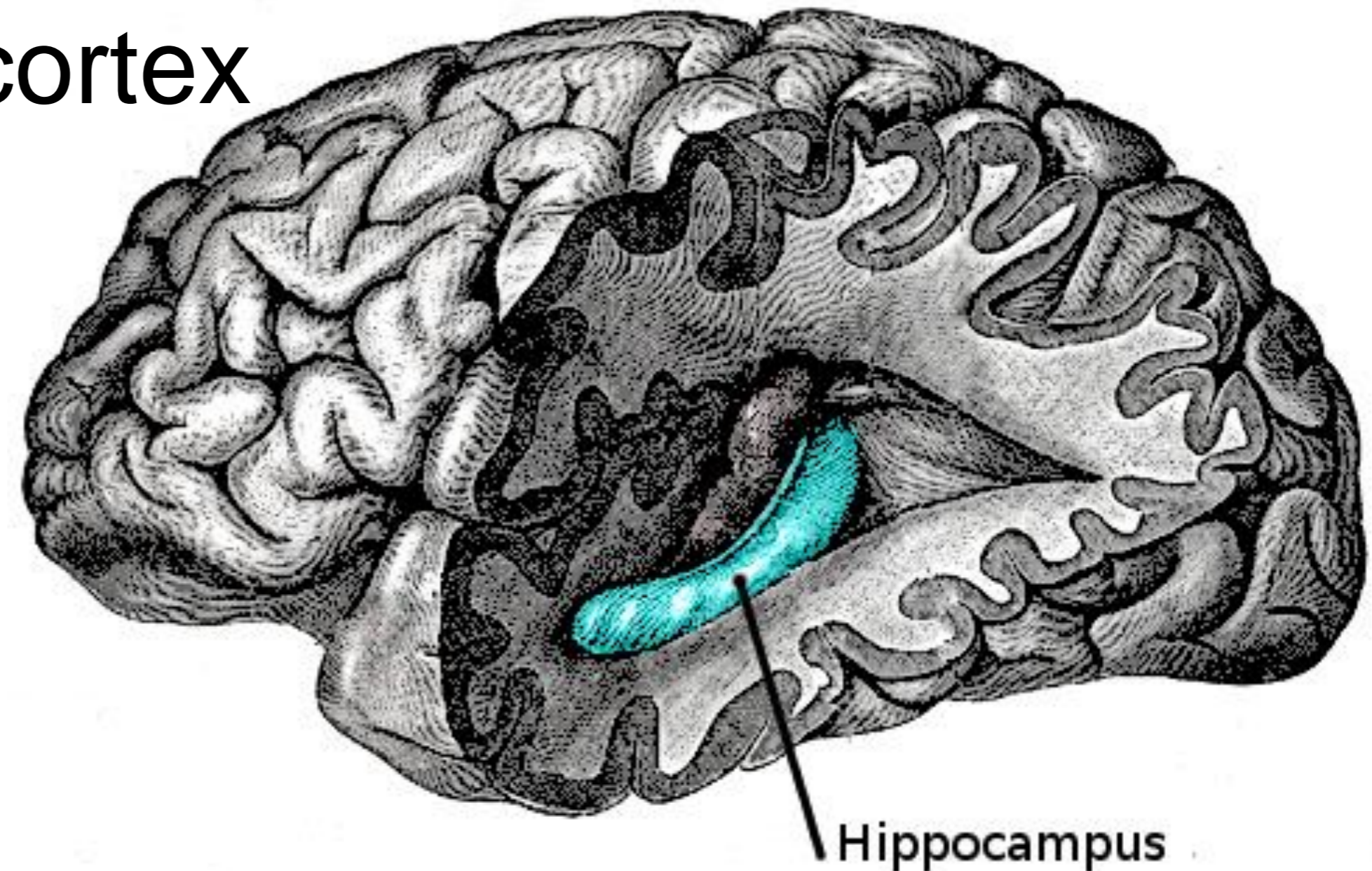


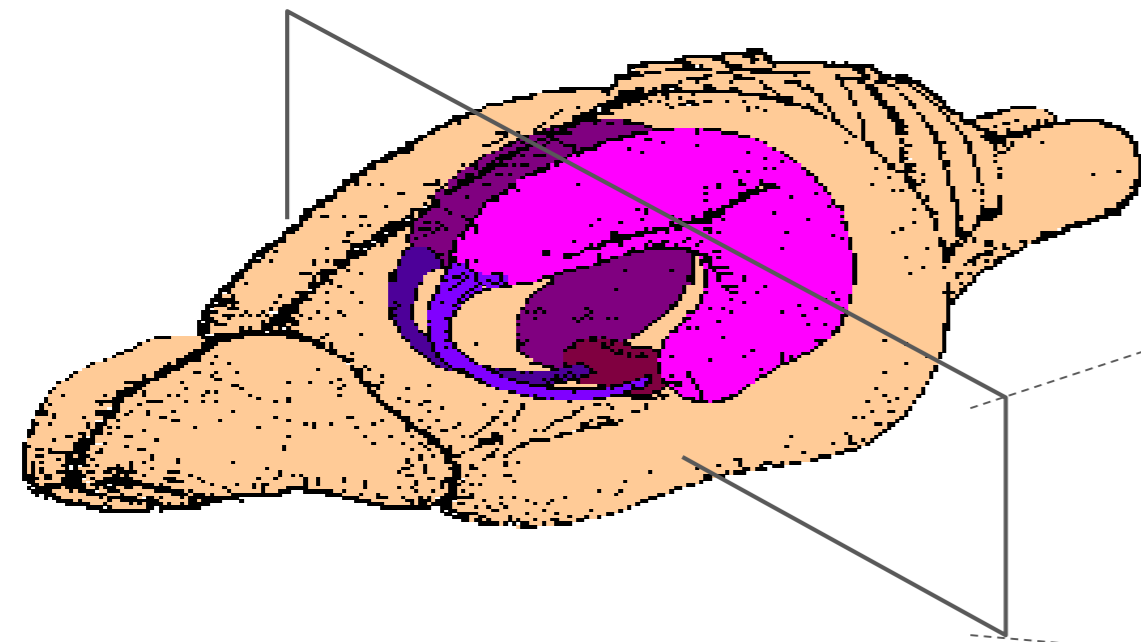
fig: Wikipedia

[Henry Gray](#) (1918) *Anatomy of the Human Body*

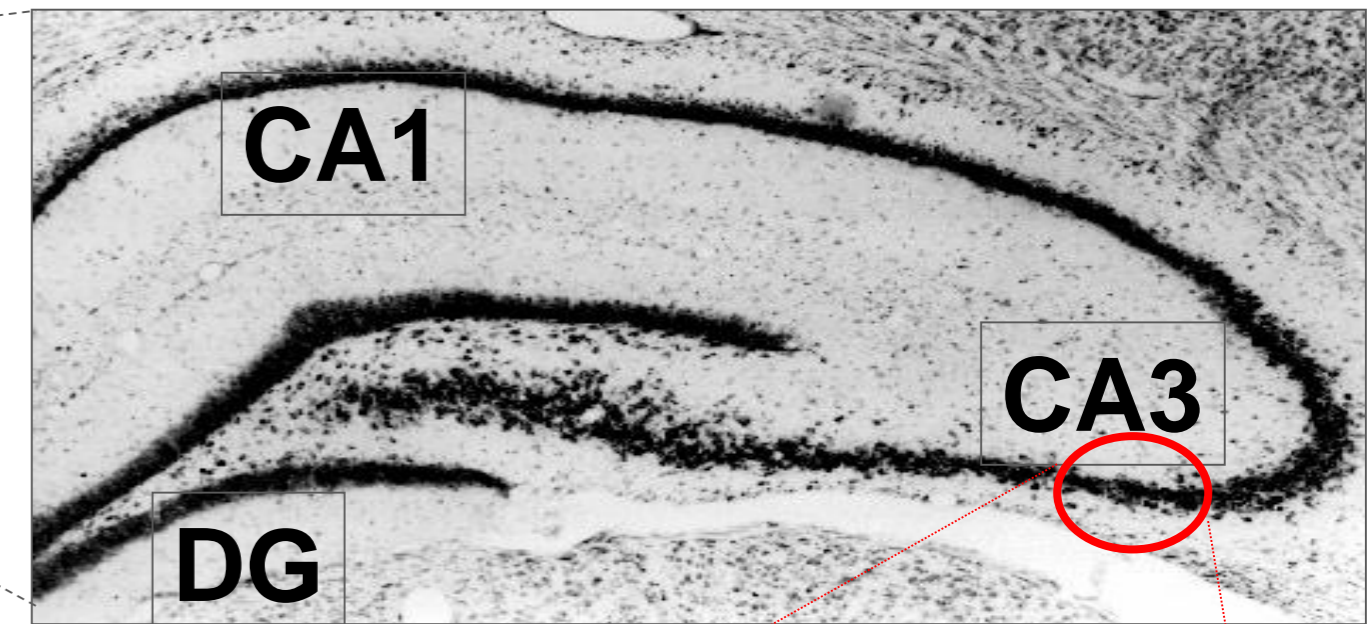
Previous slide.

the problem of navigation needs the spatical representation of the hippocampus.

Place cells in rat hippocampus

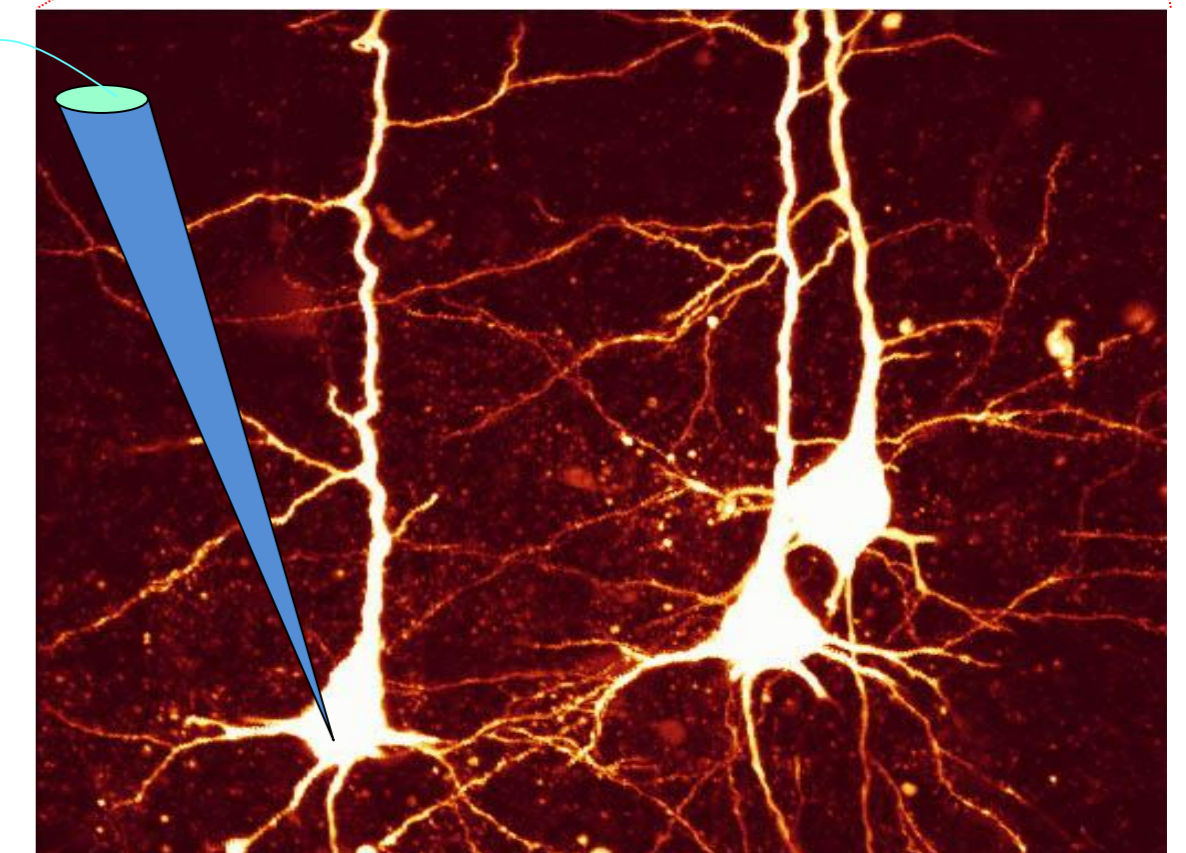
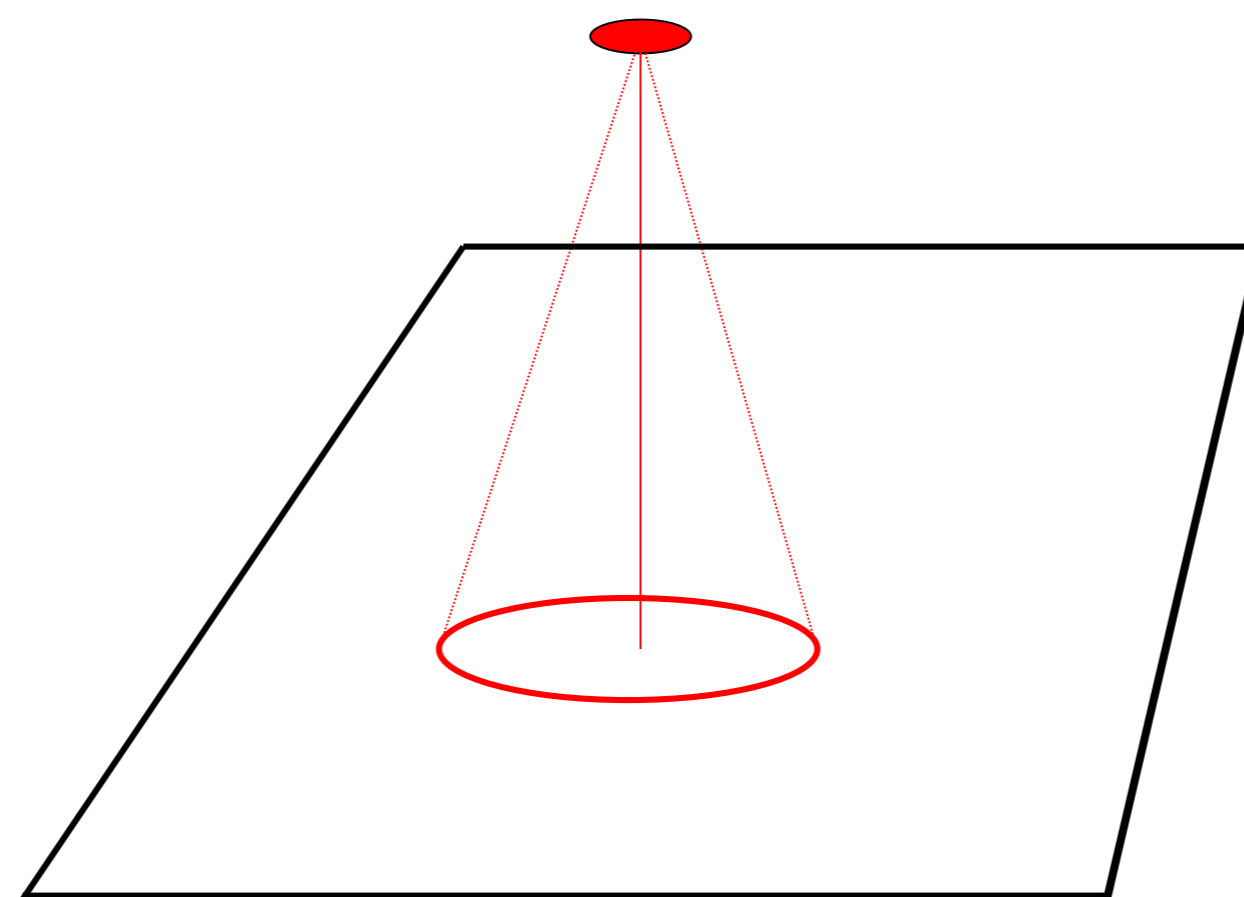
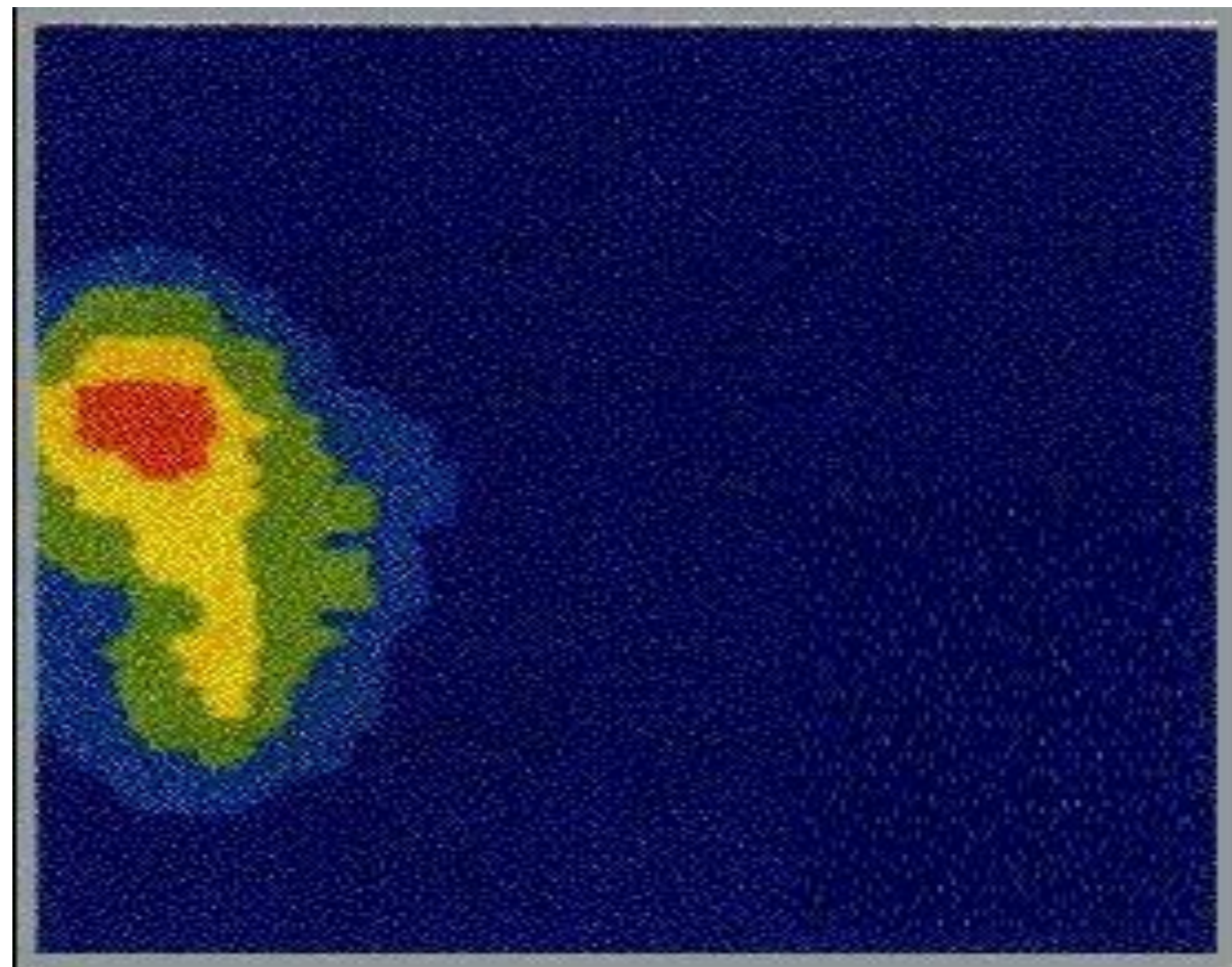


rat brain



electrode

Place fields



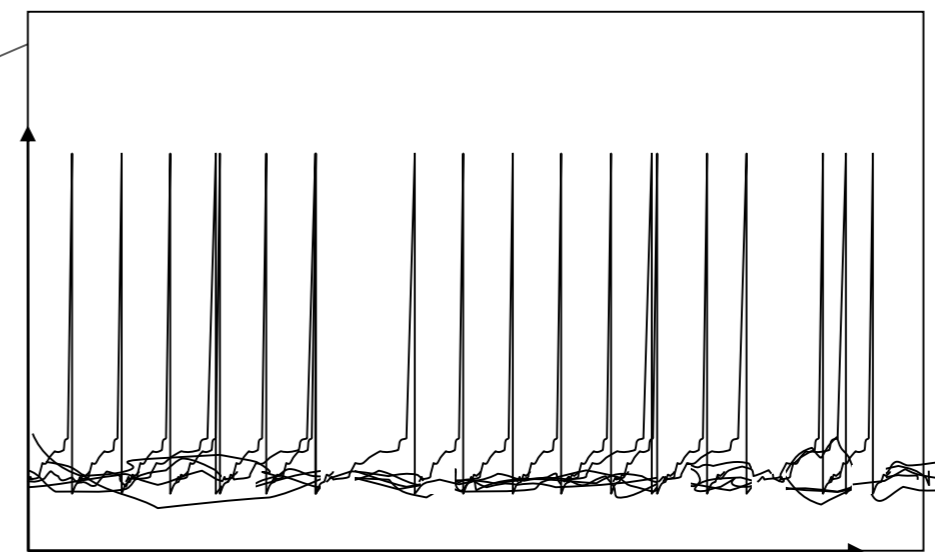
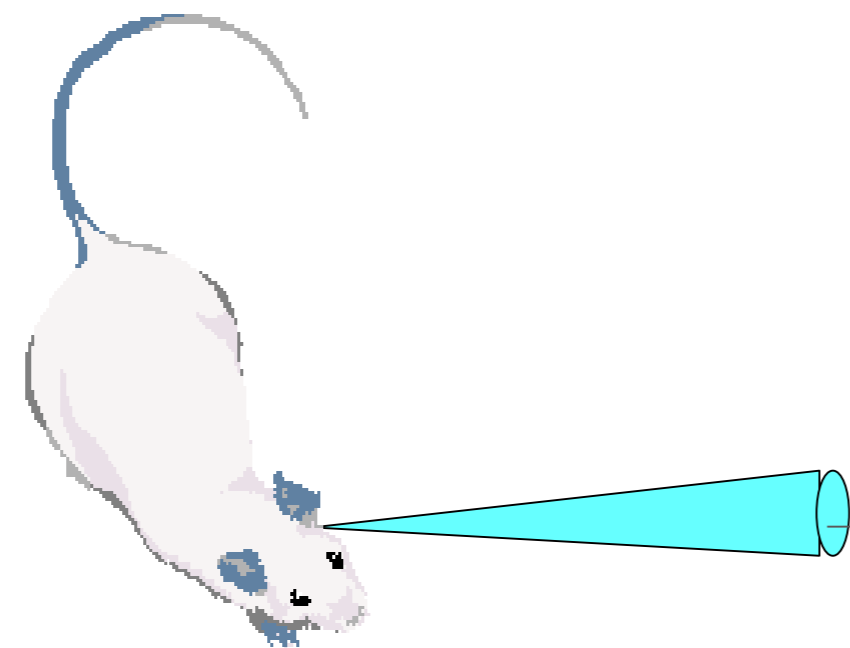
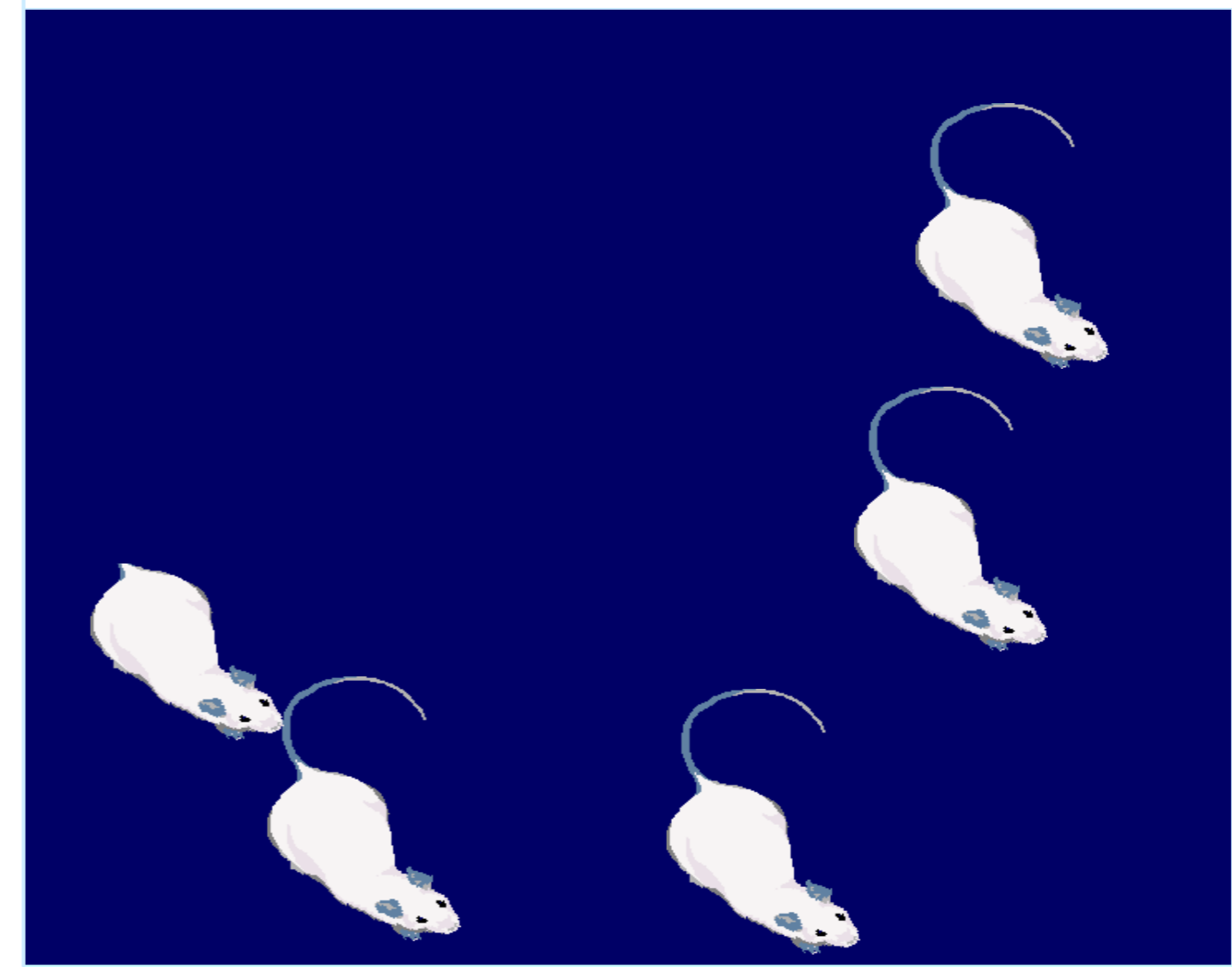
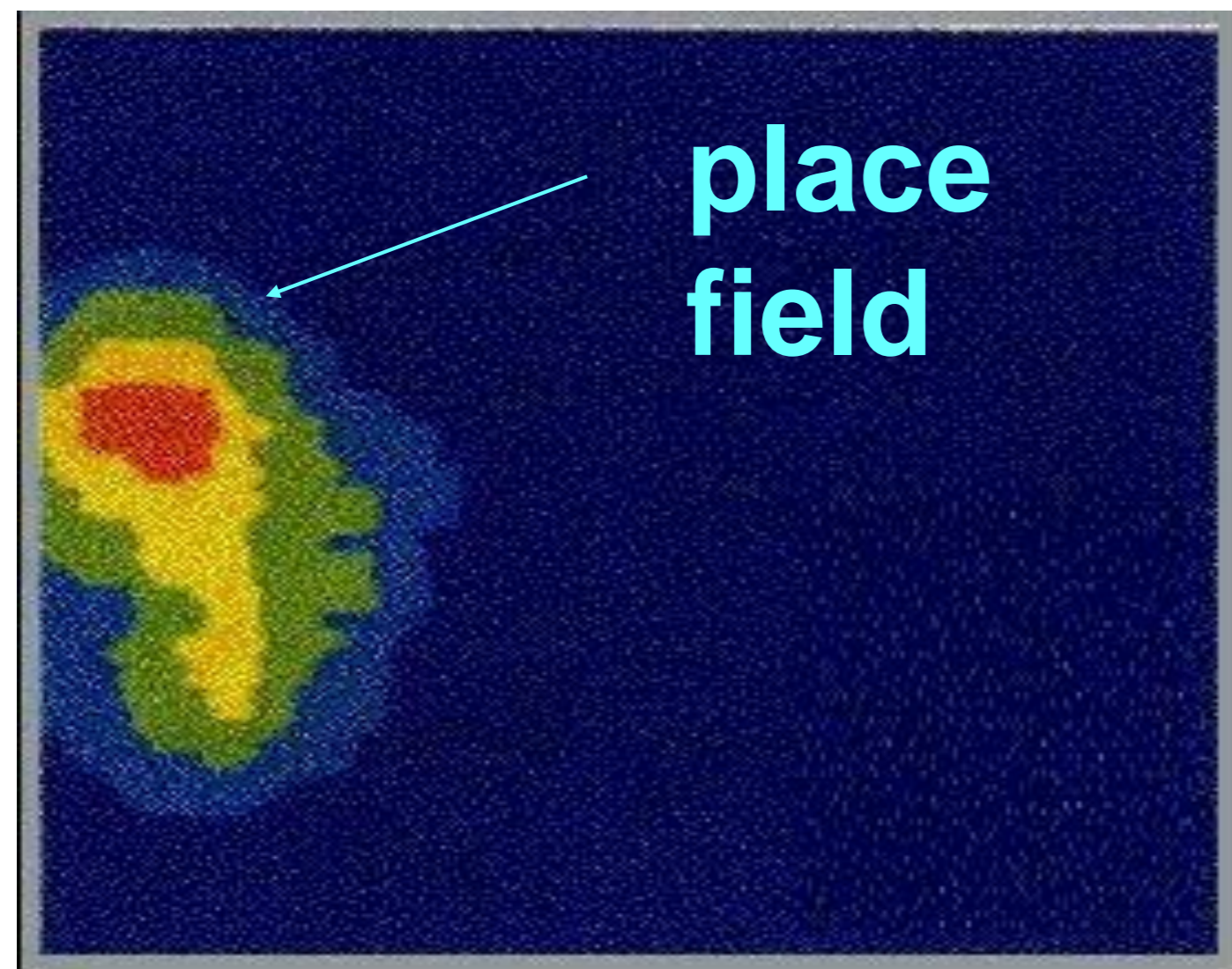
pyramidal cells

Previous slide.

the hippocampus of rodents (rats or mice) looks somewhat different to that of humans. Importantly, cells in hippocampus of rodents respond only in a small region of the environment. For this reason they are called place cells. The small region is called the place field of the cell.

Hippocampal place cells

Main property: encoding the animal's location



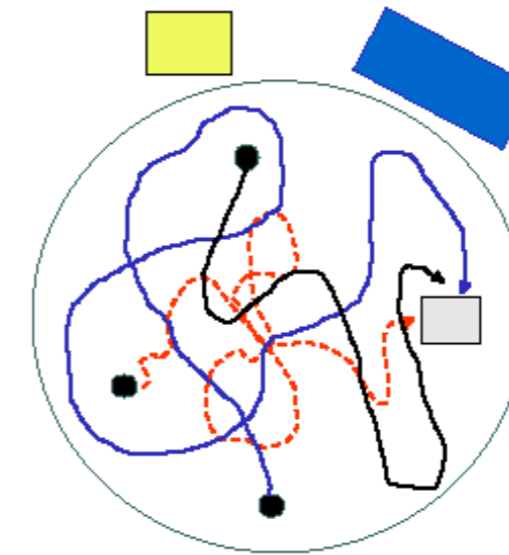
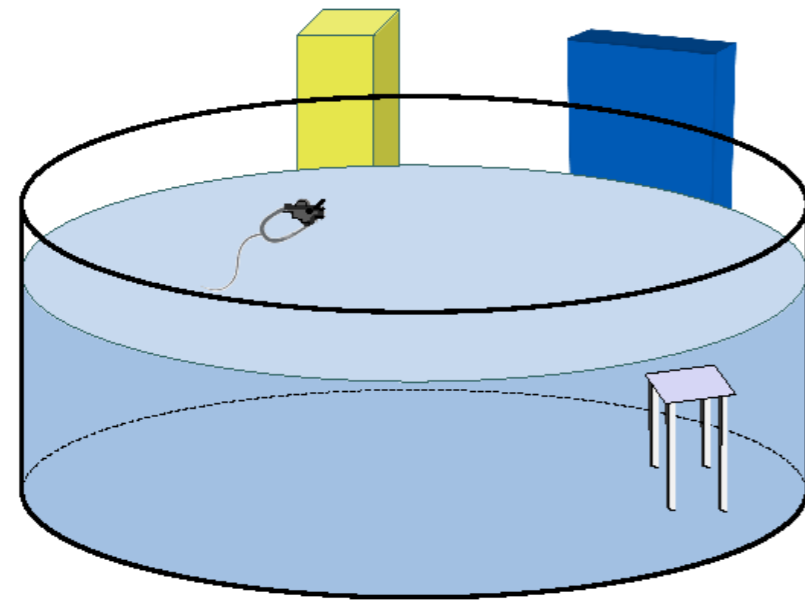
Previous slide.

Left: experimentally measured place field of a single cell in hippocampus.

Right: computer animation of place field

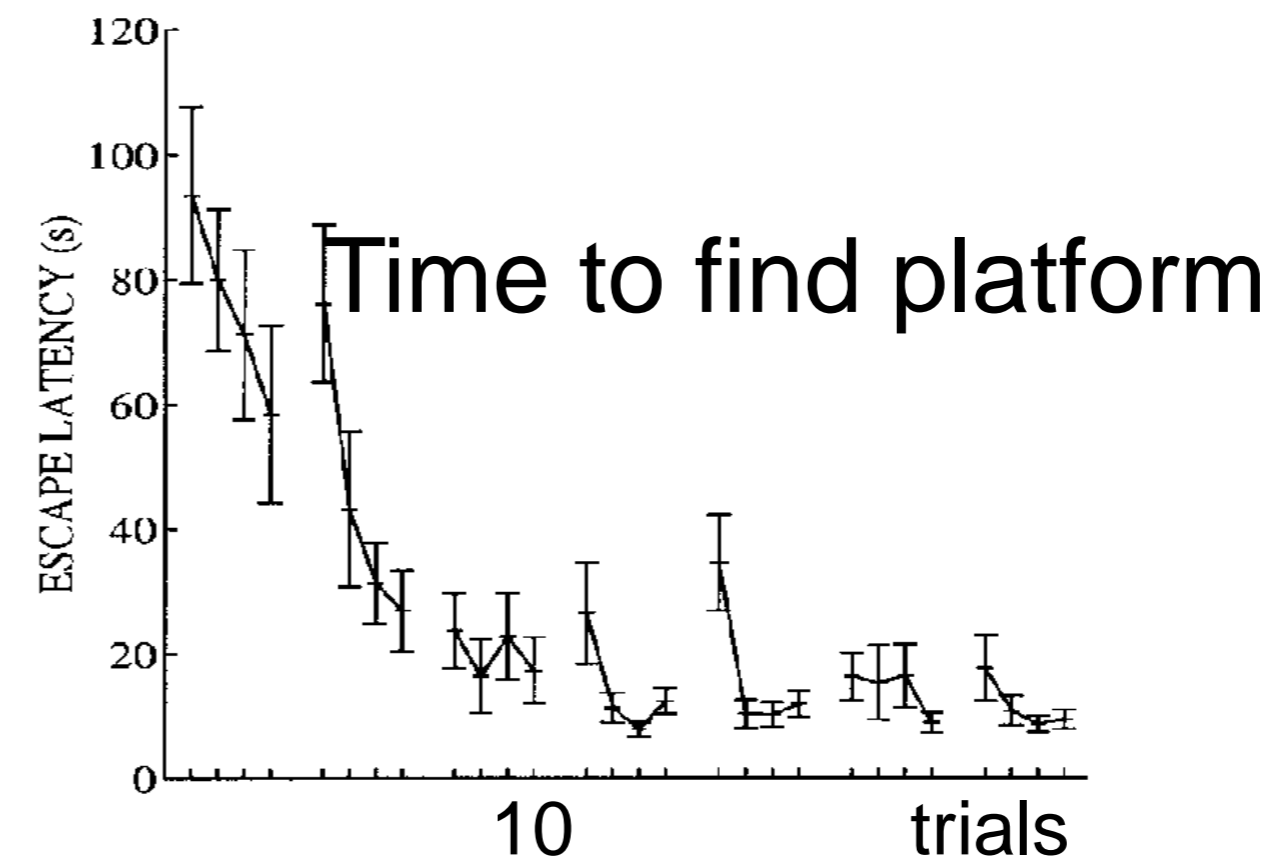
Review of Morris Water maze

Morris Water Maze



Rats learn to find the hidden platform

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



Foster, Morris, Dayan 2000

Previous slide.

Behavioral experiment in the Morris Water Maze.

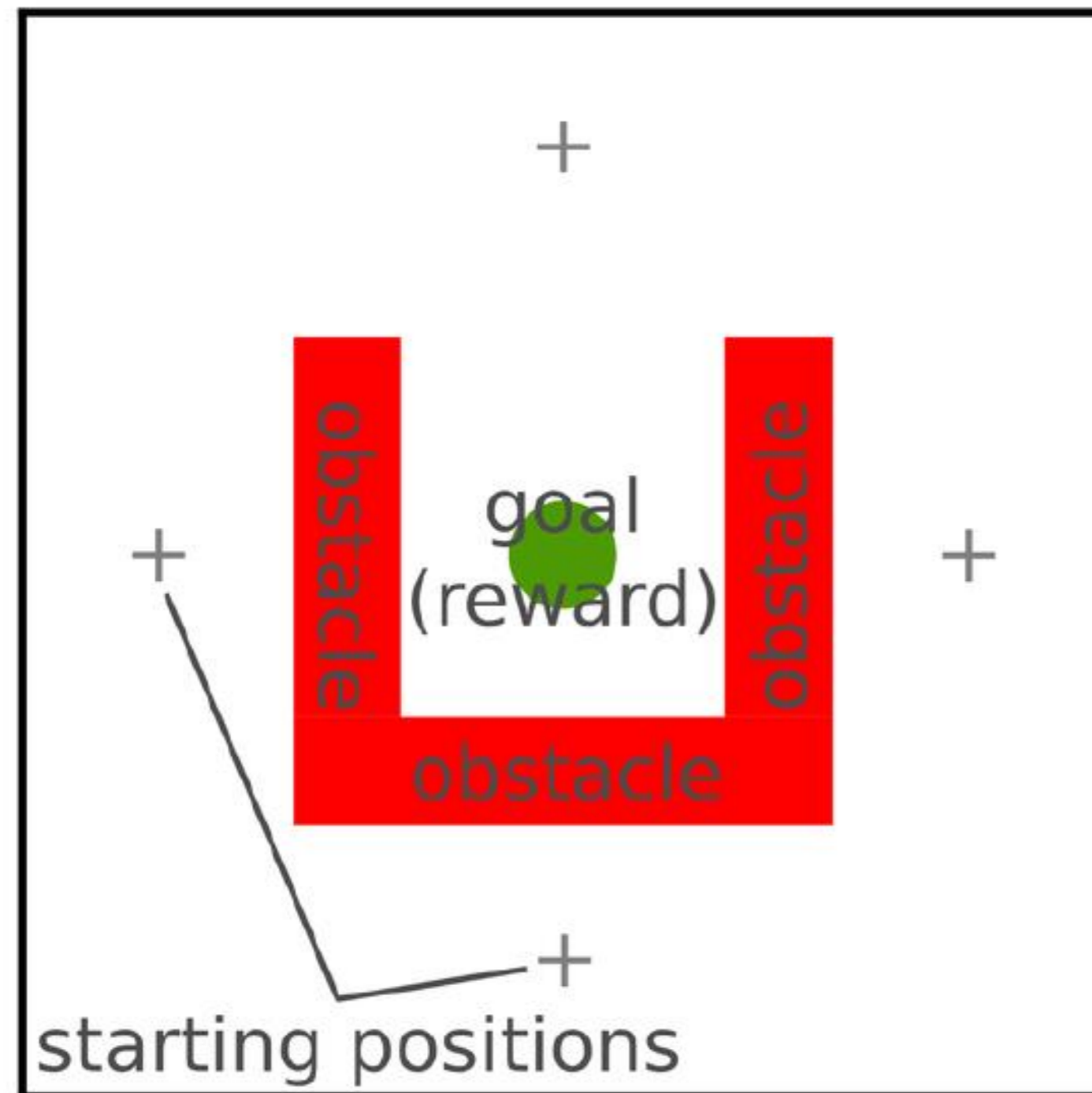
The water is milky so that the platform is not visible.

After a few trials the rat swims directly to the platform.

Platform is like a reward because the rat gets out of the cold water.

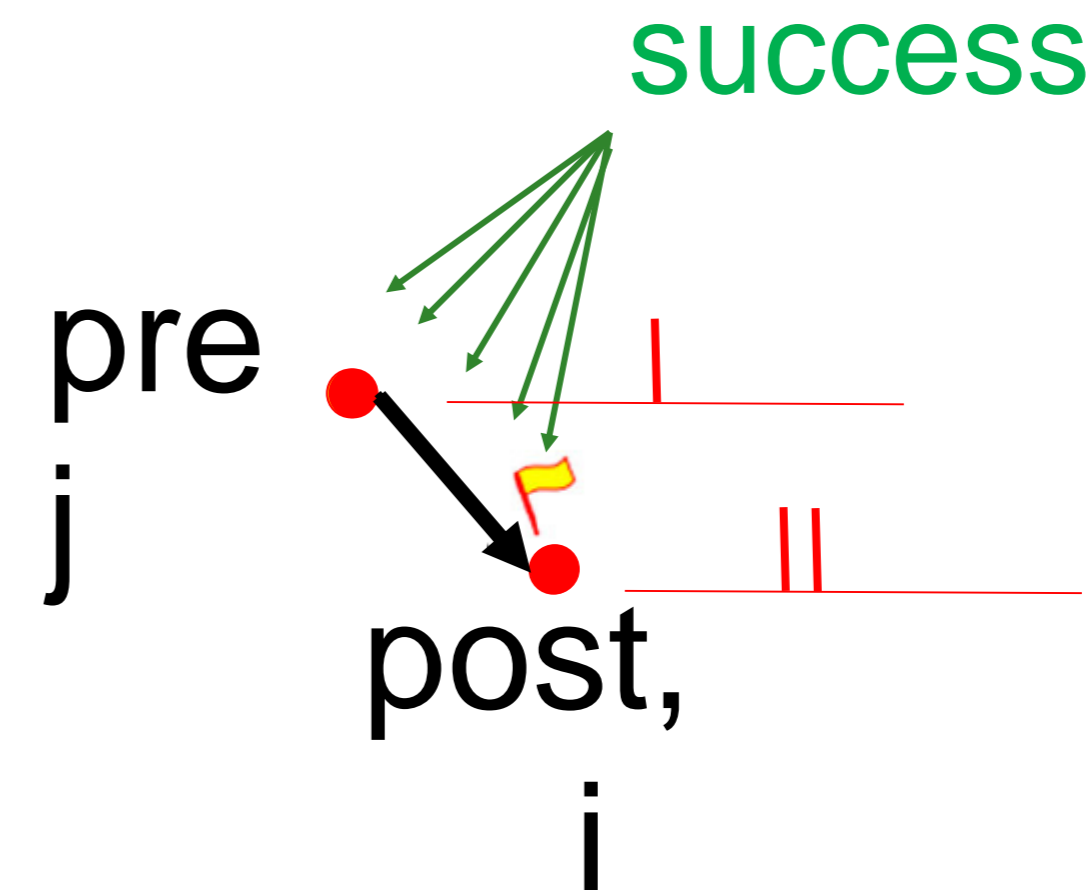
Modeling a Maze Task

Maze Task



3-factor rule

- joint activity sets eligibility trace
- success induces weight change



Previous slide.

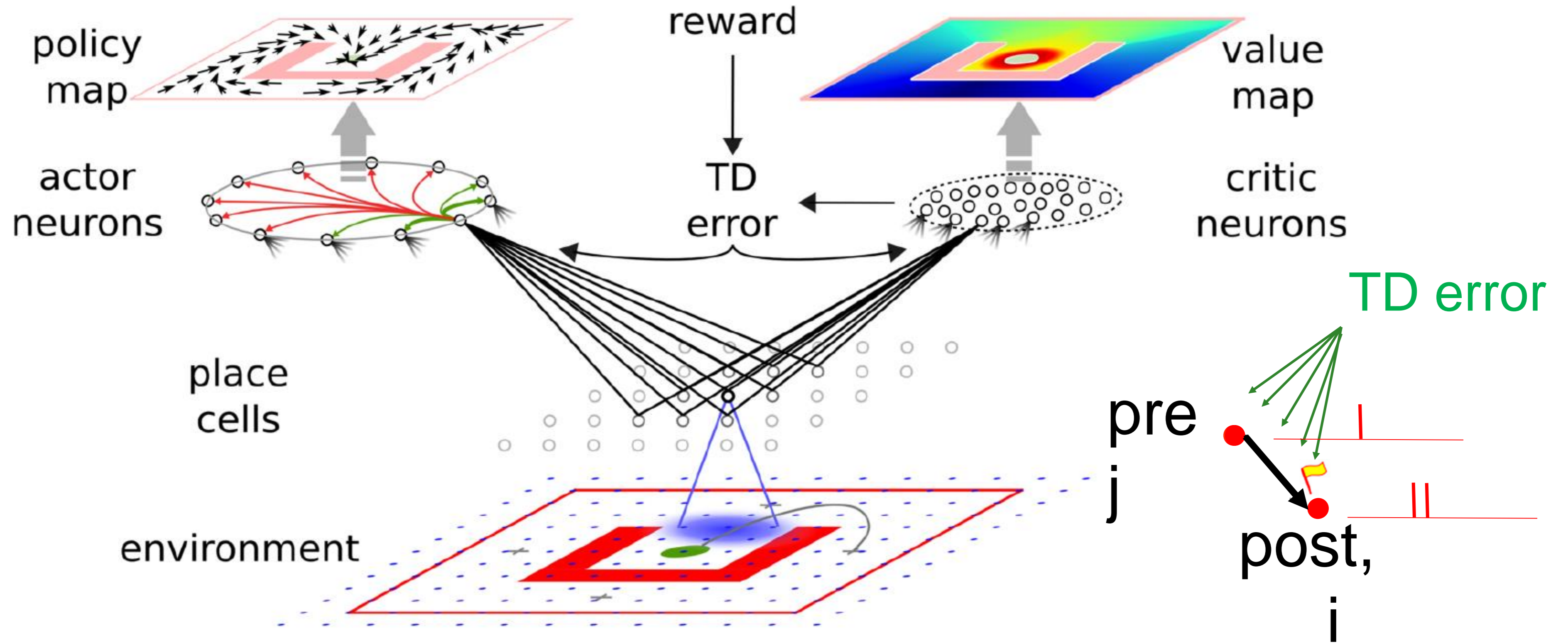
Left: The maze for the modeling work. The goal is hidden behind an obstacle. There are three different starting positions.

Right: Recall of the three-factor rule. The aim is to use learning with the three-factor rule to solve the maze task.

Maze Navigation with Actor-Critic

Continuous action space:
ring of stochastically spiking neurons

Value map:
Independent stoch. spiking neurons



Continuous state space:
Represented by spiking place cells

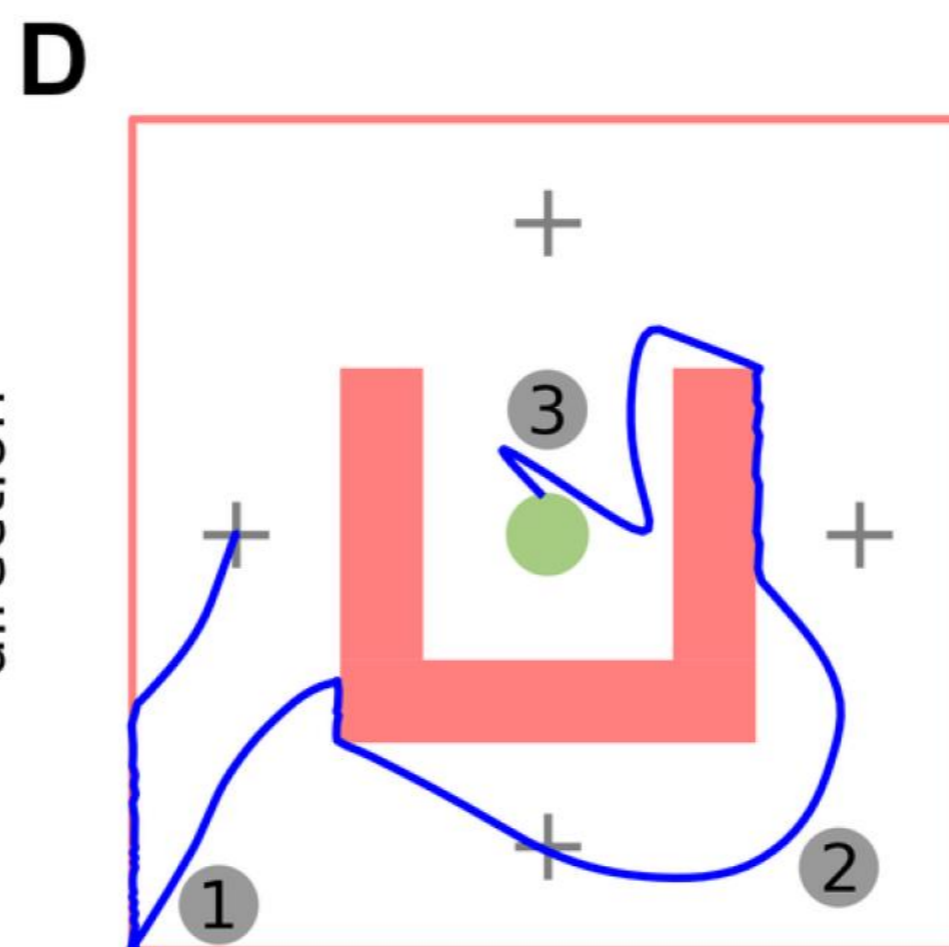
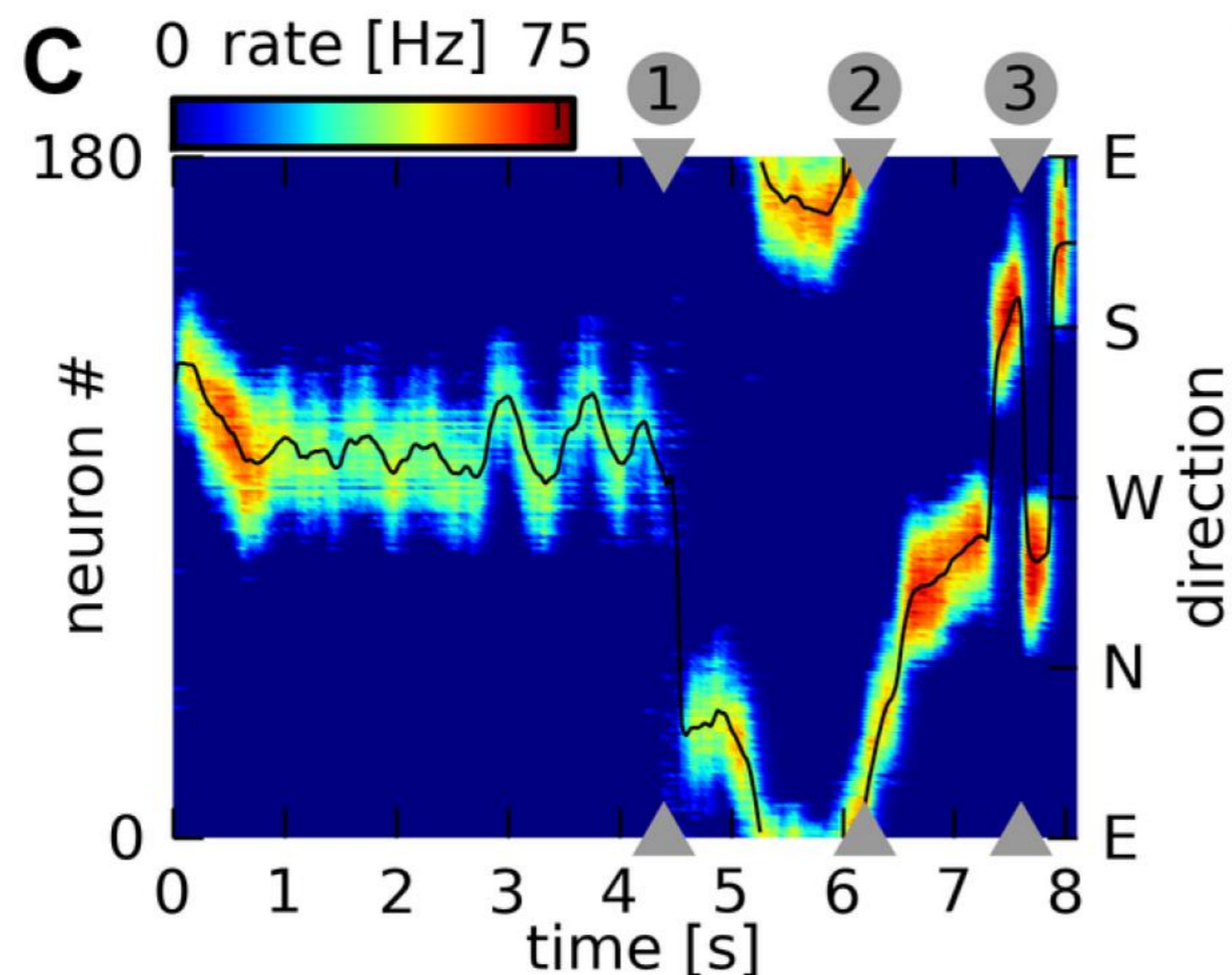
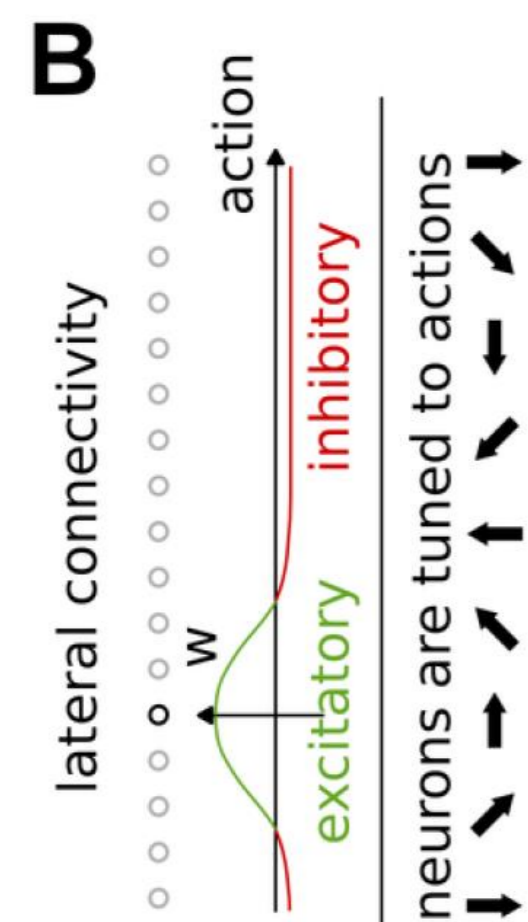
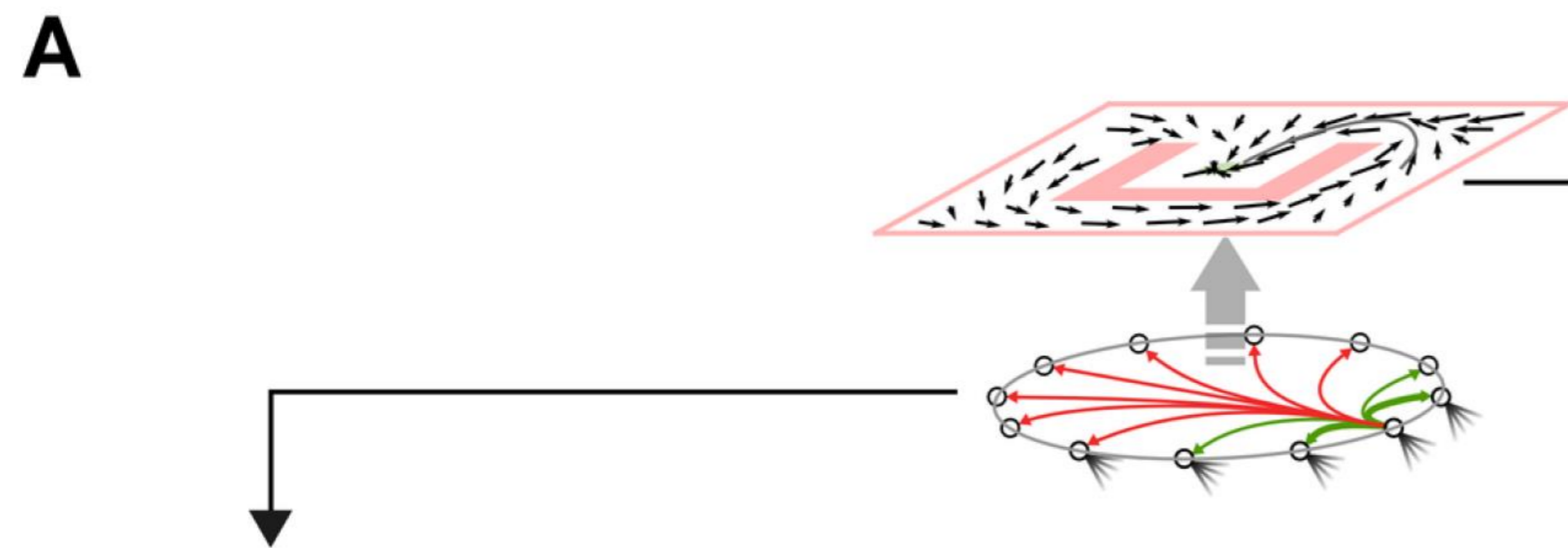
Figure 1. Navigation task and actor-critic network. From bottom to top: the simulated agent evolves in a maze environment, until it finds the reward area (green disk), avoiding obstacles (red). Place cells maintain a representation of the position of the agent through their tuning curves. Blue shadow: example tuning curve of one place cell (black); blue dots: tuning curves centers of other place cells. Right: a pool of critic neurons encode the expected future reward (value map, top right) at the agent's current position. The change in the predicted value is compared to the actual reward, leading to the temporal difference (TD) error. The TD error signal is broadcast to the synapses as part of the learning rule. Left: a ring of actor neurons with global inhibition and local excitation code for the direction taken by the agent. Their choices depending on the agent's position embody a policy map (top left).

doi:10.1371/journal.pcbi.1003024.g001

Ring of Actor neurons implements policy

Note: no need to formally define a softmax function

- Local excitation
- Long-range inhibition
- Not a formal softmax



Ring of actor neurons

Actor neurons (previous slide).

A: A ring of actor neurons with lateral connectivity (bottom, green: excitatory, red: inhibitory) embodies the agent's policy (top).

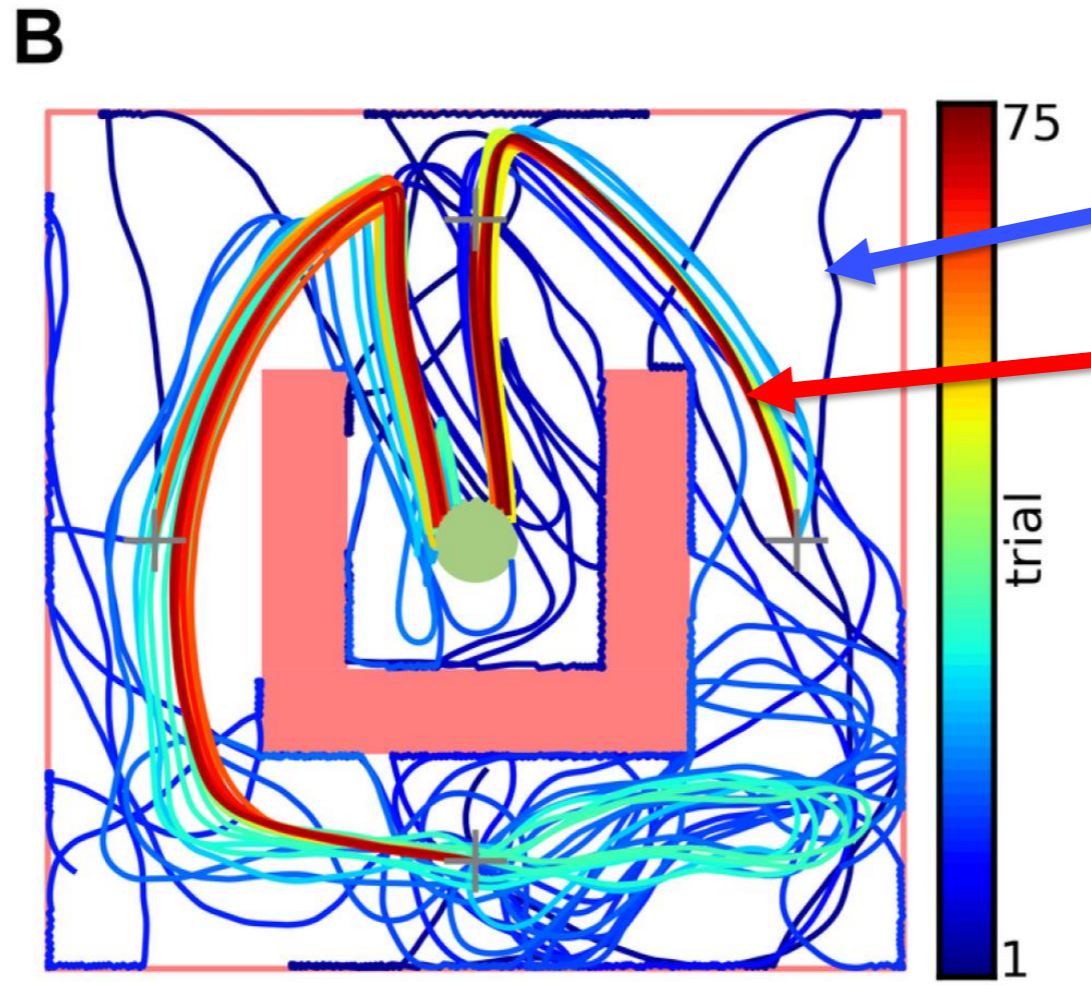
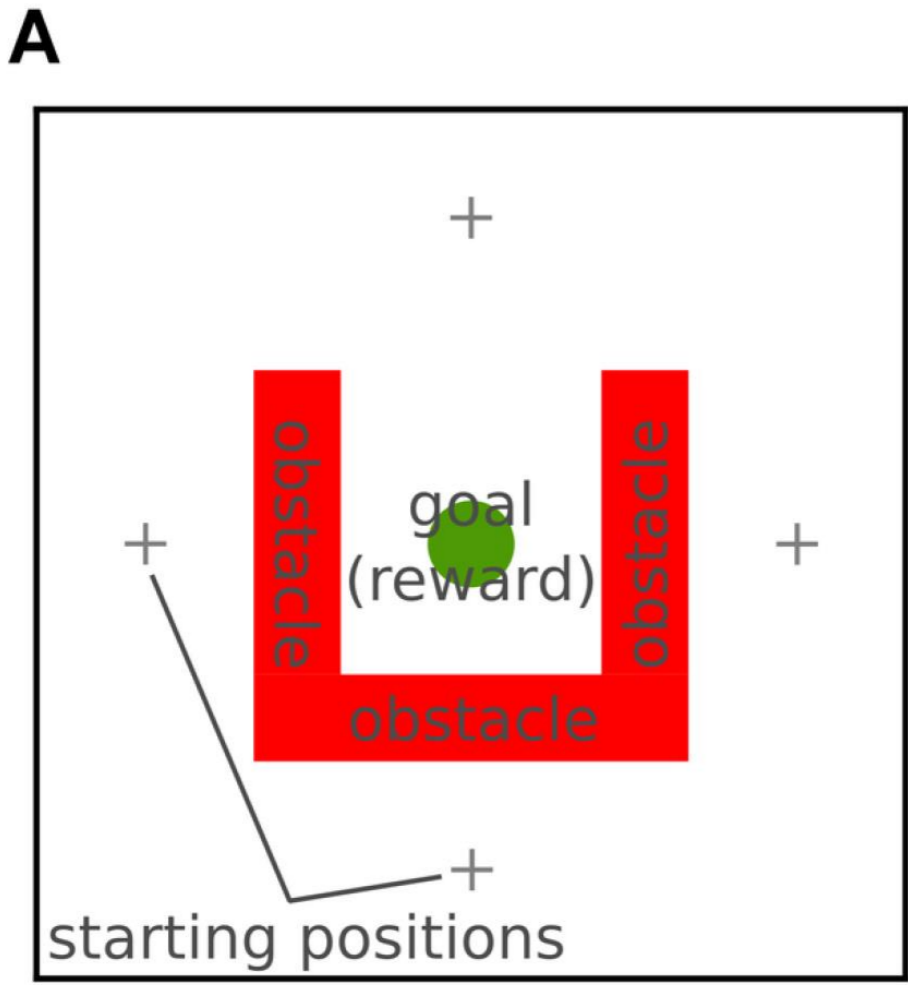
B: Lateral connectivity. Each neuron codes for a distinct motion direction.

Neurons form excitatory synapses to similarly tuned neurons and inhibitory synapses to other neurons.

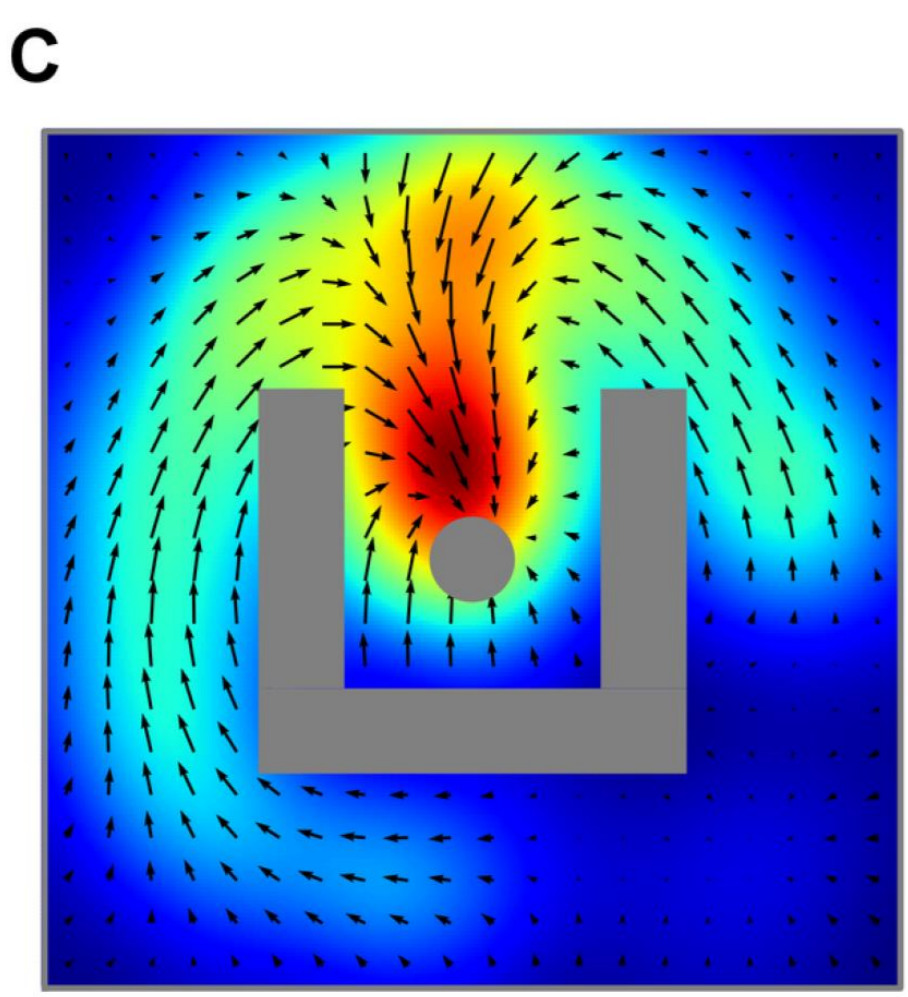
C: Activity of actor neurons during an example trial. The activity of the neurons (vertical axis) is shown as a color map against time (horizontal axis). The lateral connectivity ensures that there is a single bump of activity at every moment in time. The black line shows the direction of motion (right axis; arrows in panel B) chosen as a result of the neural activity.

D: Maze trajectory corresponding to the trial shown in C. The numbered position markers match the times marked in C.

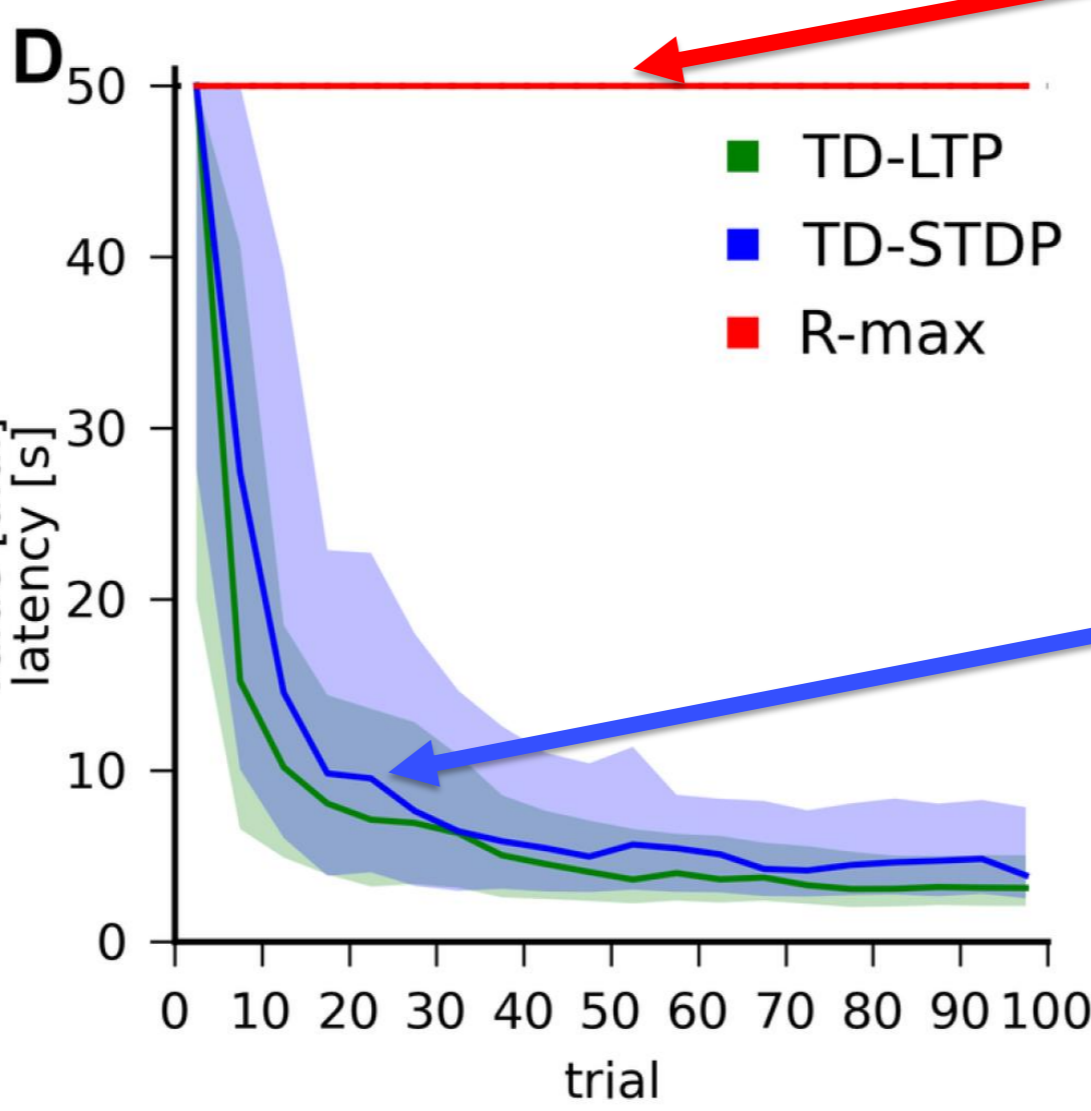
Maze Navigation: Actor-Critic with eligibility and spiking neurons



early trial
Late trial



value map



R-max:
Policy gradient without the critic. The goal was never found within 50s.

pre-post-TD (TD-STDP/TD-LTP)
After 25 trials, the goal was found within 20s.

Maze Navigation: Actor-Critic with spiking neurons

Maze navigation learning task.

A: The maze consists of a square enclosure, with a circular goal area (green) in the center. A U-shaped obstacle (red) makes the task harder by forcing turns on trajectories from three out of the four possible starting locations (crosses).

B: Color-coded trajectories of an example TD-LTP agent during the first 75 simulated trials. Early trials (blue) are spent exploring the maze and the obstacles, while later trials (green to red) exploit stereotypical behavior.

C: Value map (color map) and policy (vector field) represented by the synaptic weights of the agent of panel B after 2000s simulated seconds.

D: Goal reaching latency of agents using different learning rules. Latencies of $N \sim 100$ simulated agents per learning rule. The solid lines shows the median shaded area represents the 25th to 75th percentiles. The R-max agent were simulated without a critic and enters times-out after 50 seconds.

Actor-Critic with spiking neurons

- Learns in a few trials (assuming good representation)
- Works in continuous time.
- No artificial 'events' or 'time steps'
- Works with spiking neurons
- Works in continuous space and for continuous actions
- Uses a biologically plausible 3-factor learning rule
- Critic implements value function
- TD signal calculated by critic
- Actor neurons interact via synaptic connections
- No need for algorithmic 'softmax'

Previous slide.
Summary of findings

Summary

TD learning in an actor-critic framework maps to brain:

- Sensory representation: Cortex and Hippocampus
- Actor : Dorsal Striatum
- Critic : Ventral Striatum (nucleus accumbens)
- TD-signal: Dopamine
- 3-factor rule with delayed dopamine

Model shows:

- Learning in a few trials (not millions!) possible, if the sensory presentation is well adapted to the task
- asynchronous online, on-policy, 3-factor rule
- implementable in biology or neuromorphic hardware

Previous slide.
Summary of findings

Artificial Neural Networks

Deep Reinforcement Learning

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Part 7: Model-based versus model-free

1. Deep Q-learning
2. From Policy gradient to Deep RL
3. Actor-Critic
4. Eligibility traces for policy gradient and actor-critic
5. Application: animal tasks
6. Application: Learning to find a goal
7. **Model-based versus model-free**

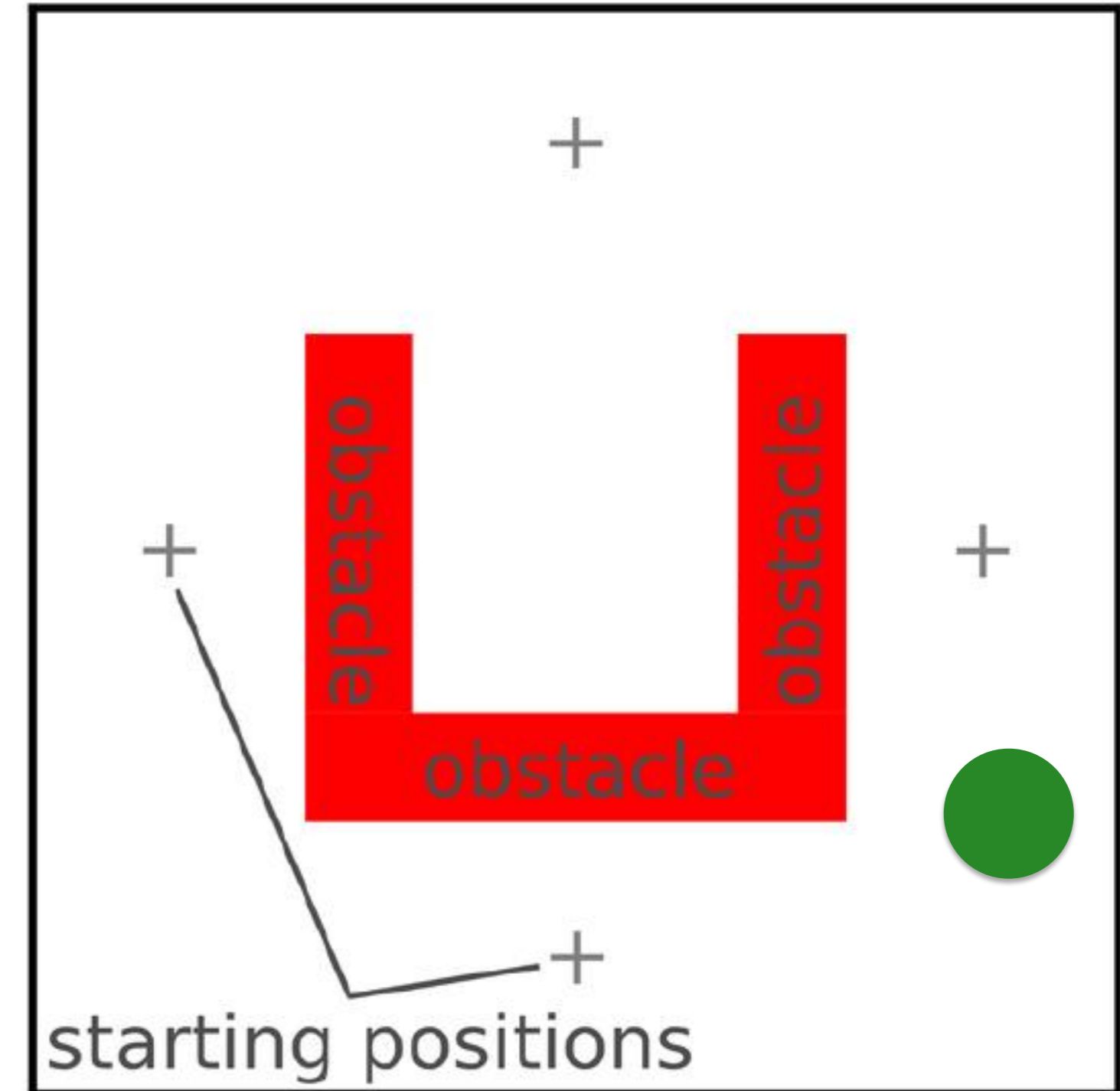
Previous slide.

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

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?

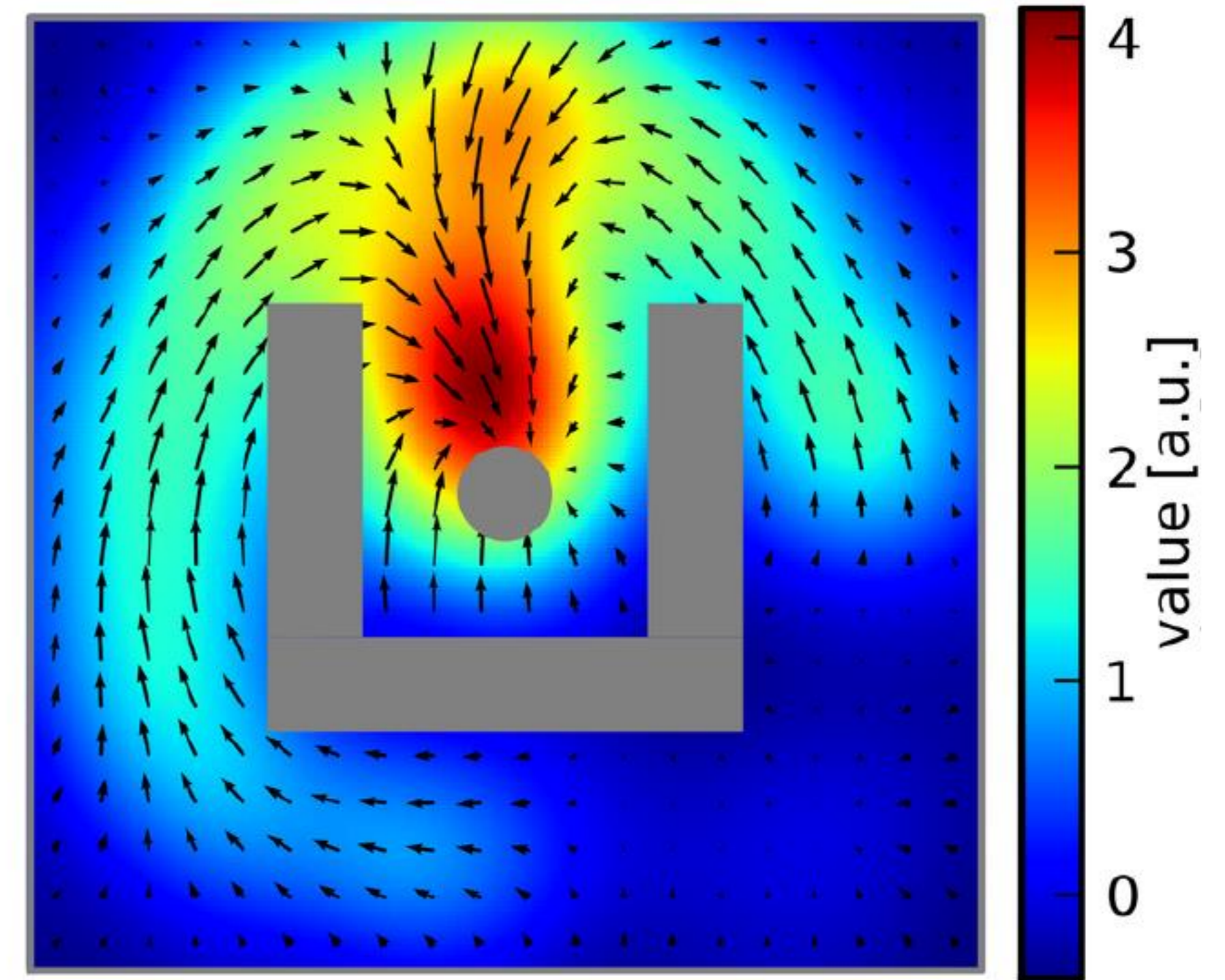
Imagine that the target location is shifted in the SAME environment.

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'.

Model-based versus Model-free Reinforcement Learning

Definition:

Reinforcement learning is **model-based**, if the agent also learns a **model of the environment**.

Examples: Model of the environment

- state s_1 is a neighbor of state s_{17} .
- if I take action a_5 in state s_7 , I will go to s_8 .
- The distance from s_5 to s_{15} is 10m.
- etc

Previous slide.

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

Model-based versus Model-free Q-learning

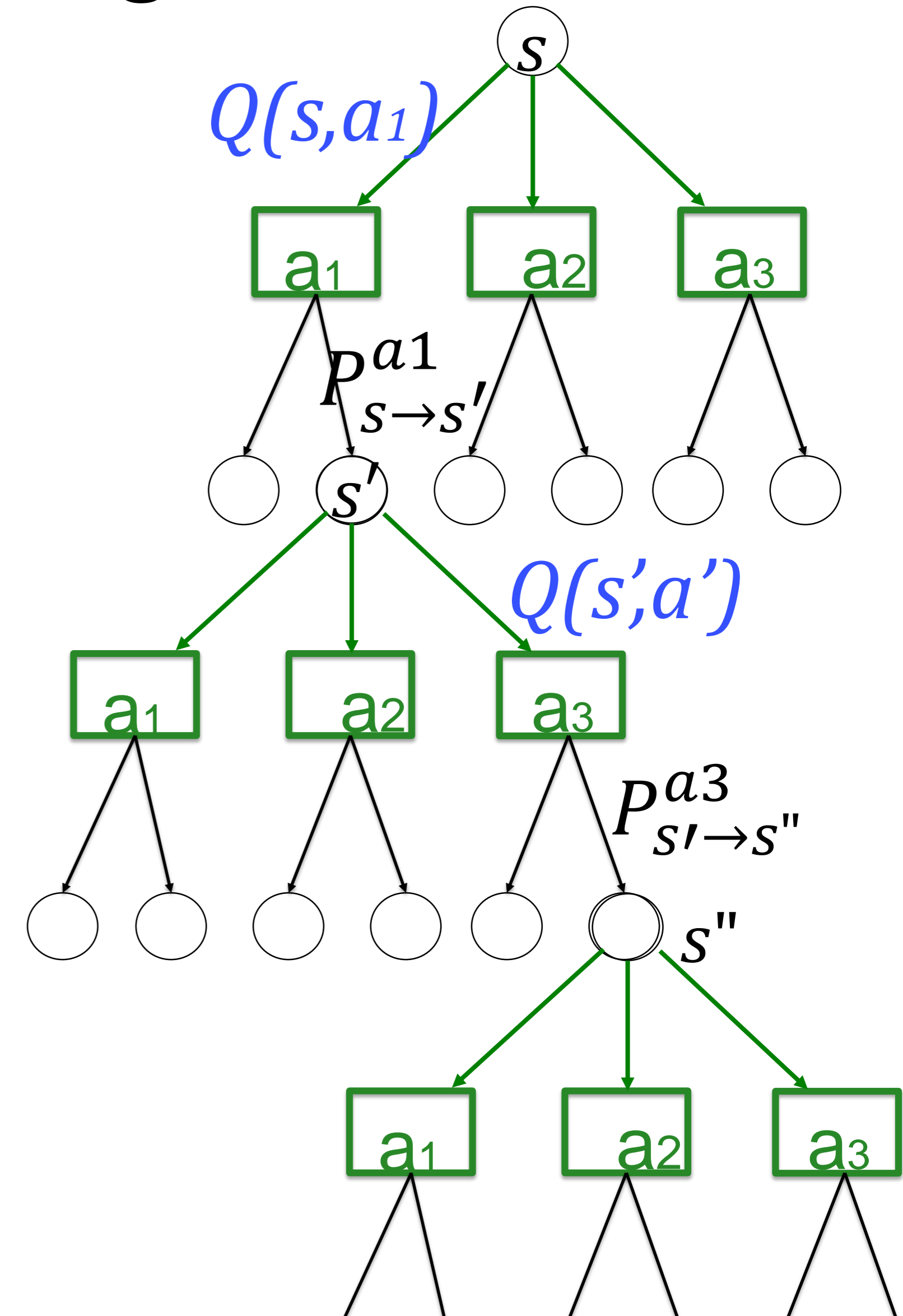
Model-free:

the agent learns directly and only the Q-values

Model-based:

the agent learns the Q-values and also the transition probabilities

$$P_{s \rightarrow s'}^{a1}$$



Previous slide.

Let us go back to our 'tree'. If the algorithm knows the transition probabilities, then this means that it is a model-based algorithm

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'.

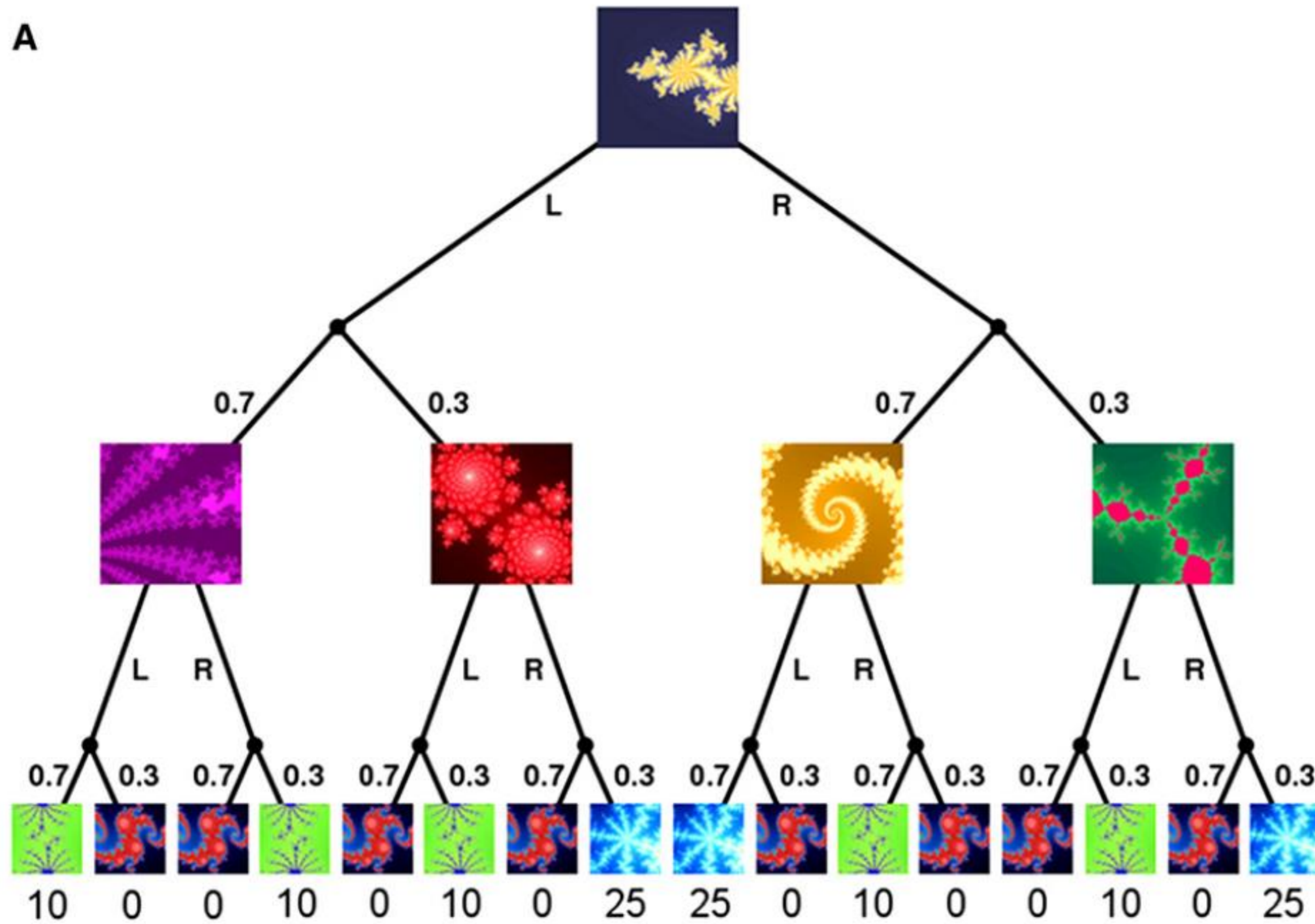
next slide.

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 for running things in the background is cheap.

Model-based learning

A



B Session 1: State Space Exposure (no choices)



Model based:
You know what state to expect given current state and action choice.
'state prediction'

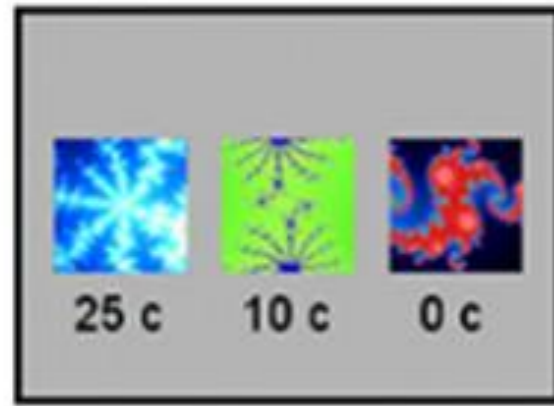
State and Reward Prediction Task (previous slide)

In order to check whether humans use model-based or model-free RL:

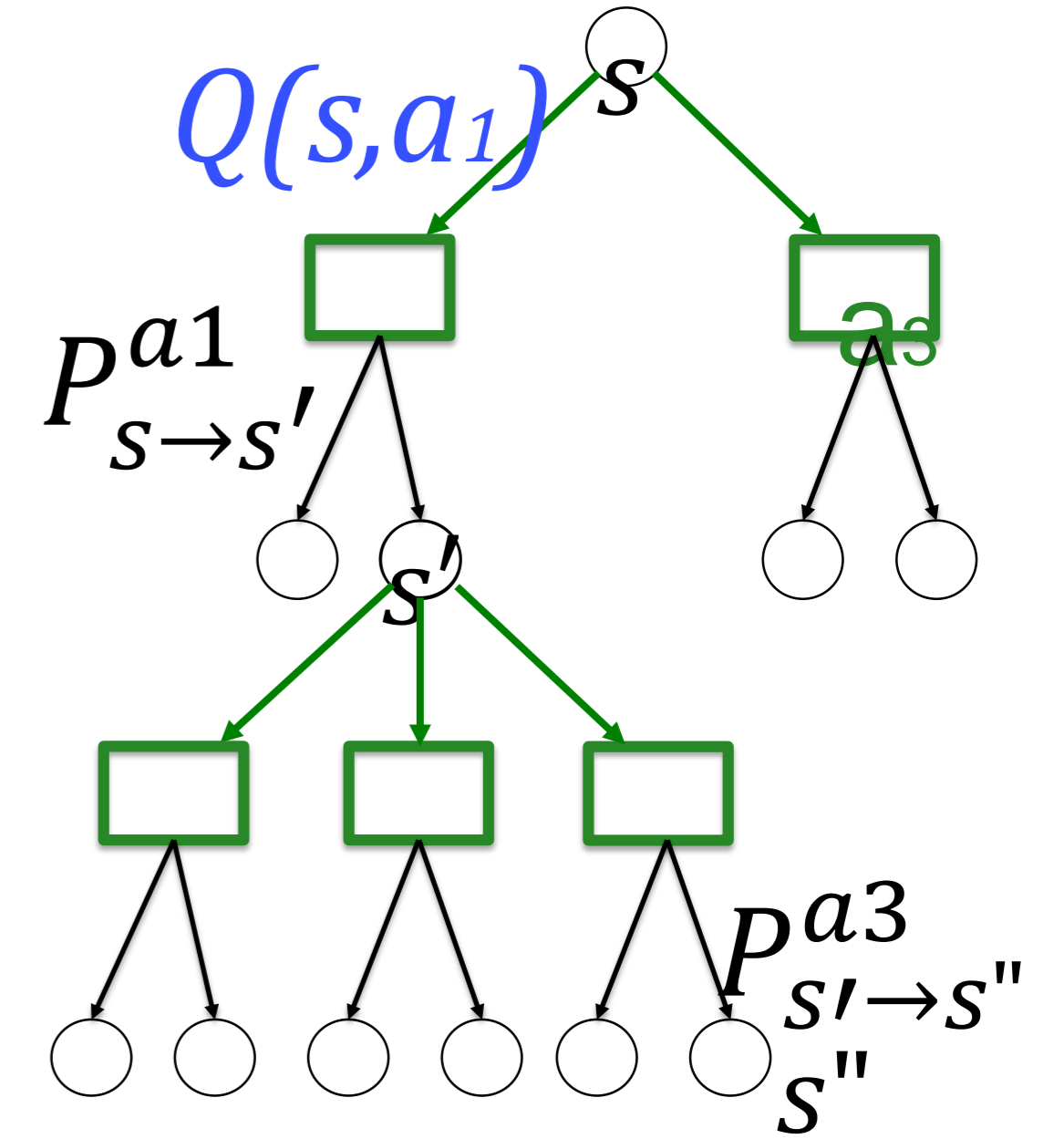
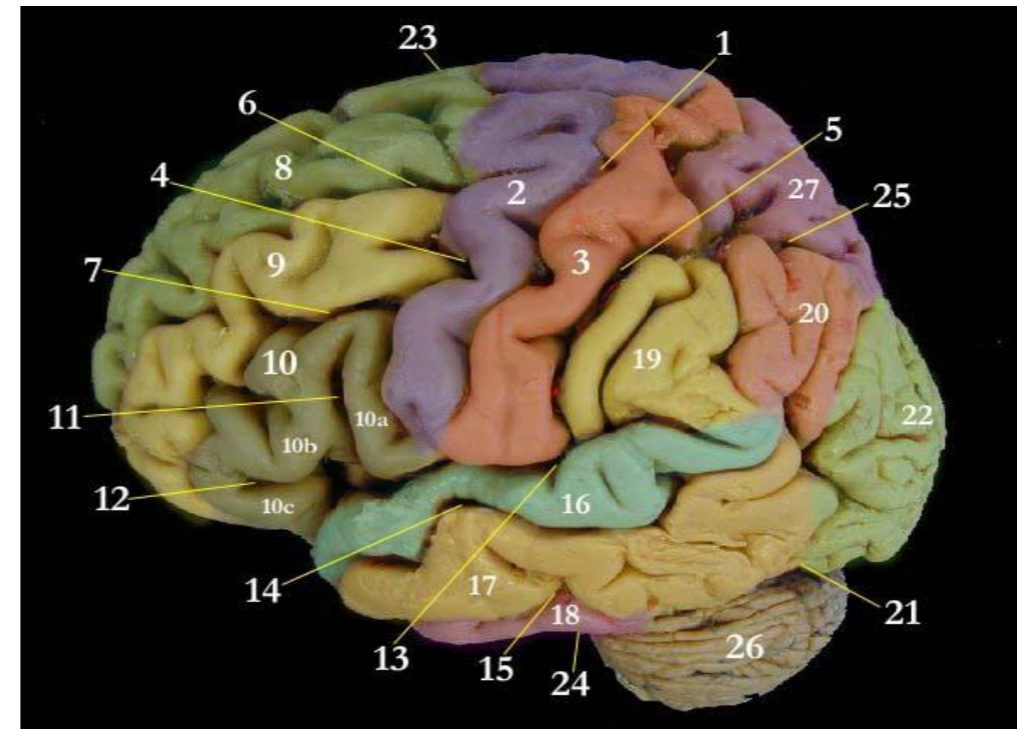
(A) A specific experimental task was a sequential two-choice Markov decision task in which all decision states are represented by fractal images. The task design follows that of a binary decision tree. Each trial begins in the same state. Subjects can choose between a left (L) or right (R) button press. 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).

Model-based Reinforcement learning

Reward Exposure



$$Q(s, a) = \sum_{s'} P_{s \rightarrow s'}^a \left[R_{s \rightarrow s'}^a + \gamma \sum_{a'} \pi(s', a') Q(s', a') \right]$$



explore potential future paths in your 'mind'

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 before taking the real actions.

Summary: Model-based versus Model-free

Model-free (Standard SARSA, Q-Learning, Policy Gradient)

- you are just choosing the best next action in current state
- no planning
- difficulties if changes in reward-scheme

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)

Previous slide.
Summary of findings