Artificial Neural Networks: Lecture 10
Policy Gradient Methods

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Objectives for today:
- **basic idea of policy gradient: learn actions, not Q-values**
- **log-likelihood trick: getting the correct statistical weight**
- **policy gradient algorithms**
- **why subtract the mean reward?**
- **actor-critic framework**
- **eligibility traces as 'candidate parameter updates'**

Reading for this week:

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

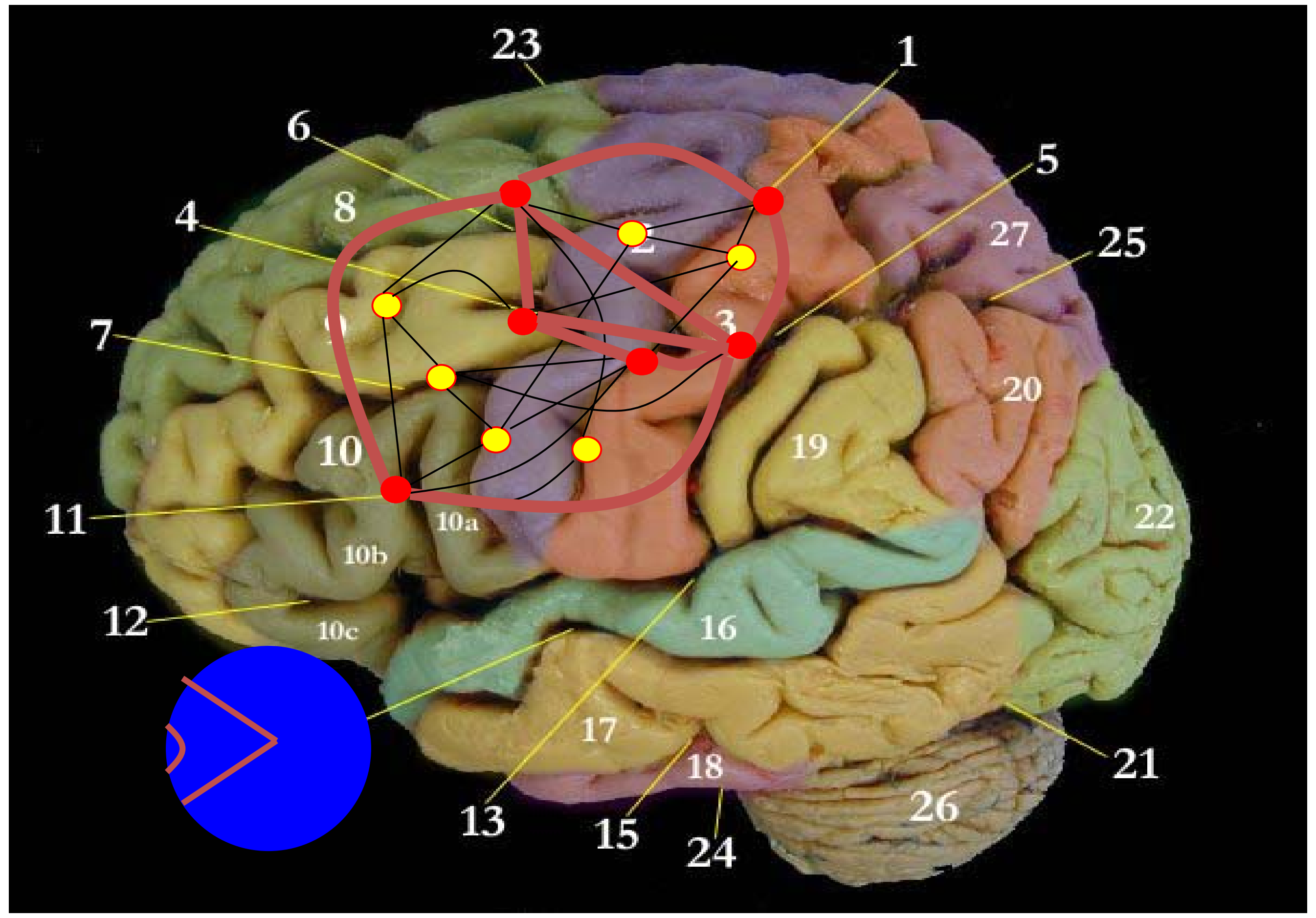
Chapter: 13.1-13.5

Background reading: none

- 3rd miniproject,
  deadline Monday June 11

- Fraud detection interviews
  Friday June 1st and May 25th

1. Review: Artificial Neural Networks for action learning



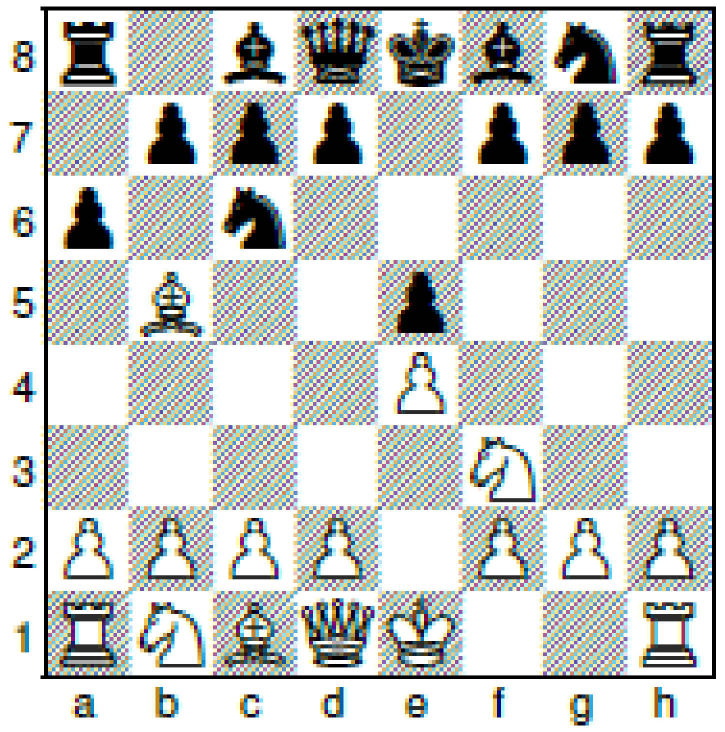
Where is the supervisor?
Where is the labeled data?

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

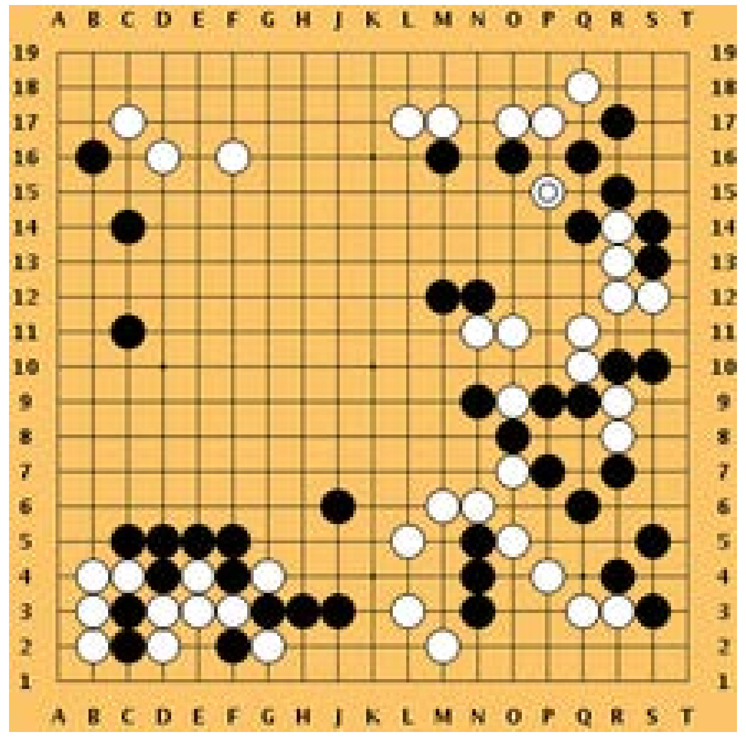
BUT:
Reward is rare:
'sparse feedback' after a long action sequence

1. Review: Deep reinforcement learning

Chess



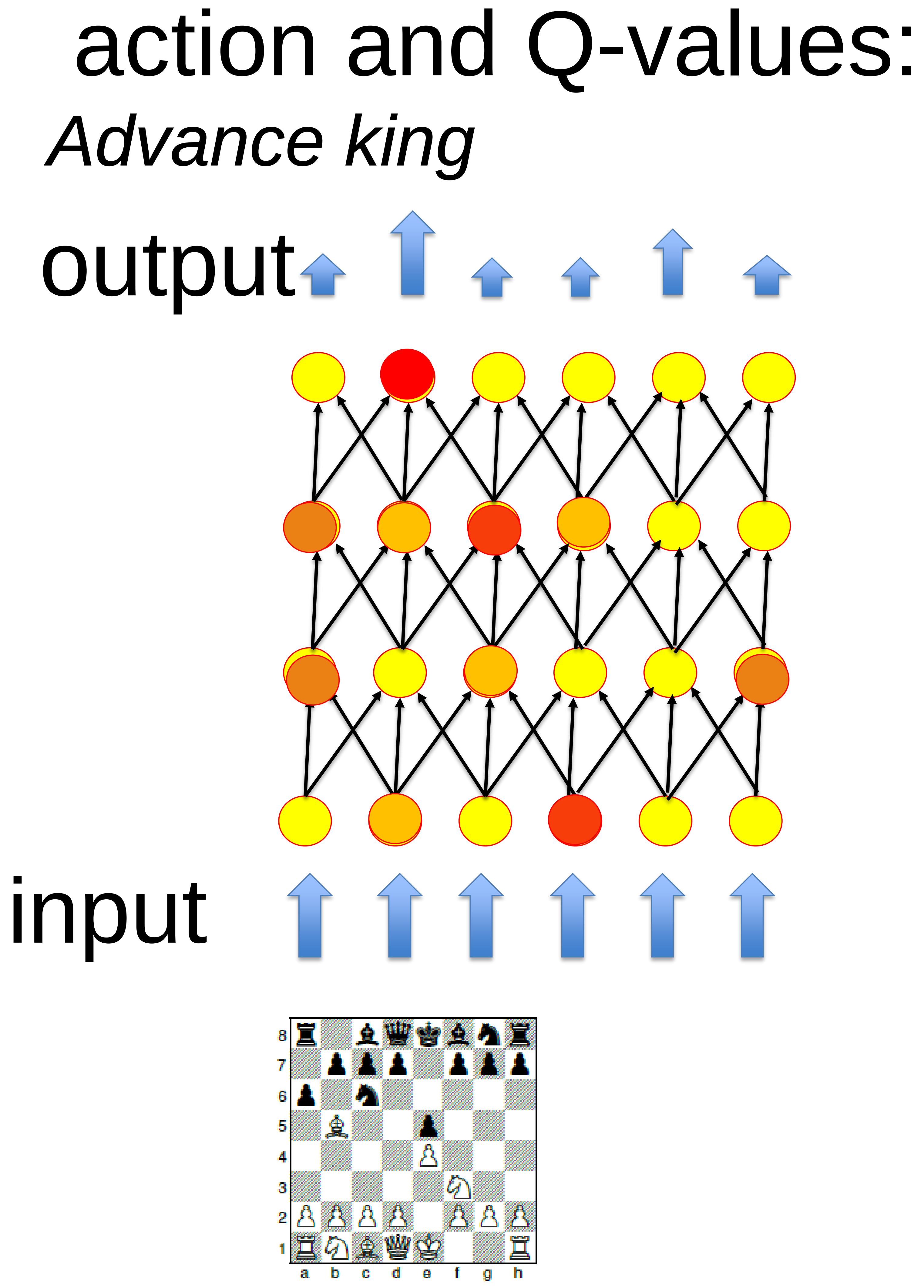
Go



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

In Go, it beats Lee Sedol

1. Review: Backprop for deep Q-learning

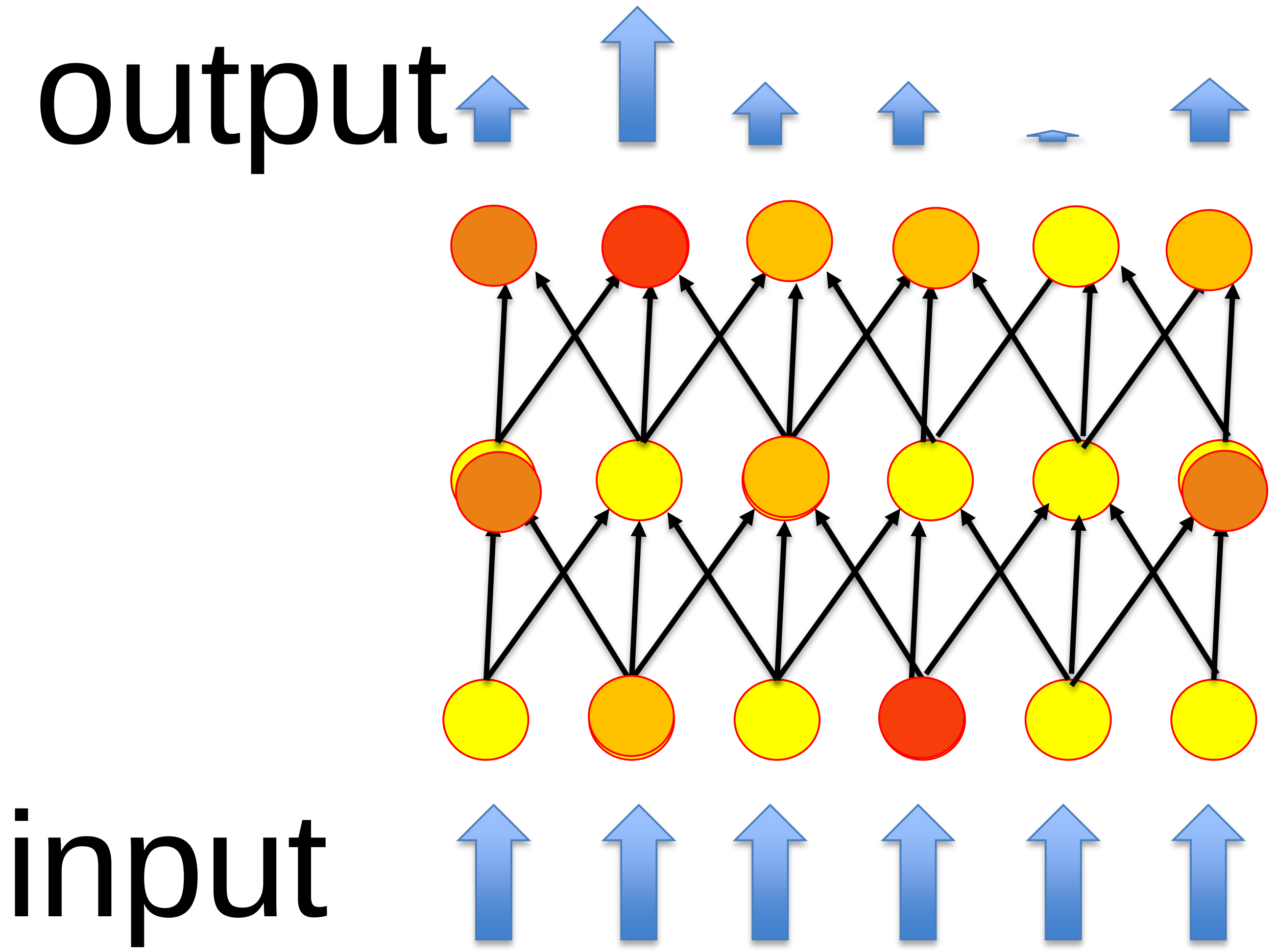


Outputs are Q-values
→ actions are easy to choose

1. Review: Backprop for deep Q-learning

action and Q-values:

Move piece



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

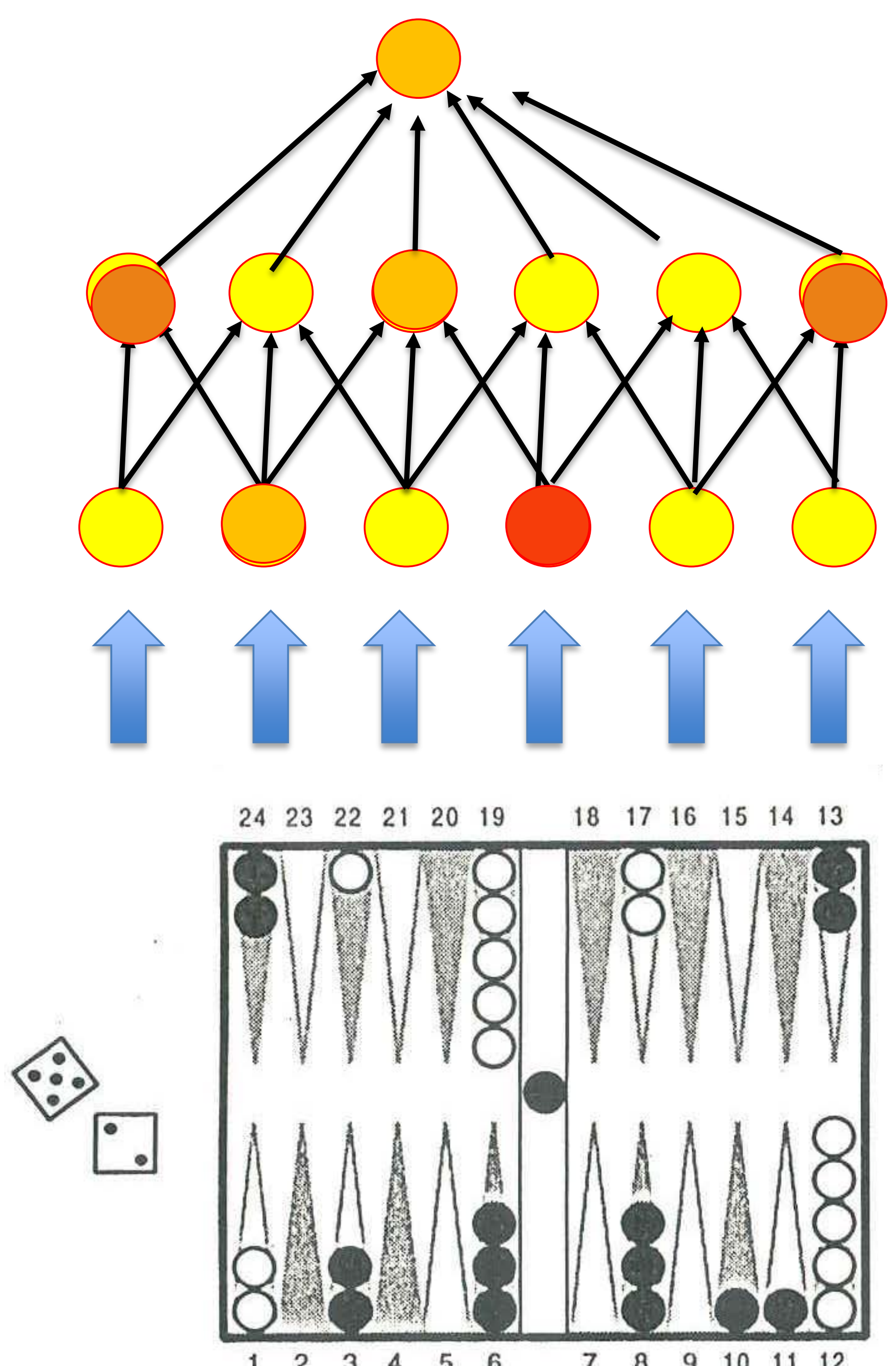
One output for one action a.

Learn weights by playing against itself.

1. Review: Deep Neural Network for Value function

Action: move piece by epsilon
greedy so as to increase V-value
in each step

output: V-values:



input

- Neural network parameterizes V-values as a function of state s.
- One single output.
- Learn weights by playing against itself.
- Minimize TD-error of V-function
- use eligibility traces

TD-Gammon

Tesauro, 1992,1994, **1995**, 2002

1. Review: Deep Neural Network for TD learning

In all TD learning methods
(includes n-step SARSA, Q-learning, TD(lambda))

- V-values OR Q-values are the central quantities

- actions are taken with softmax, greedy, or epsilon-greedy policy **derived from Q-values/V-values**

Aim for today:

- learn actions directly
- no need for Q-value estimation

→ **Policy Gradient**

Artificial Neural Networks: Lecture 10
Policy Gradient Methods

Wulfram Gerstner

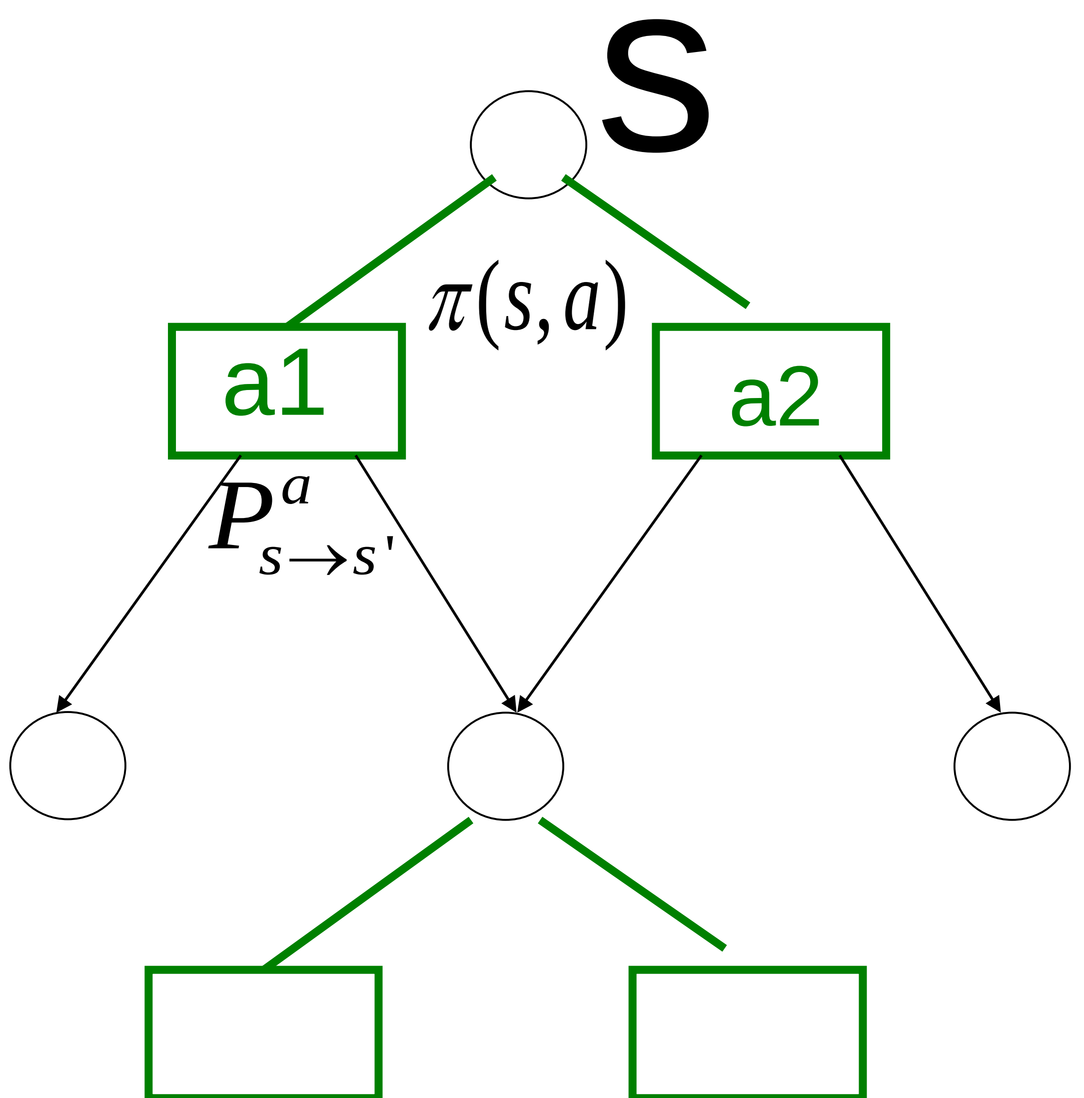
EPFL, Lausanne, Switzerland

1. **Review**
2. **Basic idea of policy gradient**

Disadvantages of Q-learning, SARSA, or TD-learning

- For continuous states, **function approximation** is necessary (which are potentially unstable).
- Even in fully observable (Markov) settings, off-policy TD algorithms (e.g. Q-learning) can diverge using **function approximation**.
- In **partially observable** environments (non-Markov), TD algorithms are problematic
- **Continuous actions** are difficult to represent using TD methods.

World is not a Markov Process
World is not fully observable
World is not tabular (not discrete states)



1. Policy Gradient methods: basic idea

- Forget Q-values
- Optimize directly the reward
- Associate actions with stimuli stochastically

Table in Q-learning:
(state,action) → Q

	a_1	a_2	a_3
s_1	$Q(s_1,a_1)$	$Q(s_1,a_2)$	
s_2	$Q(s_2,a_1)$		
s_3			
s_4			

Table in Policy gradient:
state → Prob(action|state)

	a_1	a_2	a_3
s_1	0.1	0.8	0.1
s_2	0.75	0.1	0.15
s_3	0.01	0.02	0.97
s_4	0.5	0.5	0.0

1. Policy Gradient methods: basic idea

- Forget Q-values
- Optimize directly the reward
- Associate actions with stimuli using a stochastic policy
- **Change parameters so as to maximize rewards**

stochastic policy

$\pi(a|s,\theta)$

parameter

Table in Policy gradient:
state → Prob(action|state,parameters)

	a_1	a_2	a_3
s_1	0.1	0.8	0.1
s_2	0.75	0.1	0.15
s_3	0.01	0.02	0.97
s_4	0.5	0.5	0.0

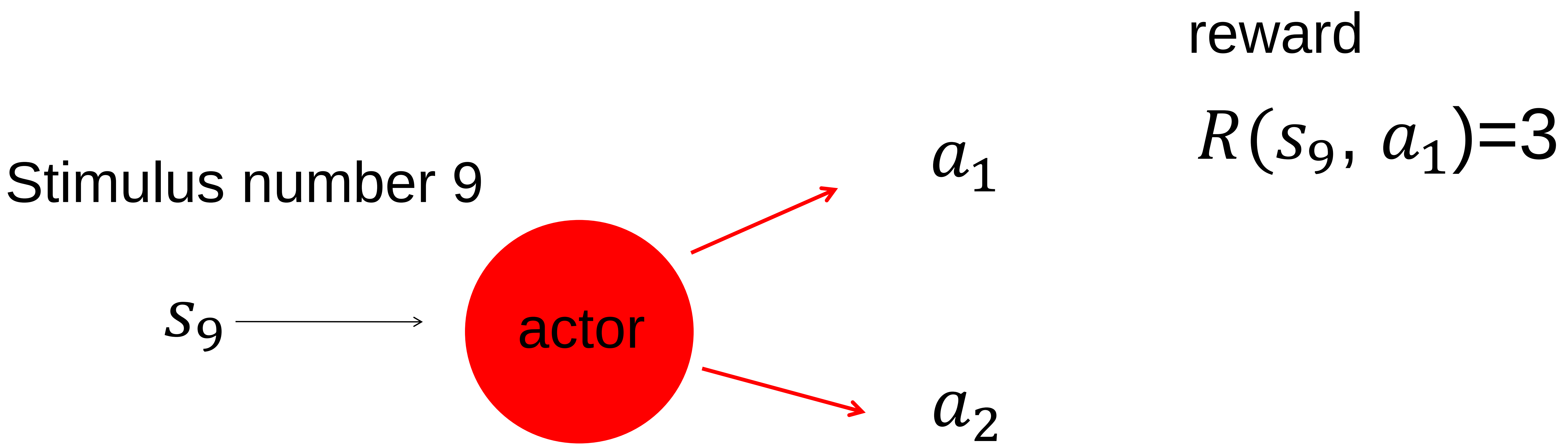
Artificial Neural Networks: Lecture 1p
Policy Gradient Methods

Wulfram Gerstner
EPFL, Lausanne, Switzerland

1. **Review**
2. **Basic idea of policy gradient**
3. **Example: 1-step horizon**

3. Policy Gradient methods: 1-step horizon

- Associate actions with stimuli
- Optimize directly the reward



3. Policy Gradient methods: 1-step horizon

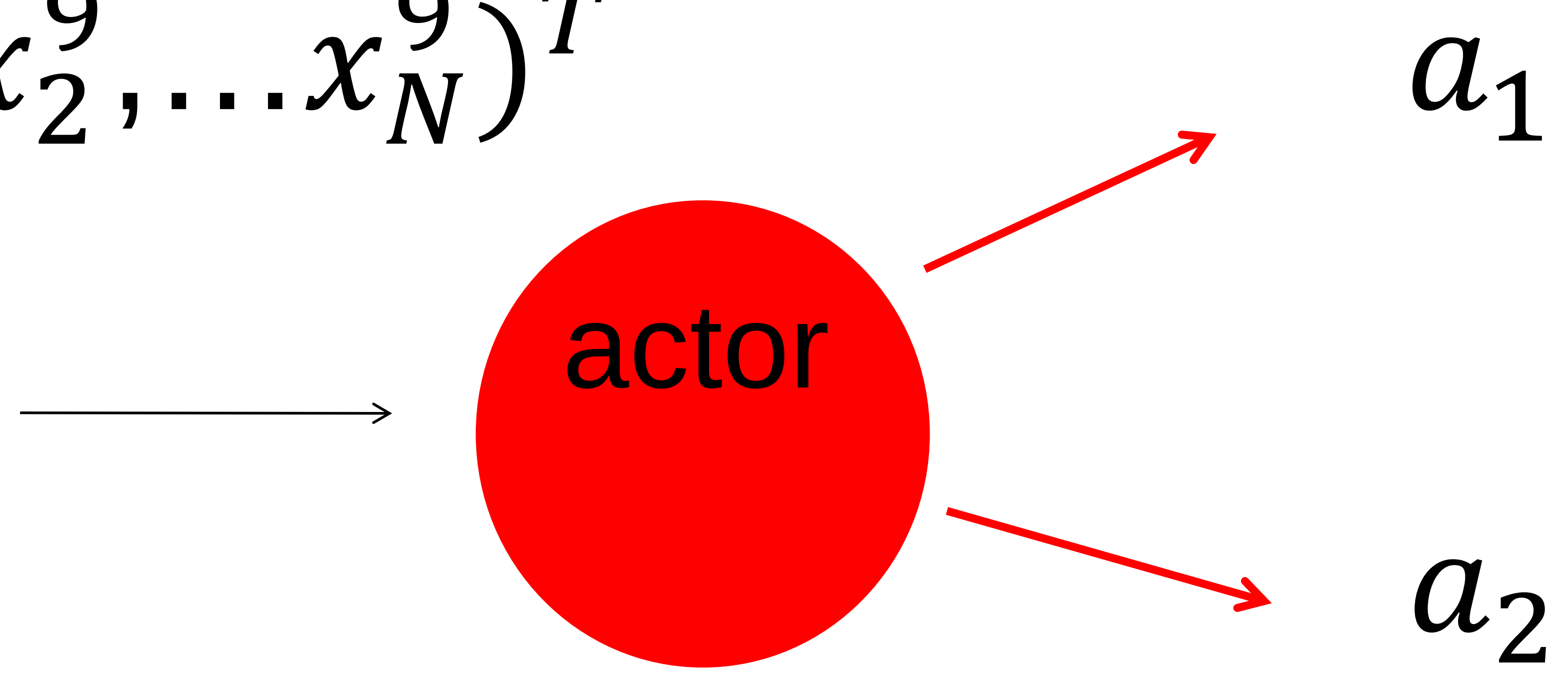
stimulus=state=input vector

Stimulus number 9 is a vector

$\vec{x}^9 = (x_1^9, x_2^9, \ldots x_N^9)^T$

$R(\vec{x}^9, a_1)$=3

$s_9 = \vec{x}^9$



3. Policy Gradient methods: 1-step horizon

Exercise 1 now (8min)

Aim: change weights of neuron
→Maximize expected reward!

Stimulus number 9

$$\vec{x}^9 = (x_1^9, x_2^9, \ldots x_N^9)^T$$

$s_9 = \vec{x}^9$

actor
=neuron

Output
of neuron

$a_1 \rightarrow y = 1$

$a_2 \rightarrow y = 0$

Choice of actions
(policy)

$$\pi(a1|s, \vec{w}) = prob(y = 1|\vec{x}, \vec{w}) = g(\sum_k^N w_k x_k)$$

Exercise 1: maximize expected reward

Exercise 1 now (8min)

3. Policy Gradient methods: 1-step horizon

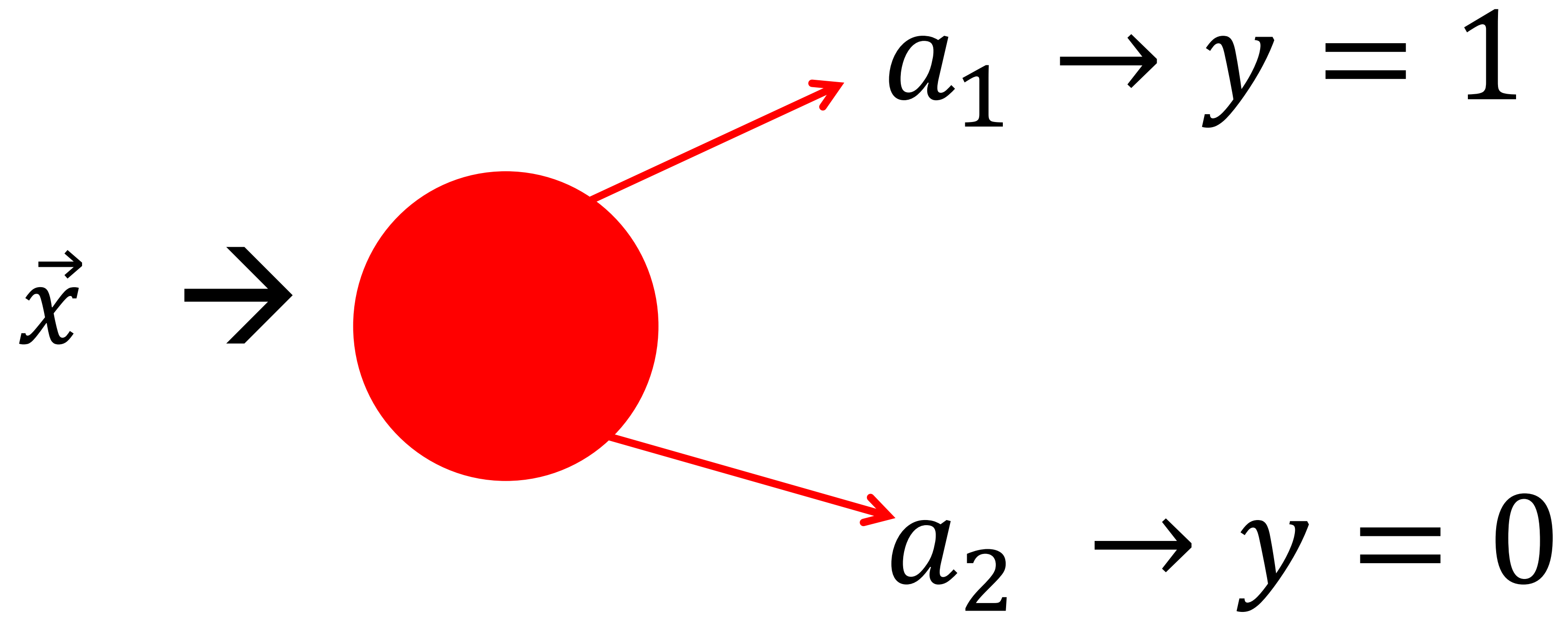
blackboard1

reward

$$R(y, \vec{x})$$

policy

$$\pi(y = 1 | s, \vec{w}) = g\left(\sum_{k}^{N} w_k x_k\right)$$



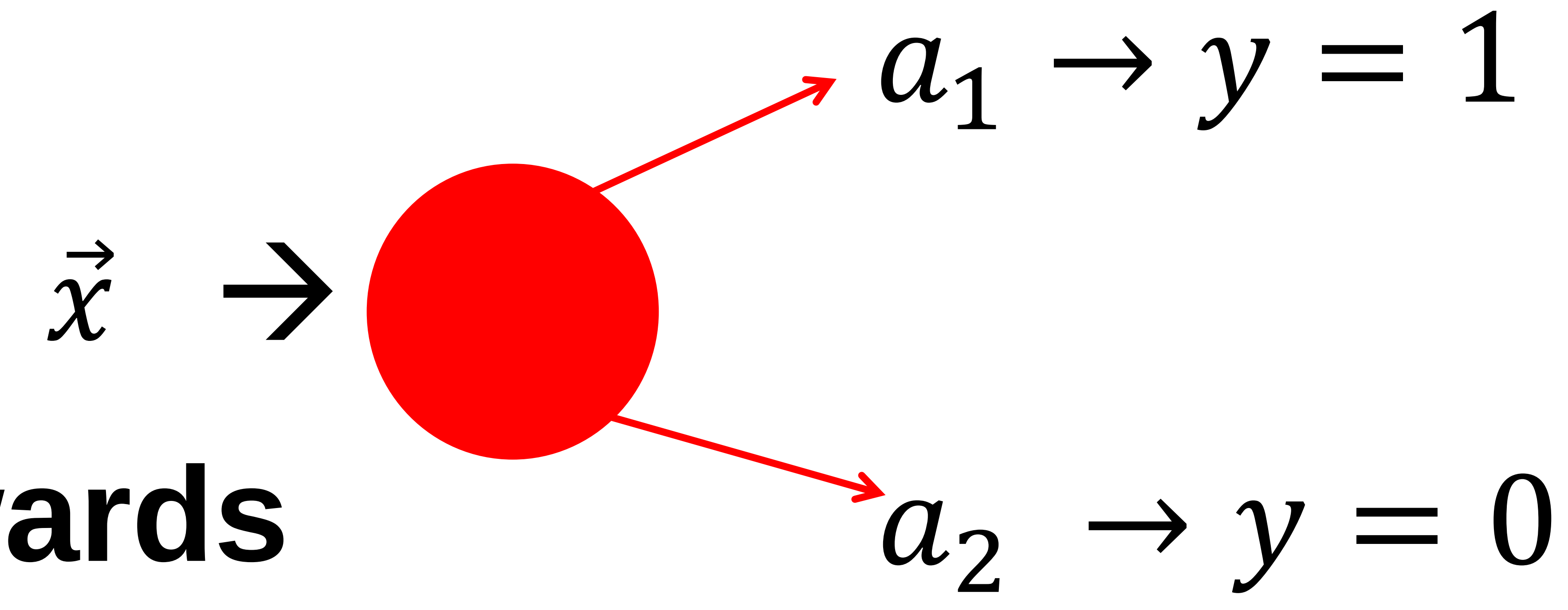
3. Policy Gradient methods: 1-step horizon

reward

$R(y, \vec{x})$

policy

$$\pi(y = 1|s, \vec{w}) \;= g(\sum_k^N w_k x_k)$$



Update parameters to maximize rewards

If $y = 1$: $\Delta w_j = \eta \frac{g'}{g} R(1, \vec{x}) x_j$

If $y = 0$: $\Delta w_j = \eta \frac{-g'}{(1-g)} R(0, \vec{x}) x_j$

3. Policy Gradient methods: Batch-to-Online

Attention at transition 'Batch to Online':
→ natural statistical weight must be correct!

We have a stochastic starting point with weight $p(s)$
as well as $\underbrace{\text{stochastic transitions}}$ and a $\underbrace{\text{stochastic policy}}$

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

weighting factor
for 'next state'

$$\sum_{a'} \pi(a'|s, \vec{w})$$

weighting factor
for 'next action'

Artificial Neural Networks: Lecture 10
Policy Gradient Methods

Wulfram Gerstner
EPFL, Lausanne, Switzerland

1. **Review**
2. **Basic idea of policy gradient**
3. **Example: 1-step horizon**
4. **Log-likelihood trick**

4. Log-likelihood trick

Blackboard 2

4. Log-likelihood trick

$$
\begin{aligned}
\nabla_\theta J &= \int \nabla_\theta p(H) R(H) dH \\
&= \int \frac{p(H)}{p(H)} \nabla_\theta p(H) R(H) dH \\
&= \int p(H) \nabla_\theta \log p(H) R(H) dH
\end{aligned}
$$

J = function you want to optimize

H = ensemble over which you integrate

4. Policy gradient derivation

$$\nabla_\theta J \quad = \quad \int p(H) \nabla_\theta \log p(H) R(H) dH.$$

Taking the sample average as Monte Carlo (MC) approximation of this expectation by taking N trial histories we get

$$\nabla_\theta J = \mathbf{E}_H \Big[\nabla_\theta \log p(H) R(H) \Big] \approx \frac{1}{N} \sum_{n=1}^{N} \nabla_\theta \log p(H^n) R(H^n).$$

which is a fast approximation of the policy gradient for the current policy

4. Policy gradient evaluation: Example from Exercise 1

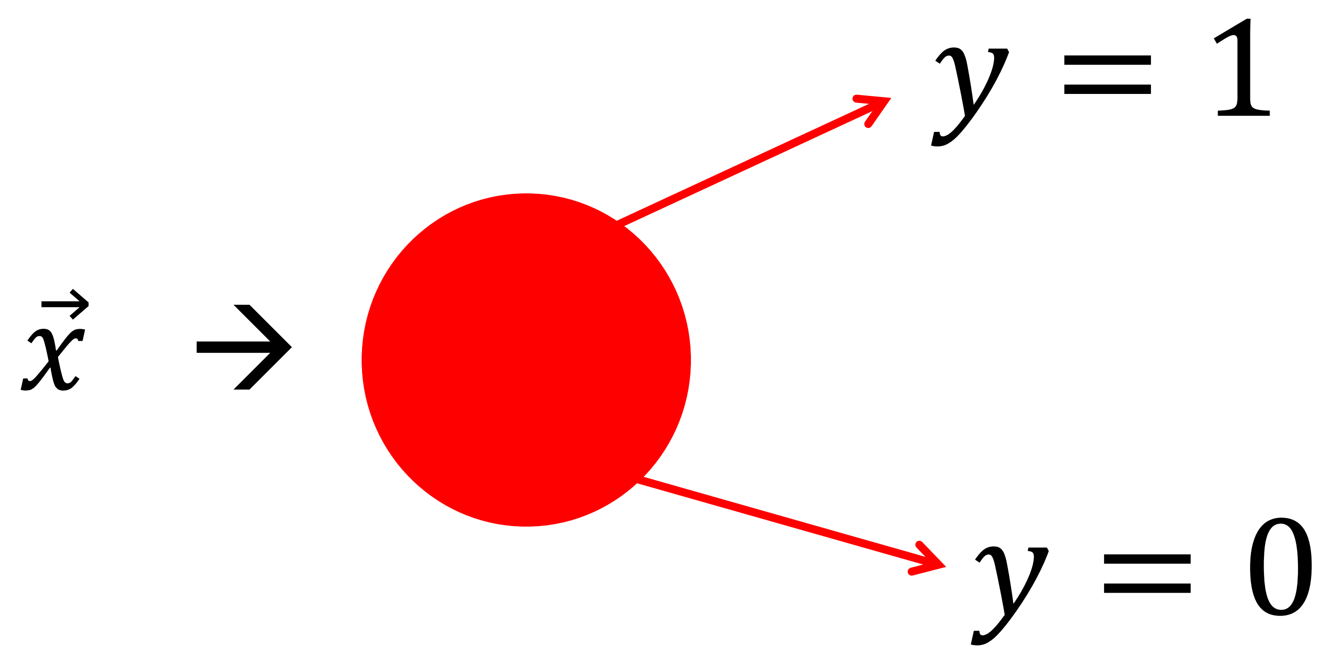
Blackboard 2b

reward

$R(y, \vec{x})$

policy

$$\pi(y = 1|s, \vec{w}) = g\left(\sum_{k}^{N} w_k x_k\right)$$



4. Update rule for Exercise 1

observe input $\vec{x}$, output y, and reward $R(y,\vec{x})$

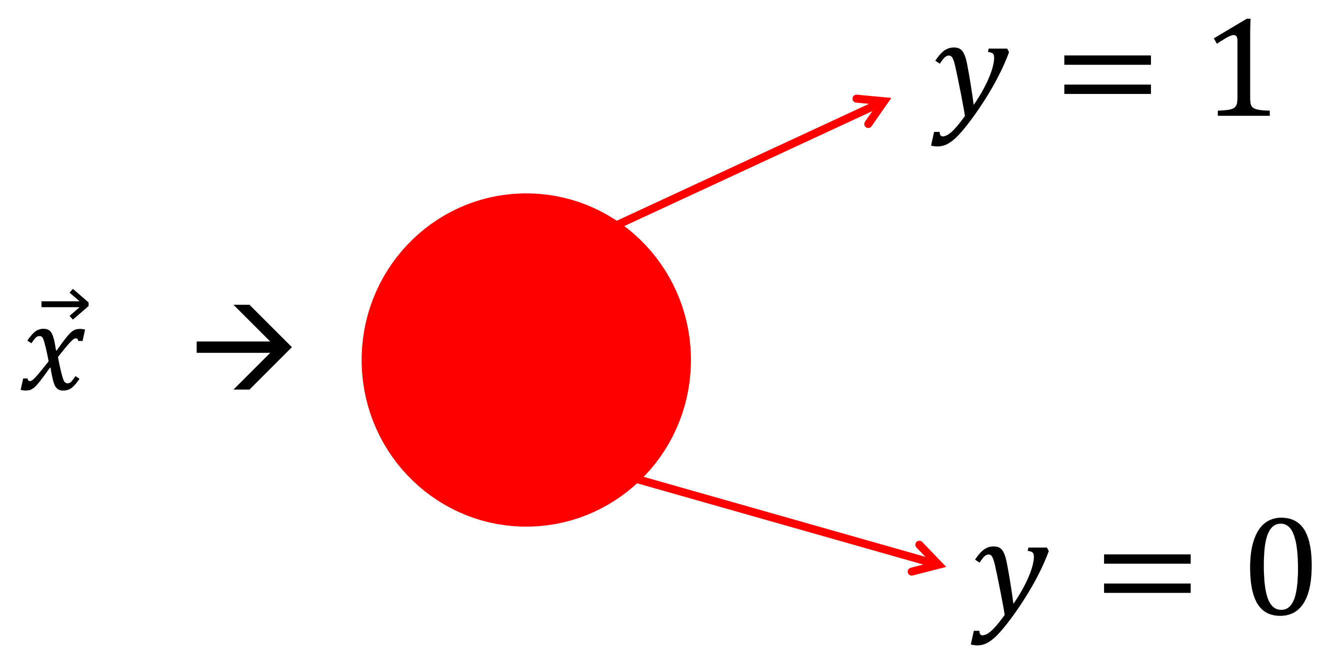
Earlier result:

If $y = 1$: $\quad \Delta w_j = \eta \, \frac{g'}{g} \cdot R(1,\vec{x}) x_j$

If $y = 0$: $\quad \Delta w_j = \eta \frac{-g'}{(1-g)} R(0,\vec{x}) x_j$

policy

$$\pi(y = 1|s,\vec{w}) = g\left(\sum_k^N w_k x_k\right)$$



Now rewritten as:

$$\Delta w_j = \eta \, \frac{g'}{g(1-g)} R(y,\vec{x}) [\, y - \langle y \rangle] x_j$$

Note: $\langle y \rangle = g(\sum_k^N w_k x_k)$

4. Comparison with Perceptron

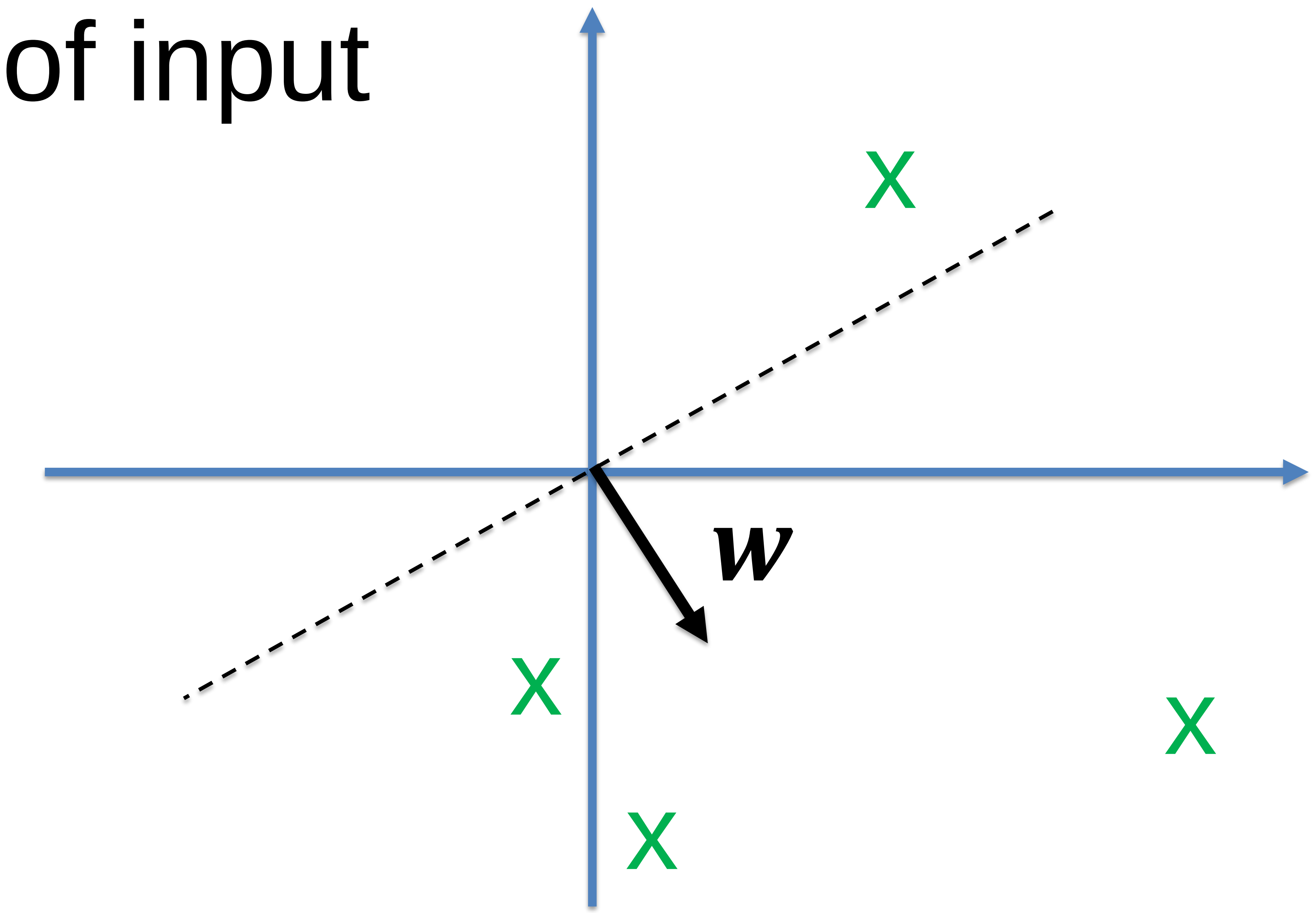
parameter = weight w_j

$$\Delta w_j \propto R(y,\vec{x})[\,y - \langle y \rangle]x_j$$

Weight vector turns in direction of input

$$\Delta \boldsymbol{w} \propto \ \pm \boldsymbol{x}$$

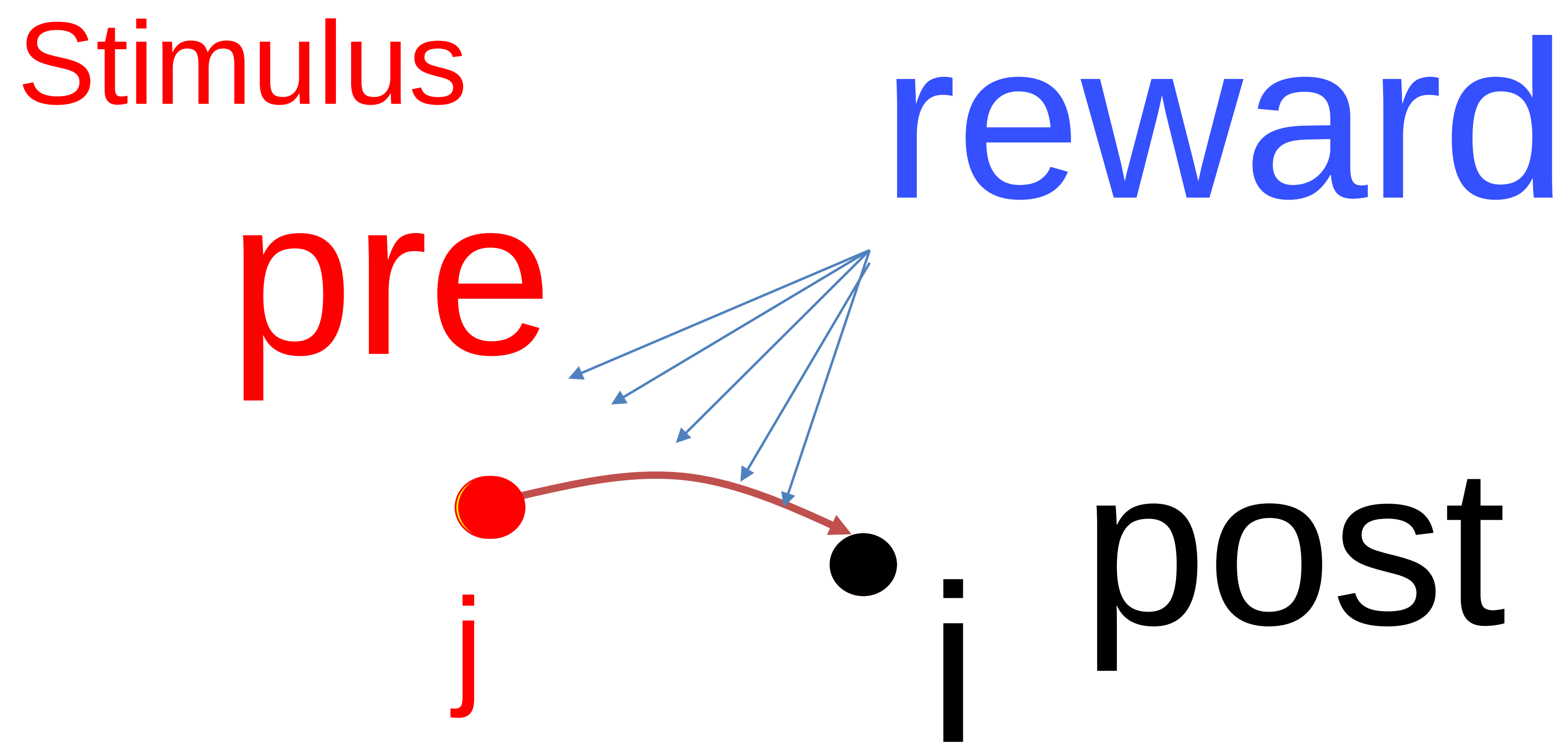
R>0 and y=1 → $\Delta \boldsymbol{w} \propto +\boldsymbol{x}$



4. Comparison with Biology

parameter = weight w_j

$$\Delta w_j \propto R(y, \vec{x})[\,y - \langle y \rangle]x_j$$



Weight vector turns in direction of input

Three factors: **reward** **post** **pre**

$$\Delta w_{ij} = \eta \frac{g'}{g(1-g)} R(\vec{y}, \vec{x}) \overbrace{[\,y_i - \langle y_i \rangle]} x_j$$

postsynaptic factor is

'activity – expected activity'

4. Generalization: subtract a reward baseline

we derived this online gradient rule

$$\Delta w_j \propto R(y, \vec{x})[\, y - \langle y \rangle] x_j$$

But then this rule is also an online gradient rule

$$\Delta w_j \propto [R(y, \vec{x}) - b][\, y - \langle y \rangle] x_j$$

with the same expectation (see exercise 2)

(because a baseline shift drops out if we take the gradient)

4. Quiz: Policy Gradient and Reinforcement learning

Your friend has followed over the weekend a tutorial in reinforcement learning and claims the following. Is he right?

[ ] All reinforcement learning algorithms work either with Q-values or V-values

[ ] The transition from batch to online is always easy: you just drop the summation signs and bingo!

[ ] All reinforcement learning algorithms try to optimize the expected total reward (potentially discounted discounted if there are multiple time steps)

[ ] The derivative of the log-policy is some abstract quantity that has no intuitive meaning.

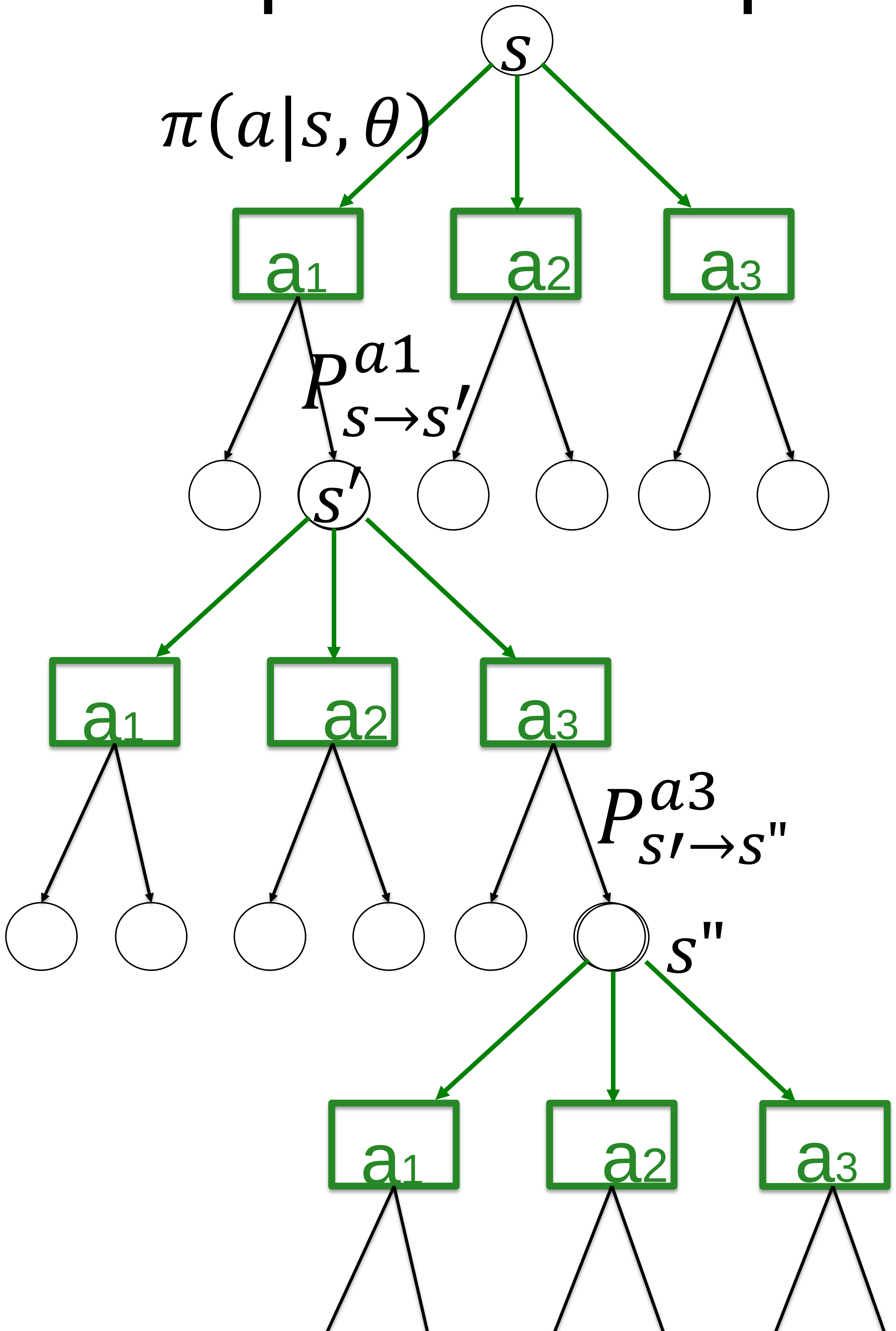
Artificial Neural Networks: Lecture 10
Policy Gradient Methods

Wulfram Gerstner
EPFL, Lausanne, Switzerland

1. **Review**
2. **Basic idea of policy gradient**
3. **Example: 1-step horizon**
4. **Log-likelihood trick**
5. **Multiple time steps**

5.Policy Gradient methods over multiple time steps

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



5.Policy Gradient methods over multiple time steps

Blackboard 3

5.Policy Gradient methods over multiple time steps

Calculation yields several terms of the form

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

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

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

$+ \ldots$

Policy Gradient methods over multiple time steps:

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (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)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

From book:
Sutton and Barto, 2018

Different states $S_0, S_1, S_2, \ldots$ during one episode

G = total accumulated reward during the episode starting at S_t;

All updates done AT THE END of the episode

Algorithm maximizes expected discounted rewards starting at S_0

Artificial Neural Networks: Lecture 10
Policy Gradient Methods

Wulfram Gerstner
EPFL, Lausanne, Switzerland

1. **Review**
2. **Basic idea of policy gradient**
3. **Example: 1-step horizon**
4. **Log-likelihood trick**
5. **Multiple time steps**
6. **Subtracting the mean via the value function**

6. Review: subtract a reward baseline

we derived this online gradient rule (for 1-step horizon)

$$\Delta w_j \propto R(y, \vec{x})[\, y - \langle y \rangle]x_j$$

But then this rule is also an online gradient rule

$$\Delta w_j \propto [R(y, \vec{x}) - b][\, y - \langle y \rangle]x_j$$

with the same expectation (see exercise 2)

(because a baseline shift drops out if we take the gradient)

6. Subtract a reward baseline

we derived this online gradient rule for multi-step horizon

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

But then this rule is also an online gradient rule

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

with the same expectation

(because a baseline shift drops out if we take the gradient)

6. 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 $V(s)$

6. Deep reinforcement learning: alpha-zero

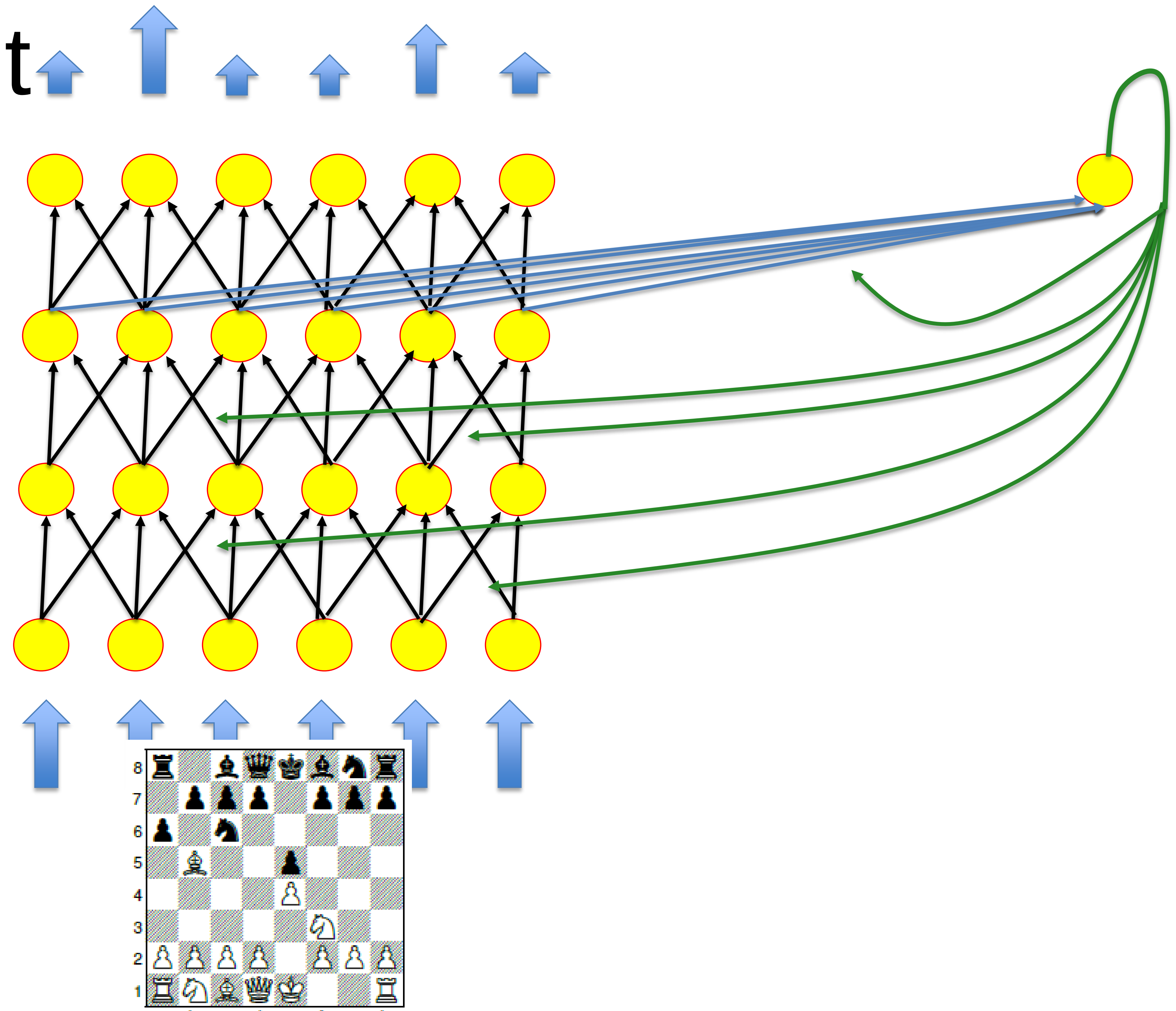
Network for choosing action

action:
Advance king

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

$V(s)$

output



input

learning:
→ change connections

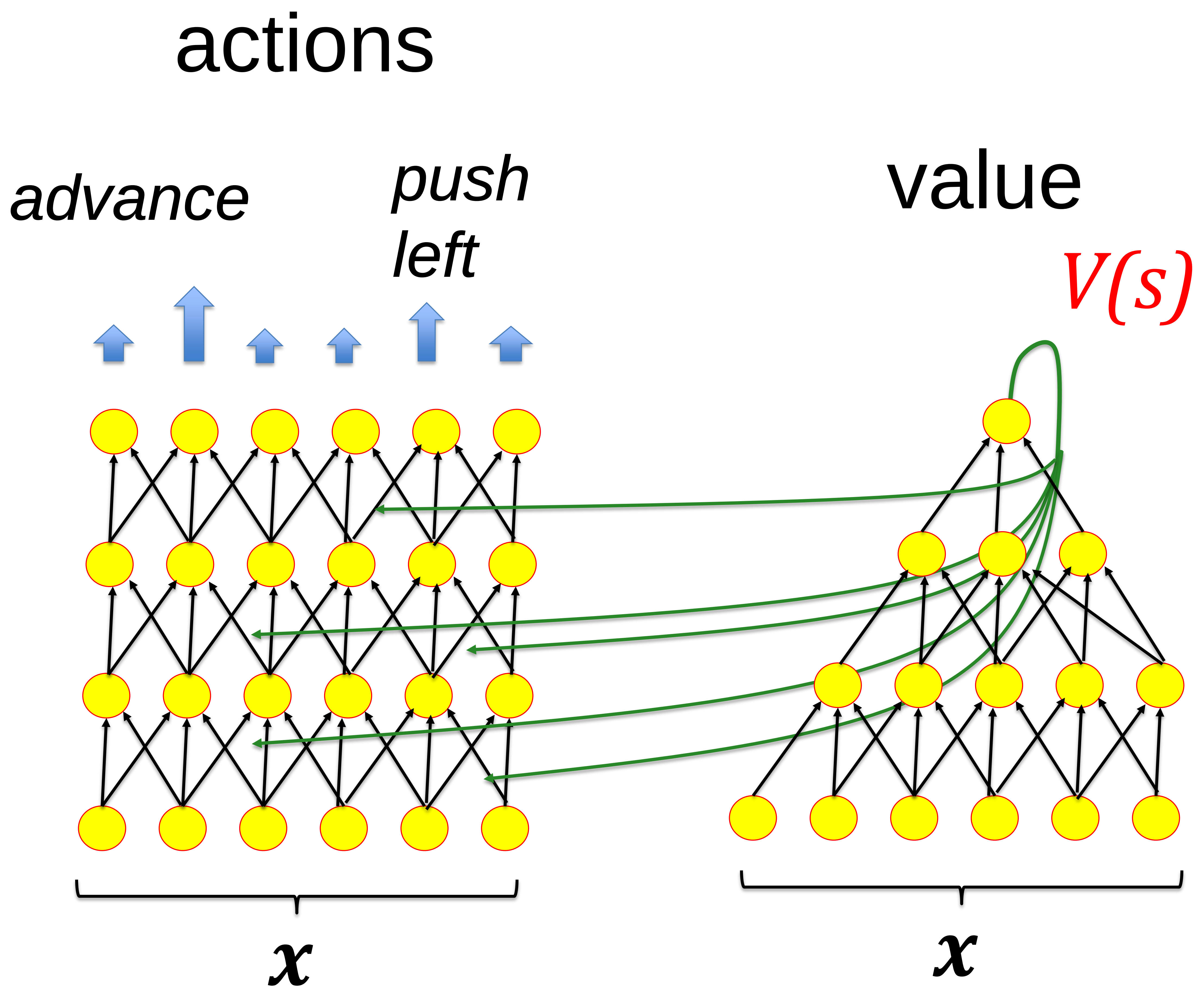
aims:

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

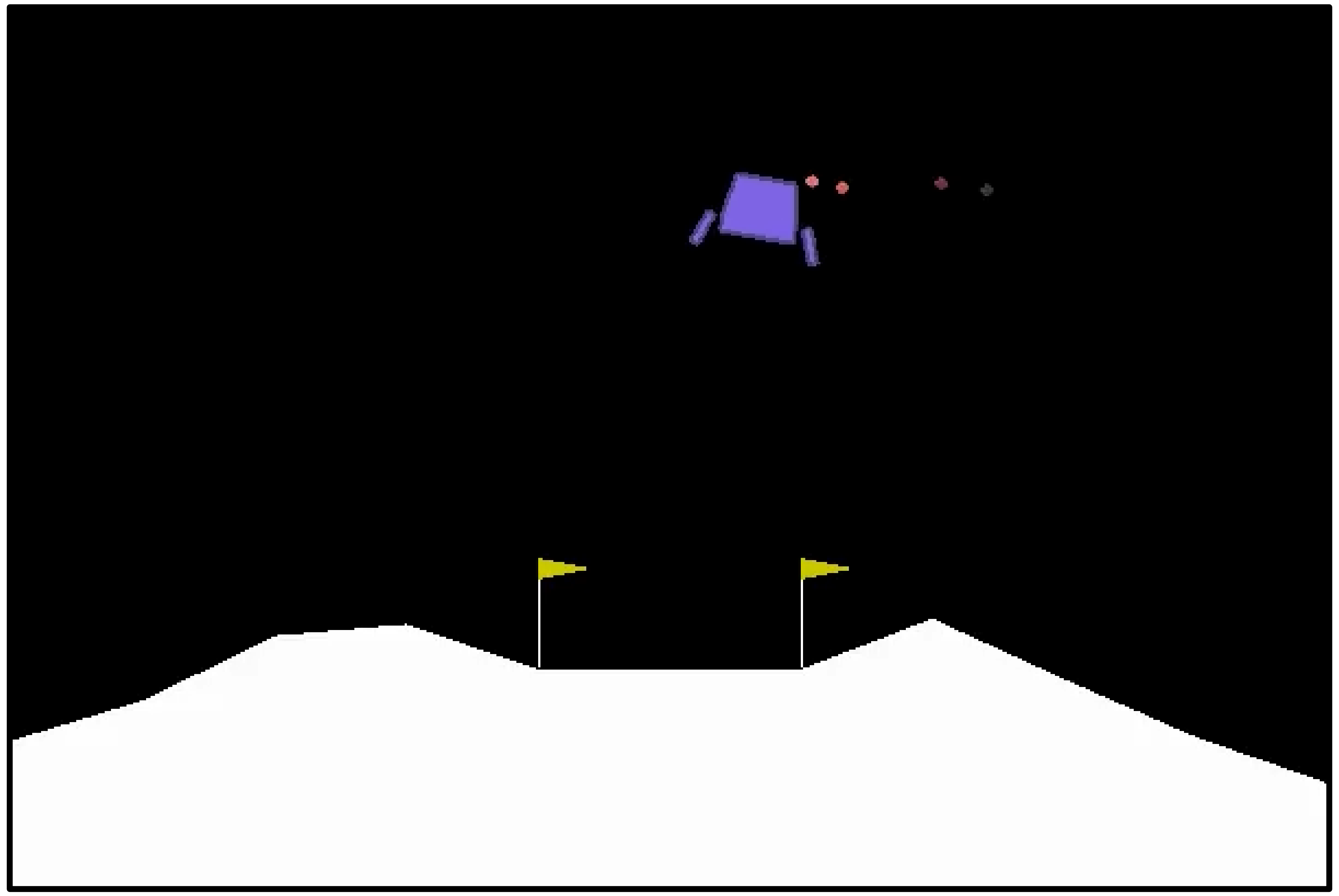
Learning signal:

- η[actual Return - $V(s)$]

6. Deep Reinforcement Learning: Lunar Lander (miniproject)



Aim: land between poles



6. Learning two Neural Networks: actor and value

Actions:

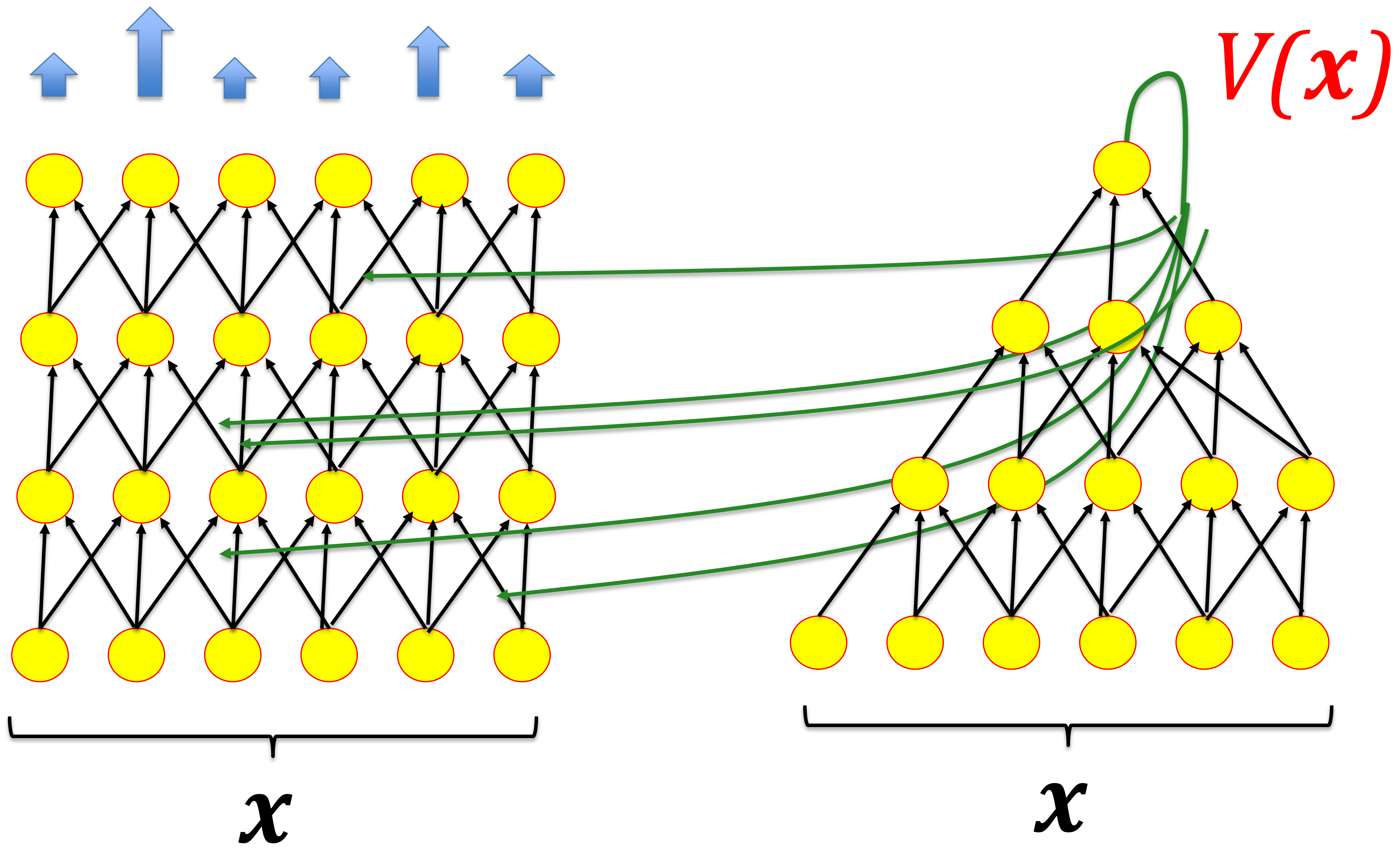
-Learned by
Policy gradient

- Uses $V(\boldsymbol{x})$ as baseline

Value function:

- Estimated by Monte-Carlo

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



$\boldsymbol{x}$ = states from episode:

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

'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)
        $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \gamma^t \delta \nabla \hat{v}(S_t,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

6. Why subtract the mean?

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

→ exercise 2.

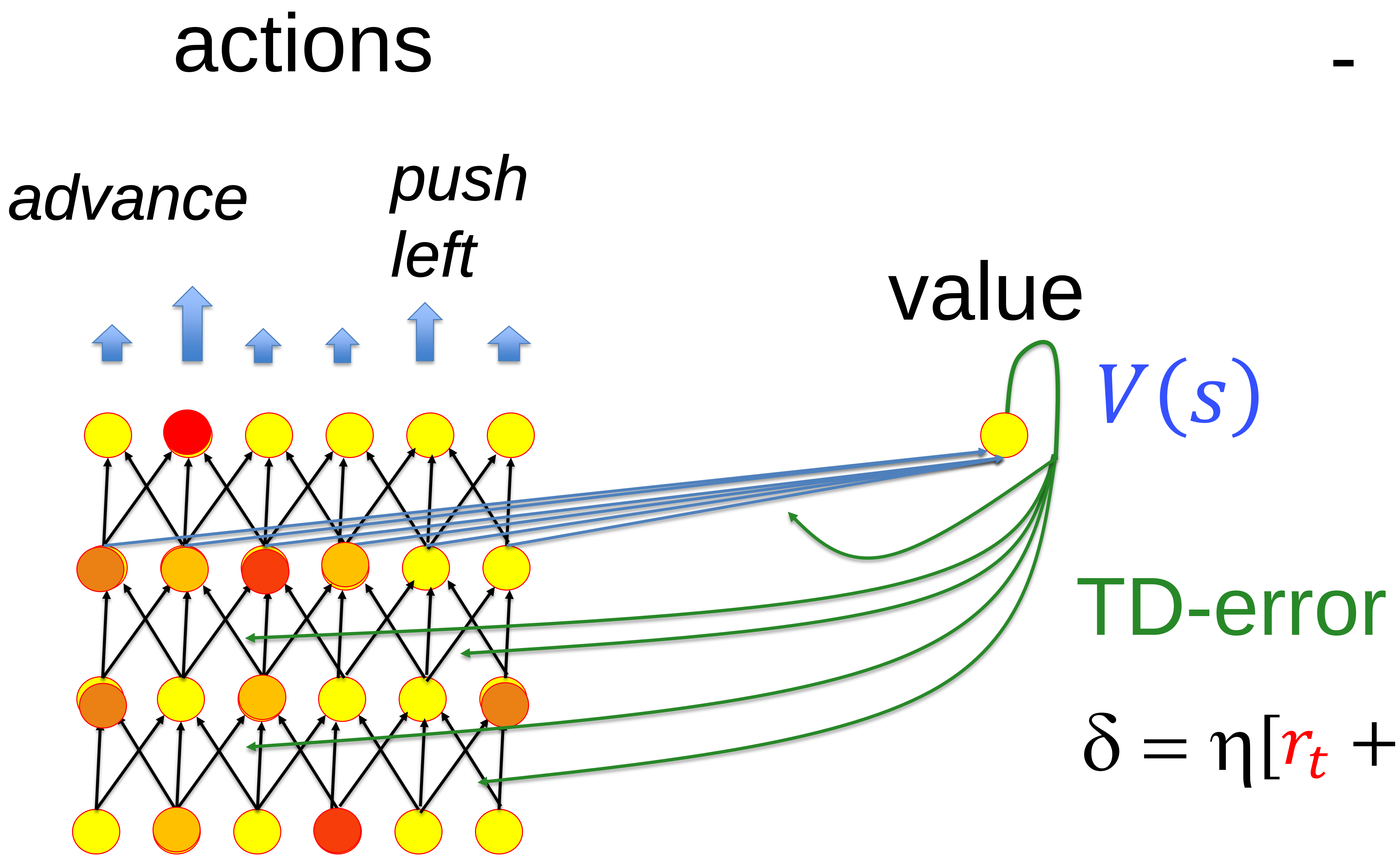
Artificial Neural Networks: Lecture 10
Policy Gradient Methods

Wulfram Gerstner
EPFL, Lausanne, Switzerland

1. **Review**
2. **Basic idea of policy gradient**
3. **Example: 1-step horizon**
4. **Log-likelihood trick**
5. **Multiple time steps**
6. **Subtracting the mean via the value function**
7. **Actor-Critic.**

7. Actor-Critic = 'REINFORCE' with TD bootstrapping

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

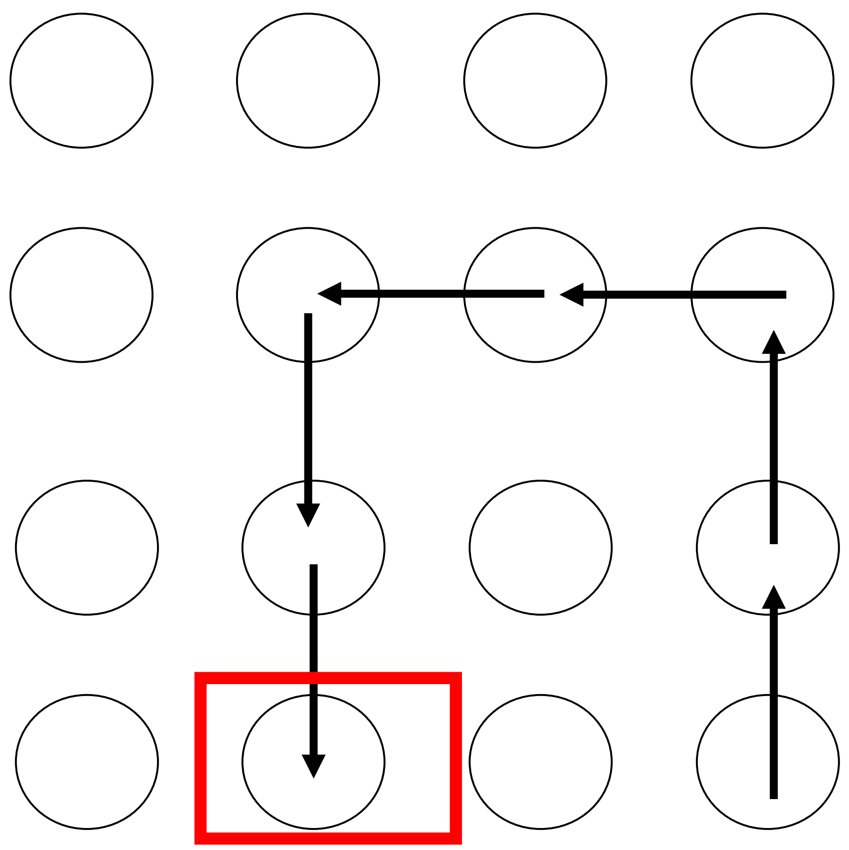


$$\delta = \eta[r_t + \gamma V(s') - V(s)]$$

7. Actor-Critic with eligibility traces

- Online algorithm
- Actor learns by policy gradient
- Critic learns by TD-learning
- Update eligibility traces while moving
- For each parameter, one eligibility trace
- Update weights proportional to TD-delta

7. Review: Eligibility Traces

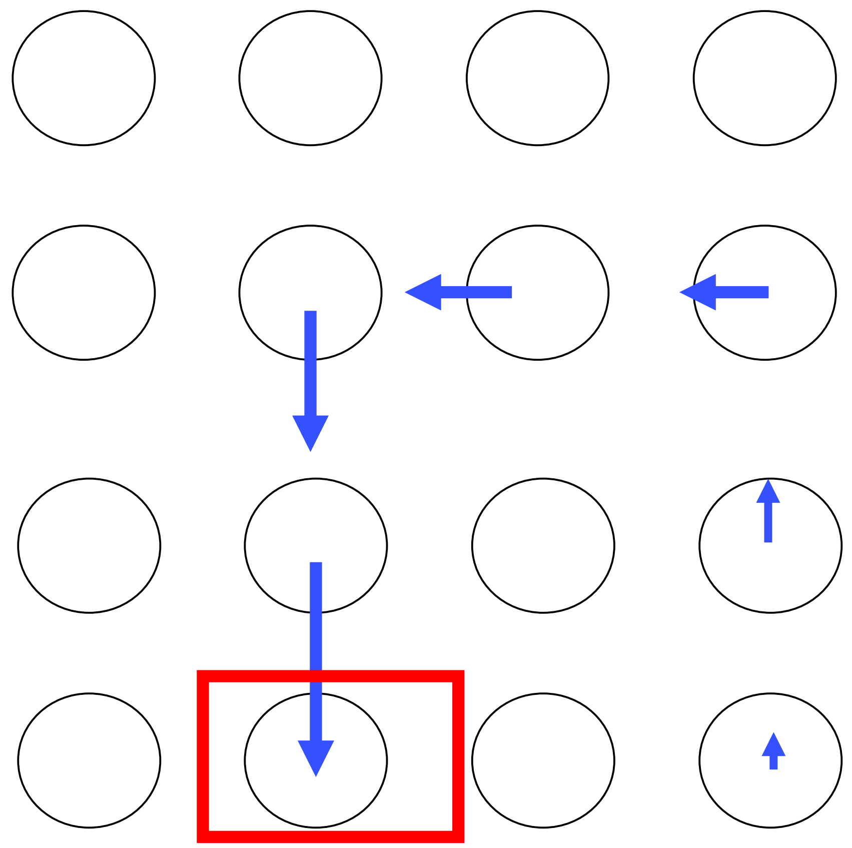


Idea:

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

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

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



- update **all** Q-values:

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

Here: SARSA with eligibility trace

7. Eligibility Traces

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

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

- update **all** parameters:

$$\Delta w_k = \eta \; \underbrace{[r-(V(s)-\gamma V(s'))]}_{\text{TD-delta}} \; z_k$$

Here: policy gradient with eligibility trace

Actor-Critic with Eligibility traces bootstrapping

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
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{z}^{\mathbf{w}} \leftarrow \gamma \lambda^{\mathbf{w}} \mathbf{z}^{\mathbf{w}} + I \nabla \hat{v}(S,\mathbf{w})$
    $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \gamma \lambda^{\boldsymbol{\theta}} \mathbf{z}^{\boldsymbol{\theta}} + I \nabla \ln \pi(A|S, \boldsymbol{\theta})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \mathbf{z}^{\mathbf{w}}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \delta \mathbf{z}^{\boldsymbol{\theta}}$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

4. Quiz: Policy Gradient and Reinforcement learning

Your friend has followed over the weekend a tutorial in reinforcement learning and claims the following. Is he right?

[ ] Even some policy gradient algorithms use V-values

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

[ ] The actor-critic has basically the same architecture as REINFORCE with baseline

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

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

Objectives for today:

- **basic idea of policy gradient: learn actions, not Q-values**
  → gradient descent of total expected discounted reward
- **log-likelihood trick: getting the correct statistical weight**
  → enables transition from batch to online
- **policy gradient algorithms**
  → updates of parameter propto $[R \quad]\frac{d}{d\theta_j}\ln[\pi]$
- **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