Wulfram Gerstner
EPFL, Lausanne, Switzerland

Artificial Neural Networks: Lecture 9
Variants of TD-learning methods and continuous space

Objectives for today:
- TD learning refers to a whole class of algorithms
- **There are many Variations of SARSA**
- **All set up to iteratively solve the Bellman equation**
- **Eligibility traces and n-step Q-learning to extend over time**
- **Continuous space and ANN models**
- **Models of actions and models of value**

Reading for this week:

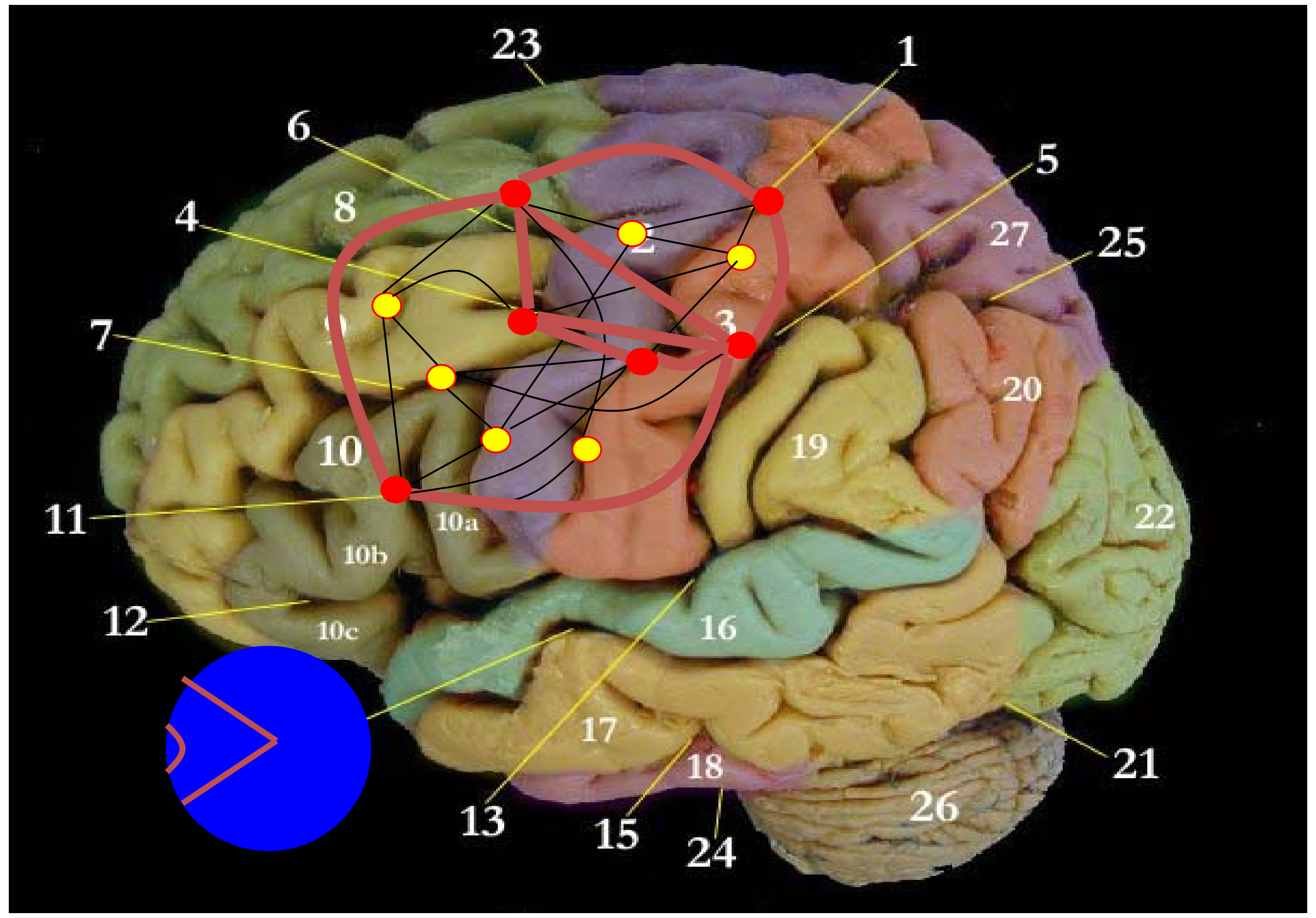
Sutton and Barto, Reinforcement Learning (MIT Press, 2nd edition 2018, also online)

Chapter: 5.1-5.4 and 6.1-6.3 and 6.5-6.6, and 7.1-7.2 and 9.3

Background reading:

Temporal Difference Learning and TD-Gammon
by Gerald Tesauro (1995) pdf online

1. Review: Artificial Neural Networks for action learning



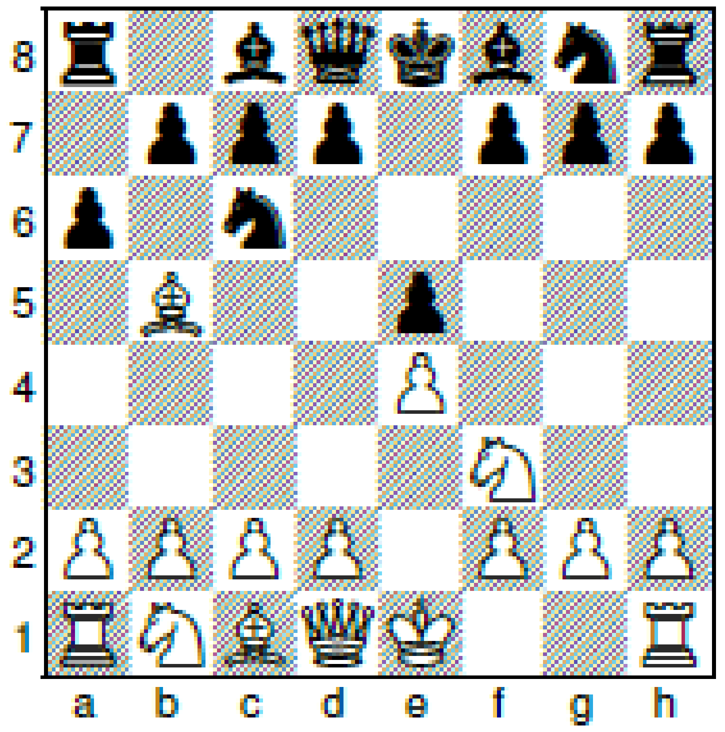
Where is the supervisor?
Where is the labeled data?

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

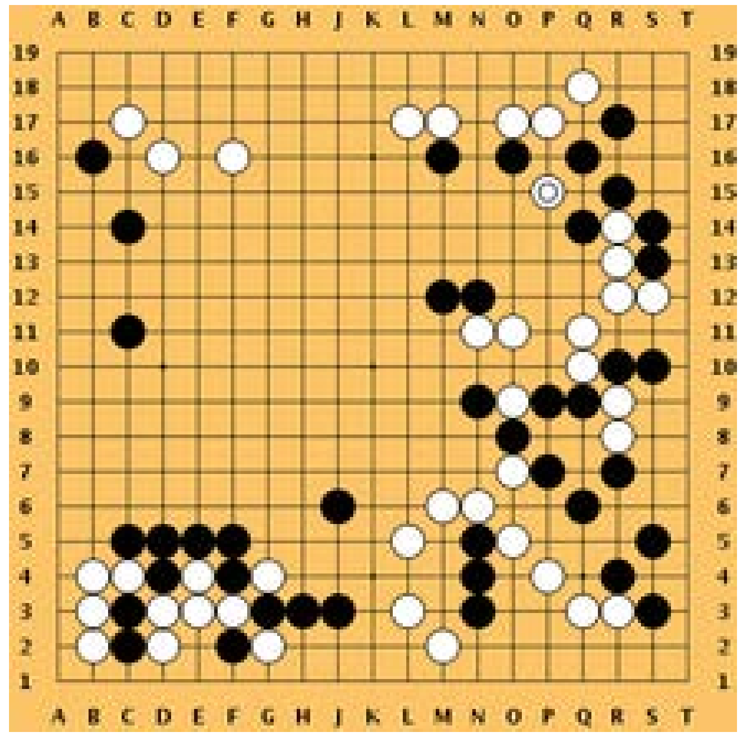
BUT:
Reward is rare:
‘sparse feedback’ after
a long action sequence

1. Review: Deep reinforcement learning

Chess



Go

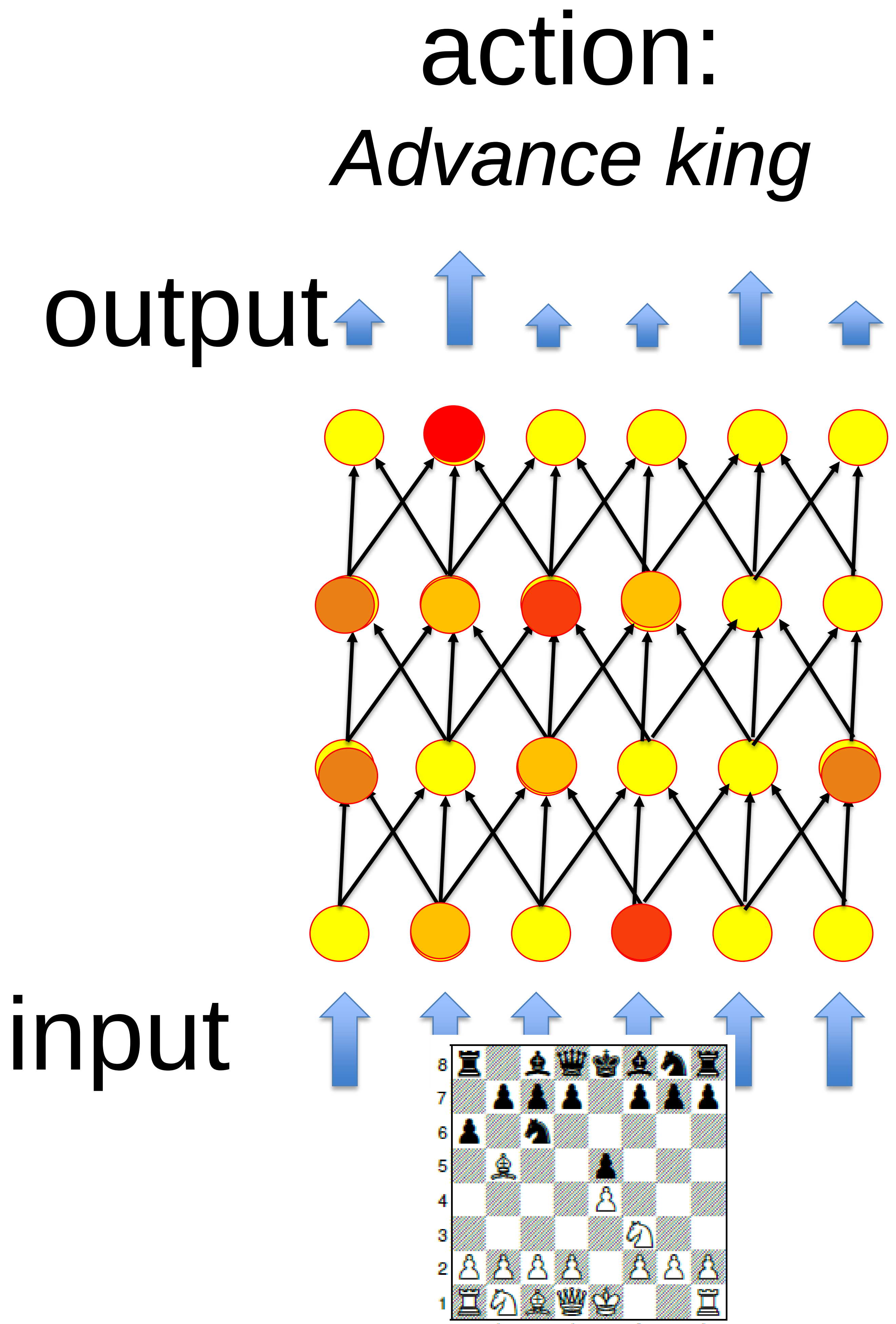


Artificial neural network (*AlphaZero*) discovers different strategies by playing against itself.

In Go, it beats Lee Sedol

1. Review: Deep reinforcement learning

Network for choosing action

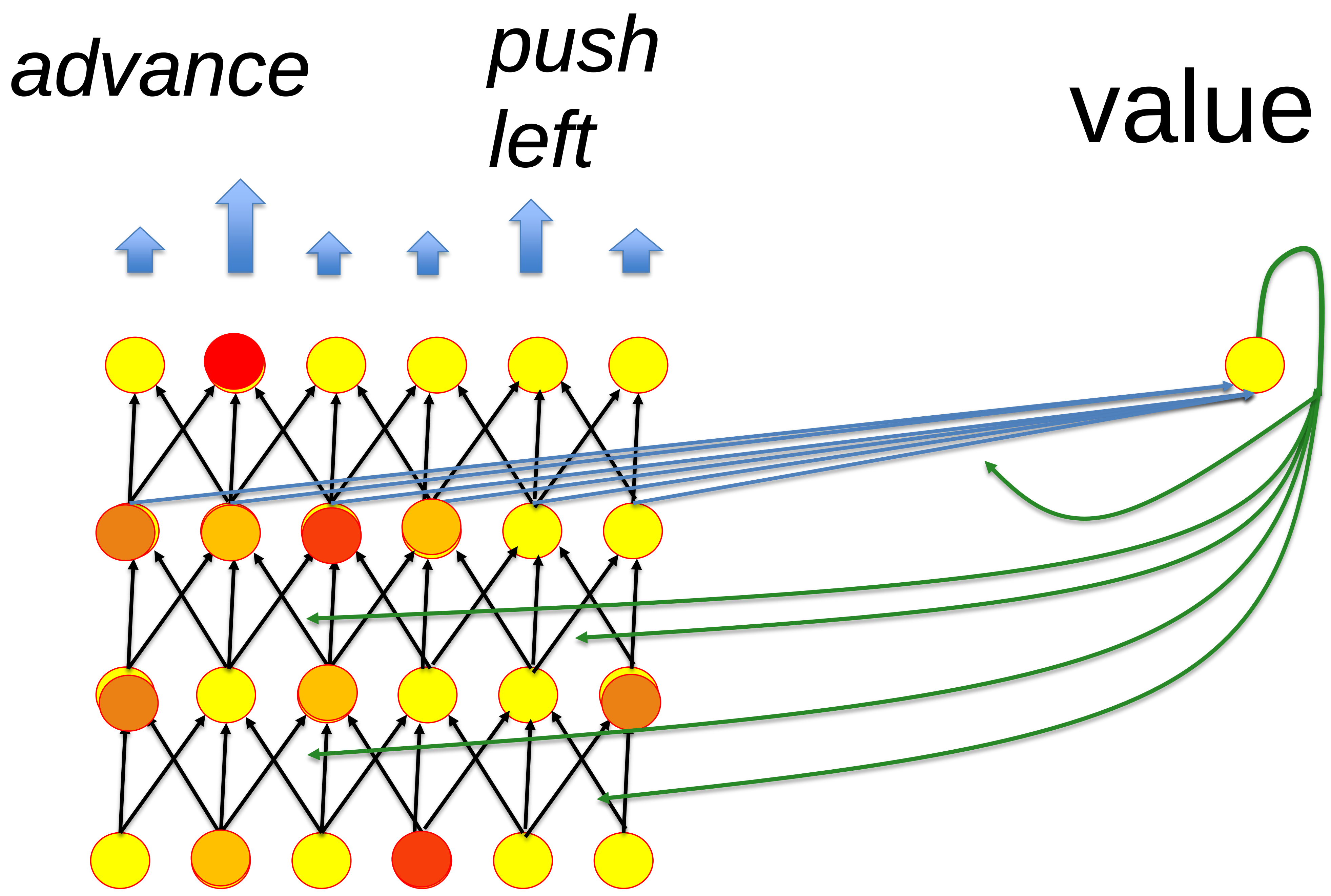


Today:

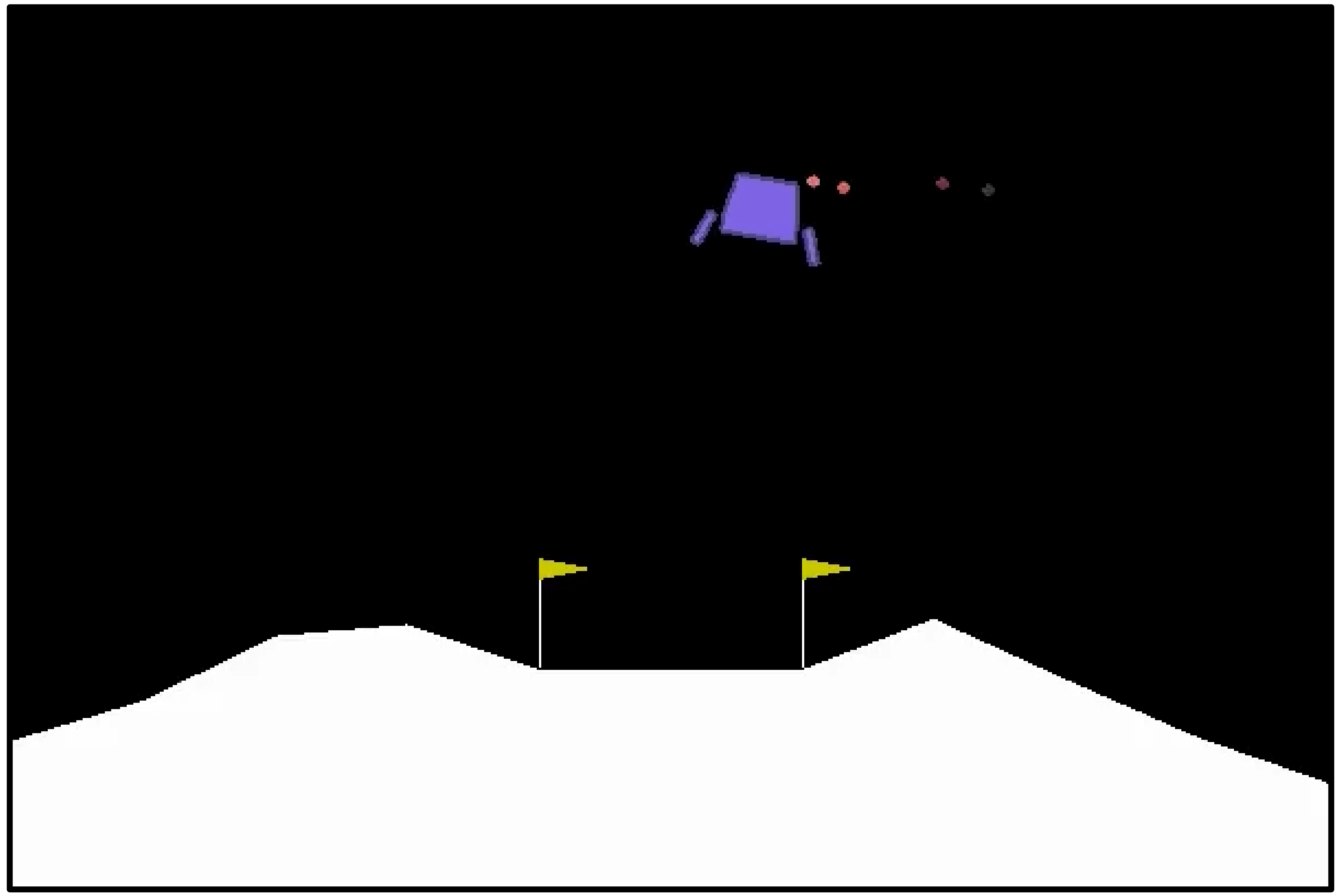
- How can we set-up such a network?
- What is the error function?
- How can we optimize weights?

1. Deep Reinforcement Learning: Lunar Lander (miniproject)

actions



Aim: land between poles



Policy gradient → Next week

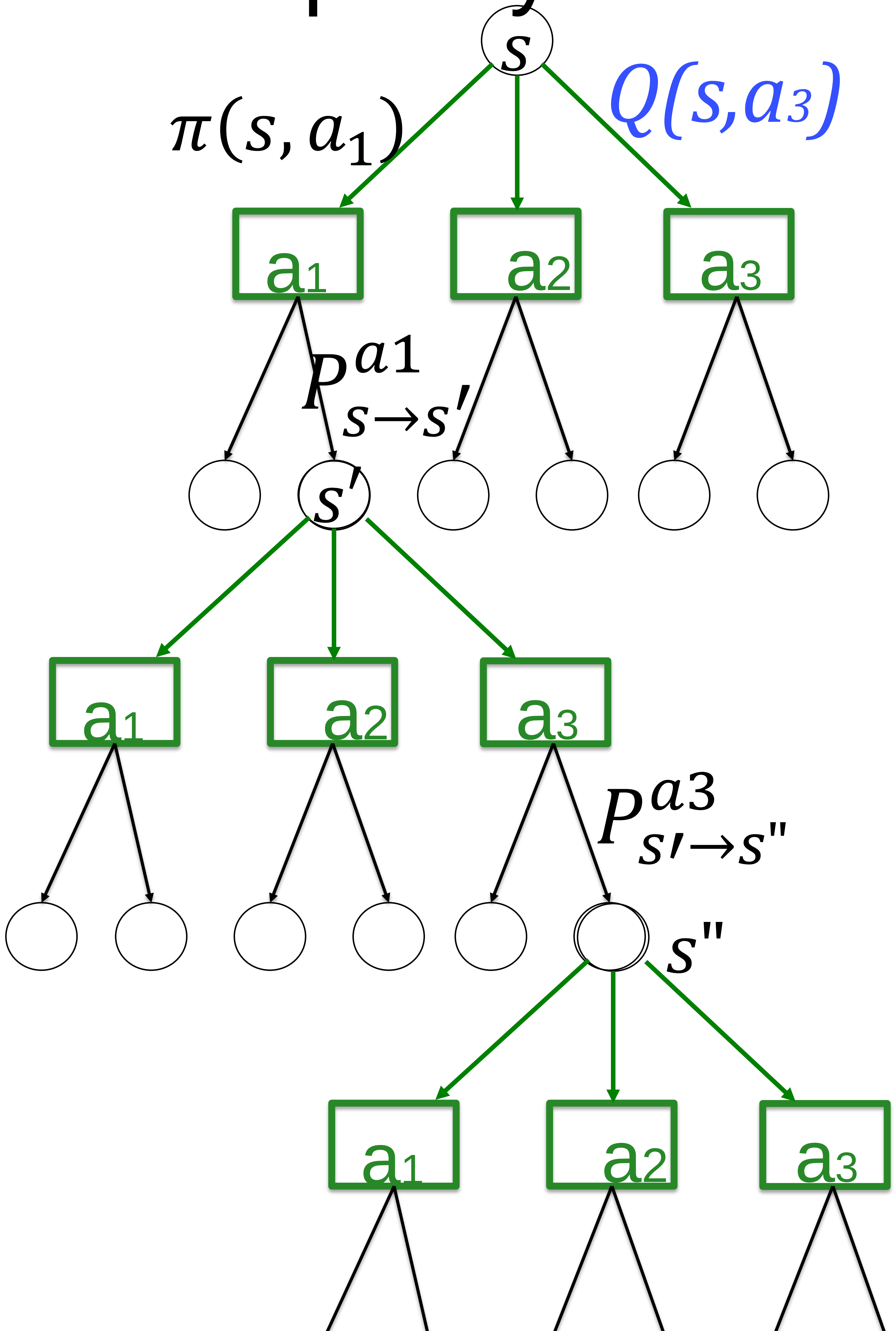
1. Review: Branching probabilities and policy

Policy $\pi(s, a)$

probability to choose action a in state s

$1=\sum_{a'} \pi(s, a')$

Examples of policy:
-epsilon-greedy
-softmax



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

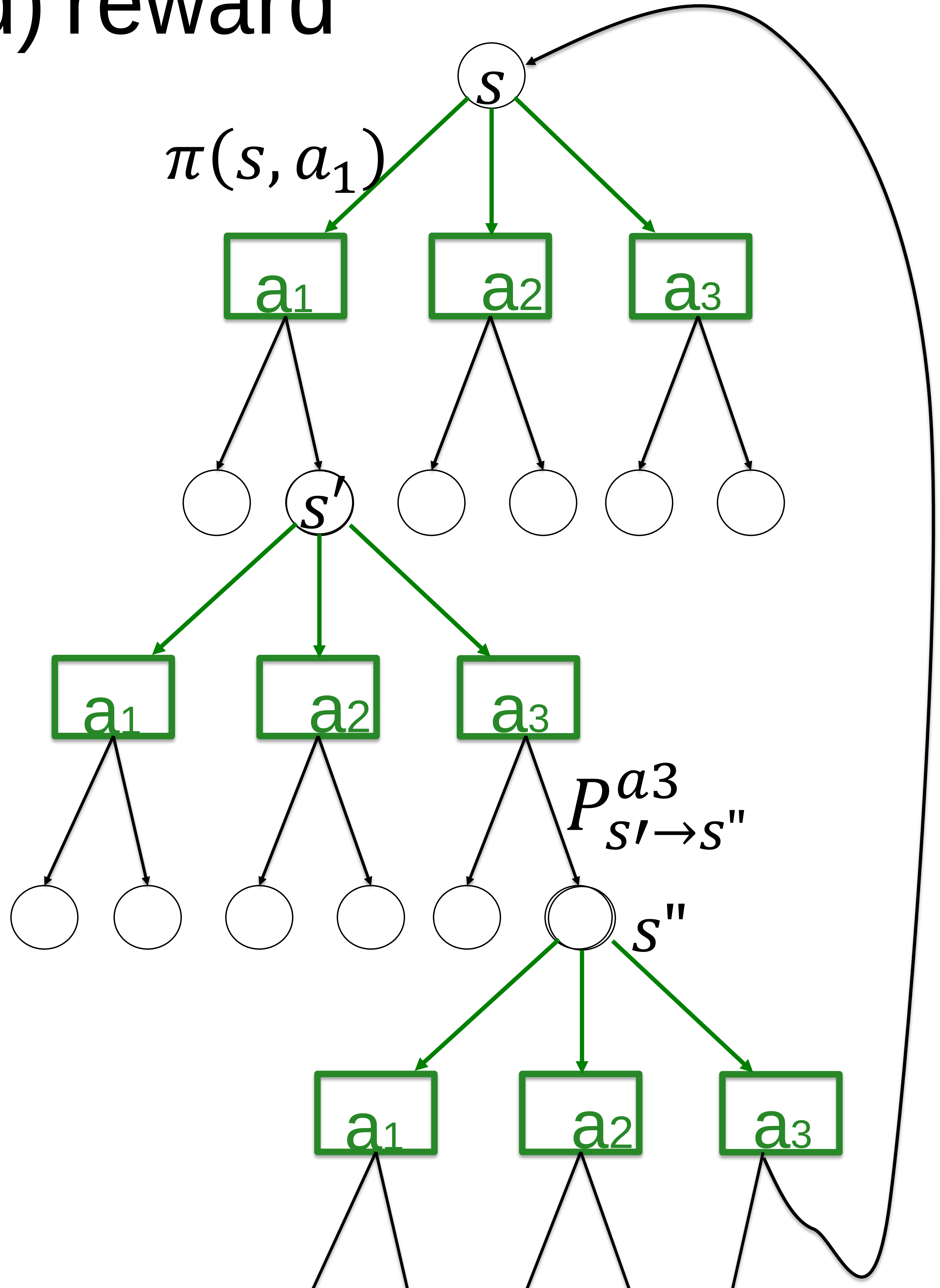
probability to end in state s' taking action a in state s

1. Review Total expected (discounted) reward

Starting in state s with action a

$Q(s,a)$ =

$$\left\langle r_t + \gamma\, r_{t+1} + \gamma^2 r_{t+2} + \gamma^3\, r_{t+3} + \cdots \right\rangle$$



Discount factor: γ <1

-important for recurrent networks!

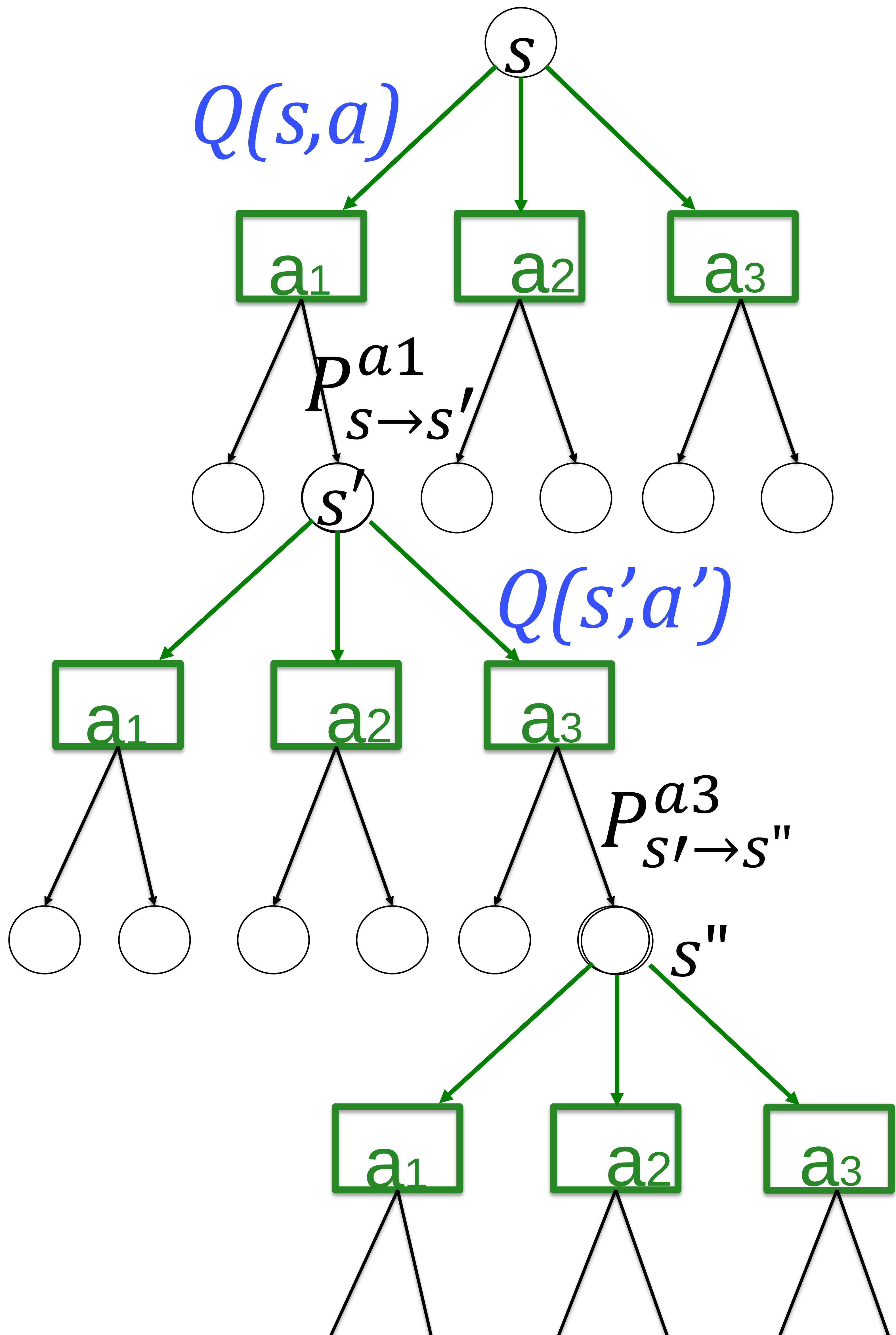
-avoids blow-up of summation

-gives less weight to reward in **far** future

1. Review: Bellman equation

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

Bellman equation =
value consistency of
neighboring states



5. Review: SARSA algorithm

Initialise Q values
Start from initial state s

1) being in state **s**
choose action **a**
[according to policy $\pi(s, a)$]

2) Observe reward **r**
and next state **s'**

3) Choose action **a'** in state s'
[according to policy $\pi(s, a)$]

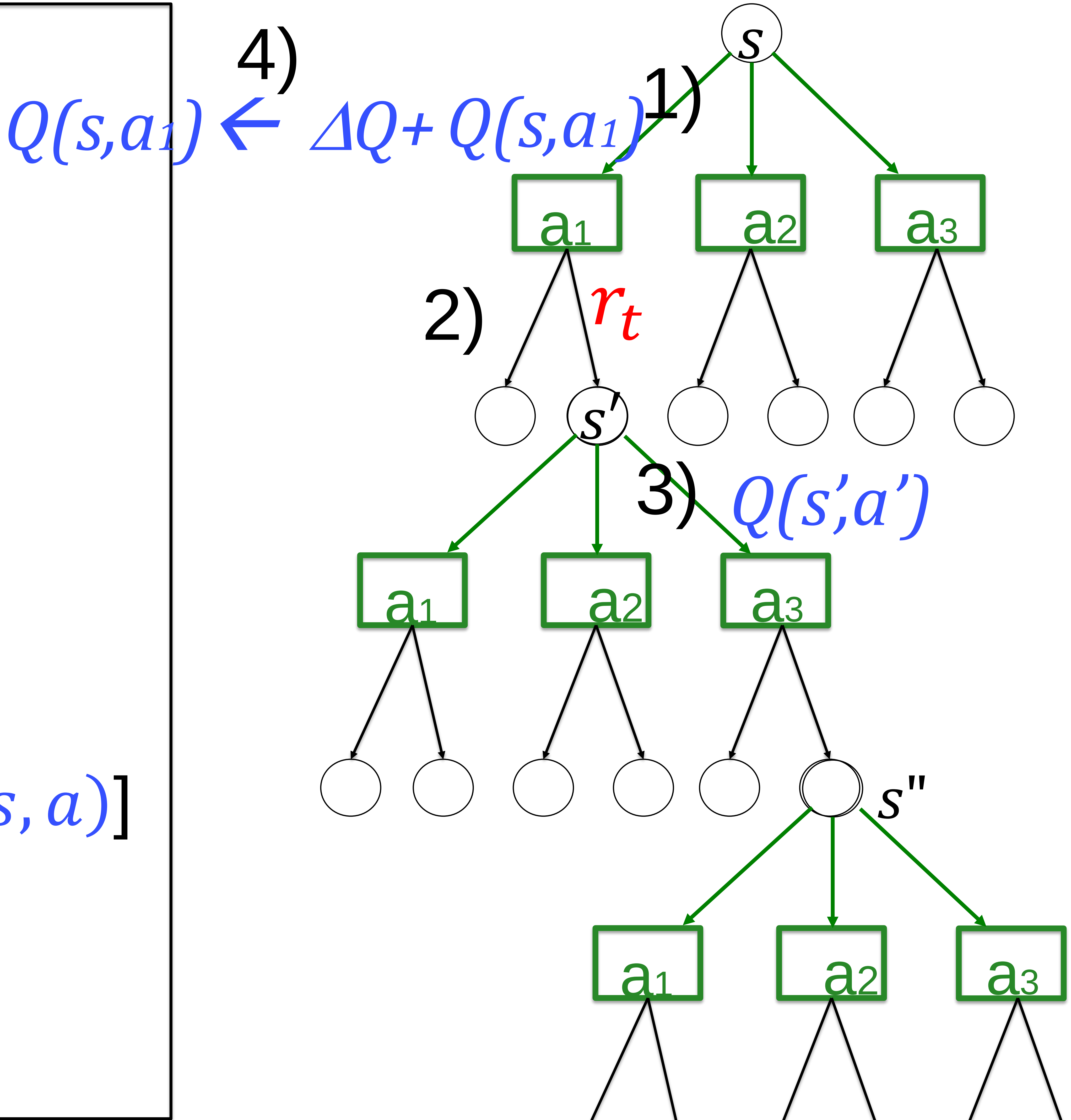
4) Update with SARSA update rule

$$\Delta Q(s, a) = \eta\ [r_t + \gamma Q(s', a') - Q(s, a)]$$

5) set: s ← s'; a ← a'

6) Goto 1)

Stop when all Q-values have converged

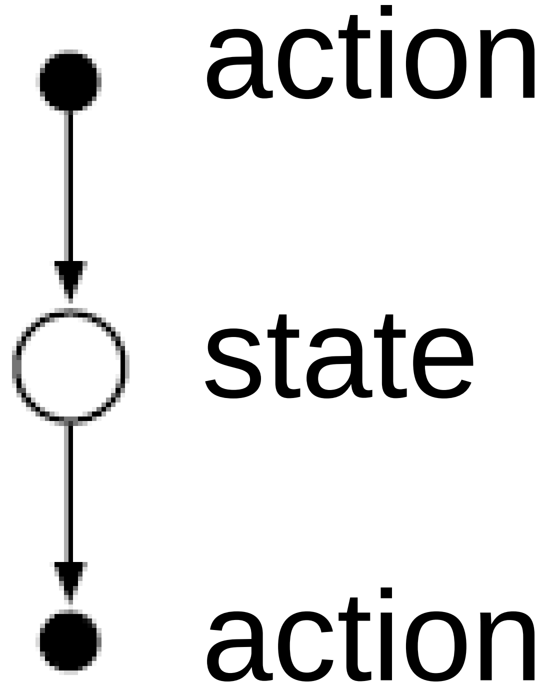


Blackboard:
Backup diagram

5. Review: SARSA algorithm

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal



Sarsa

pick next action a' before you update

Sutton and Barto, Ch. 6.4

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Artificial Neural Networks: Lecture 9

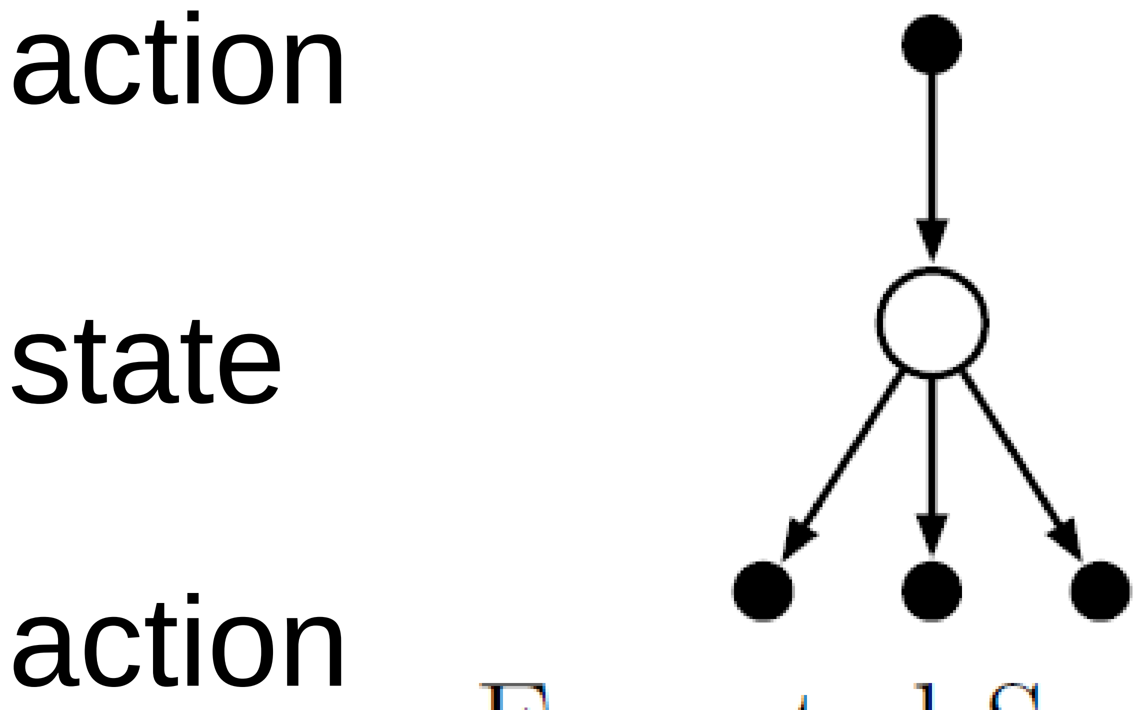
Variants of TD-learning methods and continuous space

1. Review and introduction of BackUp diagrams
2. Variations of SARSA

2. Expected SARSA

Expected SARSA for estimating $Q \approx q_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1},a) - Q(S_t,A_t)\right]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal



Expected Sarsa

Sutton and Barto, Ch. 6.6

2. Bellman equation

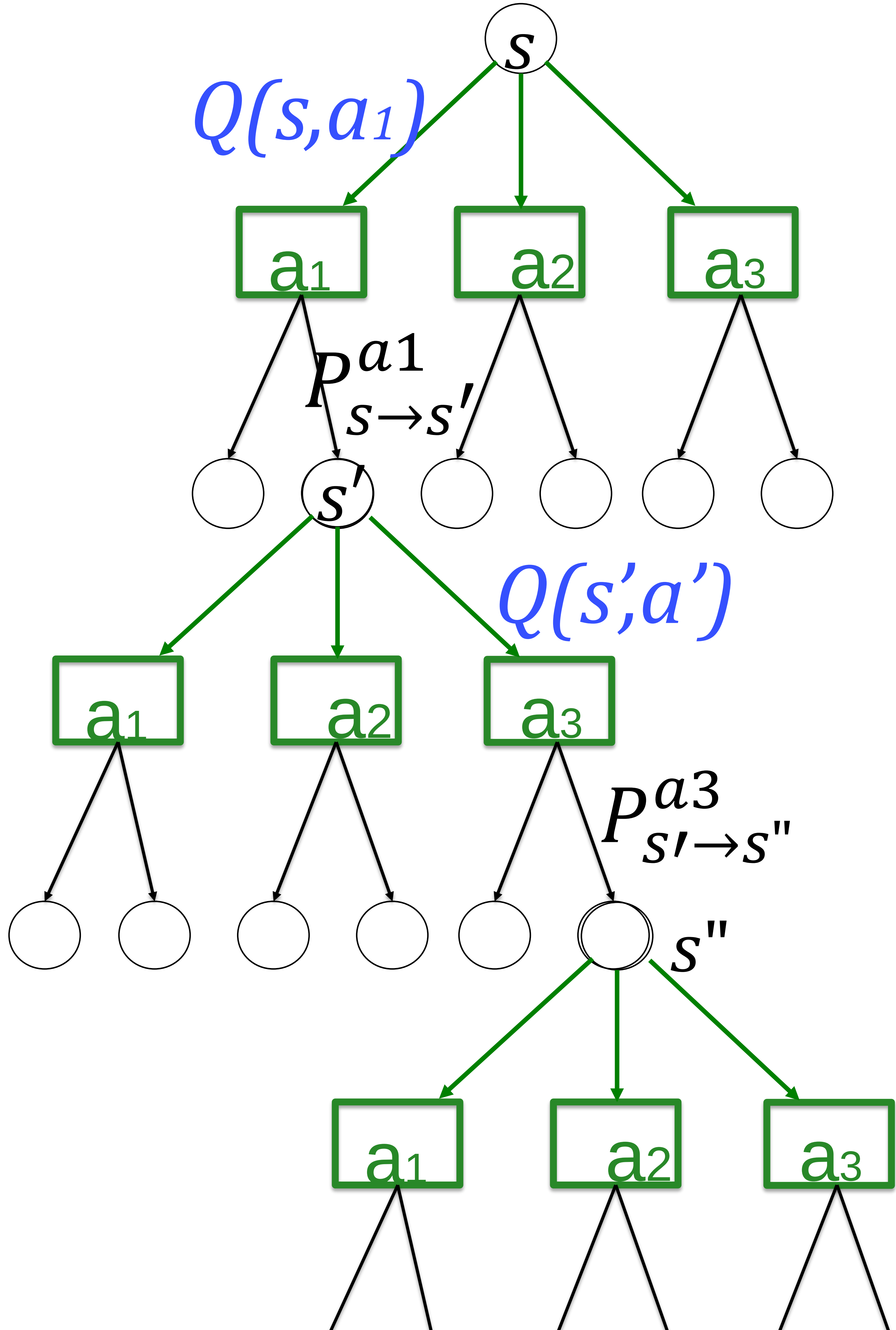
$$Q(s,a) = \sum_{s'} P^a_{s\to s'}\left[R^a_{s\to s'} + \gamma \sum_{a'} \pi(s',a') Q(s',a') \right]$$

Bellman equation = value consistency of neighboring states

Remark:

Sometimes Bellman equation is written for greedy policy: $\pi(s,a) = \delta_{a,a*}$

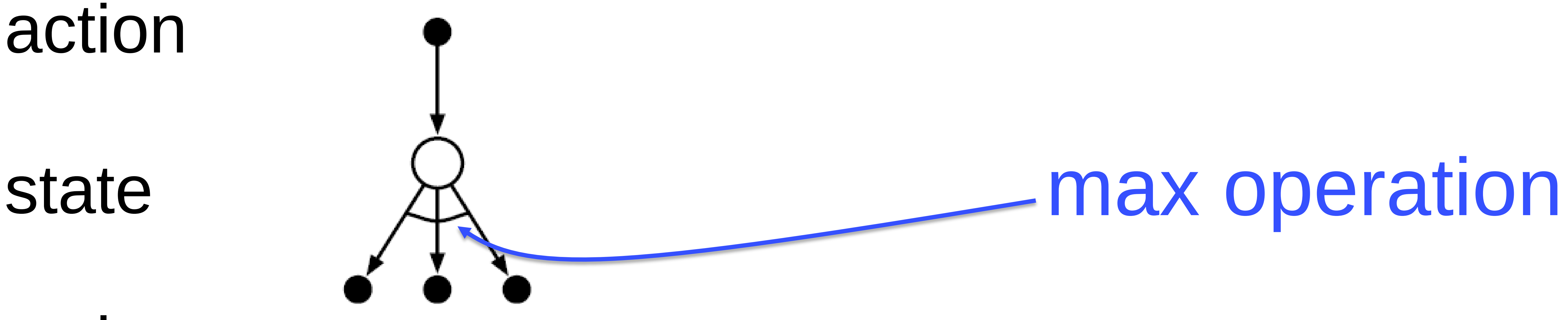
with action $a^* = \max_{a'} Q(s,a')$



2. Q-Learning algorithm

Q-learning (off-policy TD control) for estimating

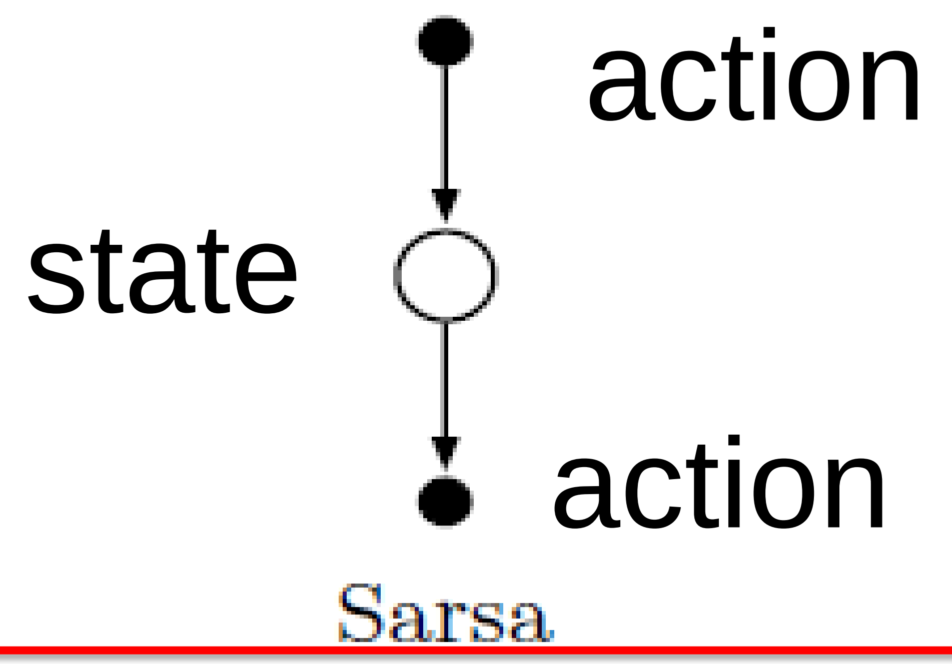
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$
    until S is terminal



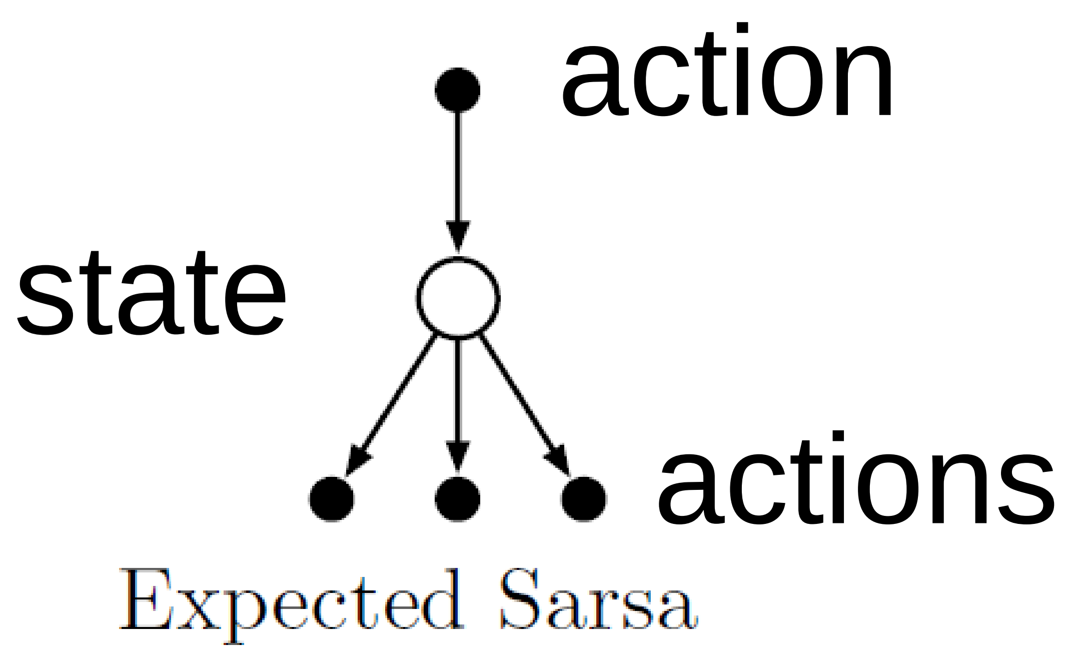
Q-learning

Sutton and Barto, Ch. 6.5

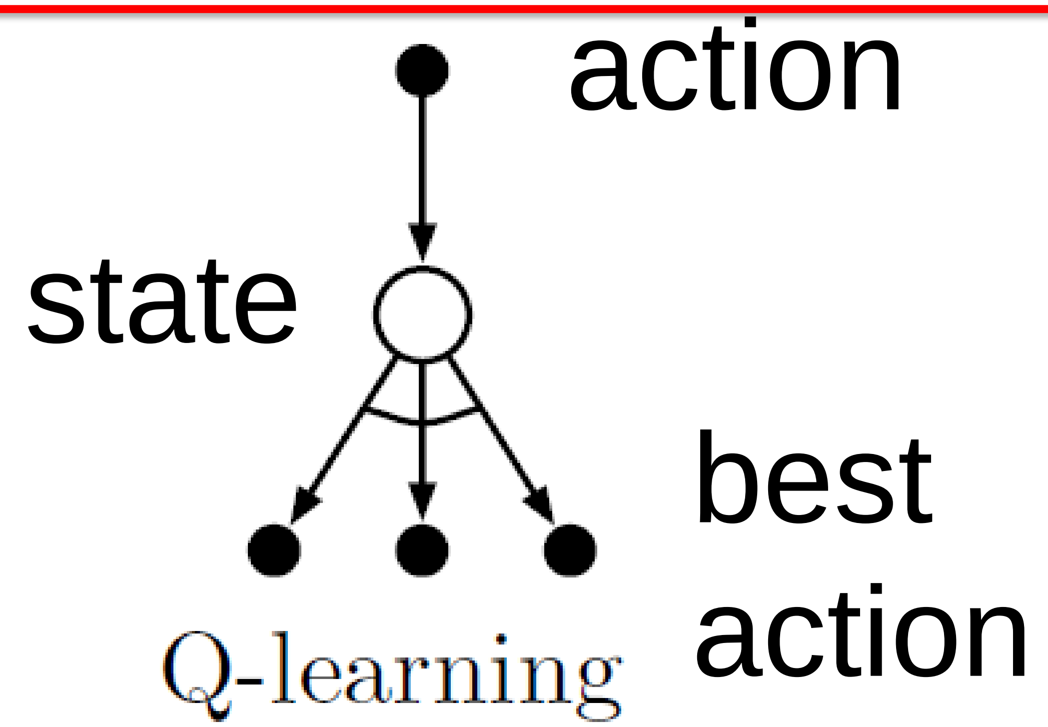
2. SARSA and related algorithms



SARSA: you actual perform **next** action, and then you update Q(s,a)



Exp. SARSA: you look ahead and average over **potential next** actions to update Q(s,a) and then you update Q(s,a)



Q-learning: you look ahead and **imagine greedy next** action to update Q(s,a) (but you perform the actual next action based on your current policy)

Artificial Neural Networks: Lecture 9

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Variants of TD-learning methods and continuous space

1. Review
2. Variations of SARSA
3. TD – learning (Temporal Difference)

3. TD-learning as bootstrap estimation

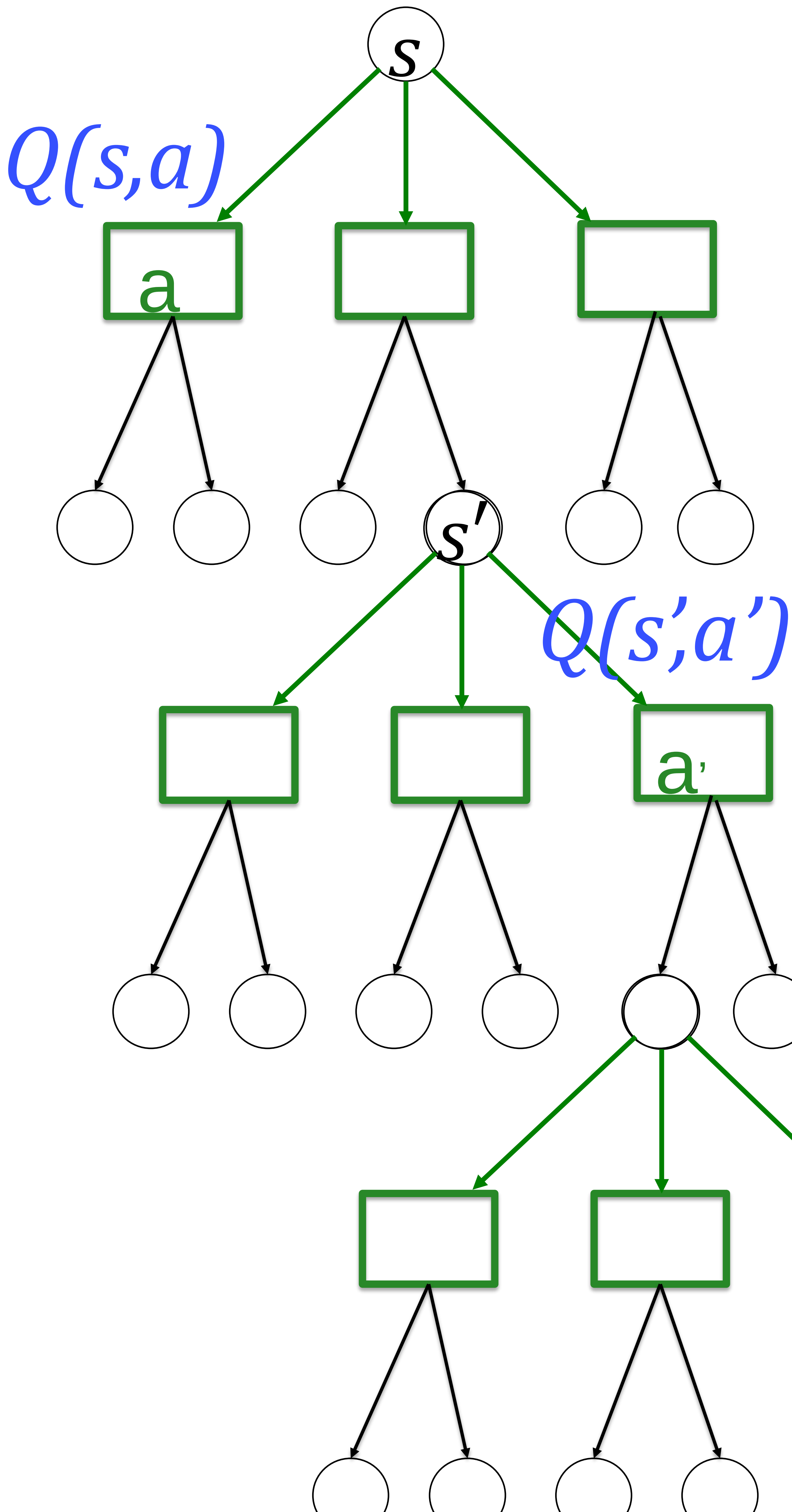
'bootstrap': summary of previous information

Temporal Difference

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

Bellman equation = value consistency of neighboring states

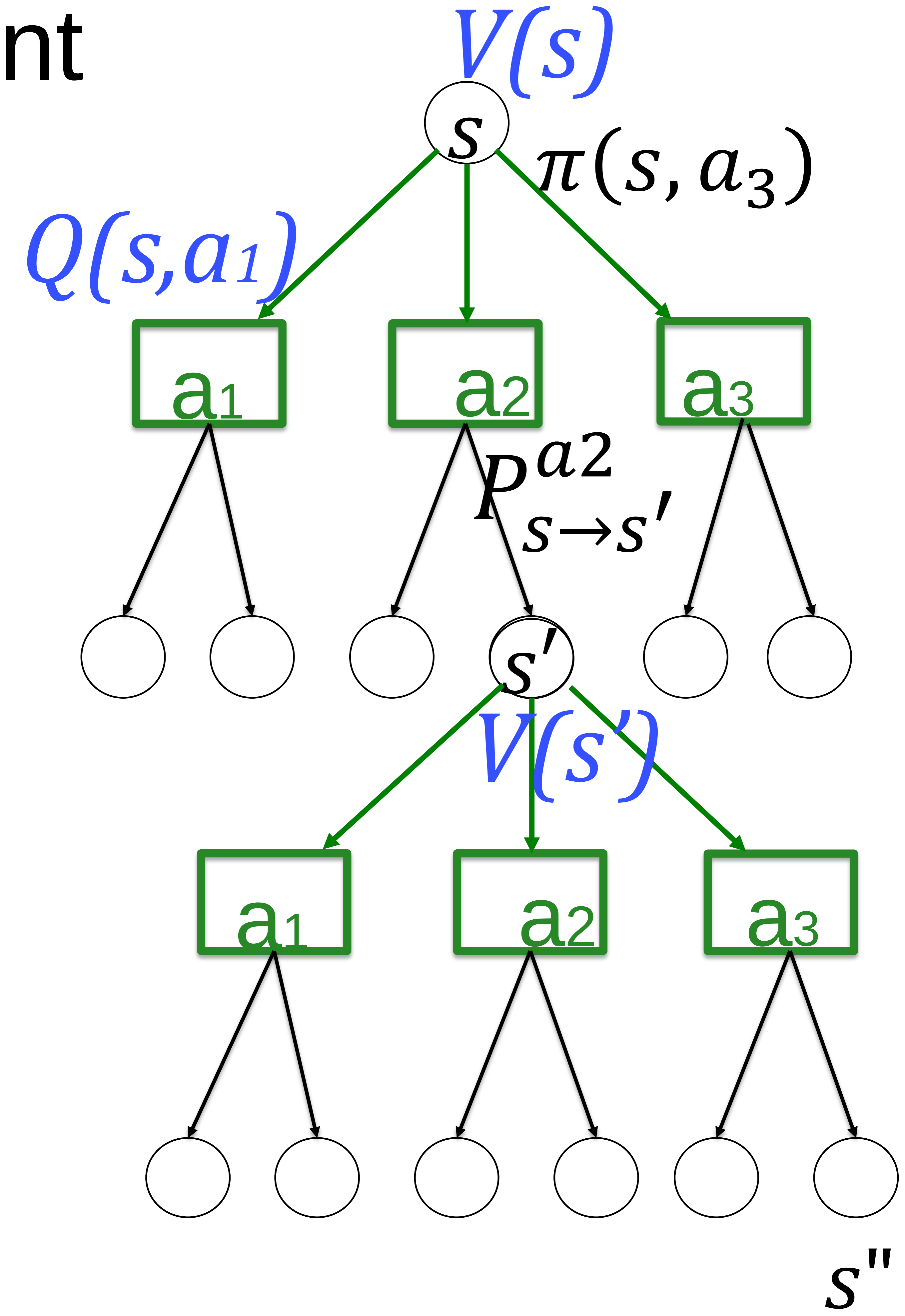
Neighboring states → neighboring time steps



3. State-values V

Value $V(s)$ of a state s
= total (discounted) expected reward the agent
gets starting from state s

$$V(s) = \sum_a \pi(s,a) Q(s,a)$$



Bellman equation for *V(s)*

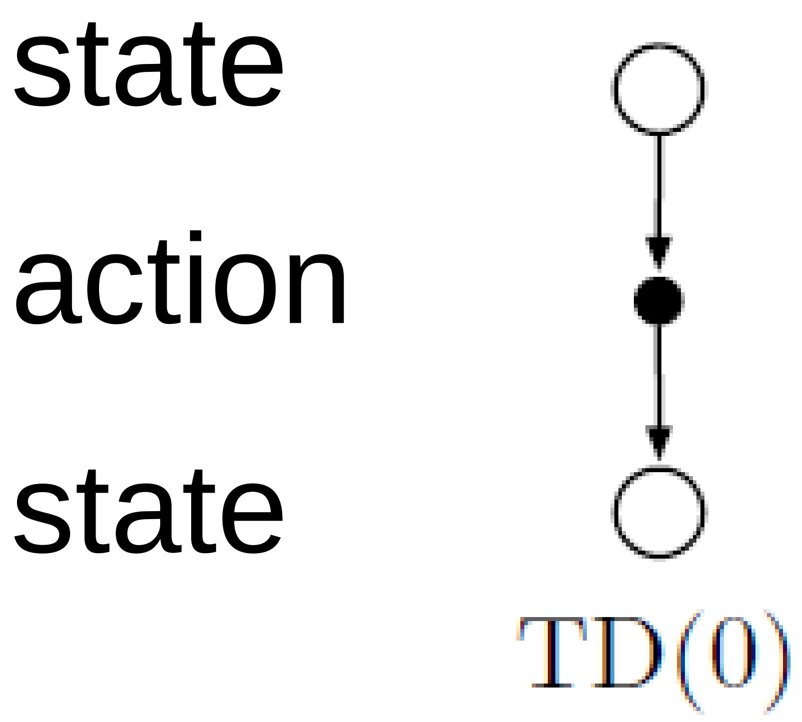
$$V(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{s\rightarrow s'} [R^a_{s\rightarrow s'} + \gamma V(s')]$$

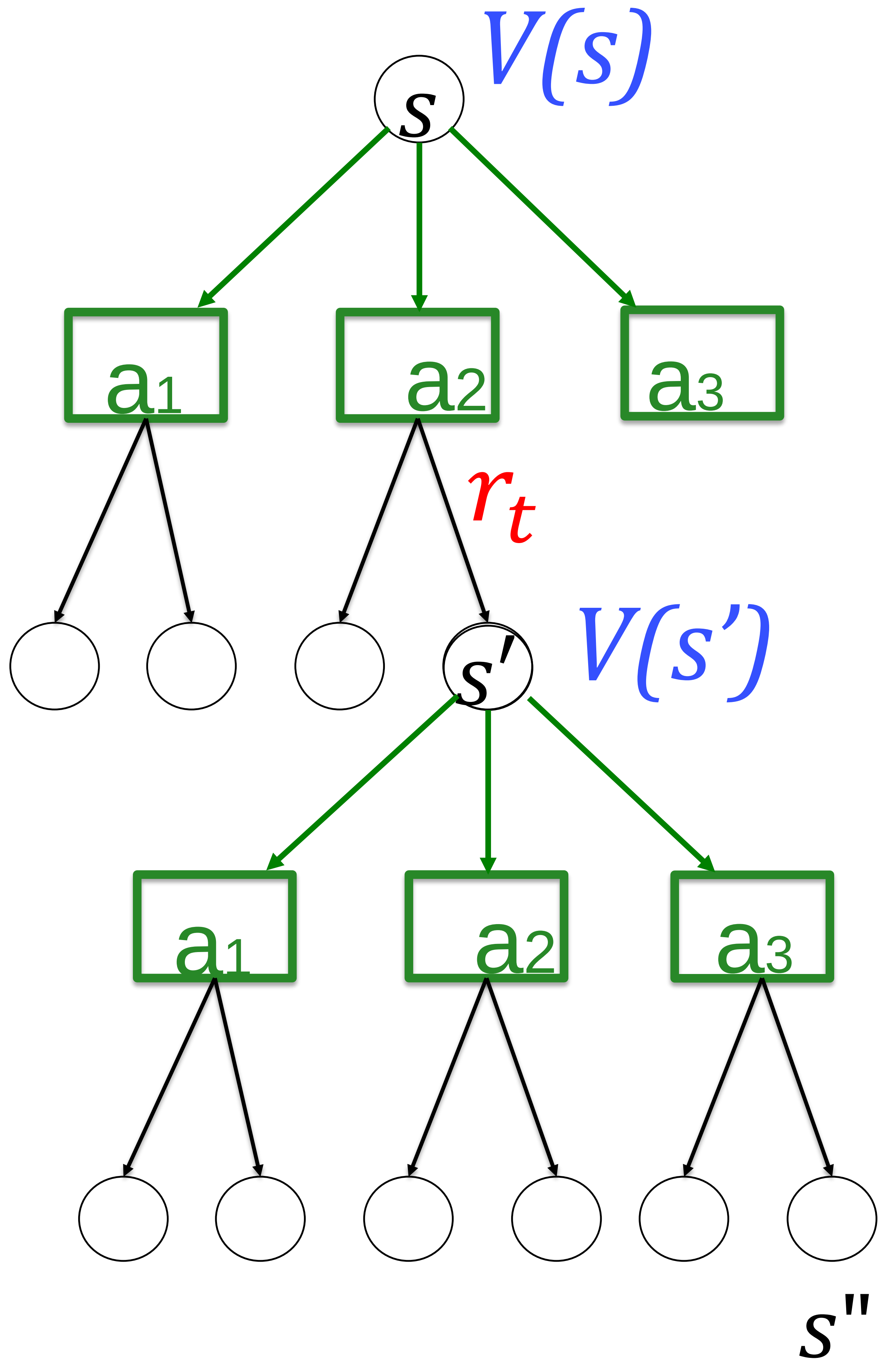
3. Standard TD-learning

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

$$\Delta V(s) = \eta\,[r_t + \gamma V(s', a') - V(s)]$$





Quiz: TD methods Rewards in Reinforcement Learning

[ ] SARSA is a TD method
[ ] expected SARSA is a TD method
[ ] Q-learning is a TD method
[ ] TD learning is an on-policy TD method
[ ] Q-learning is an on-policy TD method
[ ] SARSA is an on-policy TD method

3. TD-learning as bootstrap estimation

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

Bellman equation = value consistency of
neighboring states

Neighboring states → neighboring time steps

Temporal Difference Methods (TD methods)

- explore graph over time
- compare values (Q-values or V-values) at neighboring **time steps**
- ‘bootstrap’ estimation of values
- update after next time step, based on ‘temporal difference’

Artificial Neural Networks: Lecture 9

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Variants of TD-learning methods and continuous space

1. Review

2. Variations of SARSA

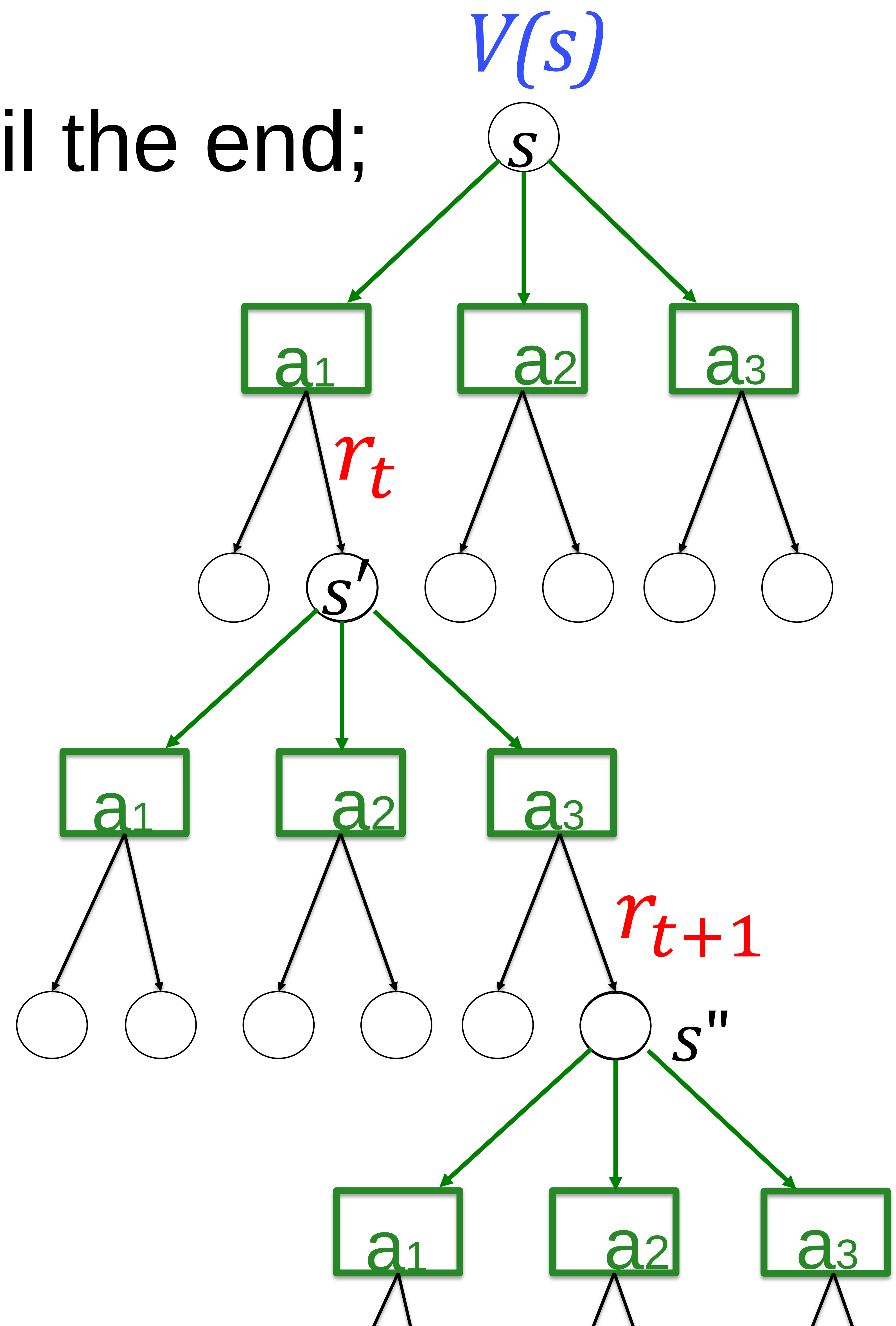
3. TD – learning (Temporal Difference)

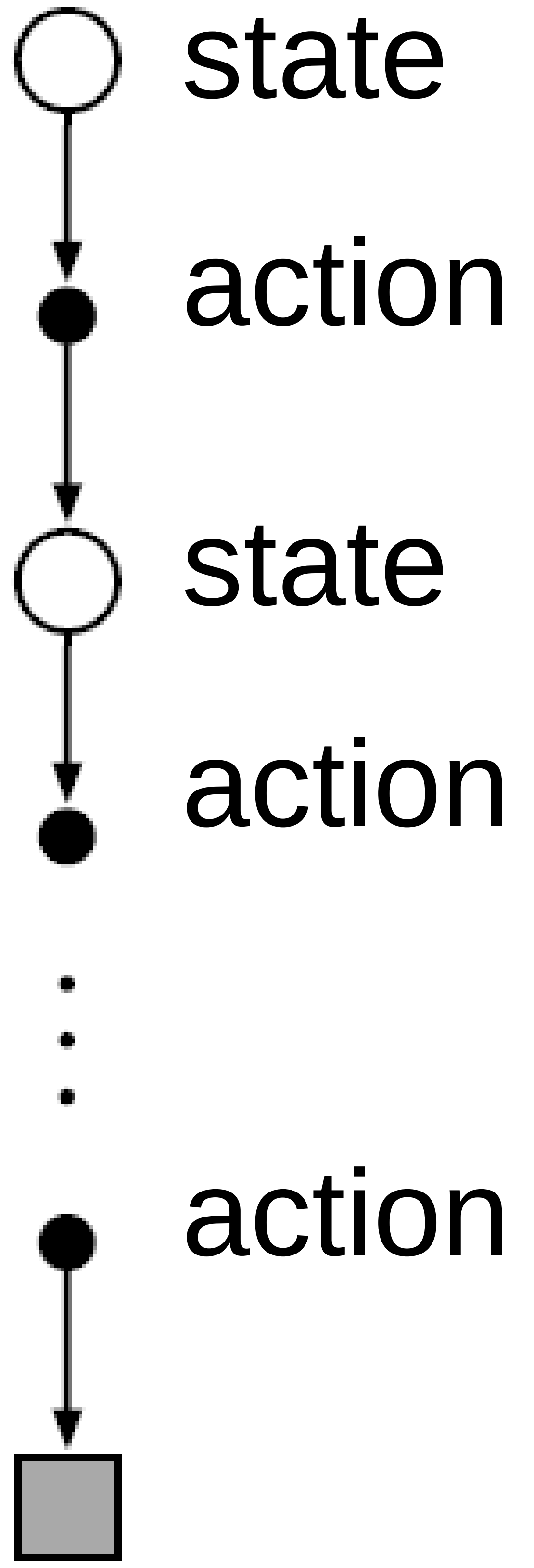
4. Monte-Carlo methods

4. Monte-Carlo Estimation

play a trial (episode) until the end;

then update, using
the total accumulated
reward (='return')





end of trial

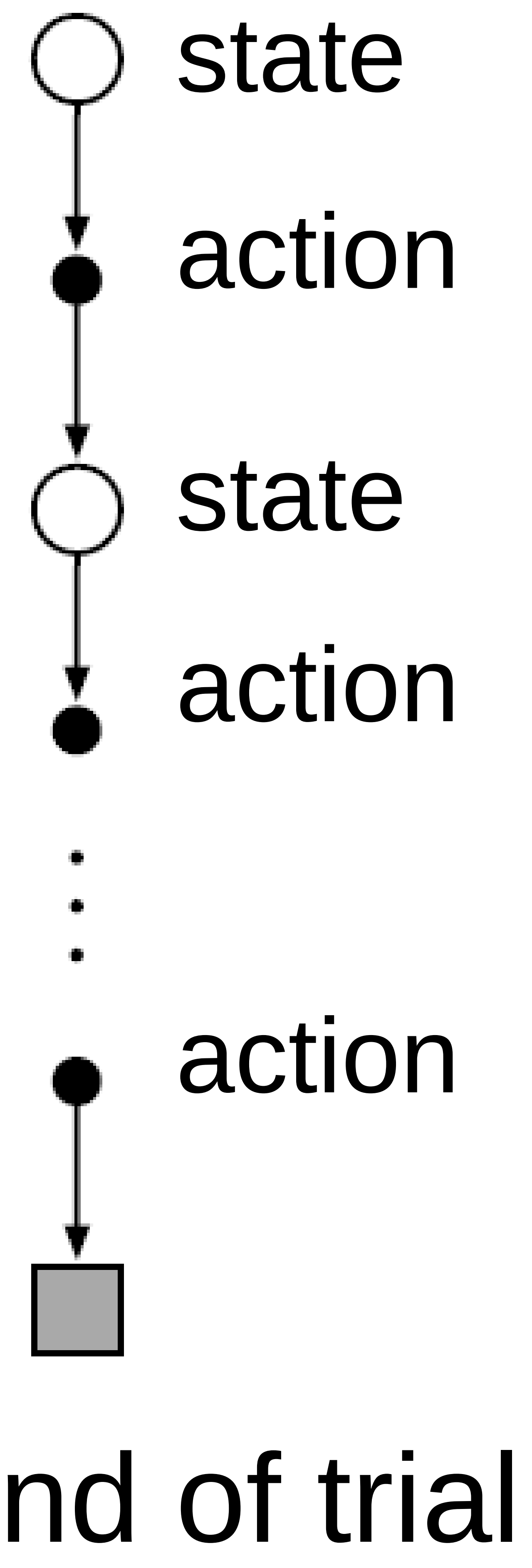
4. Monte-Carlo Estimation of V-values

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

First-visit MC prediction, for estimating $V \approx v_\pi$

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

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



4. Monte-Carlo Estimation of Q-values (batch)

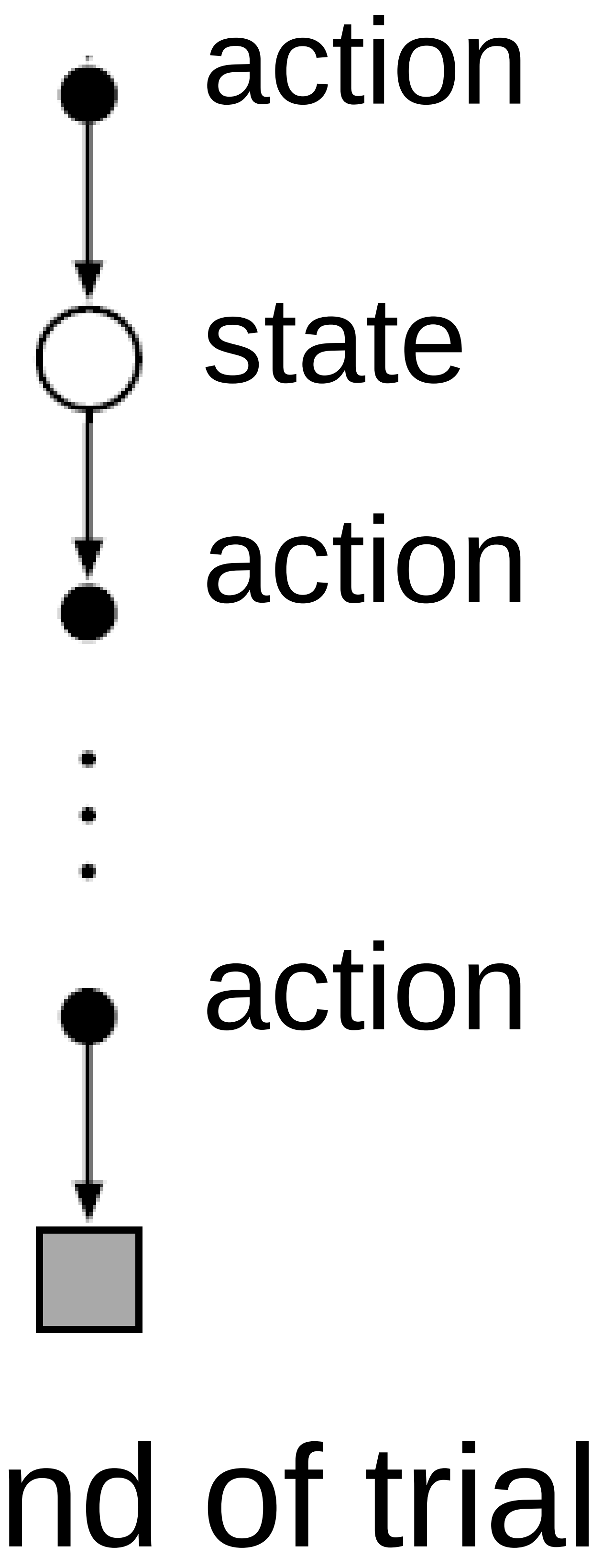
Start at a random state-action pair (s,a) (exploring starts)

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

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \leftarrow$ arbitrary
    $\pi(s) \leftarrow$ arbitrary
    $Returns(s,a) \leftarrow$ empty list

Repeat forever:
    Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability > 0
    Generate an episode starting from S_0, A_0, following π
    For each pair s, a appearing in the episode:
        $G \leftarrow$ the return that follows the first occurrence of s, a
        Append G to $Returns(s,a)$
        $Q(s,a) \leftarrow \text{average}(Returns(s,a))$
    For each s in the episode:
        $\pi(s) \leftarrow \arg\max_a Q(s,a)$



$$Q(s,a) = average[Return(s,a)]$$

Note: single episode also allows to update Q(s'a') of children

Oh, so many, many variants ….

Question:

We have three variants to estimate Q-values:

1) Q-learning (online, like in SARSA)
2) Monte-Carlo (Batch)
3) Bellman equation (Batch)

We have played N trials.

How do they rank?

Which one is best? → commitment:

write down 1 or 2 or 3

4. Monte-Carlo versus TD methods (Exercise 1, now, 8 min.)

example trials:

1: **s,a2**→s',a4→ r=0

2: s',a3→ r=1

3: s',a4→ r=0

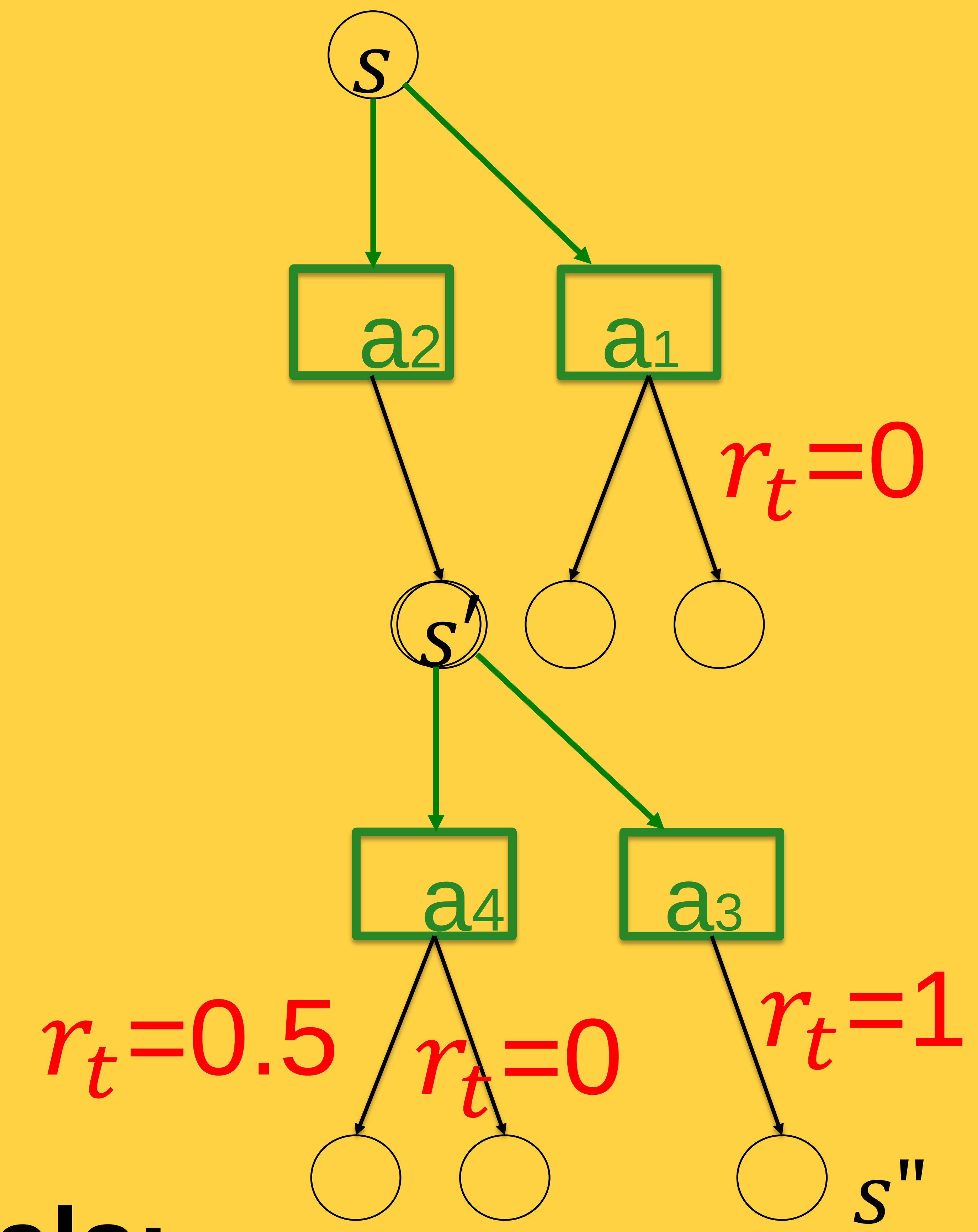
4: s',a3→ r=1

5: **s,a1** → r=0

6: s',a4→ r=0

7: s',a4→r=0.5

8: s',a3→r=0



Batch update of Q-values after all 8 trials:

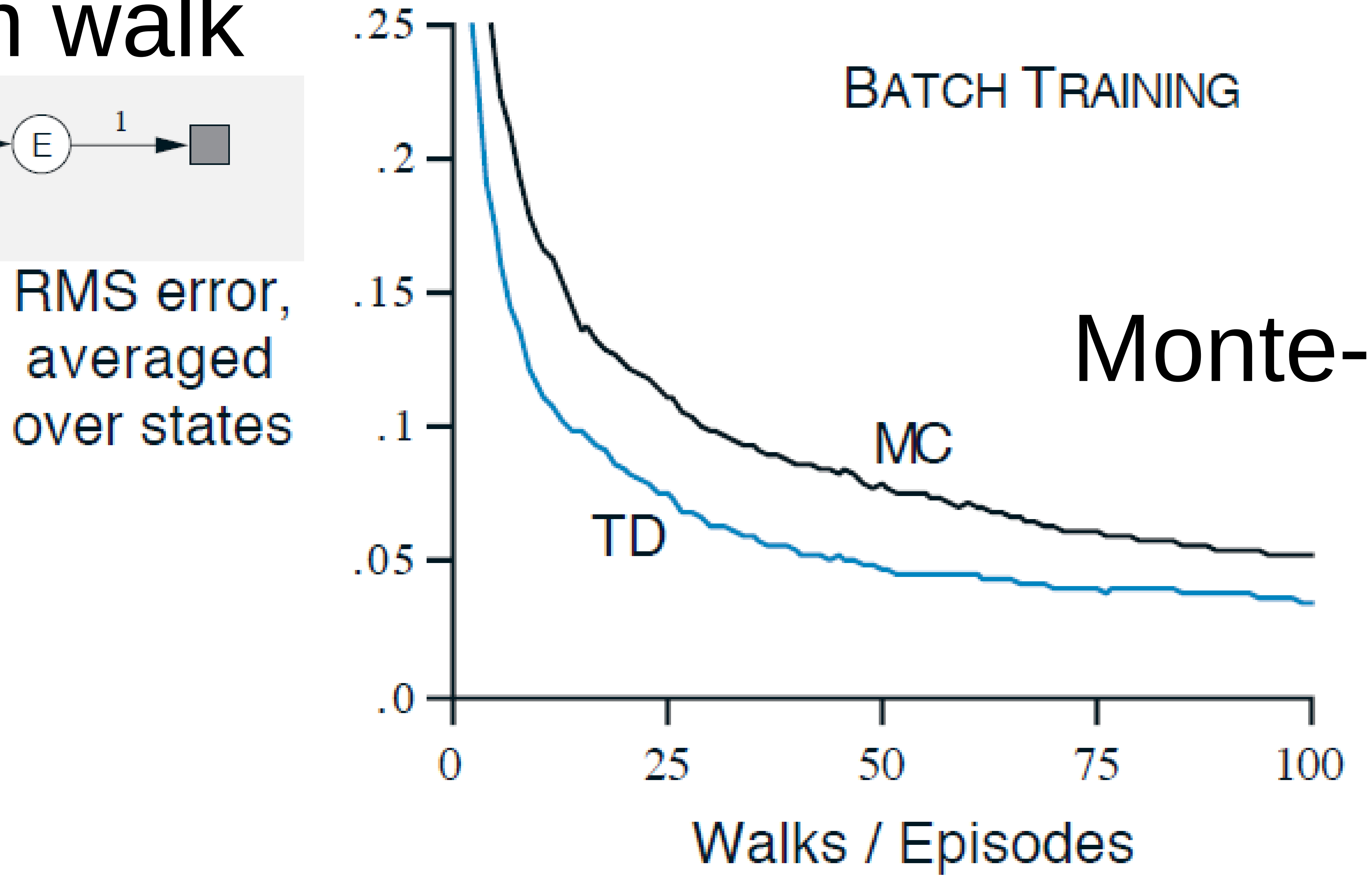
(i) Monte-Carlo: average over accumalted reward for each (a,s)

(ii) SARSA batch update (with eta =1/number of examples)

4. Monte-Carlo versus TD methods:

Comparison in **batch mode**: We have observed N episodes, and update (once) after these N episodes.

Example: 1d random walk



TD is better than Monte-Carlo

Figure 6.2: Performance of TD(0) and constant-α MC under batch training on the random walk task.■

4. Monte-Carlo versus TD methods:

TD is better than Monte Carlo

The averaging step in TD methods is more efficient (compared to Monte Carlo methods)
to propagate information back into the graph,
since information from different starting states is combined and compressed in a Q-value or V-value.

→similar to Dynamic programming

4. Monte-Carlo Estimation of Q-values

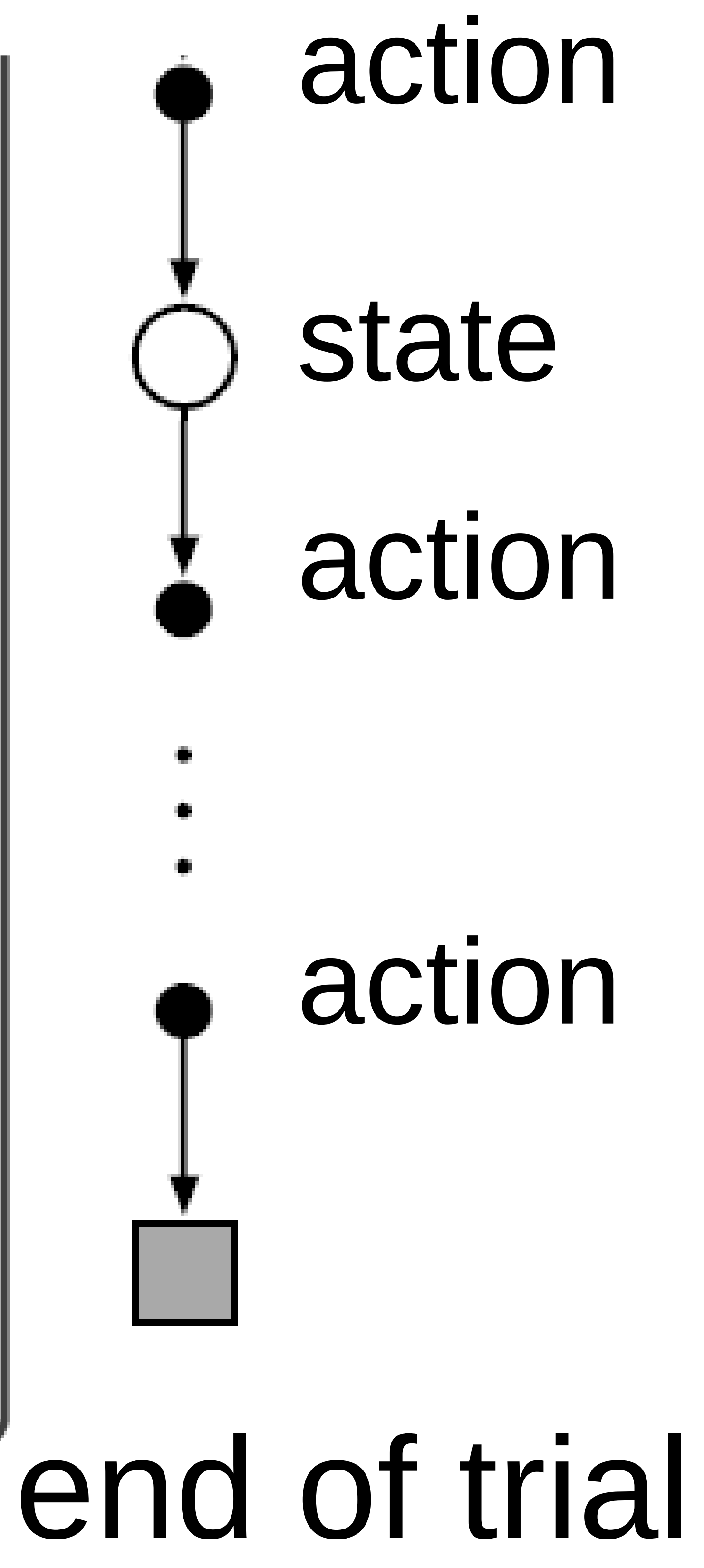
Combine epsilon-greedy policy with Monte-Carlo Q-estimates

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s, a) \leftarrow$ arbitrary
  $Returns(s, a) \leftarrow$ empty list
  $\pi(a|s) \leftarrow$ an arbitrary ε-soft policy (e.g., epsilon-greedy)

Repeat forever:
  (a) Generate an episode using π
  (b) For each pair s, a appearing in the episode:
    $G \leftarrow$ the return that follows the first occurrence of s, a
    Append G to $Returns(s, a)$
    $Q(s, a) \leftarrow$ average$(Returns(s, a))$
  (c) For each s in the episode:
    $A^* \leftarrow \arg\max_a Q(s, a)$ (with ties broken arbitrarily)
    For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$



Quiz: Monte Carlo methods

We have a network with 1000 states and 4 action choices in each state. There is a single terminal state.
We do Monte-Carlo estimates of total return to estimate Q-values

Our episode starts with (s,a) that is 400 steps away from the terminal state. How many return R(s,a) variables do I have to open in this episode?

[ ] one, i.e. the one starting at (s,a)
[ ] about 100 to 400
[ ] about 400 to 4000
[ ] potentially even more than 4000

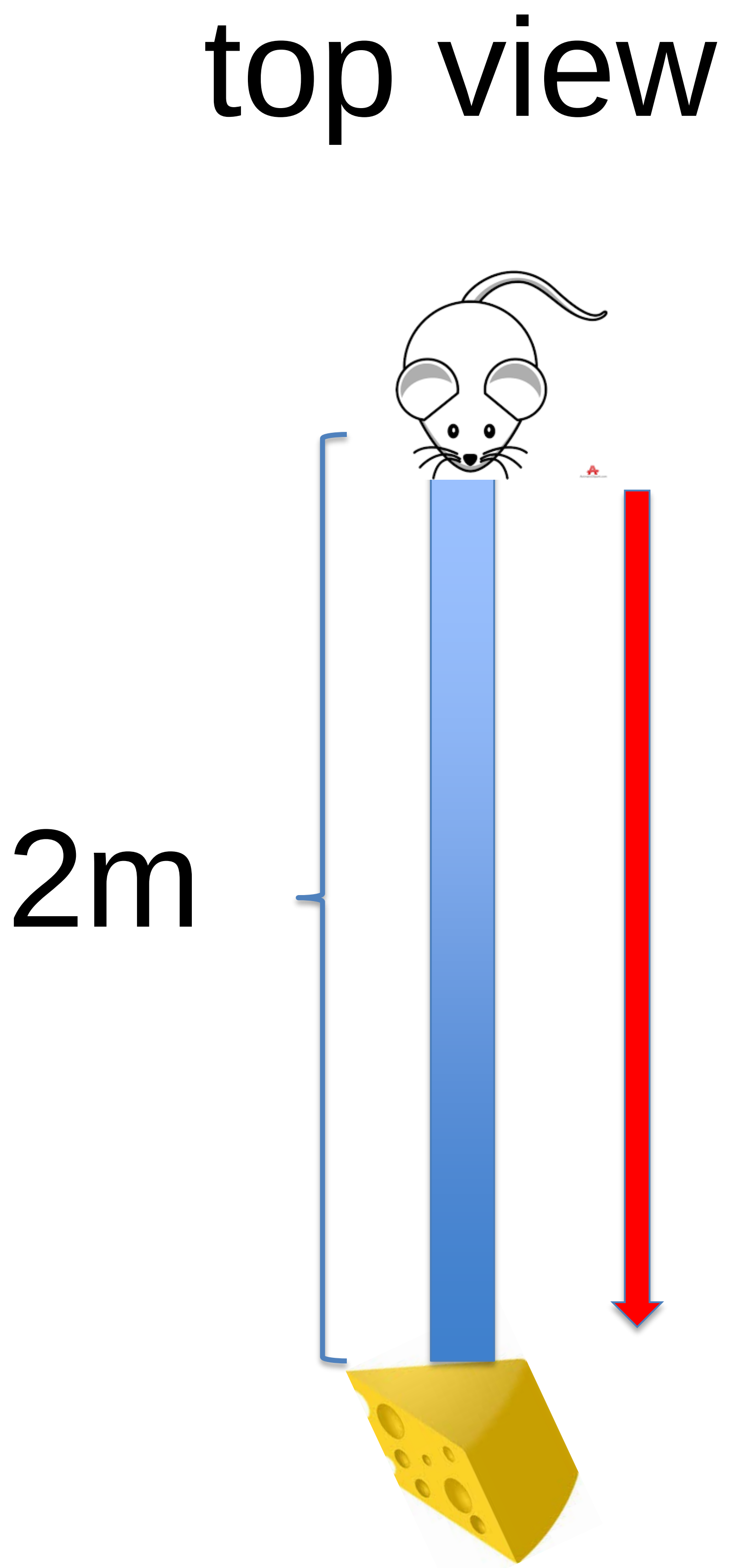
Artificial Neural Networks: Lecture 9

Wulfram Gerstner
EPFL, Lausanne, Switzerland

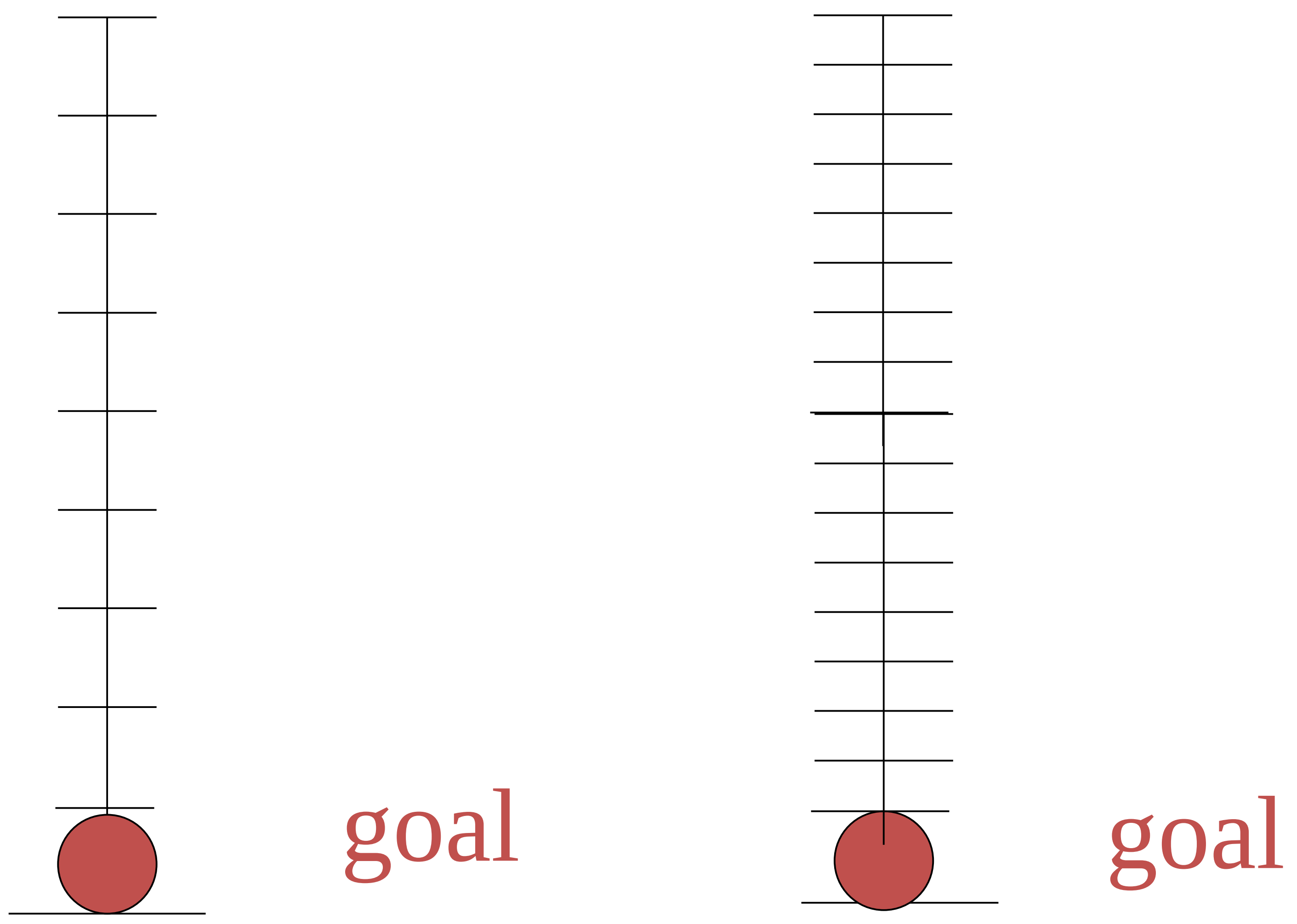
Variants of TD-learning methods and continuous space

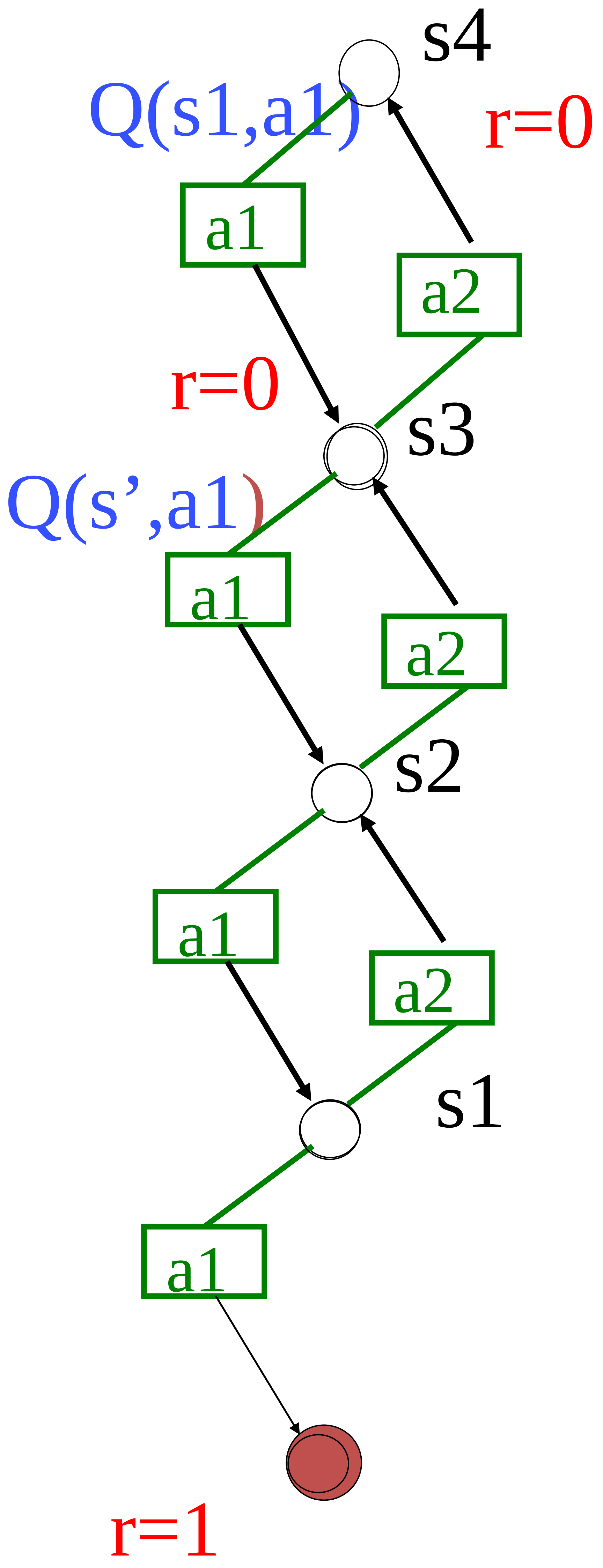
1. Review
2. Variations of SARSA
3. TD – learning (Temporal Difference)
4. Monte-Carlo methods
5. Eligibility traces and n-step methods

Exercise from last week: one-dimensional track



Discretize state





Exercise from last week: one-dimensional track

- **Update of Q values in SARSA**

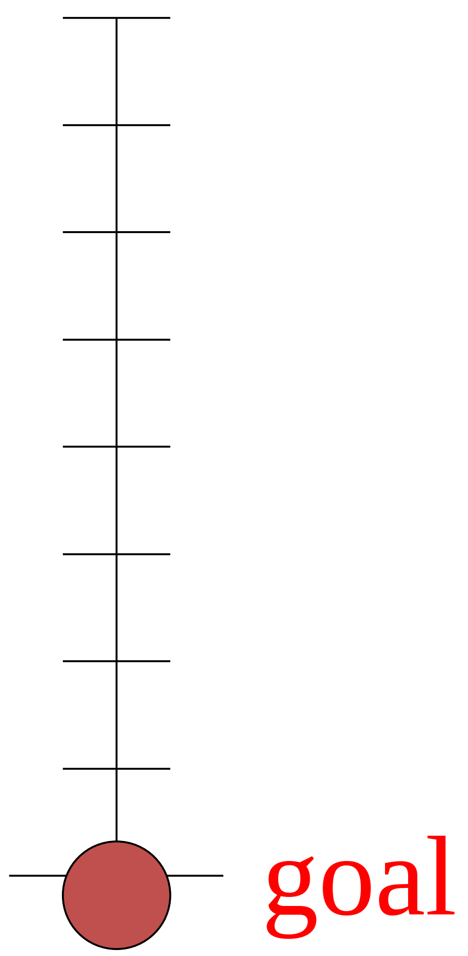
$$\Delta Q(s,a)=\eta\ [r-(Q(s,a)-Q(s',a'))]$$

- **policy for action choice:**

  Pick most often action

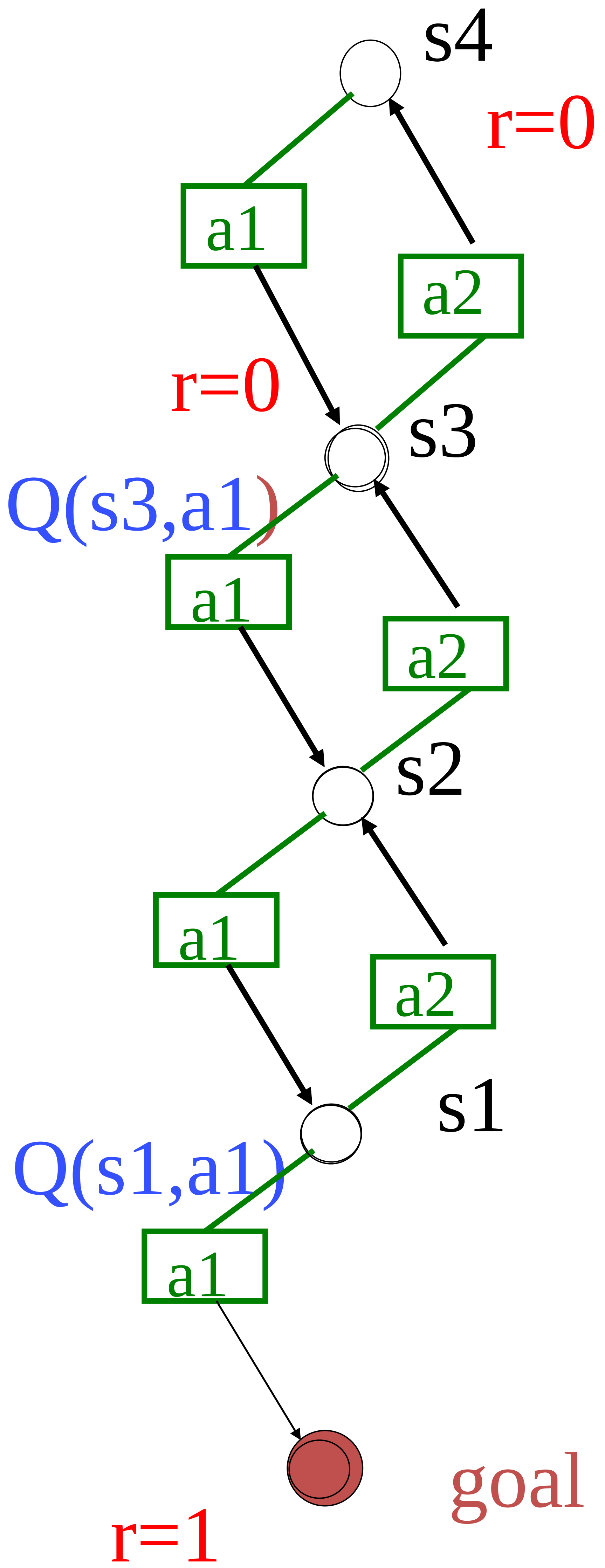
$$a_t^* = \arg\max_a Q_a(s,a)$$

Linear sequence of states.
Reward only at goal.
Actions are up or down.
Initialise Q values at 0. Start trials at top.



[ ] After 2 trials the Q-value Q(s1,a1)>0

[ ] After 2 trials the Q-value Q(s3,a1)>0



5. Problem of TD algorithms

Problem:

-'Flow of information' back from target is slow

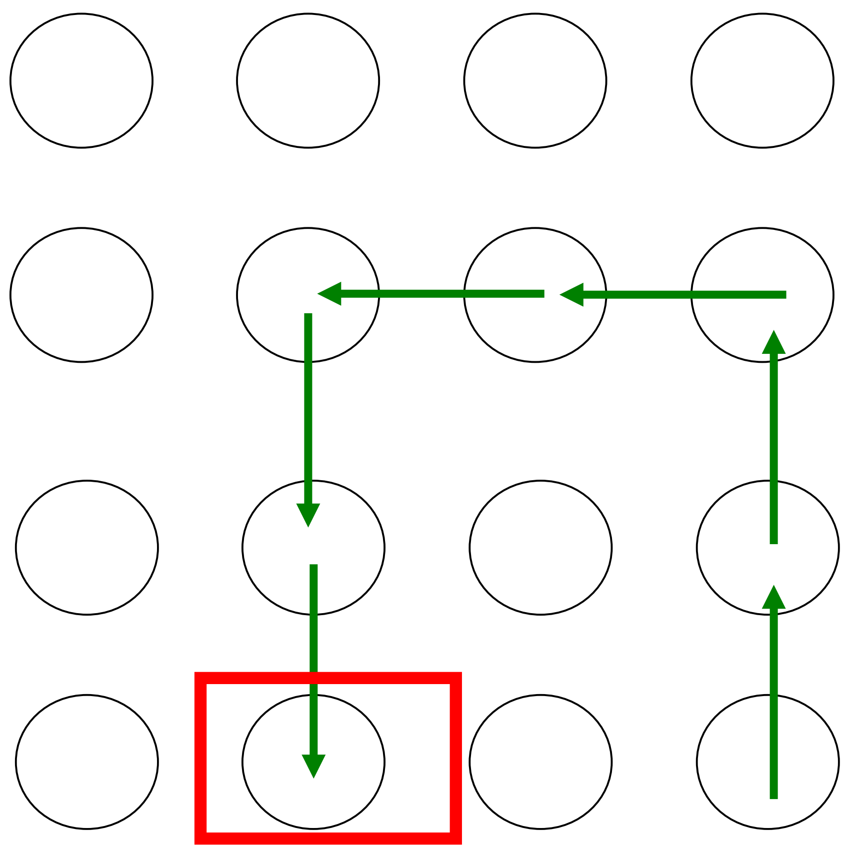
- information flows 1 step per complete trial
- 20 trials needed to get information 20 steps away from target

BUT:

- the discretization of states has been an arbitrary choice!!!

$\boxed{\rightarrow \text{Something is wrong with the discrete-state SARSA algo}}$

5. Solution 1: Eligibility Traces

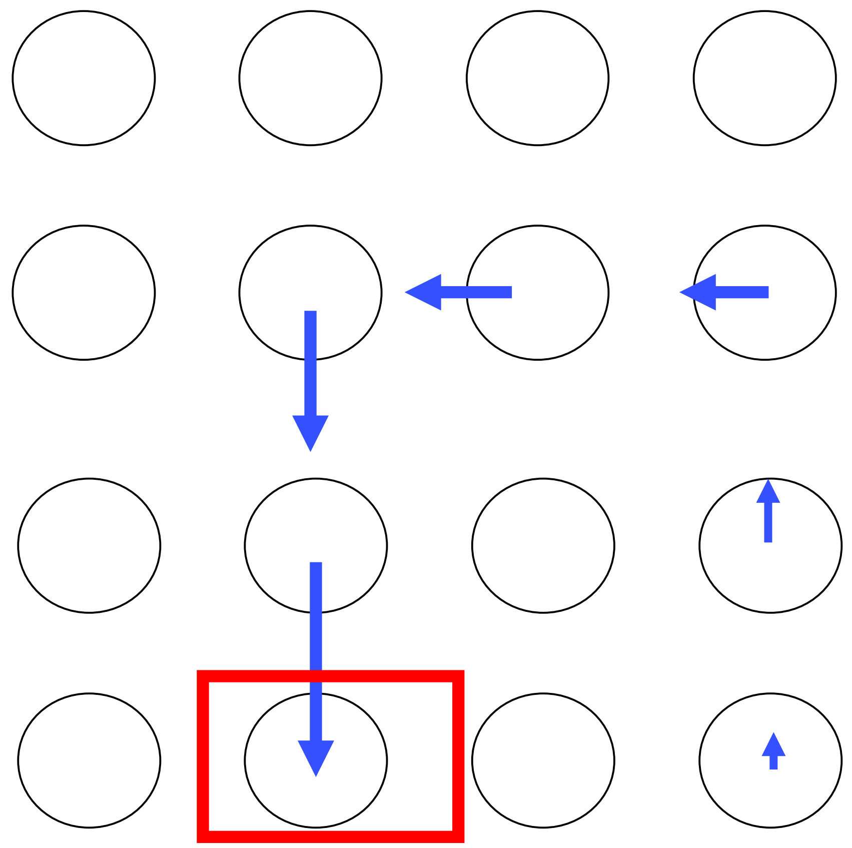


Idea:

- 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:

$$\Delta Q(s,a) = \eta \; [r - (Q(s,a) - Q(s',a'))] \; e(s,a)$$

$\longrightarrow$ SARSA(λ)

Note: lambda=0 gives standard SARSA

5. Solution 1: Eligibility Traces

7.5 *Sarsa(λ)*

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s, a and set e(s,a)=0 for all actions a and states s
    Repeat (for each step of episode):
        Take action a, observe r, s'
        Choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
        $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
        $e(s, a) \leftarrow e(s, a) + 1$
        For all s, a:
            $Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$
            $e(s, a) \leftarrow \gamma \lambda e(s, a)$
        $s \leftarrow s'; a \leftarrow a'$
    until s is terminal

Figure 7.11 Tabular Sarsa(λ).

From: Reinforcement Learning, Sutton and Barto 1998 First edition

5. Quiz: Eligibility Traces

[ ] Eligibility traces keep information of past state-action pairs.

[ ] For each Q-value $Q(s,a)$, the algorithm keeps one eligibility trace e(s,a), i.e., if we have 200 Q-values we need 200 eligibility traces

[ ] Eligibility traces enable information to travel rapidly backwards into the graph

[ ] The update of $Q(s,a)$ is proportional to $[r-(Q(s,a)-Q(s',a'))]$

[ ] In each time step all Q-values are updated

5. Problem of TD algorithms

Problem:

-‘Flow of information’ back from target is slow

- information flows 1 step per complete trial
- 20 trials needed to get information 20 steps away from target

→ First solution: eligibility traces.

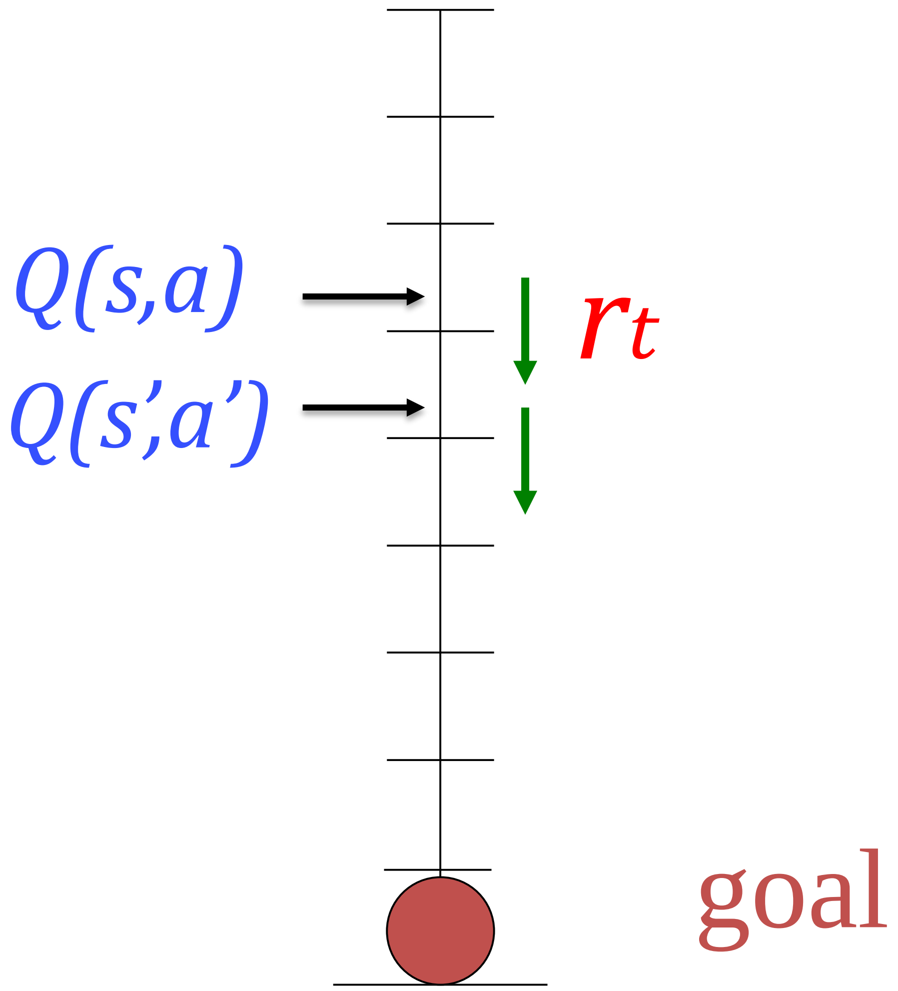
5. Solution 2: n=step SARSA

Standard SARSA

$\Delta Q(s,a) = \eta\ [r - (Q(s,a) - \gamma Q(s',a'))]$

$\Delta Q(s_t,a_t) = \eta\ [r_t - (Q(s_t,a_t) - \gamma Q(s_{t+1},a_{t+1}))]$

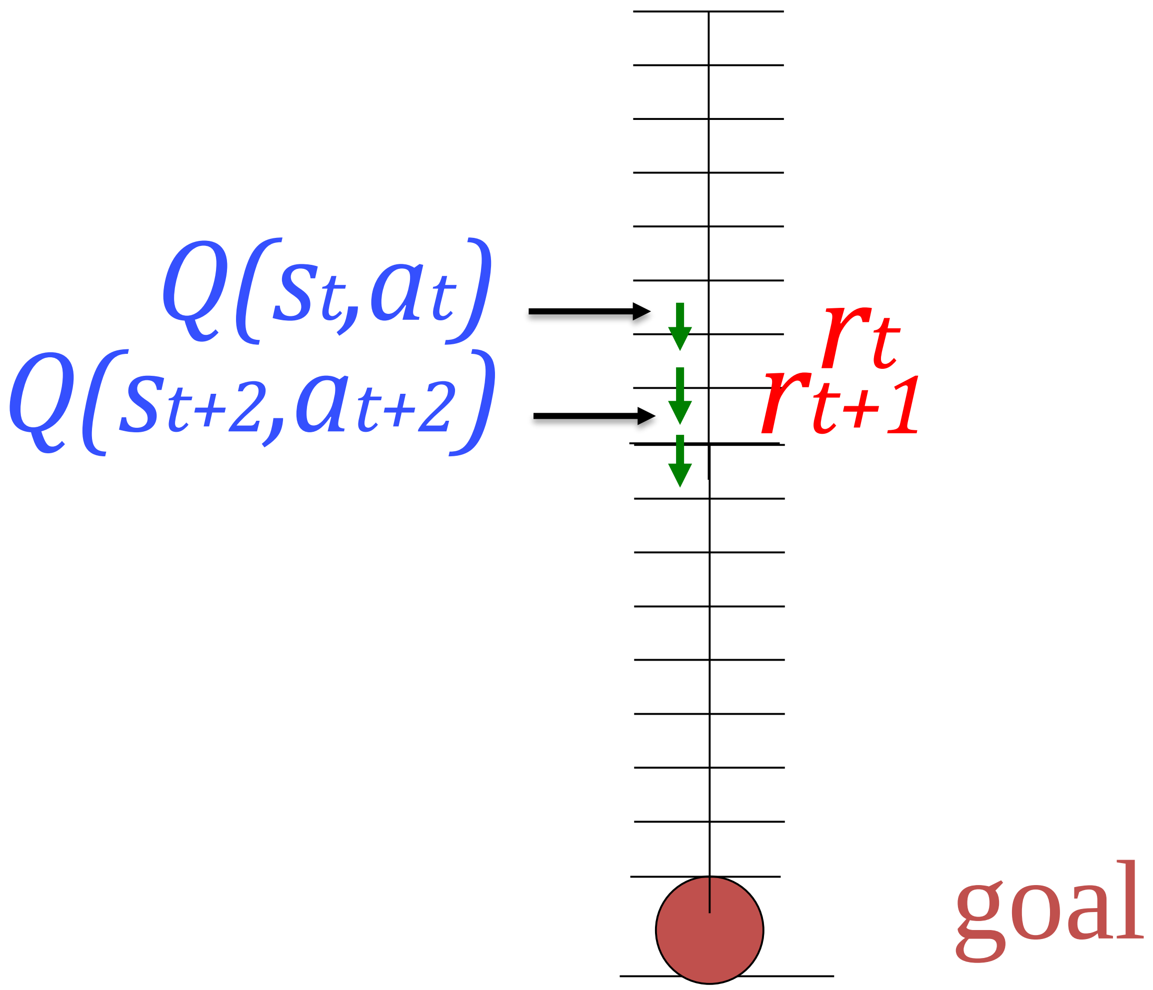
Temporal Difference (TD)



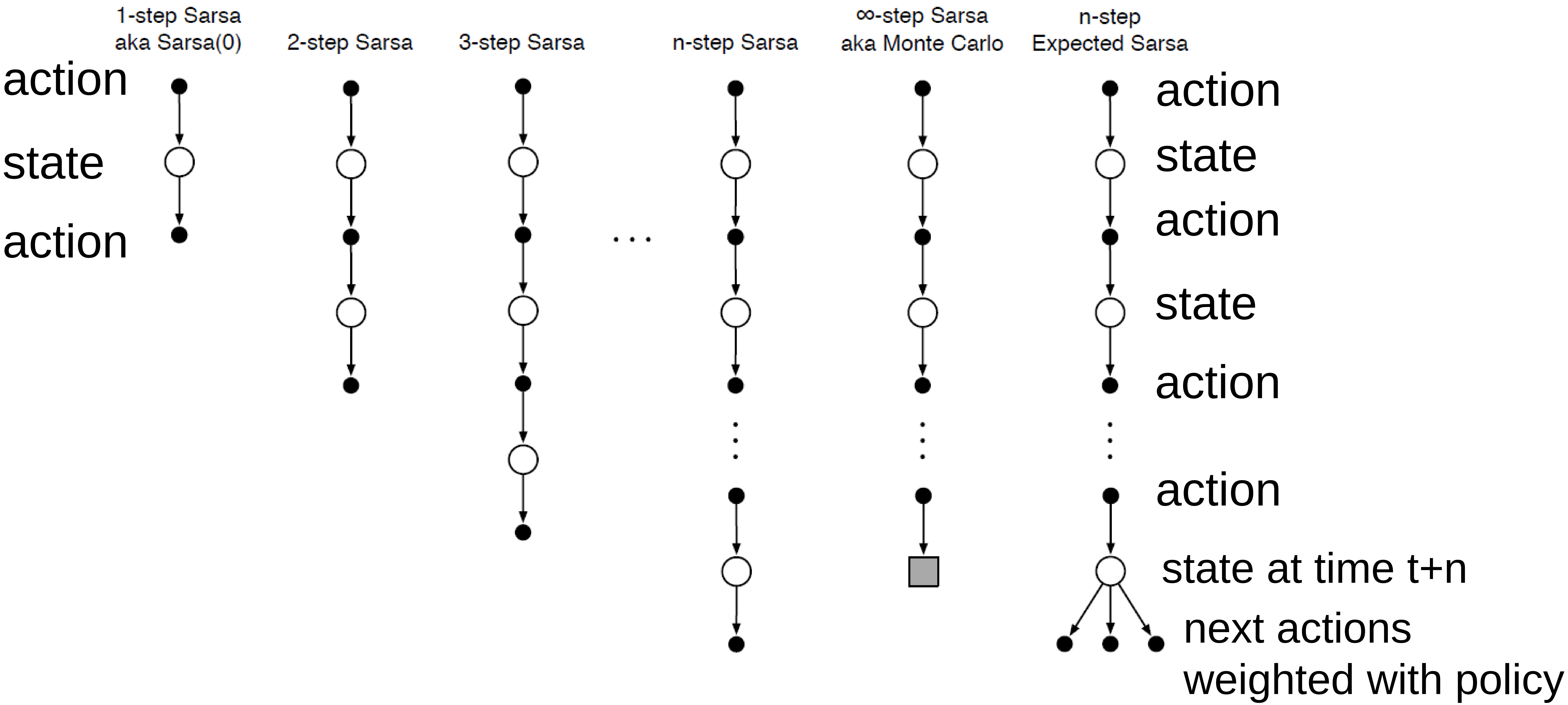
2-step SARSA

$\Delta Q(s_t,a_t) = \eta\ [r_t + \gamma r_{t+1} - (Q(s_t,a_t) - \gamma\gamma Q(s_{t+2},a_{t+2}))]$

2-step TD



5. n-step SARSA and n-step expected SARSA



5. n-step SARSA algorithm

n-step Sarsa for estimating $Q \approx q_*$, or $Q \approx q_\pi$ for a given π

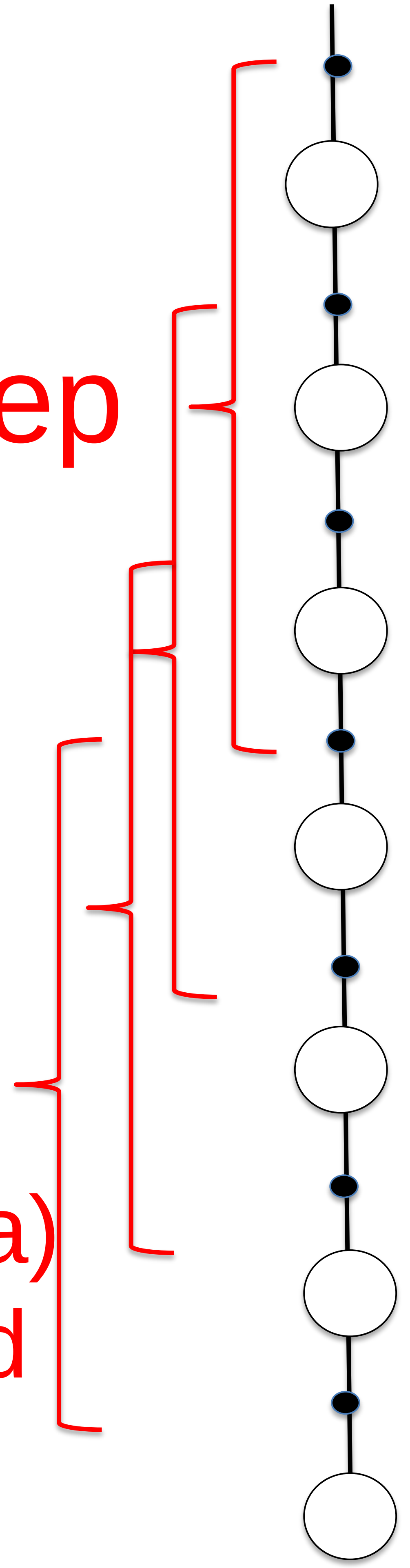
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod n

Repeat (for each episode):
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  For $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t+1$
  | else:
  | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \left[G - Q(S_\tau, A_\tau)\right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T - 1$

Take action, observe next state and reward, choose next action

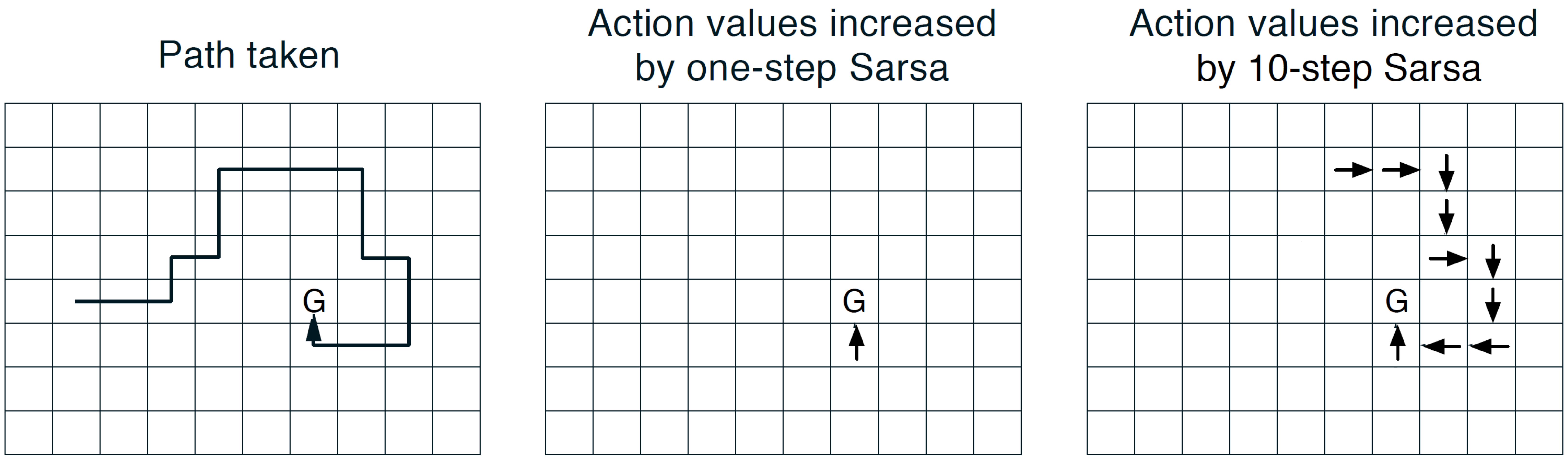
update of Q(s,a) with actions and state at time t-n

3-step



Sutton and Barto, Ch. 7.2

5. Example: 10-step SARSA



5. Scaling Problem of TD algorithms

TD algorithms do not scale correctly if the discretization is changed

either

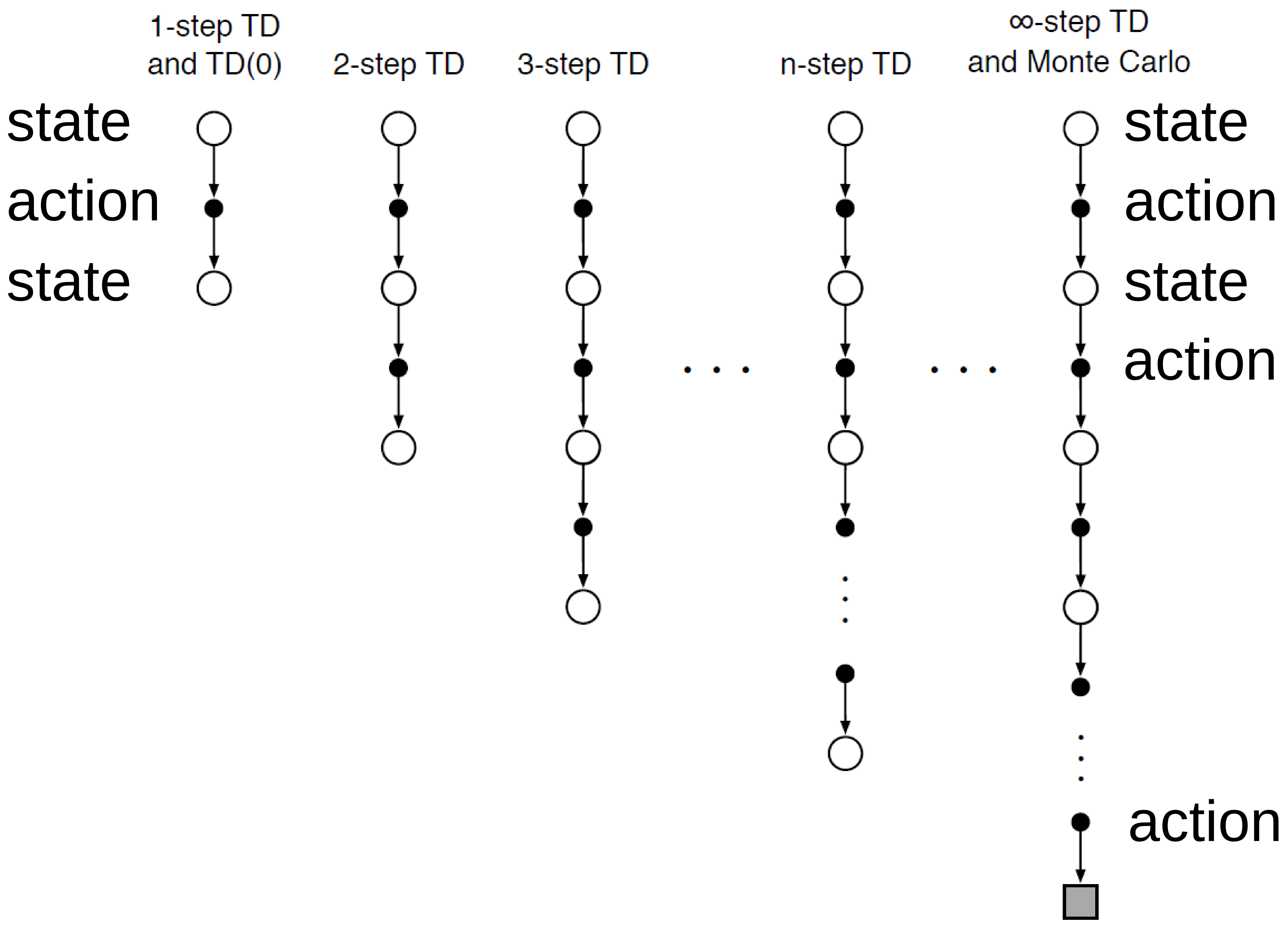
→ Introduce eligibility traces (temporal smoothing)

or

→ Switch from 1-step TD to n-step TD (temporal coarse graining)

Remark: the two methods are mathematically closely related.

4. Detour: n=step TD methods for V-values



Sutton and Barto, Ch. 7.1

5. Detour: n=step TD methods for V-values

n-step TD for estimating $V \approx v_\pi$

Initialize $V(s)$ arbitrarily, $s \in \mathcal{S}$
Parameters: step size $\alpha \in (0, 1]$, a positive integer n
All store and access operations (for S_t and R_t) can take their index mod n

Repeat (for each episode):
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  For $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t + 1$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$

Sutton and Barto, Ch. 7.1

Artificial Neural Networks: Lecture 9

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Variants of TD-learning methods and continuous space

1. Review
2. Variations of SARSA
3. TD – learning (Temporal Difference)
4. Monte-Carlo methods
5. Eligibility traces and n-step methods
6. Modeling the input space

6. Problem of TD algorithms: representation of input

All algorithms so far are 'tabular':

Q-learning or SARSA:
→ build a table $Q(s,a)$ with entries for all states s and actions a

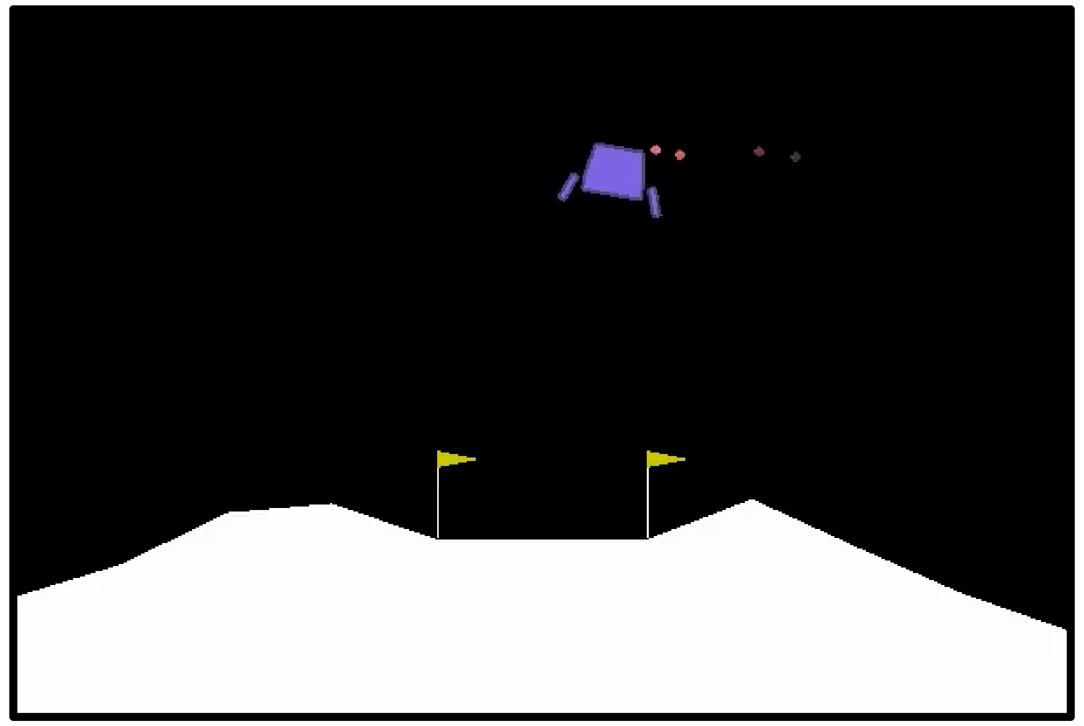
TD-learning of V-values:
→ build a table $V(s)$ for all states s

discrete states and actions

6. Problem of TD algorithms: representation of input

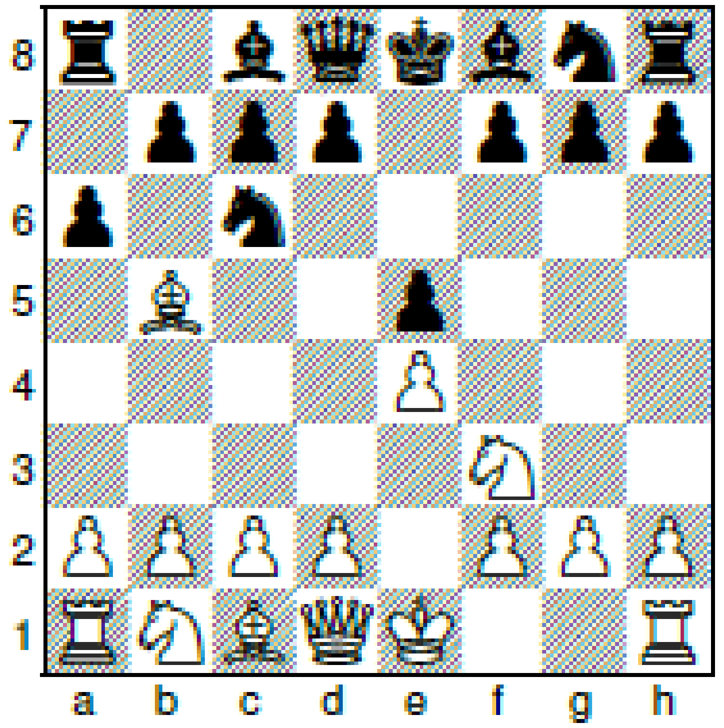
- for control problems, input space is naturally continuous

Moon lander
Aim: land between poles

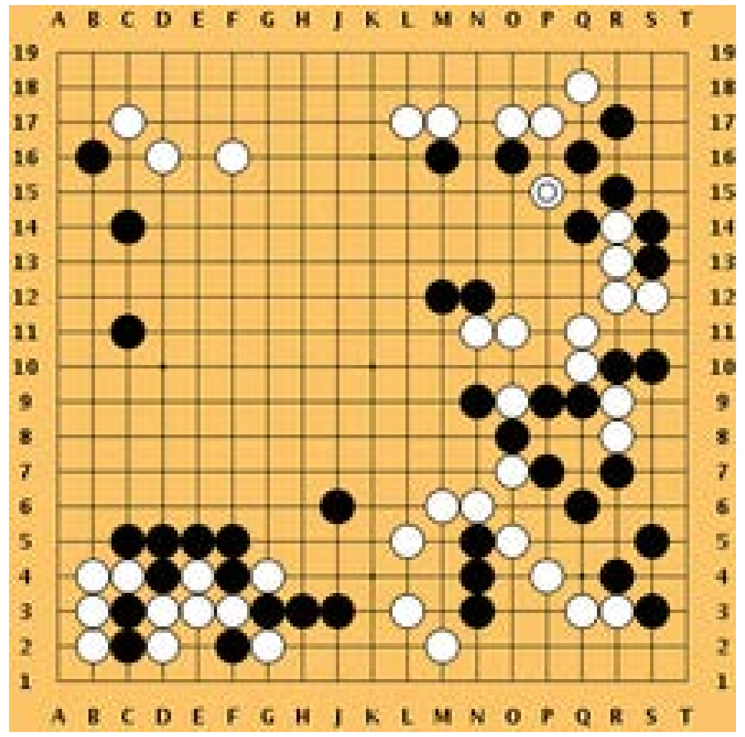


- for discrete games, the input space often too big

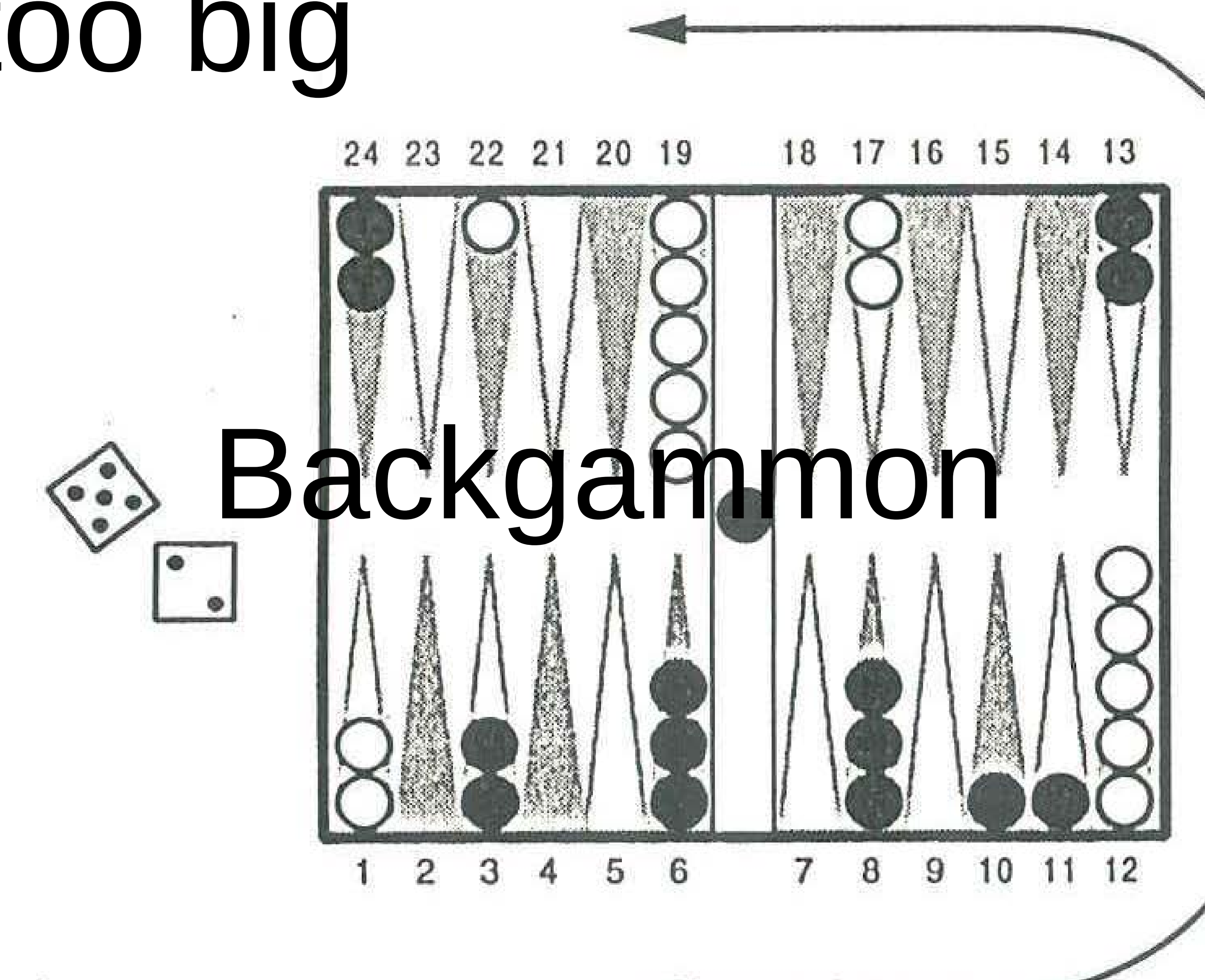
Chess



Go



Backgammon



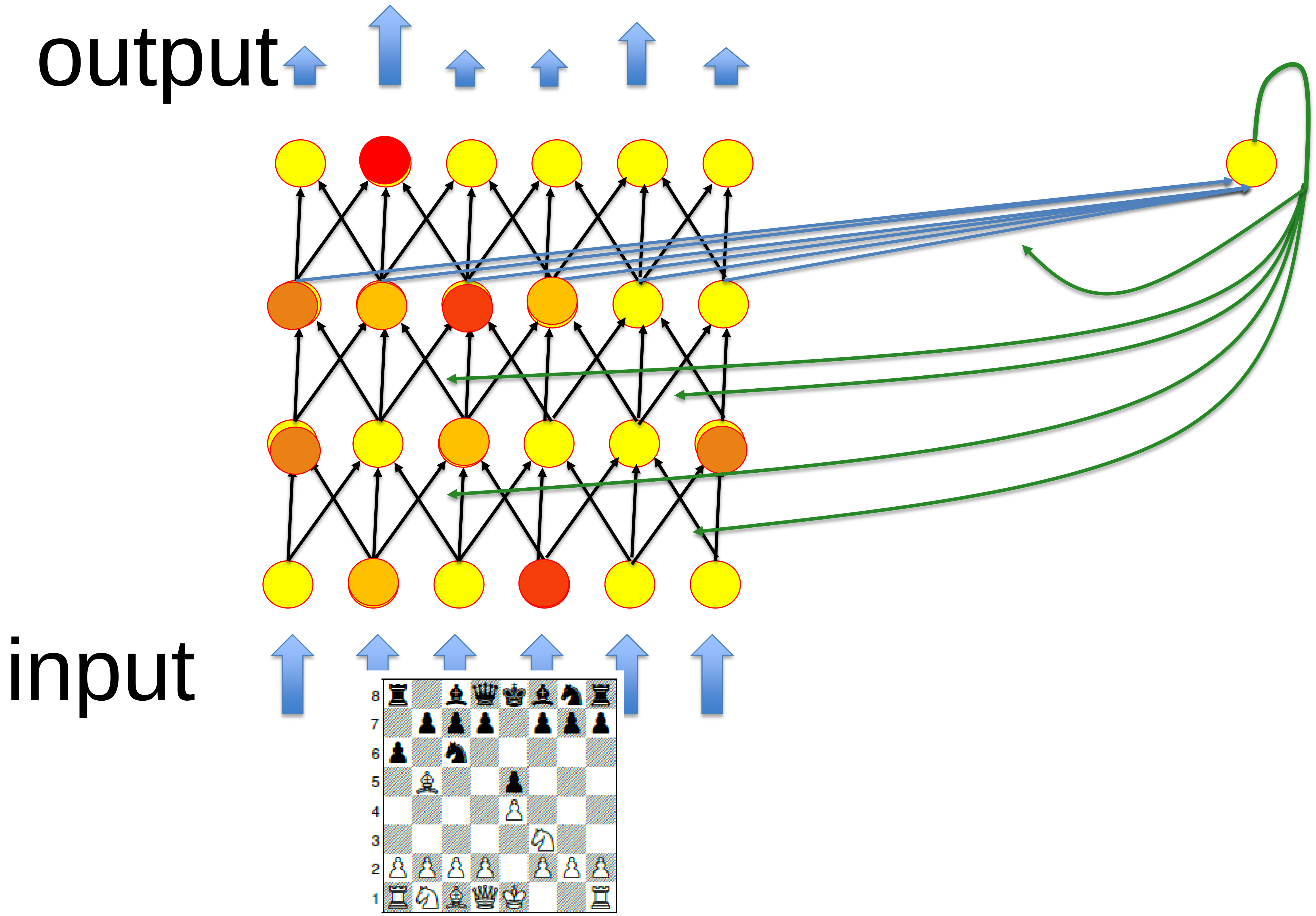
6. Solution: Neural Network to represent input configuration

Schematically (theory will follow):

action:
Advance king

2^e output for V-value
for current situation:

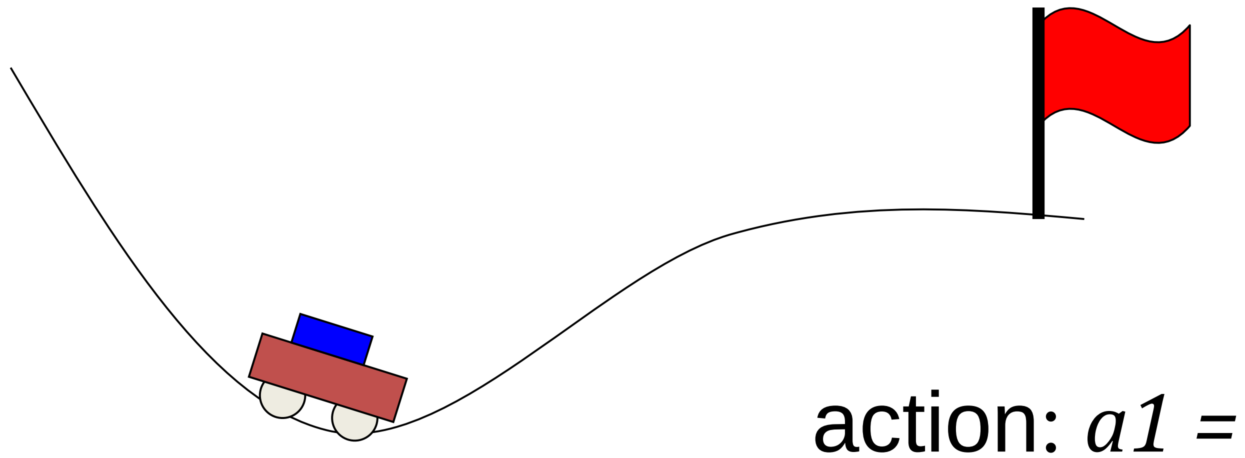
Note: alternatively,
action outputs could present
Q-values



learning:
- change connections

aim:
- Predict value of position
- Choose next action to win

6. Solution: Continuous input representation

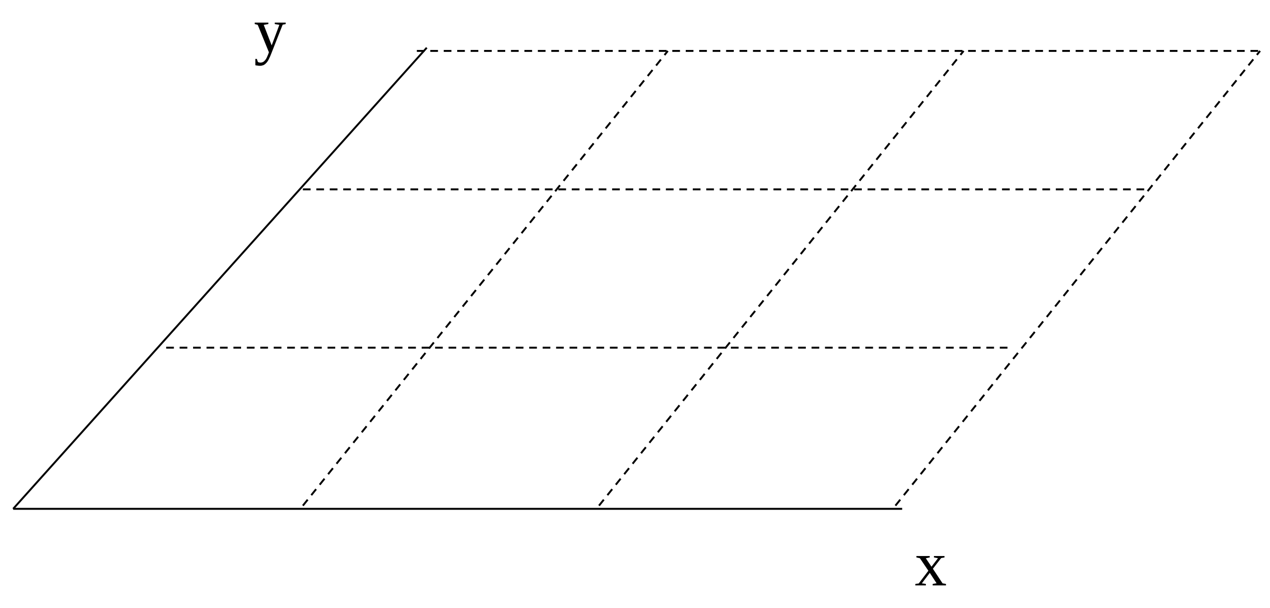


Example: Mountain Car

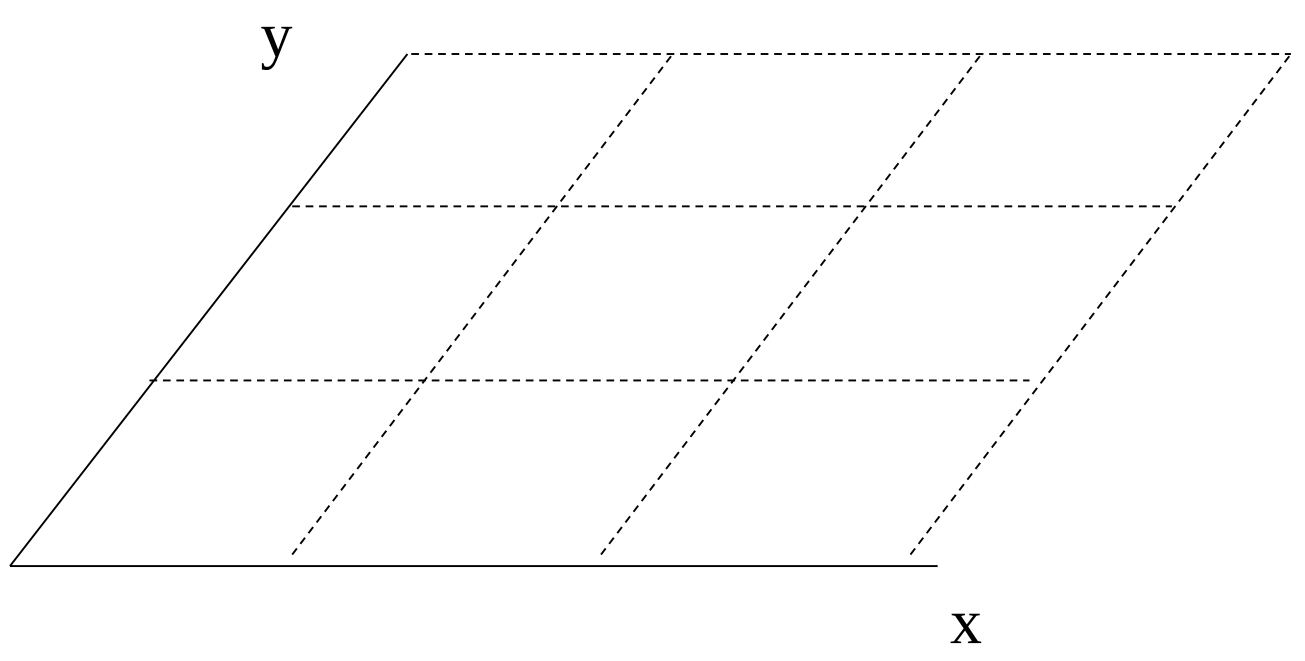
action: $a1$ = *right*

$a2$ = *left*

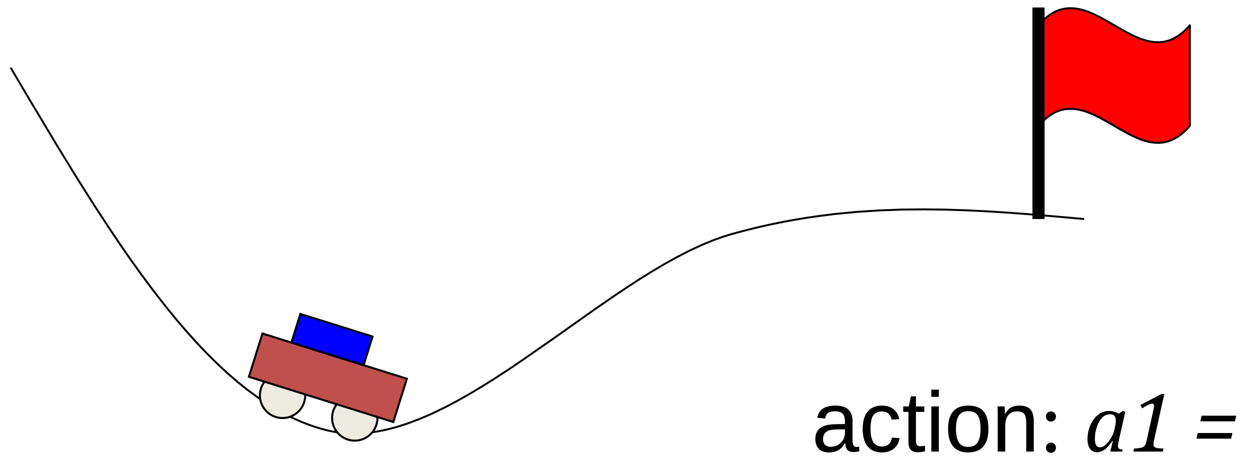
for action $a1$



for action $a2$



6. Solution: Continuous input representation

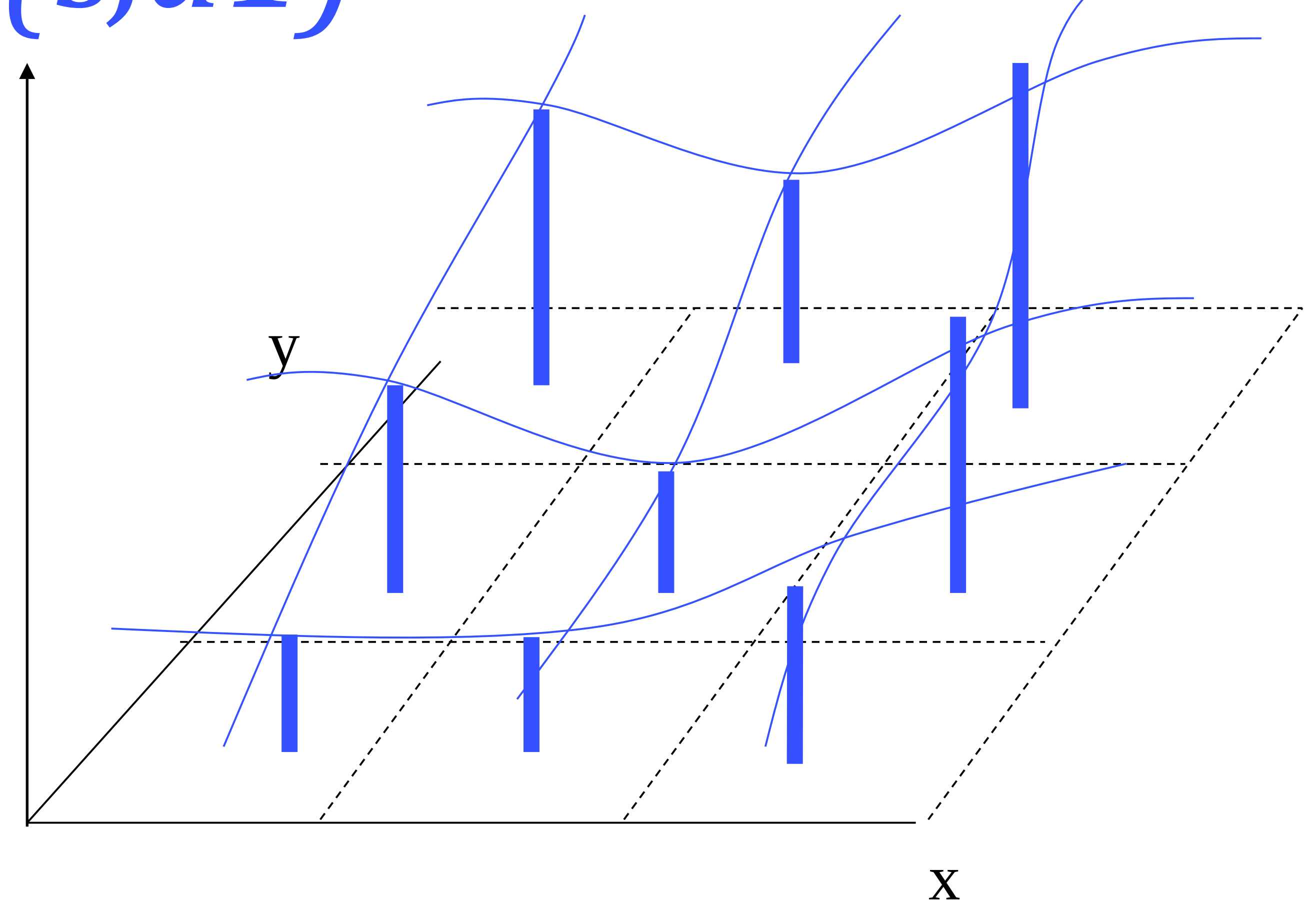


Example: Mountain Car

action: *a1 = right*
a2 = left

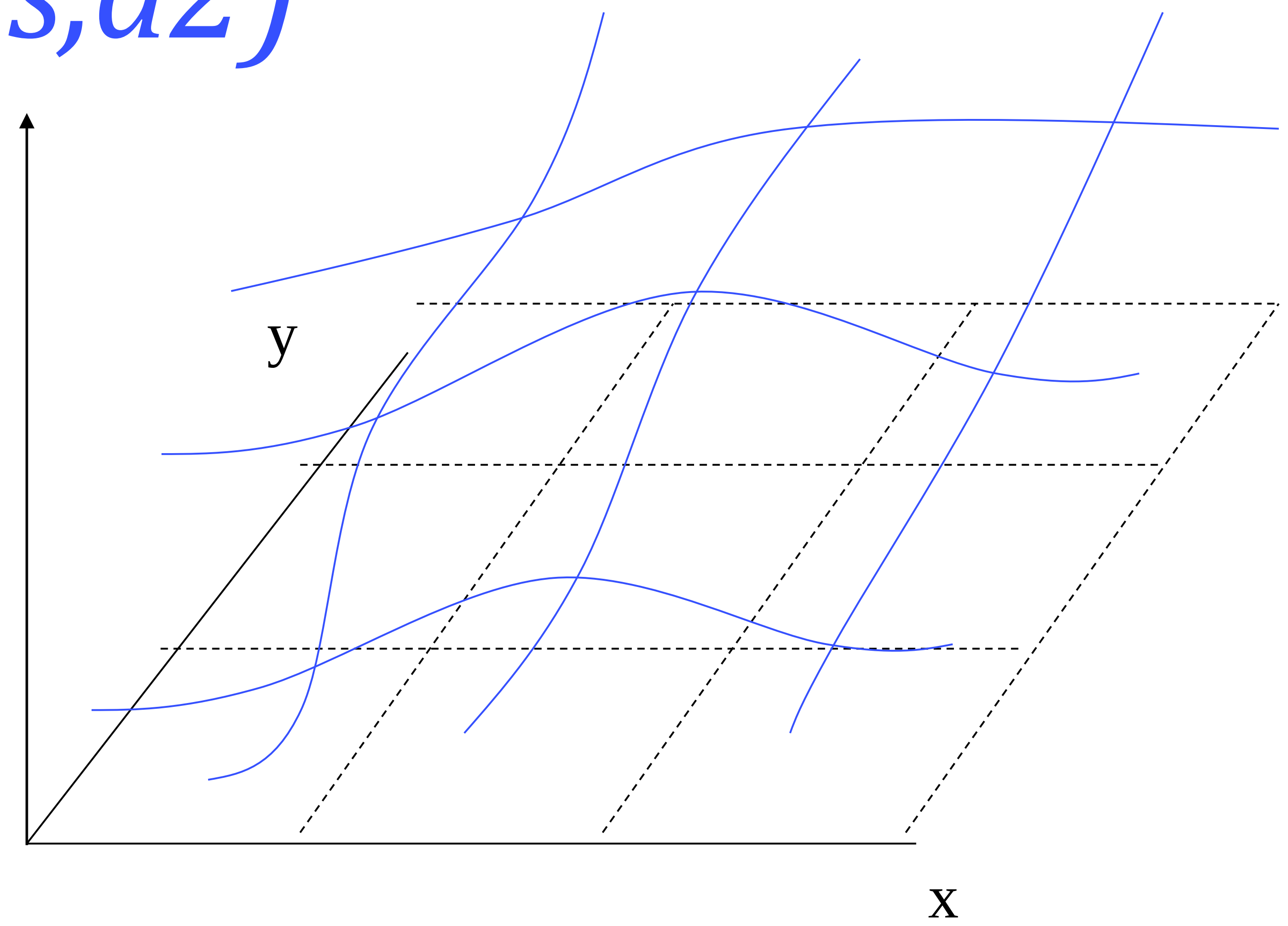
for action *a1*

Q(s,a1)



for action *a2*

Q(s,a2)



Blackboard: Radial Basis functions

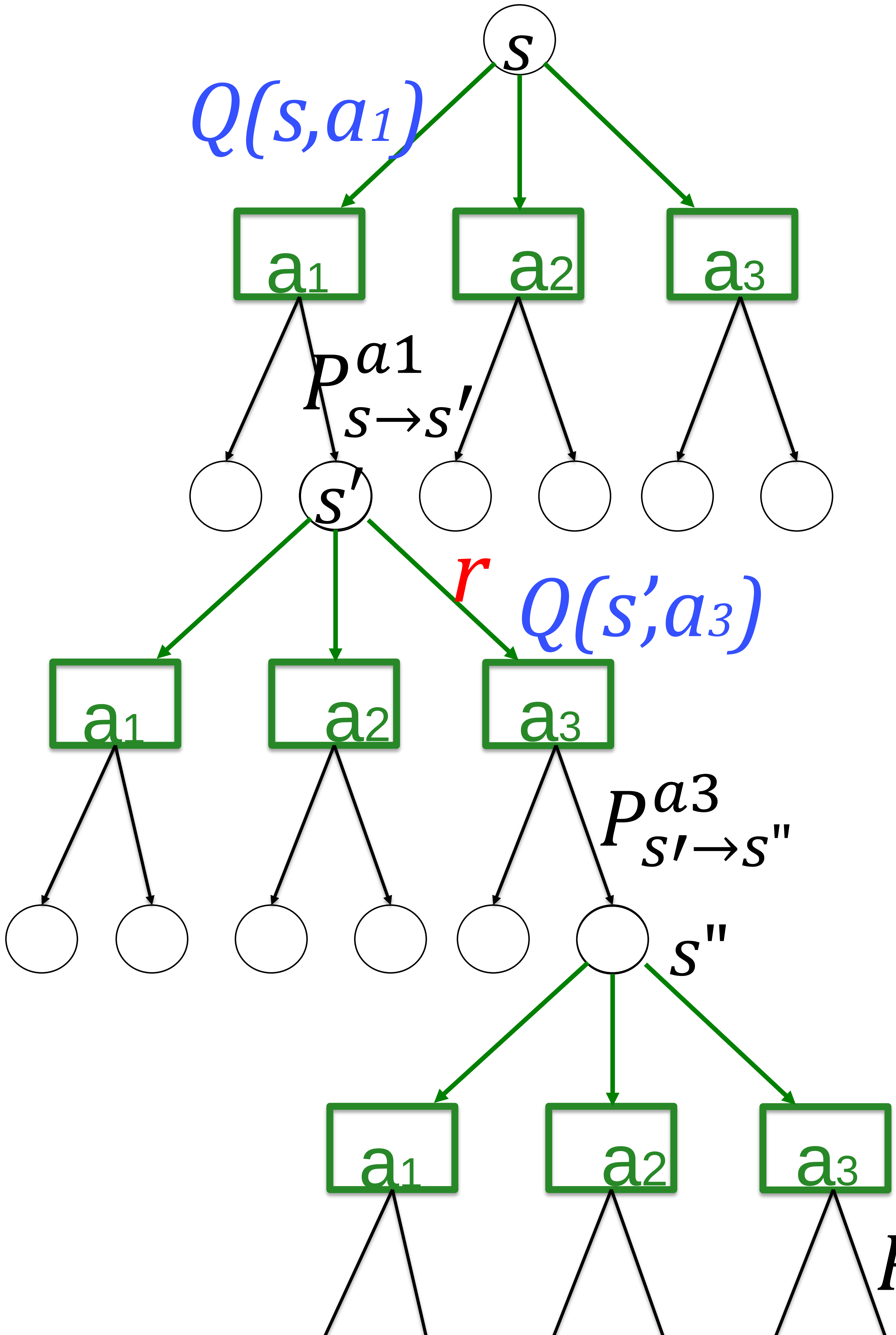
Blackboard: Radial Basis functions

6. Solution: Continuous input representation

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

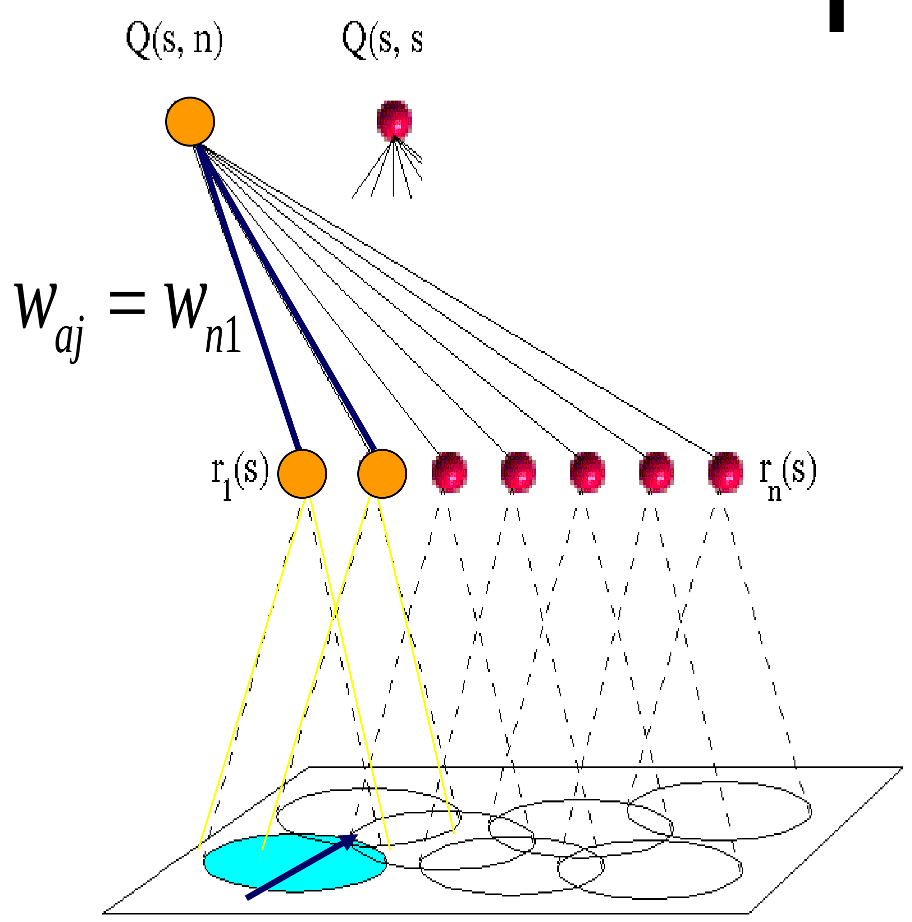
Blackboard: Error function

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



6. Solution: Continuous input representation

First idea: use basis functions



Spatial Representation

$$\boxed{Q(s,a) = \sum_j \ w_{aj}\Phi(s - s_j)}$$

Exercise now:

$$\frac{dQ(s',a')}{dw_{aj}}$$

Note: one hidden layer; only output weights are learned

6. Gradient descent on TD error (semi-gradient)

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

ignore

take gradient w.r.t $\boldsymbol{w}$

gradient descent

$$\Delta w_k = -\eta \frac{d}{dw_k} E(\boldsymbol{w})$$

$$\Delta w_k = \eta \, [r_t + \gamma \hat{V}(S'|\boldsymbol{w}) - \hat{V}(S|\boldsymbol{w})] \frac{d}{dw_k} \hat{V}(S|\boldsymbol{w})$$

'semi-gradient' because of the part we 'ignore'

6. Gradient descent on TD error (semi-gradient)

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

ignore

take gradient w.r.t $\boldsymbol{w}$

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$

Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose $A \sim \pi(\cdot|S)$
        Take action A, observe R, S'
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
        $S \leftarrow S'$
    until S' is terminal

6. SARSA (λ) in continuous space (for output weights in Neural Net)

$$\boxed{Q(s,a) = \sum_j w_{aj} \Phi(s - s_j)}$$

0) initialise

1) Use policy for action choice a

Pick most often action

$$a_t^* = \arg\max_a Q_a(s,a)$$

2) Observe R, s', choose next action a'

3) Calculate TD error in SARSA

$$\boxed{\delta_t = R_{t+1} - [Q_a(s,a) - \gamma \cdot Q_{a'}(s',a')]}$$

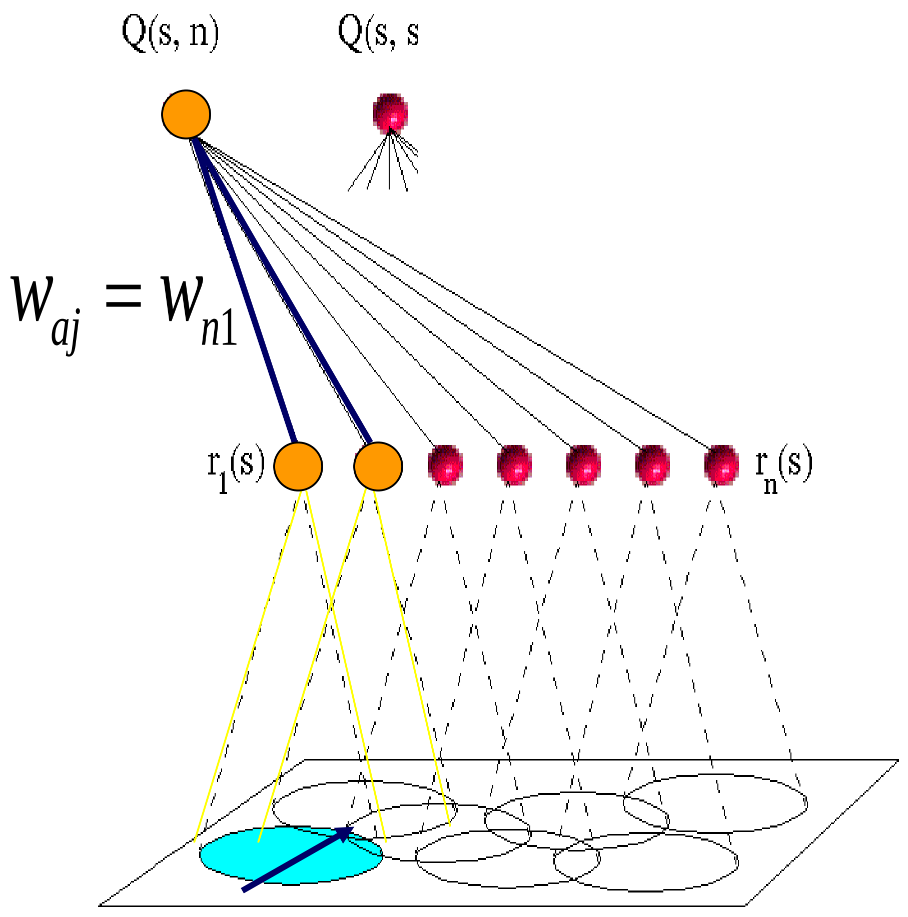
4) Update eligibility trace

$$e_{aj}(t) = \gamma\lambda\, e_{aj}(t - \Delta t) + \begin{cases} r_j & \text{if } a = \text{action taken} \\ 0 & \end{cases}$$

5) Update weights

$$\boxed{\Delta w_{aj} = \eta\, \delta_t\, e_{aj}}$$

6) Old *a'* becomes *a*, old *s'* becomes s and return to 2)

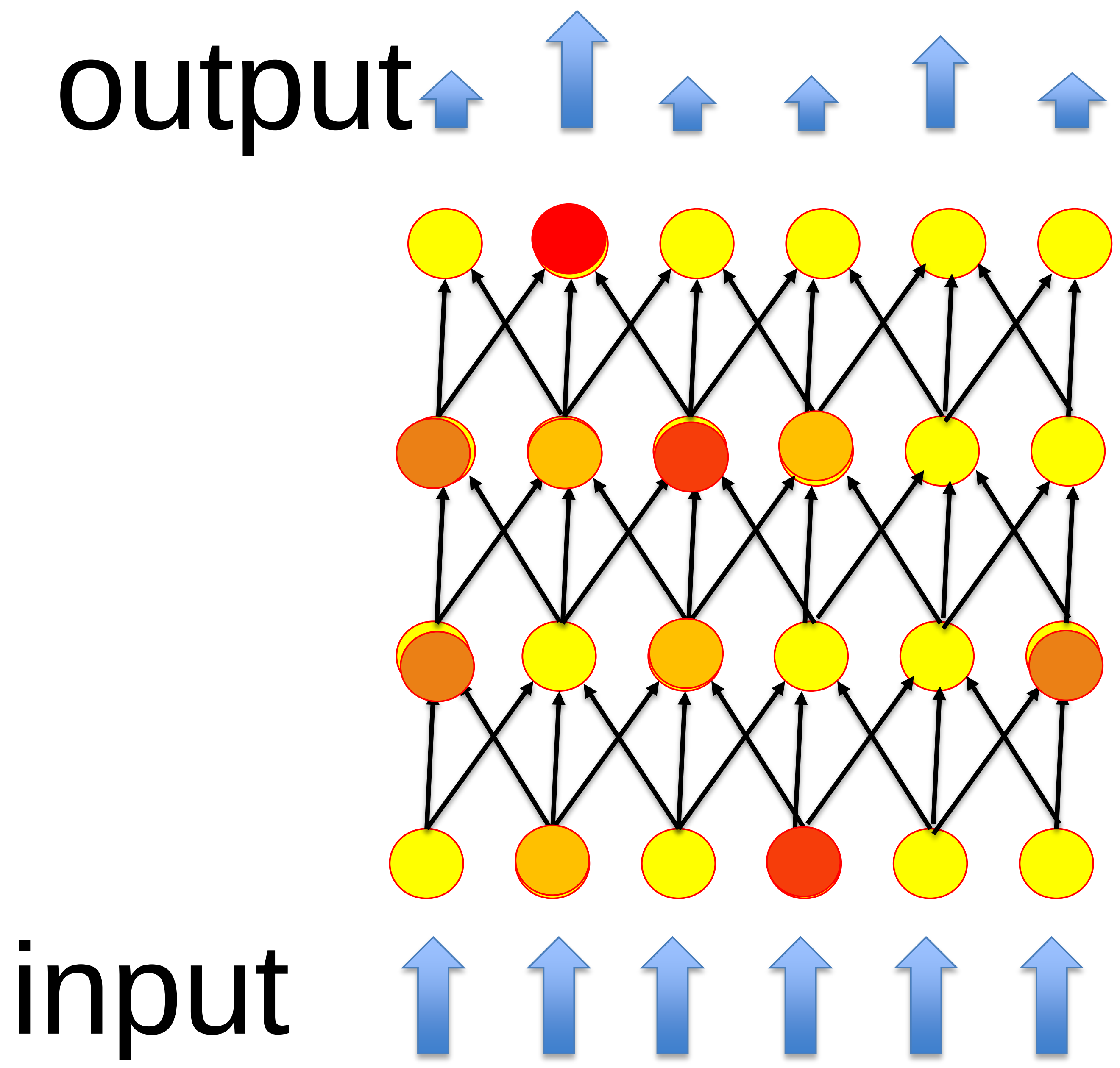


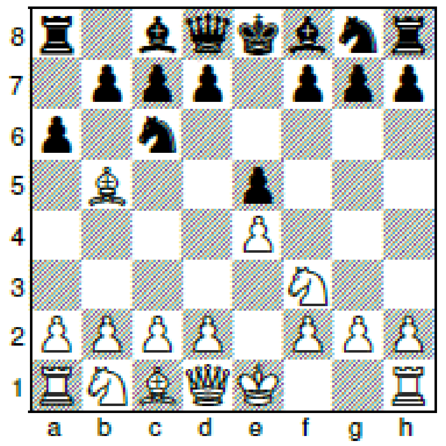
Spatial Representation

6. 2nd idea: multilayer Neural Network: Backprop

estimate Q-values of action

e.g. Advance king





Softmax strategy: take action a'
with prob

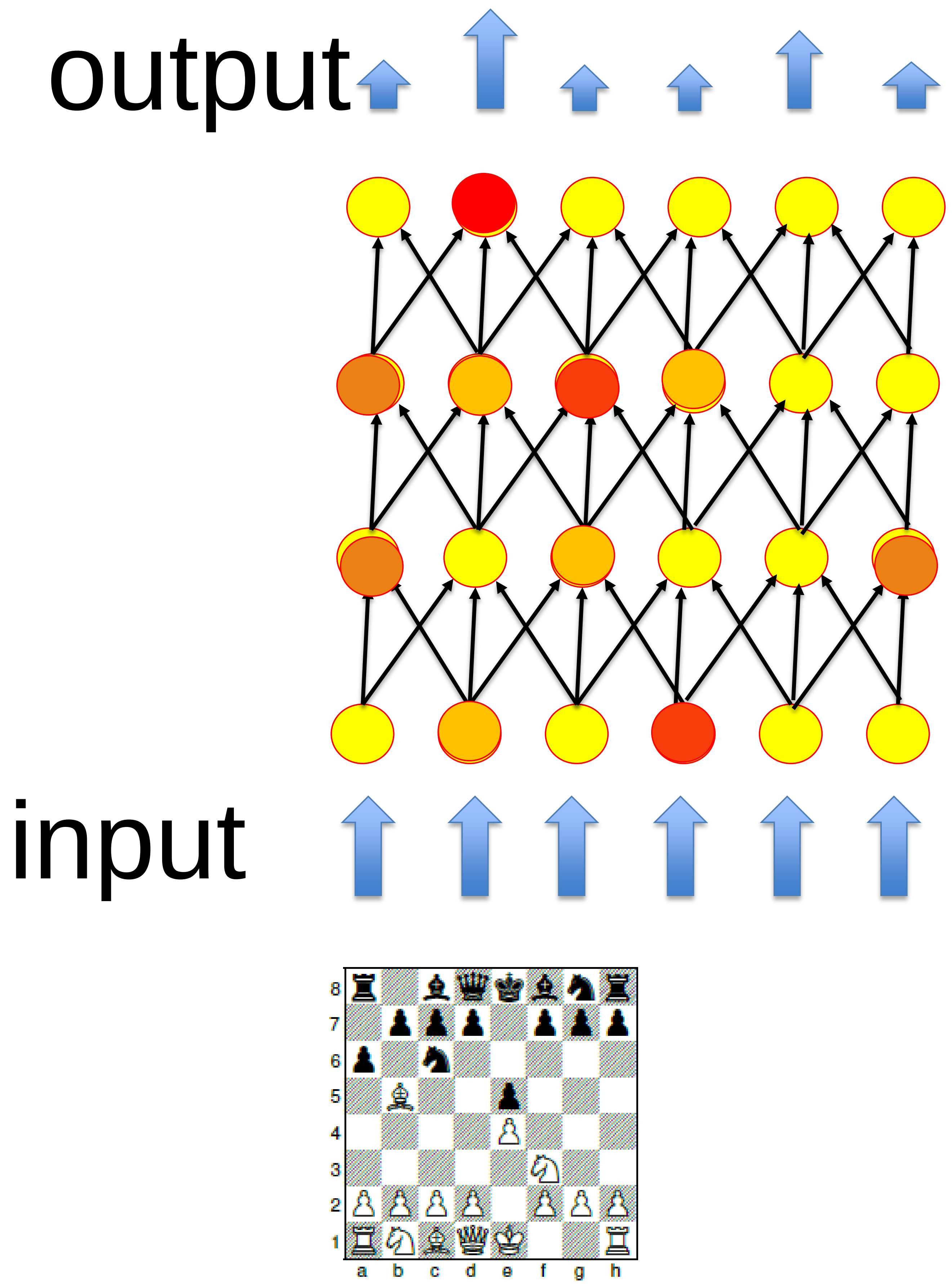
$$P(a') = \frac{\exp[\beta Q(a')]}{\sum_a \exp[\beta Q(a)]}$$

Neural network parameterizes Q-values as a function of state s.
Many outputs, one for each action a.
Learn weights by playing against itself.

6. 2nd idea: multilayer Neural Network: Backprop

estimate Q-values of action

e.g. Advance king



Softmax strategy: take action a'
with prob

$$P(a') = \frac{\exp[\beta Q(a')]}{\sum_a \exp[\beta Q(a)]}$$

Minimize TD of error of Q-values

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

use backprop to evaluate gradient

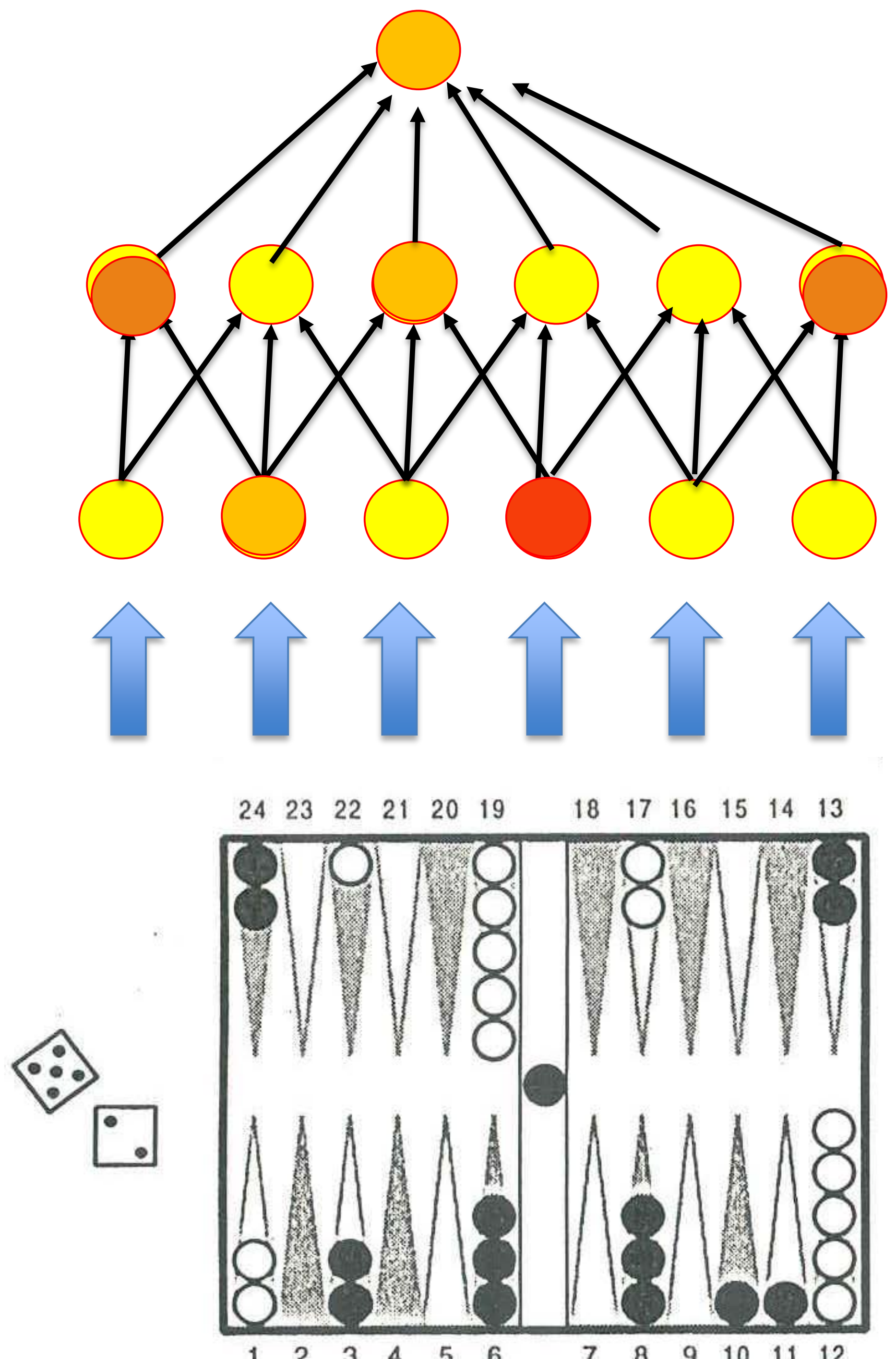
Note: we can use n-step Q-learning
instead of 1-step Q-learning

6. 2nd idea: Neural Network: Backprop

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

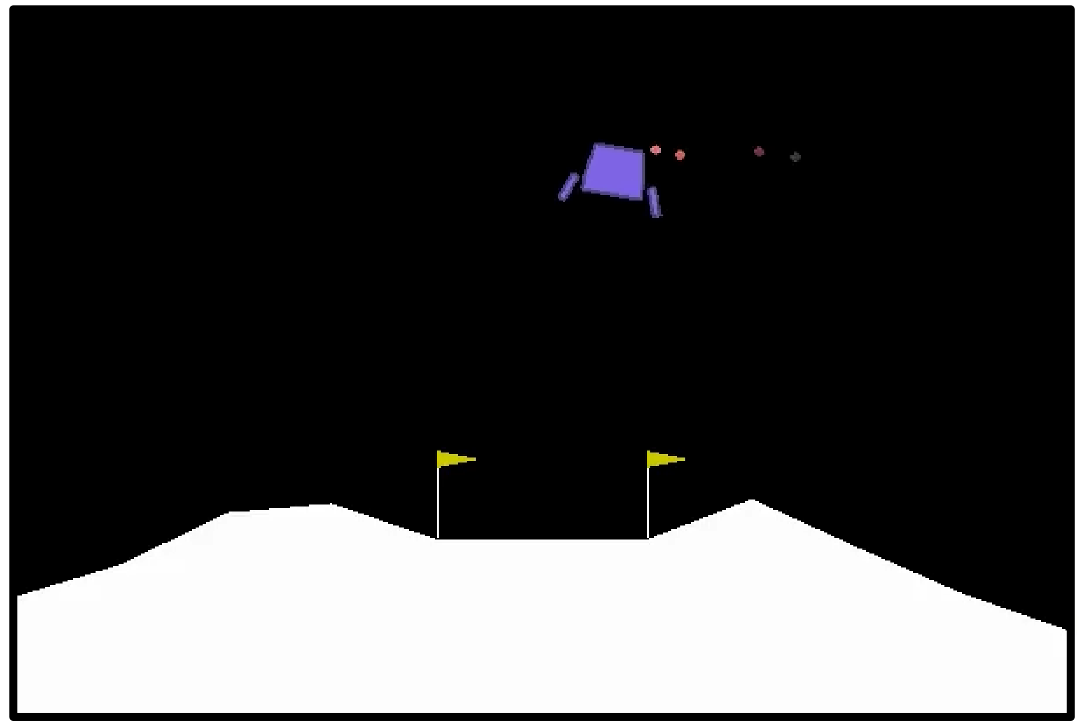
TD-Gammon

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

6. Neural networks to model input space

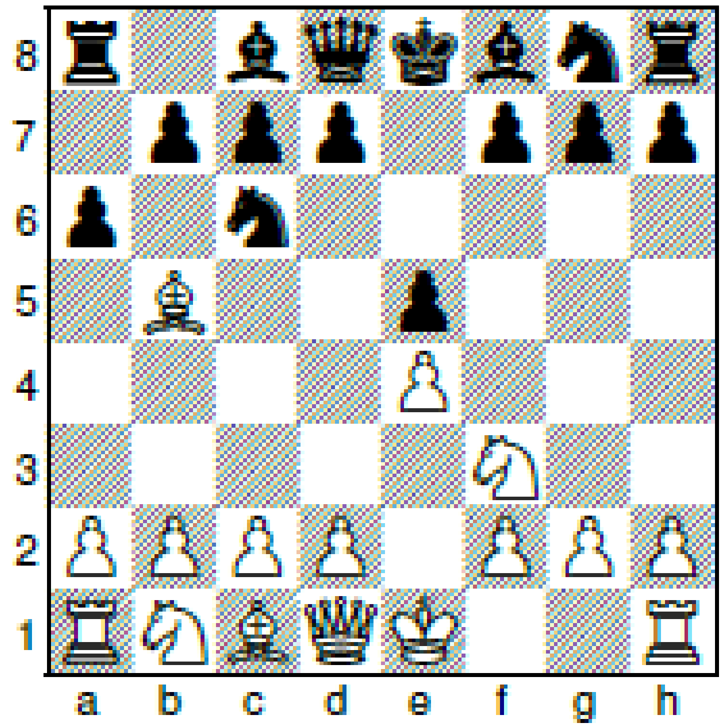
- for control problems, input space is naturally continuous

Moon lander
Aim: land between poles

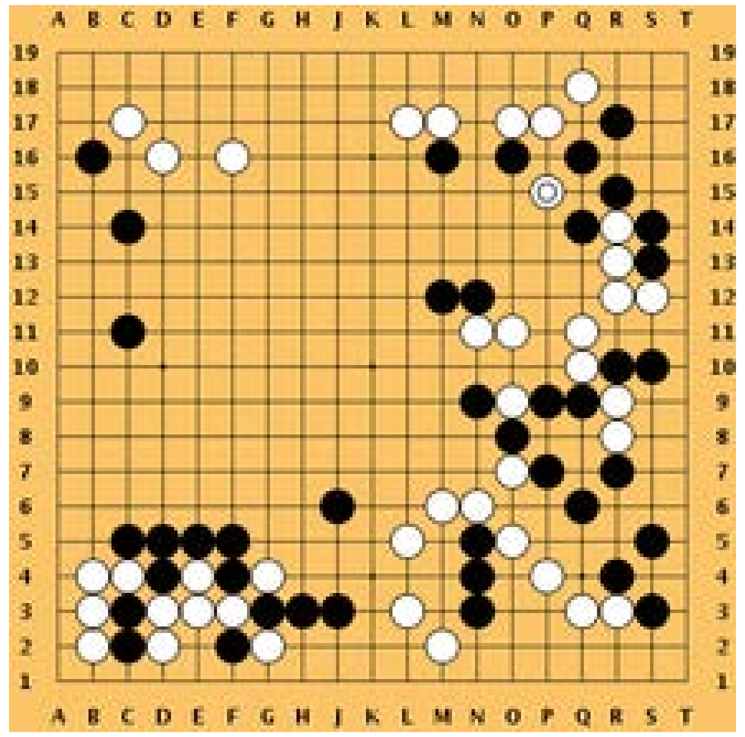


- for discrete games, the input space often too big
  → generalize via hidden states in neural networks

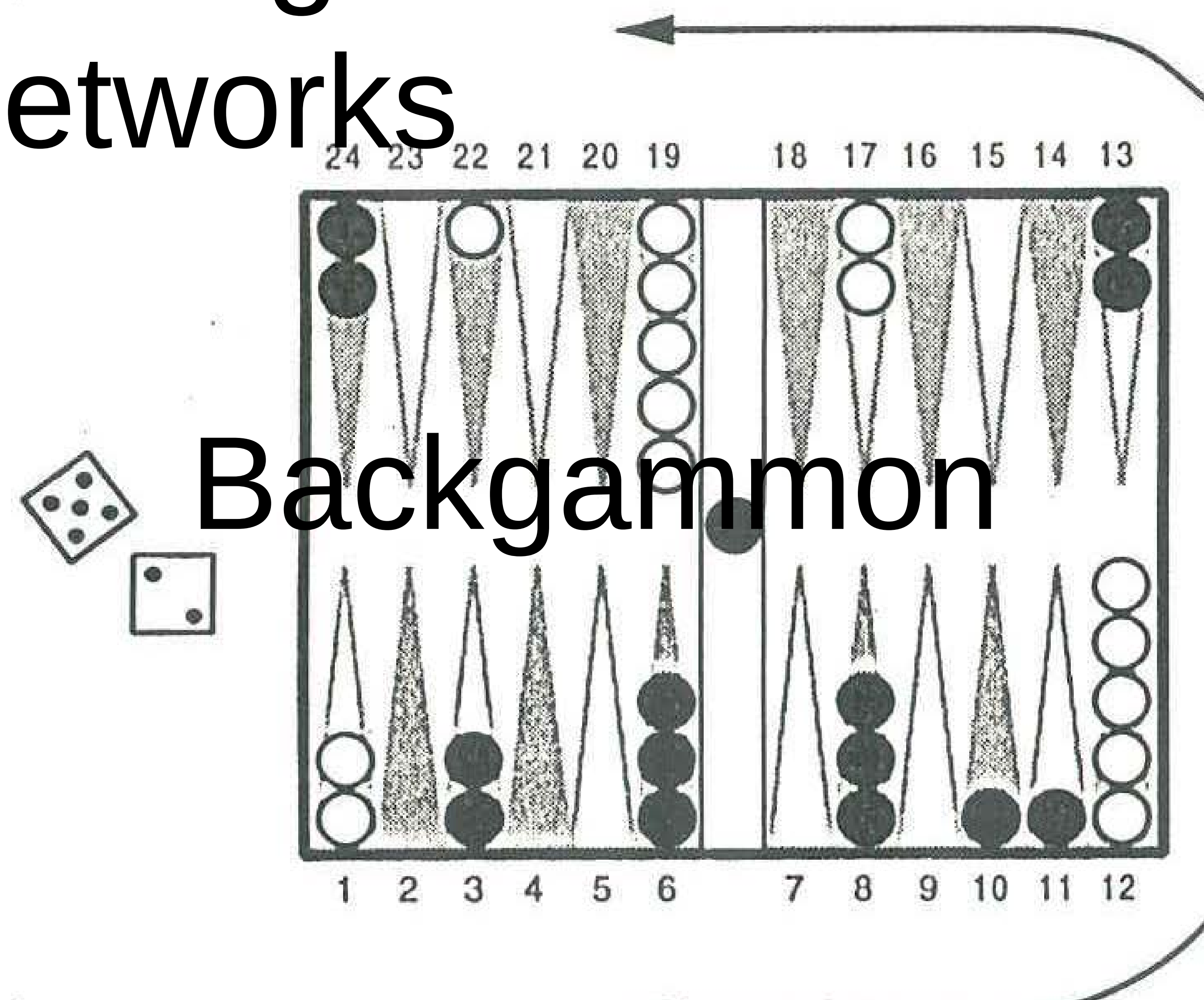
Chess



Go



Backgammon



Summary: Many Variations of a few ideas in TD learning

Objectives for today:

- TD – learning (Temporal Difference)

  → work with V-values, rather than Q-values

- **Variations of SARSA**

  - off-policy Q-learning (greedy update)
  - Monte-Carlo
  - n-step Bellman

- **Eligibility traces**

  - allows rescaling
  - similar to n-step SARSA

- **Continuous space**

  → use neural network to model and generalize

Basis of all:
iterative solution of
Bellman equation

example trials:

1: **s,a2**→s',a4→ r=0

2: s',a3→ r=1

3: s',a4→ r=0

4: s',a3→ r=1

5: **s,a1** → r=0

6: s',a4→ r=0

7: s',a4→r=0.5

8: s',a3→r=0