Statistical Machine Learning (BE4M33SSU)
Lecture 13: Reinforcement Learning

Jan Drchal

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Overview

Topics covered in the lecture:

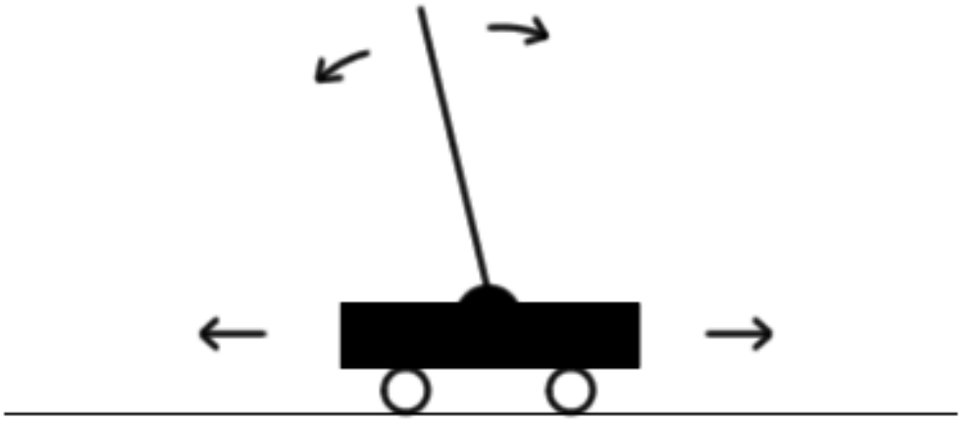
- Reinforcement Learning problem
- Markov Decision Processes (MDPs)
- Methods based on dynamic programming
- Sampling methods: Monte Carlo, Temporal Differences
- SARSA and Q-learning
- Value function approximation

Resources

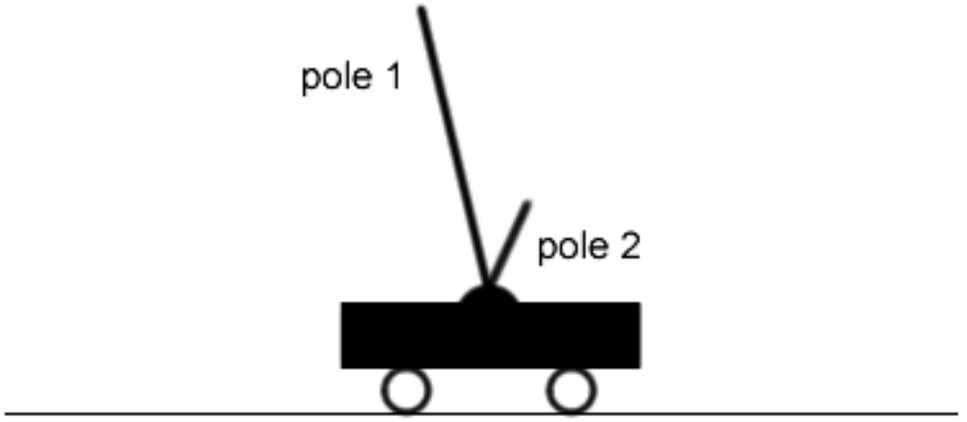
- Sutton and Barto: *An Introduction to Reinforcement Learning*, 1998
  - second edition draft free to download:
    https://webdocs.cs.ualberta.ca/~sutton/book/the-book.html
- David Silver: *UCL Course on RL*:
  http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
- Szepesvári: *Algorithms for Reinforcement Learning, 2010*
  - available free: https://sites.ualberta.ca/~szepesva/RLBook.html

What is Reinforcement Learning?

- Tasks: robot control, game playing, managing investments
- No supervisor, just *reward* signal
- Feedback is often delayed
- Time matters: no i.i.d. data, e.g., what robot sees is correlated with what it has seen a second before
- Agent takes actions $\rightarrow$ influences the environment $\rightarrow$ influences data it receives in future



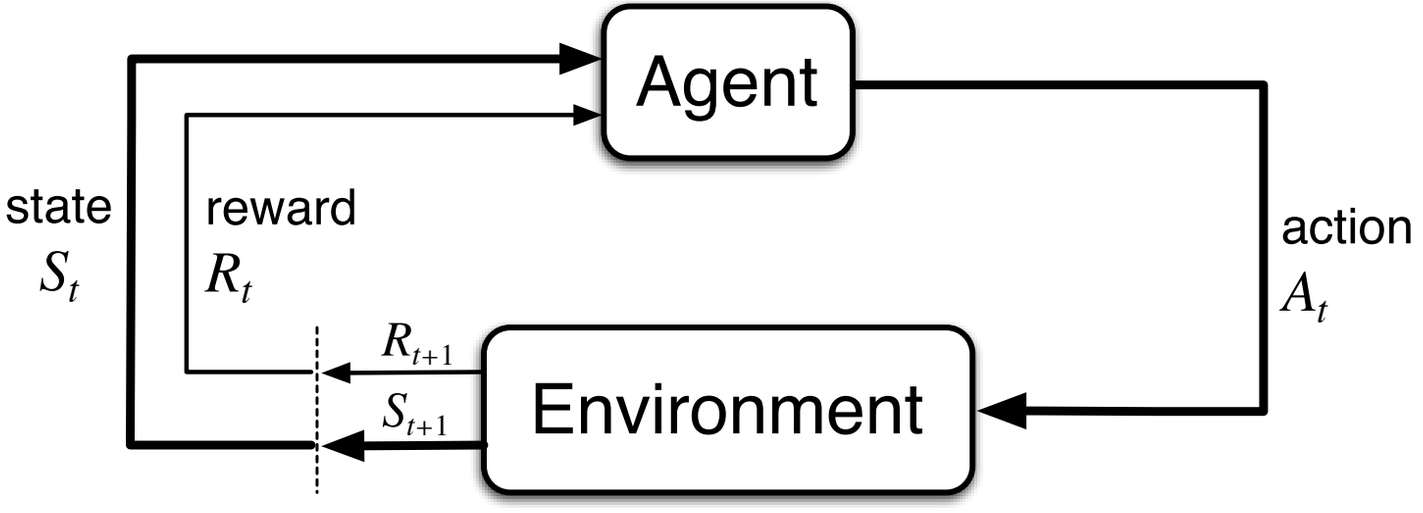
Single Pole Balancing



Double Pole Balancing

http://anji.sourceforge.net/polebalance.htm

Agent-Environment Interface



Sutton and Barto: *An Introduction to Reinforcement Learning, draft of the 2nd ed.*, 2016

- *Discrete* timesteps
- Signal S_t is a *representation* of environment's state
- Action A_t leads to a reward R_{t+1} and a new state S_{t+1}
- The sequence goes: $S_0, A_0, R_1, \ldots, S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, \ldots$
- Note that rewards may be delayed many steps from actions which caused them!

Markov Property

- We want a state signal which retains all relevant information in a compact form
- When a state depends only on a previous state and an action taken (not on a whole history):

$$\mathbb{P}(S_{t+1} \mid S_t, A_t) = \mathbb{P}(S_{t+1} \mid S_0, A_0, R_1, S_1, A_1, R_2, \ldots, R_t, S_t, A_t)$$

  we say it has a *Markov property* (we use also terms like *Markov state*, *Markovian task*, etc.)
- Agent does not need to keep any internal state (memory) to act optimally
- Methods in the following slides assume Markov state signals only, although they are often applied to Markovian approximations of non-Markovian tasks in practice

Markov Decision Process (MDP)

Definition 1. *A finite Markov Decision Process is a tuple* $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ *is a finite set of states*
- $\mathcal{A}$ *is a finite set of actions,* $\mathcal{A}(s)$ *are actions available at* $s \in \mathcal{S}$
- $\mathcal{P}$ *defines state transition probabilities:*

$$\mathcal{P}^a_{ss'} = \mathbb{P}\left(S_{t+1} = s' \mid S_t = s, A_t = a\right)$$

- $\mathcal{R}$ *is a reward function:*

$$\mathcal{R}^a_s = \mathbb{E}\left(R_{t+1} \mid S_t = s, A_t = a\right)$$

- $\gamma \in [0, 1]$ *is a discount factor*

Only stationary tasks covered here: the probabilities do not change in time

Policy

Definition 2. *A policy π is a distribution:*

$$\pi(a|s) = \mathbb{P}\left(A_t = a \mid S_t = s\right)$$

- Defines agent's behavior
- MDP policies depend on the current state not on a history
- Deterministic policy: $a = \pi(s)$

Return

Definition 3. *The return G_t is the total discounted reward from time-step t:*

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The *discount* $\gamma \in [0, 1]$ is the present value of the future rewards
- The value of receiving reward R after $k + 1$ steps is $\gamma^k R$

Why Discount?

- Mathematically convenient
  - G_t converges for R_t bounded and $\gamma \in [0, 1)$
- Humans prefer immediate rewards
- Lower discount when we do not trust the model
- Undiscounted returns ($\gamma = 1$) may be used when all sequences terminate: *episodic tasks*

Value Functions

Definition 4. *The state-value function $v_\pi(s)$ is the expected return of starting in state s and following policy π:*

$$v_\pi(s) = \mathbb{E}_\pi \left(G_t \mid S_t = s \right)$$

Definition 5. *The action-value function $q_\pi(s, a)$ is the expected return of starting in state s, taking action a and following policy π:*

$$q_\pi(s, a) = \mathbb{E}_\pi \left(G_t \mid S_t = s, A_t = a \right)$$

- The task of Reinforcement Learning is to find a policy which **maximizes the expected return**
- *Note that the subscripts used in expected values denote that the policy π is used, not that we sum over policies!*

Value Function Decomposition

- The *state-value* function can be decomposed into immediate reward and a discounted value of the successor state:

$$v_\pi(s) = \mathbb{E}_\pi \left(R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \right)$$

- The *action-value* function can be similarly decomposed to:

$$q_\pi(s, a) = \mathbb{E}_\pi \left(R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a \right)$$

Bellman Expectation Equations

We can convert between the *state-value* and *action-value* functions:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

hence we can get recursive equations:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

- We call these equations the **Bellman expectation equations**
- For a *state-value* function $v_\pi(s)$, $s \in \mathcal{S}$ we have a system of $|\mathcal{S}|$ simultaneous linear equations in $|\mathcal{S}|$ unknowns

Optimal Value Functions

Definition 6. *The optimal state-value function* $v_*(s)$ *is the maximum value function over all policies:*

$$v_*(s) = \max_\pi v_\pi(s)$$

Definition 7. *Similarly the optimal action-value function is:*

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

Optimal Policy

Definition 8. *Partial ordering over all policies:*

$$\pi \geq \pi' \text{ if and only if } v_\pi(s) \geq v_{\pi'}(s),\ \forall s \in \mathcal{S}$$

For any MDP:

- there exists at least one optimal policy $\pi_* \geq \pi, \forall \pi$
- it achieves the optimal state-value function: $v_{\pi_*}(s) = v_*(s)$
- and the optimal action-value function: $q_{\pi_*}(s,a) = q_*(s,a)$

An optimal *greedy* policy can be found:

$$\pi_*(a|s) = \mathbb{I}\left\{ a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\, q_*(s,a') \right\}$$

- Knowing $q_*(s,a)$ gives us immediately the optimal policy $\pi_*(a|s)$!
- There is always a deterministic optimal policy for any MDP

Bellman Optimality Equations

For optimal value functions we have:

$$v_*(s) = \max_a q_*(s,a)$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

hence:

$$v_*(s) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right)$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a')$$

- We call these equations the **Bellman optimality equations**
- The equations are non-linear $\Rightarrow$ generally no closed form solution
- **Iterative methods** are used to get approximate solutions in practice

Policy Evaluation

- Dynamic programming approach to evaluate v_π:
  1. initialize (e.g. randomize) v_1, iteration $k = 1$
  2. use Bellman Expectation equation to update

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

  3. $k \leftarrow k + 1$
  4. go to 2 until convergenece (e.g., $\max_s |v_k(s) - v_{k-1}(s)|$ is less than a threshold)
- The algorithm generates a sequence $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$
- All one-step transitions from s involved: *full backup*
- Policy evaluation for action-value function $q_\pi(s, a)$ is analogous
- Convergence proof using *contraction mapping theorem* (see Szepesvári)
- Asynchronous version is possible and converges too

Policy Improvement

- Can we improve our policy using the evaluated value function?
- Consider a deterministic policy: $a = \pi(s)$
- Define a new *greedy policy* w.r.t. to the actual value function:

$$\pi'(s) = \operatorname*{argmax}_a q_\pi(s, a)$$

- Using the greedy policy for just one step leads to an improvement:

$$q_\pi(s, \pi'(s)) = \max_a q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- Let us show that improvement holds for more than one step:

$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}\left(R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right) \\
&\leq \mathbb{E}_{\pi'}\left(R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right) \\
&\leq \mathbb{E}_{\pi'}\left(R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s\right) \\
&\leq \mathbb{E}_{\pi'}\left(R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right) = v_{\pi'}(s)
\end{aligned}$$

Policy Improvement (contd.)

- When improvements stop:

$$q_\pi(s, \pi'(s)) = \max_a q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$

it means Bellman optimality equation was satisfied:

$$v_\pi(s) = \max_a q_\pi(s, a)$$

and therefore the optimal policy was found: $v_\pi(s) = v_*(s)$, $\forall s \in \mathcal{S}$

- For stochastic policies:
  - assign a portion of probability to each maximizing action,
  - set zero to all other activities

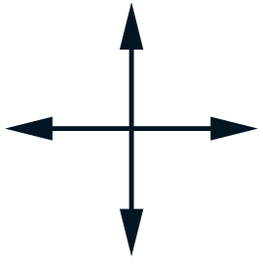
Policy Iteration

- *Policy iteration* method searches for an optimal policy π_*
- Initial policy is iteratively improved by repeating the following two steps:
  - **Evaluate** value function v_{π_k} for the actual policy π_k using the *policy evaluation*
  - **Improve** the policy using the *policy improvement*: $\pi_{k+1} = \text{greedy}(v_{\pi_k})$
- We get a chain of monotonically improving policies and value functions:

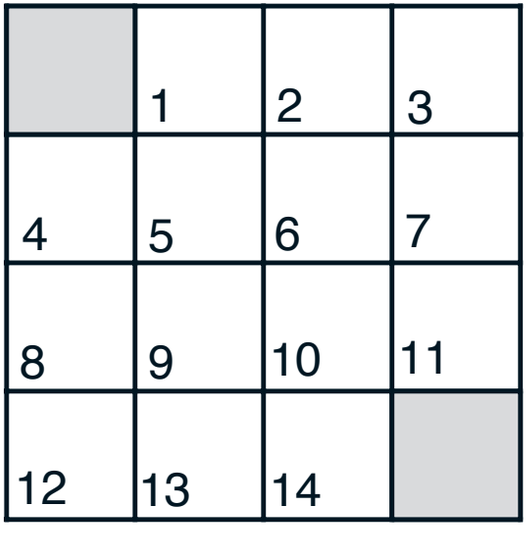
$$\pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} v_{\pi_2} \xrightarrow{I} \pi_3 \xrightarrow{E} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_{\pi_*}$$

- Finite MDP $\Longrightarrow$ finite number of policies $\Longrightarrow$ finite number of iterations

Grid World Example



actions



$R = -1$
on all transitions

Sutton and Barto: *An Introduction to Reinforcement Learning, draft of the 2nd ed.*, 2016

Grid World Example (contd.)

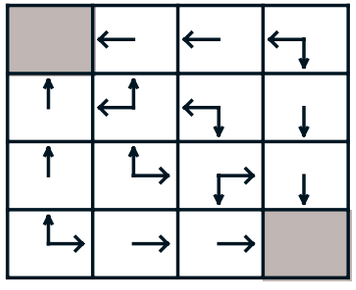
	v_k for the Random Policy	Greedy Policy w.r.t. v_k
$k = 0$	0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	← random policy
$k = 1$	0.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 0.0	
$k = 2$	0.0 -1.7 -2.0 -2.0 -1.7 -2.0 -2.0 -2.0 -2.0 -2.0 -2.0 -1.7 -2.0 -2.0 -1.7 0.0	

Sutton and Barto: *An Introduction to Reinforcement Learning, draft of the 2nd ed.*, 2016

Grid World Example (contd.)

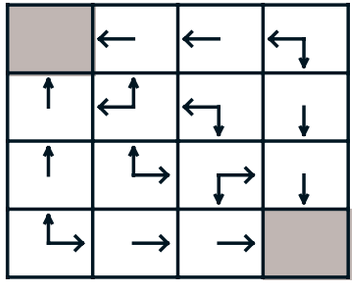
$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



$k = 10$

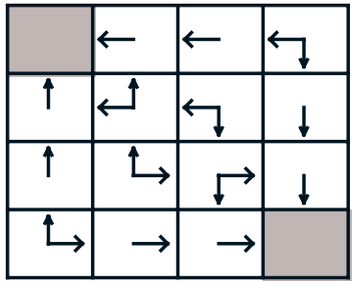
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0



optimal policy

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Sutton and Barto: *An Introduction to Reinforcement Learning, draft of the 2nd ed.*, 2016

Value Iteration

- Do we really need to exactly evaluate value v_k?
- Idea: truncate *policy evaluation*
- Truncation to single step $\rightarrow$ *value iteration*:

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

- Note that the above *value iteration* update replaces the sum over all actions by maximum in the *policy evaluation* update

Monte-Carlo Prediction

- Evaluates value functions with no prior knowledge of the environment's dynamics (transitions, rewards)
- Based on averaging sampled returns
- Limited to episodic tasks (termination needed)
- *First-visit* Monte-Carlo policy evaluation:
  - the **first** time t a state s is visited in episode:
  - increment counter: $N(s) \leftarrow N(s) + 1$
  - increment total return: $S(s) \leftarrow S(s) + G_t$
  - estimate value: $V(s) = S(s)/N(s)$
  - Convergence by law of large numbers: $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$
- Other possibility: *every-visit* Monte-Carlo policy evaluation

Monte-Carlo Prediction (contd.)

- Use moving average: $\mu_k = \frac{1}{k}\sum_{i=1}^{k} x_i = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$
- Update after each episode is then:

$$V(s) \leftarrow V(s) + \frac{1}{N(s)}(G_t - V(s))$$

- Getting $q_\pi(s, a)$ estimates $Q(s, a)$ is analogous
- What if not all needed states/actions are visited (e.g., when π is a deterministic policy)? $\rightarrow$ we will deal with that later
- For non-stationary problems we typically *forget*:

$$V(s) \leftarrow V(s) + \alpha(G_t - V(s))$$

- No *bootstraping*: an estimate for a state is not build upon other state estimates (DP methods did this)

Temporal Difference (TD) Prediction

- Temporal Difference (TD) methods can learn from incomplete episodes by *bootstrapping*
- MC updates toward an actual return G_t:

$$V(S_t) \leftarrow V(S_t) + \alpha(\ G_t\ - V(S_t))$$

- TD(0) algorithm updates toward an estimated return $R_{t+1} + \gamma V(S_{t+1})$:

$$V(S_t) \leftarrow V(S_t) + \alpha(\ R_{t+1} + \gamma V(S_{t+1})\ - V(S_t))$$

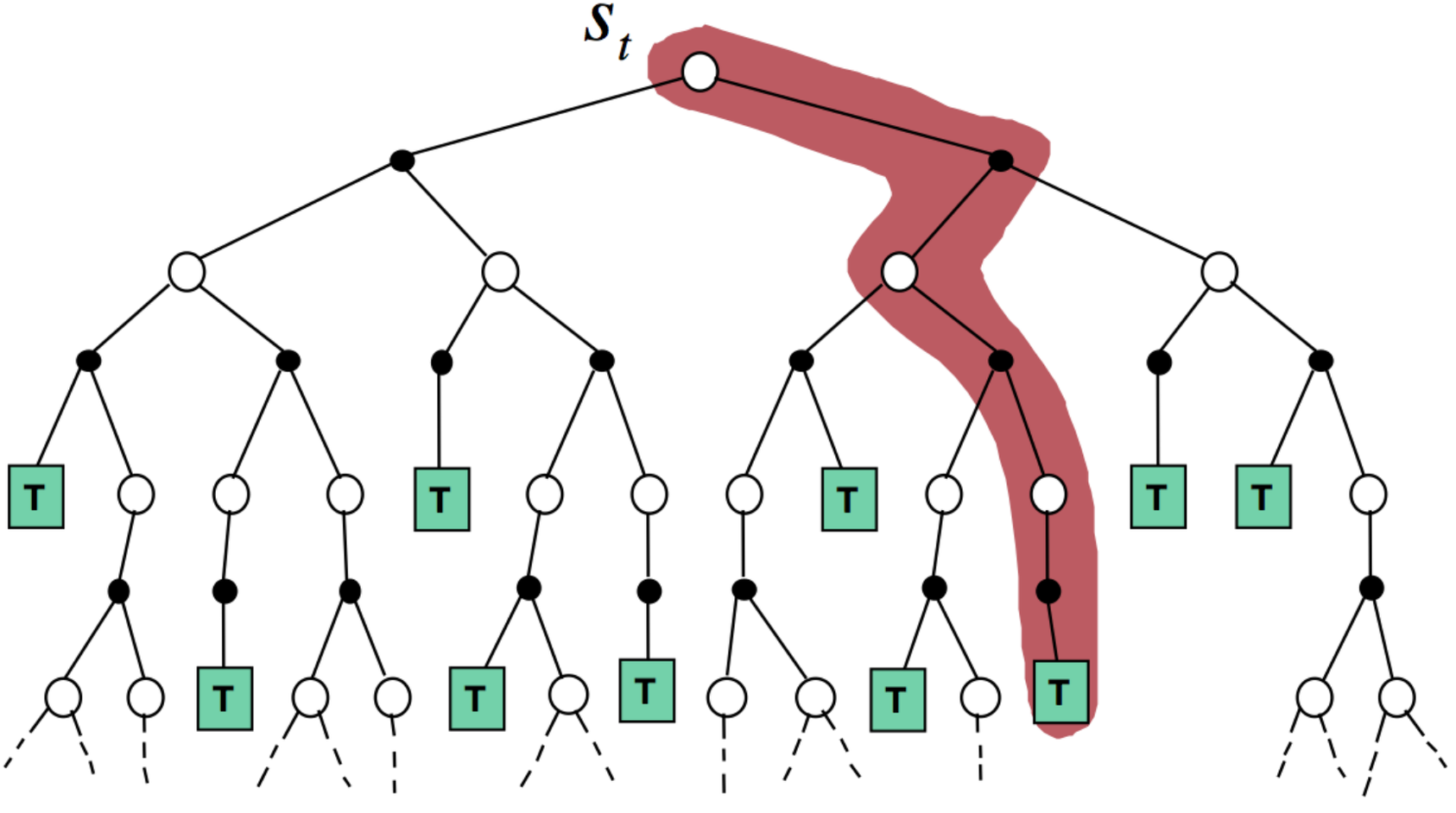
- $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

TD vs. MC

- TD can learn from incomplete sequences
  - TD can learn online after every step $\Rightarrow$ works in continuing (non-terminating) environments
  - MC must wait until an episode ends and its return is known $\Rightarrow$ works for episodic (terminating) environments only
- Convergence
  - TD is faster in practice (but still no theoretical results)
  - MC has good convergence properties (even with value function approximation)
  - TD(0) converges to $v_\pi(s)$ (not always with function approximation)
  - MC not sensitive to initial values
  - TD sensitive

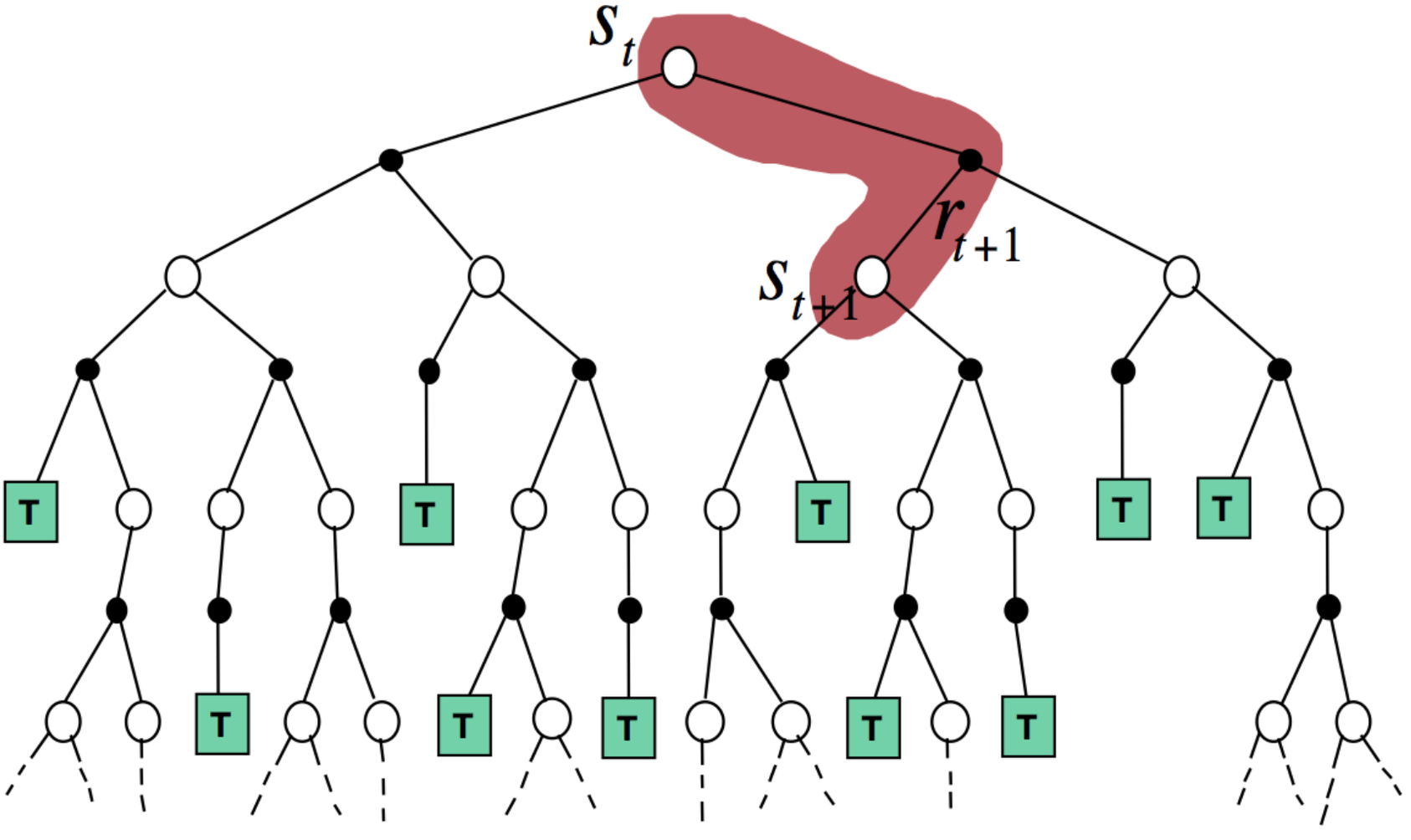
Monte-Carlo Backup

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$



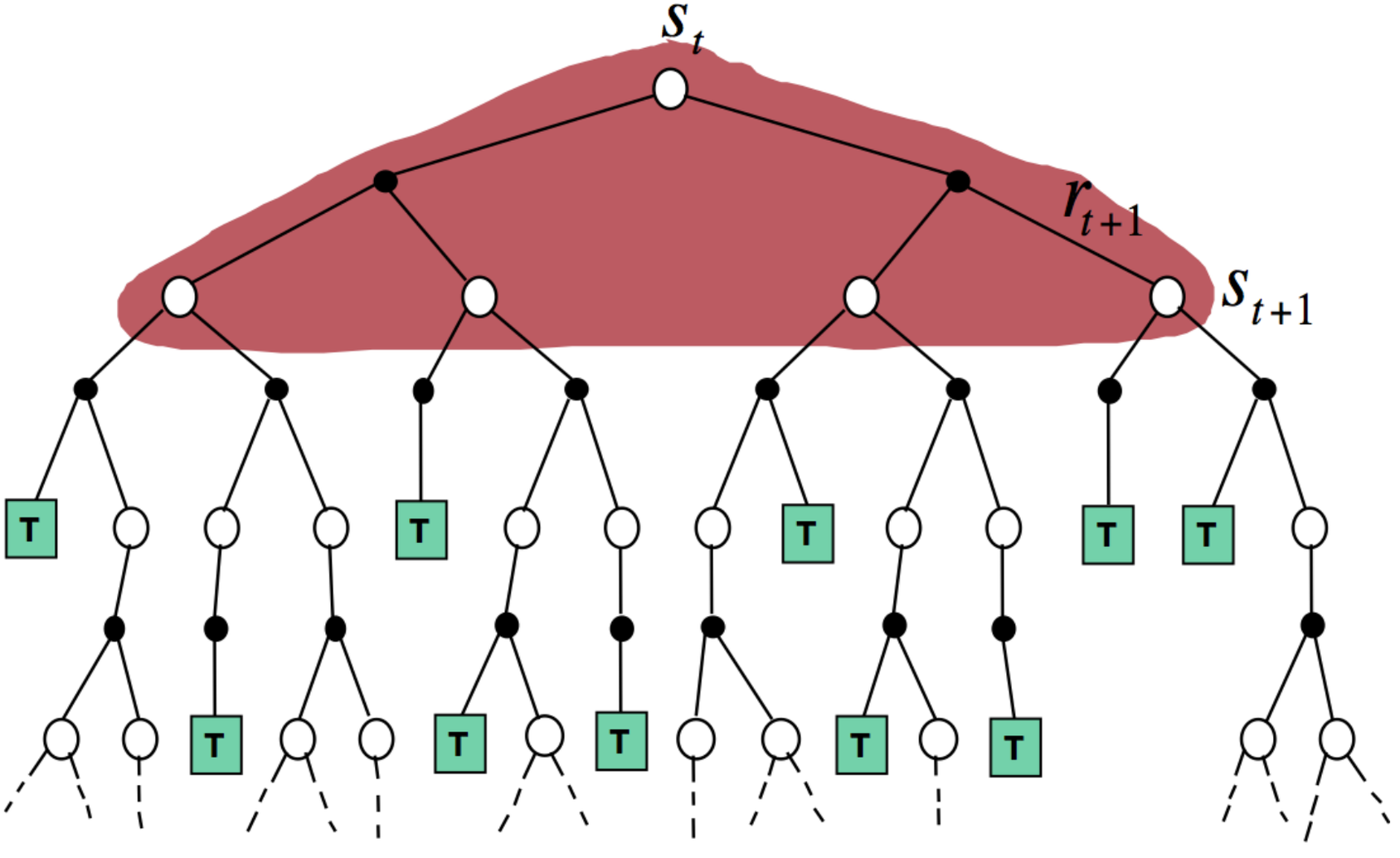
Temporal-Difference Backup

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$



Dynamic Programming Backup

$$V(S_t) \leftarrow \mathbb{E}_\pi \left(R_{t+1} + \gamma V(S_{t+1}) \right)$$



TD(λ)

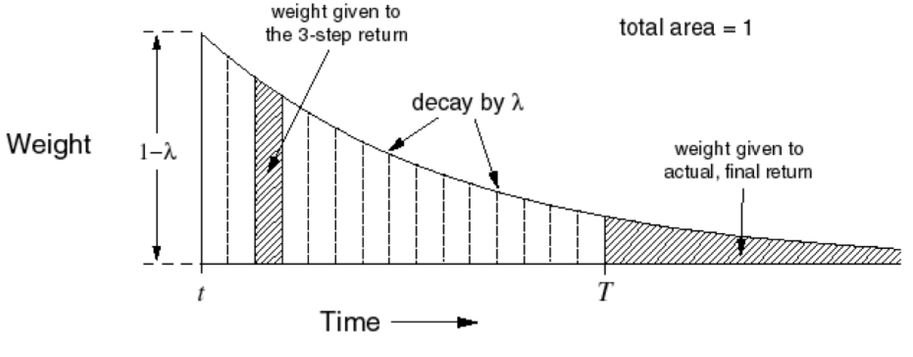
- Consider n-step returns:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- Note that $G_t^{(1)}$ is a target for TD(0), while $G_t^{(\infty)}$ a target for MC
- TD(λ) prediction combines all $G_t^{(n)}$ using weighted average:

$$G_t^{\lambda} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

- This version of the TD(λ) is limited to episodic tasks
- TD(λ) with *eligibility traces* works even for incomplete tasks
- In practice TD(λ) gives the best results



Generalized Policy Iteration

- Recall the *policy iteration* which iteratively repeats the following two steps:
  - **Evaluate** value function: estimate v_π using, e.g., iterative *policy evaluation* or *value iteration*
  - **Improve** the policy: get $\pi' \geq \pi$ by, e.g., greedy policy improvement
- Can we get convergence for different *evaluation* and *improvement* approaches?
- What about using MC for the evaluation? There are two problems:
  - greedy improvement over $V(s)$ requires a model of the MDP:
    $\pi'(s) = \text{argmax}_a \left(\mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')\right)$
    $\Longrightarrow$ **use** $Q(s,a)$ **instead**: $\pi'(s) = \text{argmax}_a Q(s,a)$,
  - **greedy strategy prevents exploration**!

Exploration by ϵ-greedy Policy

- Simplest idea for continual exploration: ϵ-greedy policy
- ϵ-greedy policy selects a random action with probability ϵ otherwise it selects maximum valued actions:

$$\pi(a|s) = \begin{cases} \frac{\epsilon}{|\mathcal{A}(s)|} \text{ for non-greedy actions} \\ 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|} \text{ for the greedy action} \end{cases}$$

- For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$.i.e., policy improvement works (see seminar)
- *MC policy iteration* = MC evaluation + ϵ-greedy policy improvement

Greedy in the Limit with Infinite Exploration (GLIE)

- We can let MC converge which might be slow...
- Or we can run it for a limited number of episodes, (e.g., a single one) before improving using ϵ-greedy policy

Definition 9. *Greedy in the Limit with Infinite Exploration (GLIE)*

- *All state-action pairs are explored infinitely many times,*

$$\lim_{k \leftarrow \infty} N_k(s, a) = \infty$$

- *The policy converges to a greedy policy:*

$$\lim_{k \leftarrow \infty} \pi_k(a|s) = \mathbb{I}\left\{ a = \operatorname*{argmax}_{a \in \mathcal{A}} Q_k(s, a') \right\}$$

- GLIE ensures convergence
- Example: ϵ-greedy is GLIE if ϵ reduces to zero at $\epsilon_k = \frac{1}{k}$

SARSA

- SARSA is a favourite method which uses TD instead of MC

Sarsa: An on-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, 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

Theorem 1. *SARSA converges, i.e.,* $Q(s,a) \Rightarrow q_*(s,a)$ *if:*

- *GLIE sequence of policies* $\pi_t(a|s)$
- *Robbins-Monro sequence of step-sizes* α_t *satisfies:*

$$\sum_{t=1}^{\infty} \alpha_t = \infty \text{ and } \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Off-Policy Learning

- Evaluate *target policy* $\pi(a|s)$ why following a different *behaviour policy* $\mu(a|s)$
- Why?
  - Learn from observing other humans (agents)
  - Re-use experience generated from old policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$
  - Learn about *optimal* policy while following *exploratory* policy

Q-Learning

- Watkins 1989
- Next action A_t is chosen using *behaviour policy* μ
- But we consider an alternative successor action A' using *target policy* π
- Update $Q(S_t, A_t)$ towards the alternative action:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t))$$

- We allow both policies to be **improved**
- The *target policy* π is **greedy** w.r.t. to $Q(s, a)$:

$$\pi(S_{t+1}) = \operatorname*{argmax}_{a'} Q(S_{t+1}, a')$$

- The behaviour policy is μ is ϵ-**greedy** w.r.t. $Q(s, a)$

Q-Learning (contd.)

- The Q-learning target simplifies to:

$$
\begin{aligned}
& R_{t+1} + \gamma Q(S_{t+1}, A') \\
= \; & R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_{a'} Q(S_{t+1}, a')) \\
= \; & R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')
\end{aligned}
$$

Q-learning: An off-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, 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

Value Function Approximation

- So far we have presented value functions $V(s)$ and $Q(s, a)$ as *lookup tables*
- This is impractical for large MDPs:
  - Too many states/actions to store in memory
  - Too slow to learn the value for each state/action
- Idea: **use function approximation**:

$$\hat{V}(S, \boldsymbol{w}) \approx V(S)$$
$$\hat{Q}(S, A, \boldsymbol{w}) \approx Q(S, A)$$

- Use any paradigm available: linear regression, neural networks, KNN, decision trees, . . .

ANNs for Value Function Approximation

- When mean-squared loss is used:

$$\mathcal{L}(\boldsymbol{w}) = \mathbb{E}_\pi \left([v_\pi(S) - \hat{V}(S, \boldsymbol{w})]^2 \right)$$

- We end up with the following Stochastic Gradient Descent update:

$$\Delta w = \eta [v_\pi(S_t) - \hat{V}(S_t, \boldsymbol{w})] \nabla_{\boldsymbol{w}} \hat{V}(S_t, \boldsymbol{w})$$

  where η is the *learning rate*, $v_\pi(S) - \hat{V}(S, \boldsymbol{w})$ is the error and $\nabla_{\boldsymbol{w}} \hat{V}(S, \boldsymbol{w})$ can be evaluated using back propagation

- Problem: we don't know $v_\pi \Longrightarrow$
  - for MC approach use the return G_t instead
  - for TD(0) use $R_{t+1} + \gamma V(S_{t+1})$
  - for TD(λ) use G_t^λ

Batch Learning

- In the previous online setup the neural network was continually updated and the samples were immediately discarded
- In many cases it is better to construct a training dataset and use batch learning, for Monte-Carlo it might be:

$$\mathcal{T} = \{(S_1, G_1), (S_2, G_2), \ldots\}$$

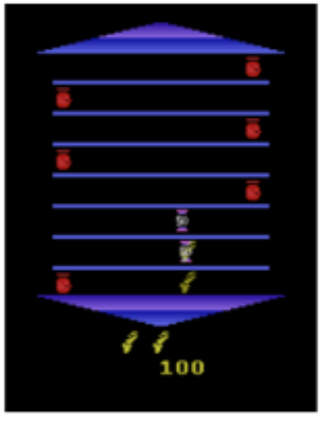
- Random sampling from $\mathcal{T}$ also "decorrelates" the samples
- This approach is called the **experience replay**

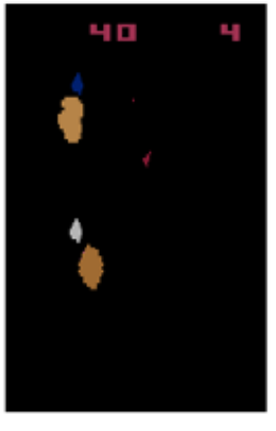
Example: Playing Atari 2600 Games

- Mnih et al.: *Human-level control through deep reinforcement learning*, 2015
- You get 160×250 RGB images at $60Hz$. The task is to find a *policy* which chooses one of 18 possible actions (9 joystick positions, fire button on/off) lead
- More than 50 games are available. See

  http://www.arcadelearningenvironment.org/





Atari: Deep Q-Network (DQN) Approach

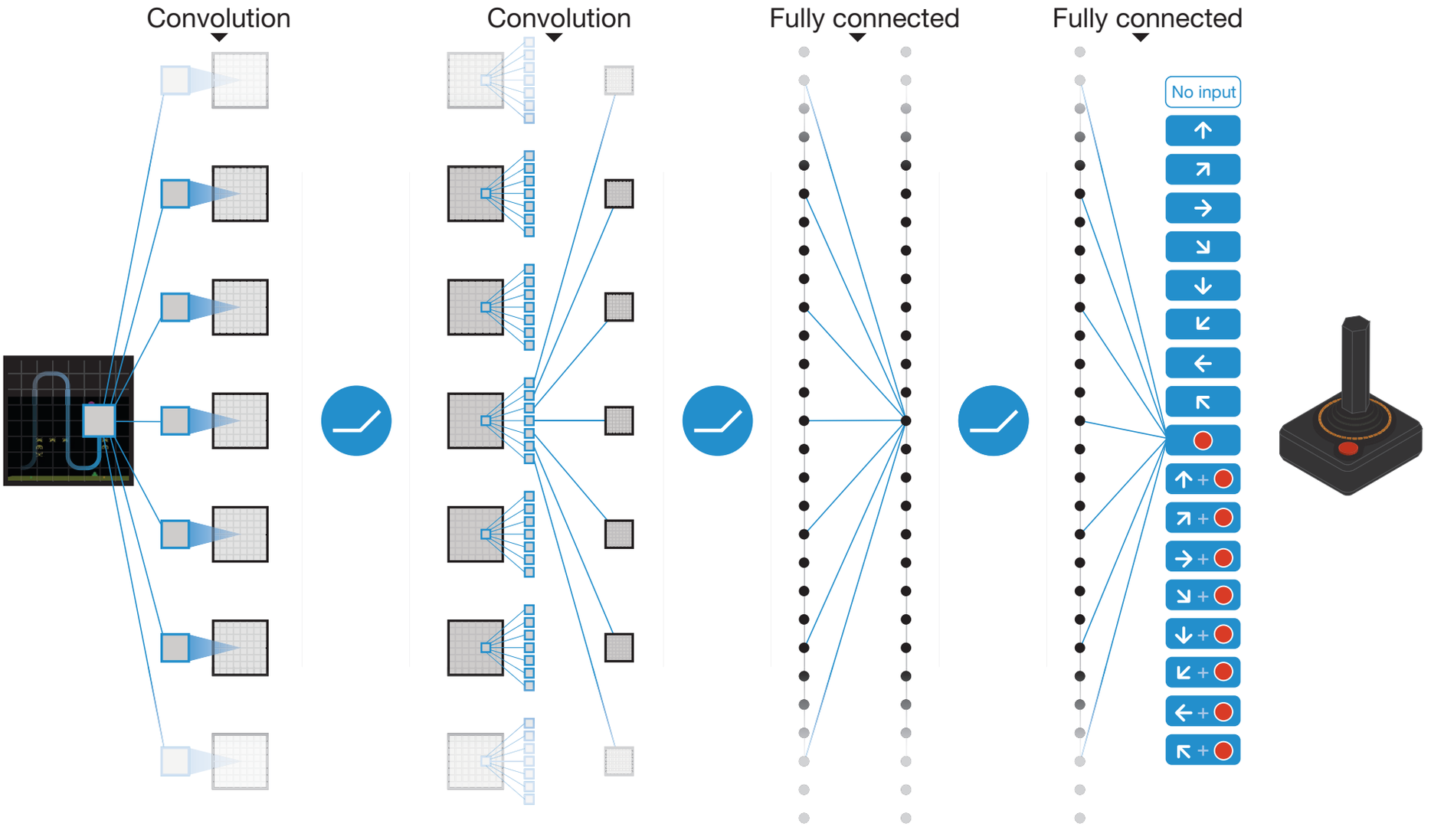
DQN uses **experience replay** and **fixed Q-targets**

- Take an action a_t according to ϵ-greedy policy
- Store a transition $(s_t, a_t, r_{t+1}, s_{t+1})$ into replay memory $\mathcal{T}$
- Sample a random mini-batch from $\mathcal{T}$
- Compute Q-learning targets w.r.t. old, fixed parameters $\bar{\boldsymbol{w}}$
- Optimize mean-squared error between Q-network and Q-learning targets:

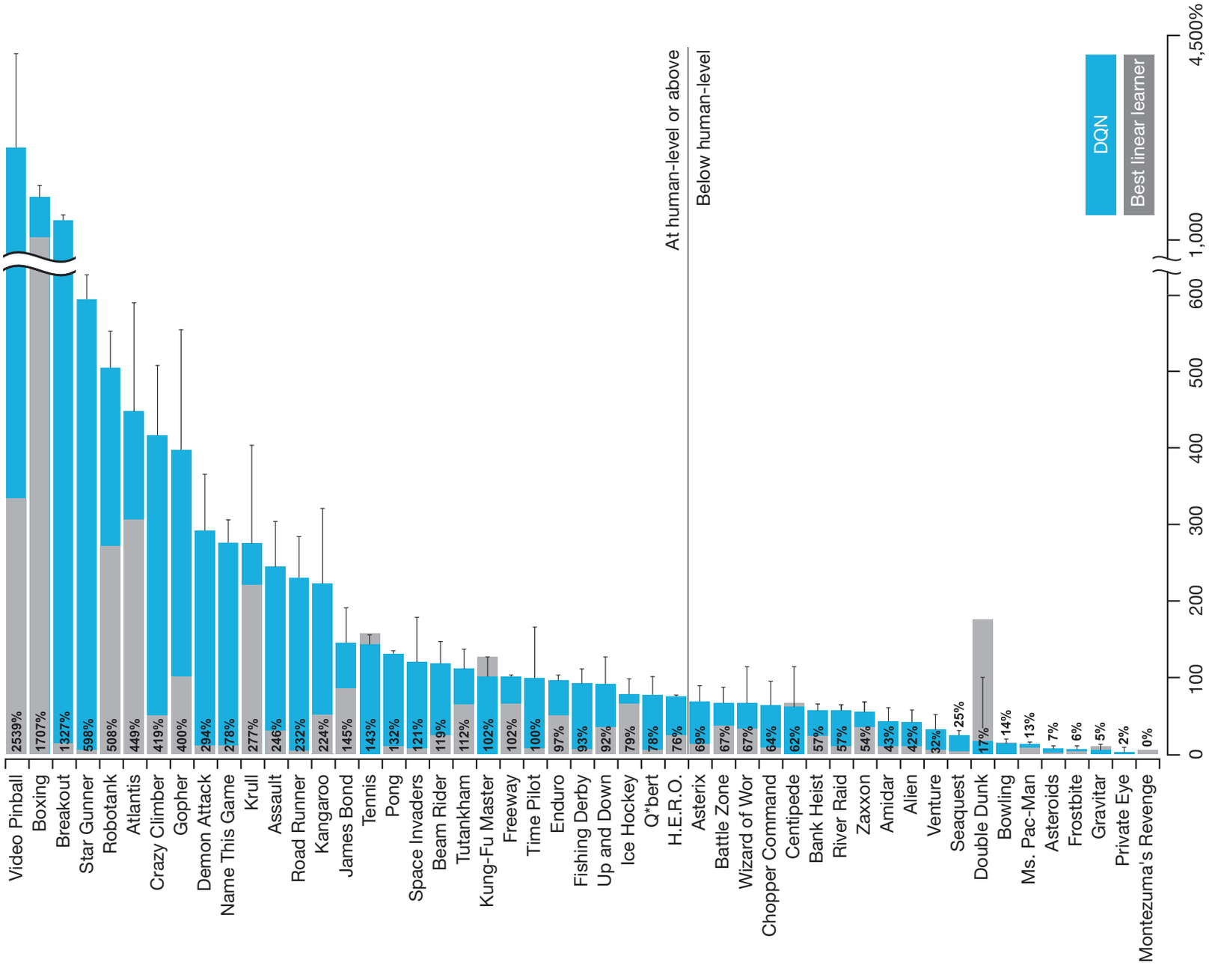
$$\mathcal{L}(\boldsymbol{w}) = \mathbb{E}_{s,a,r,s' \sim \mathcal{T}} \left(\left[r + \gamma \max_{a'} Q(s', a', \bar{\boldsymbol{w}}) - Q(s, a, \boldsymbol{w}) \right]^2 \right)$$

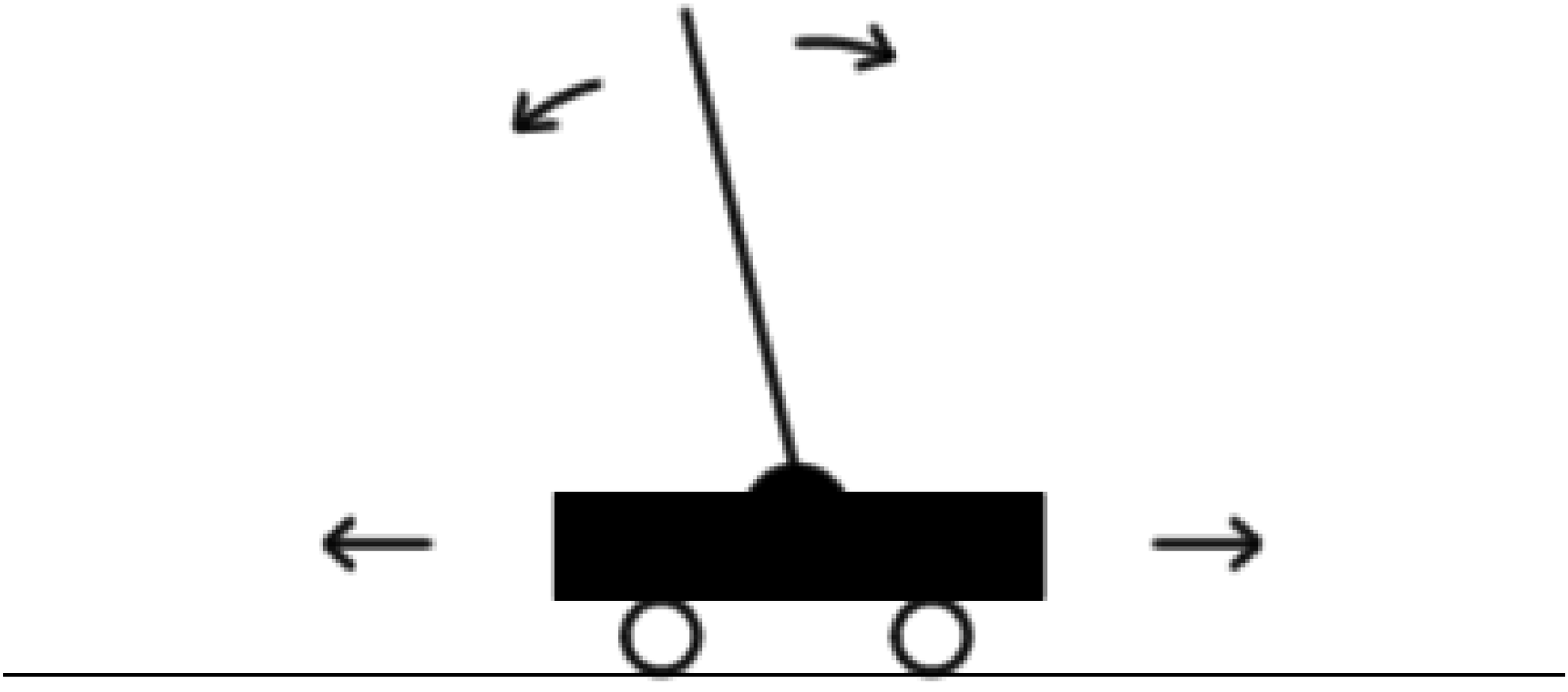
- A variant of SGD used

Atari: DQN Architecture

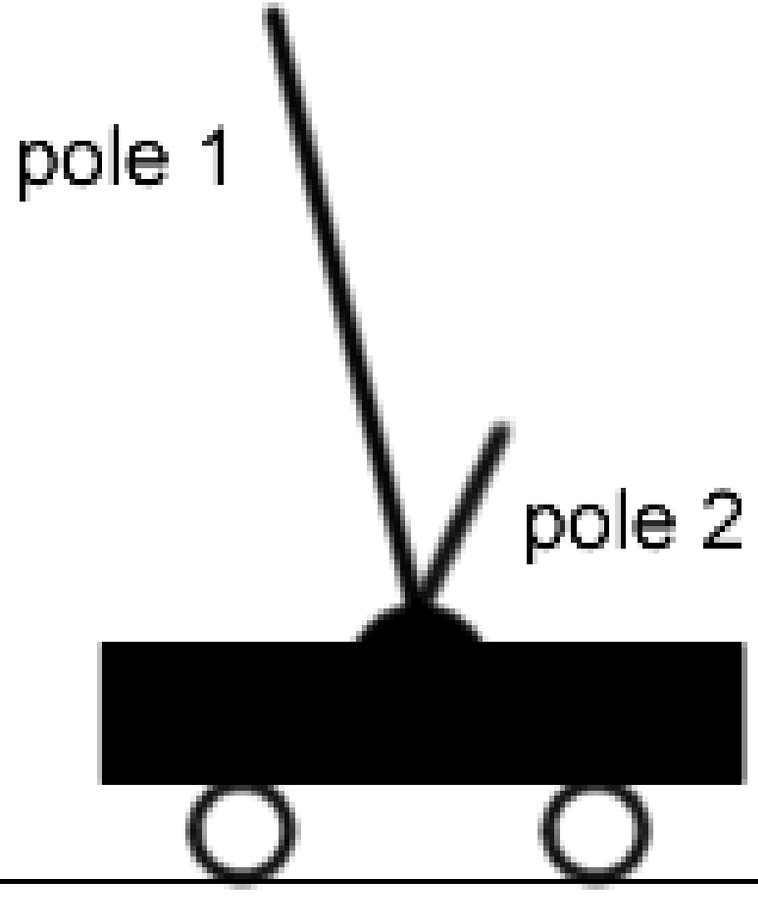


Atari: Results

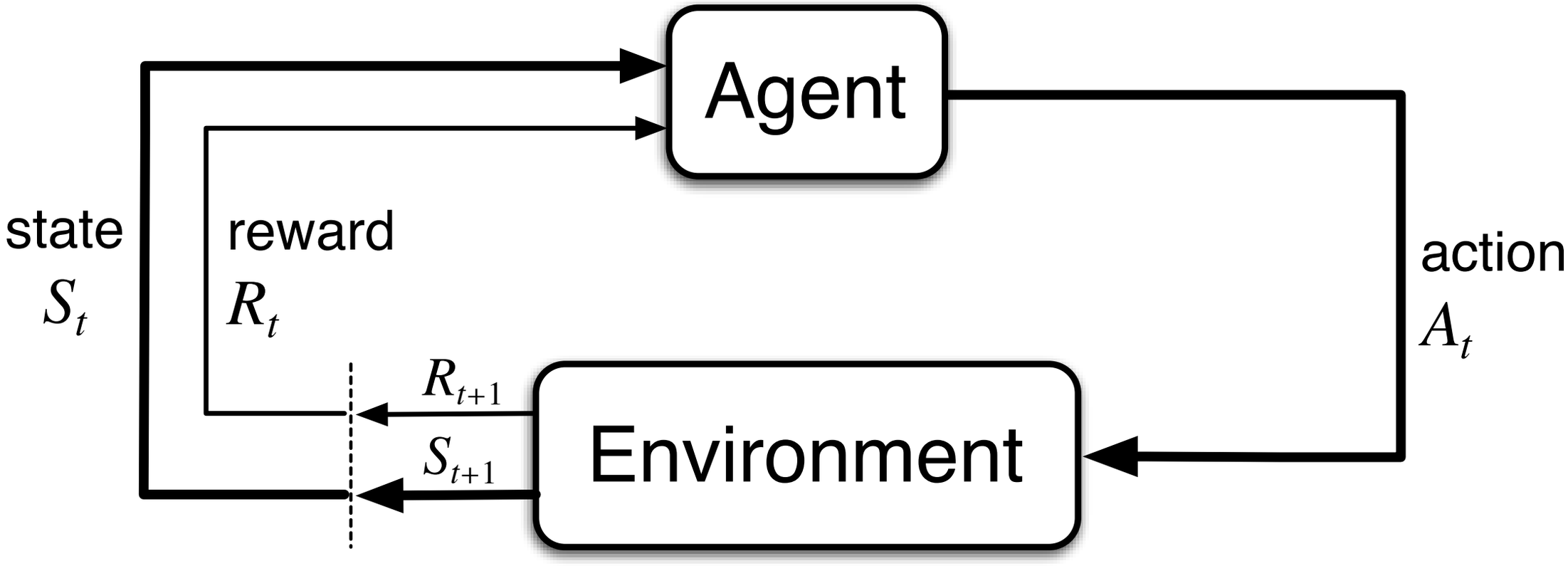


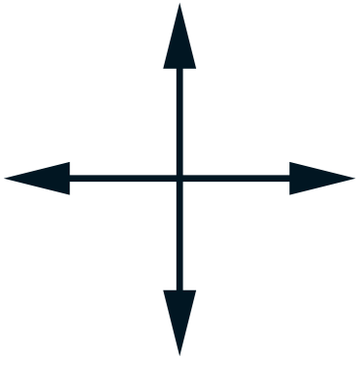


Single Pole Balancing



Double Pole Balancing


Agent
state
S_t
reward
R_t
action
A_t
R_{t+1}
S_{t+1}
Environment



actions

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

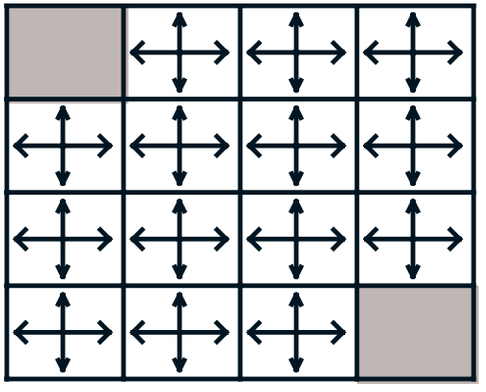
$R = -1$
on all transitions

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 0$

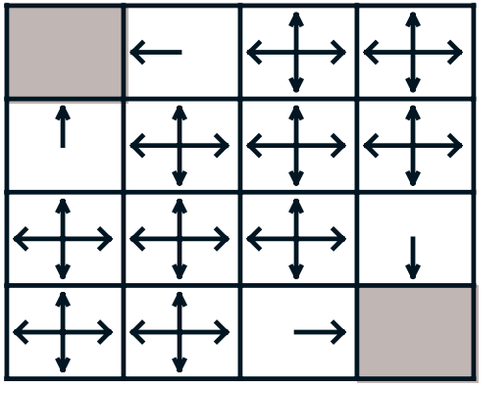
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

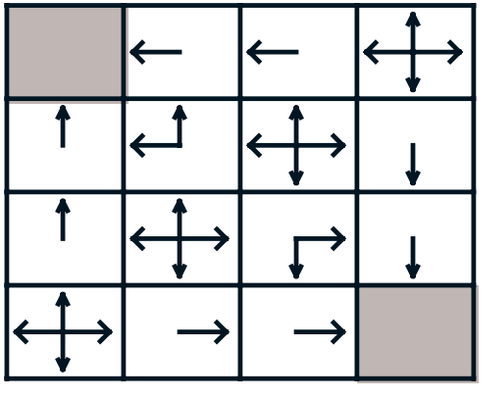
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



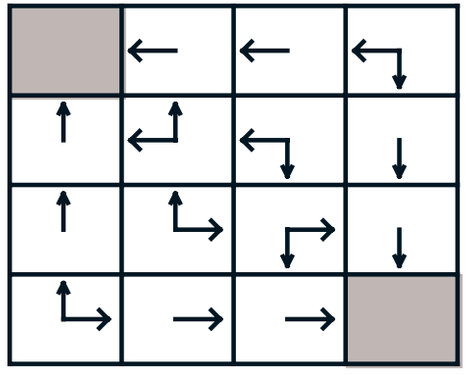
$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



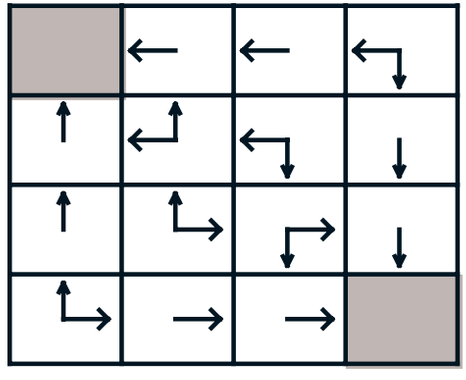
$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



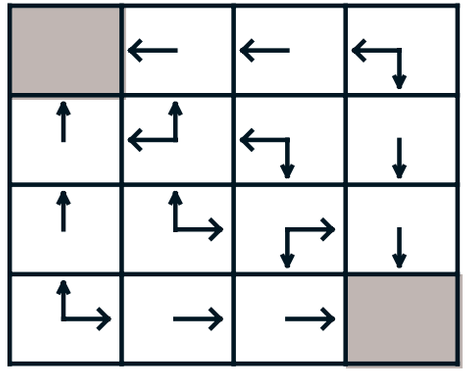
$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

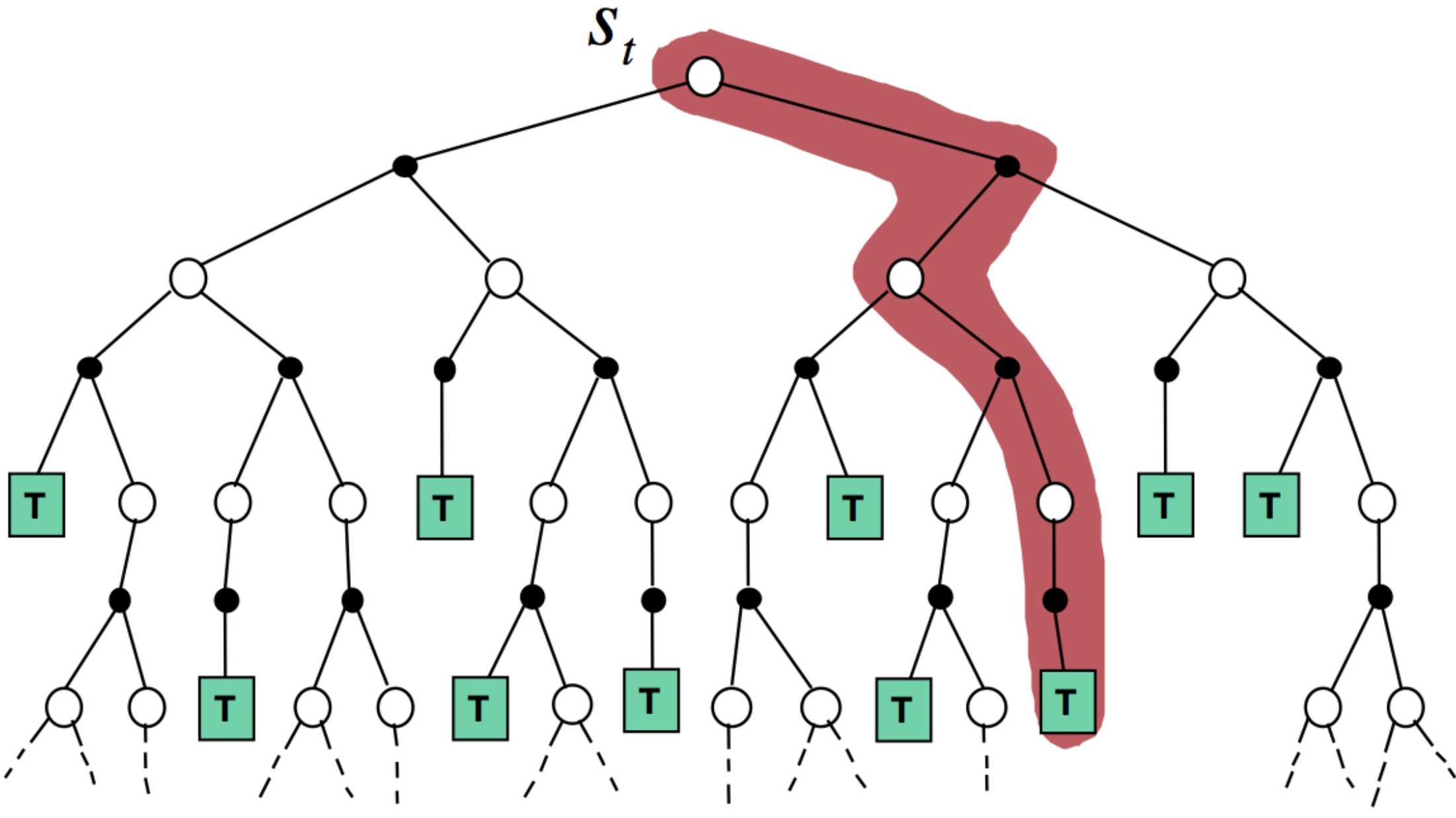


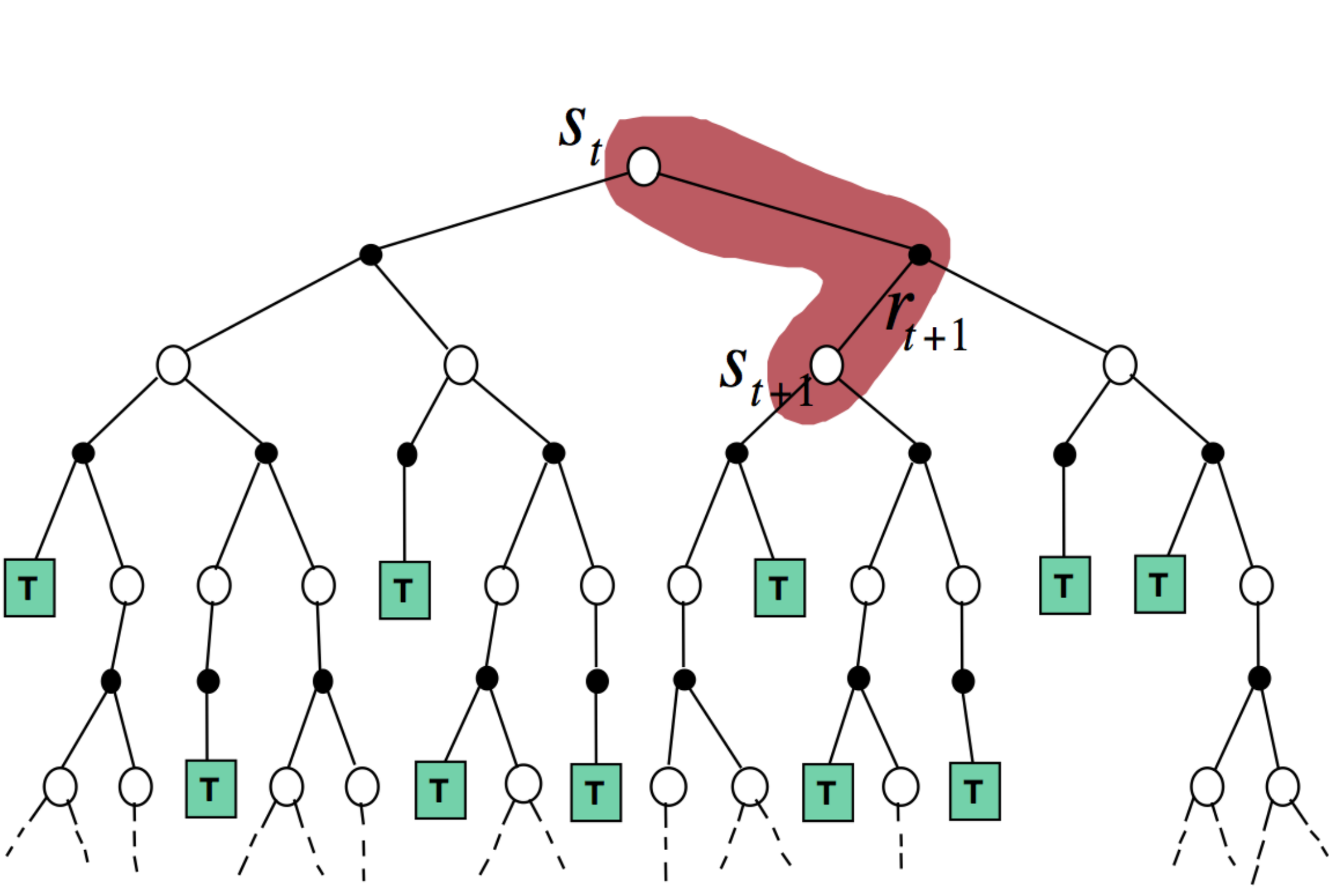
$k = \infty$

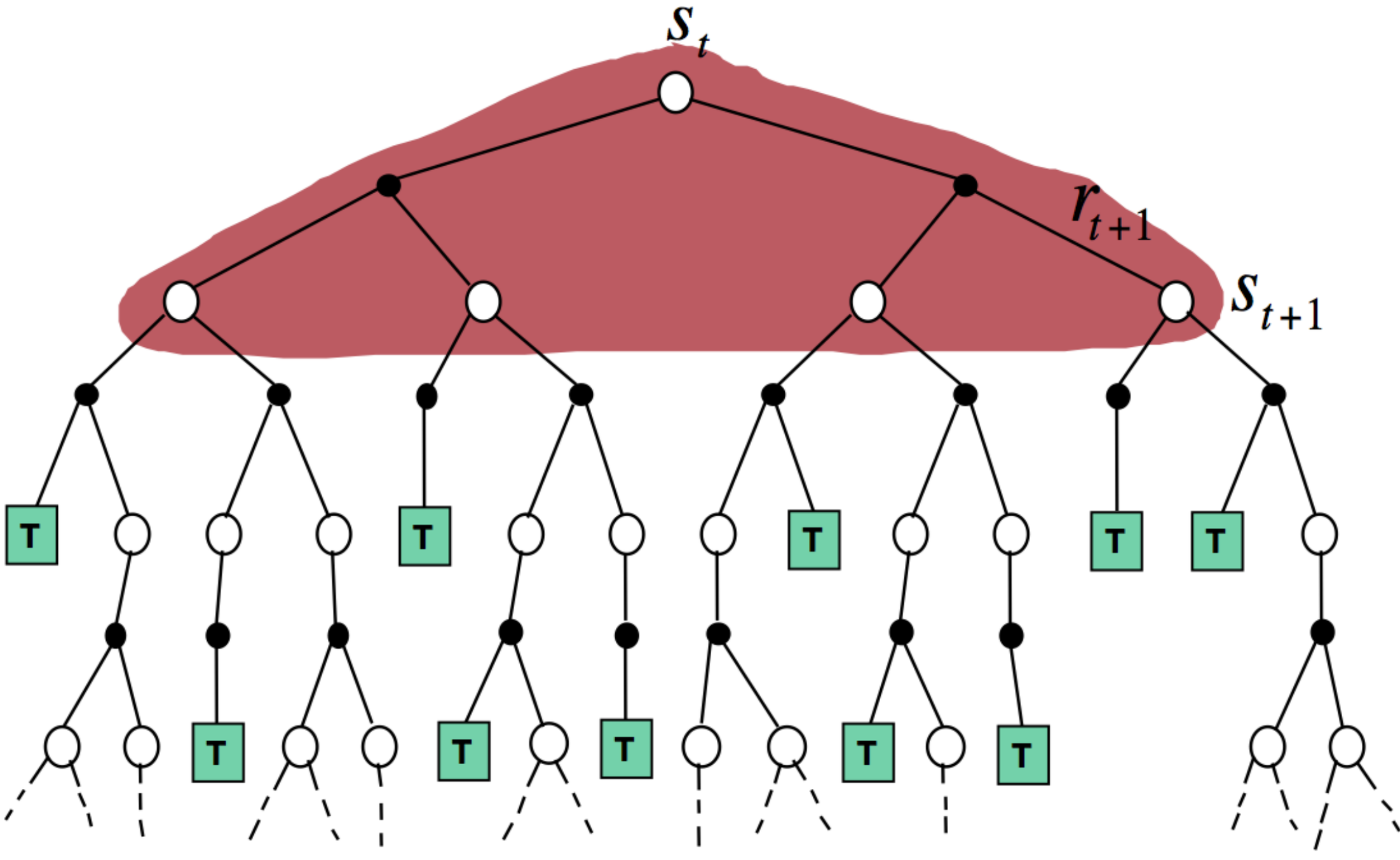
0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

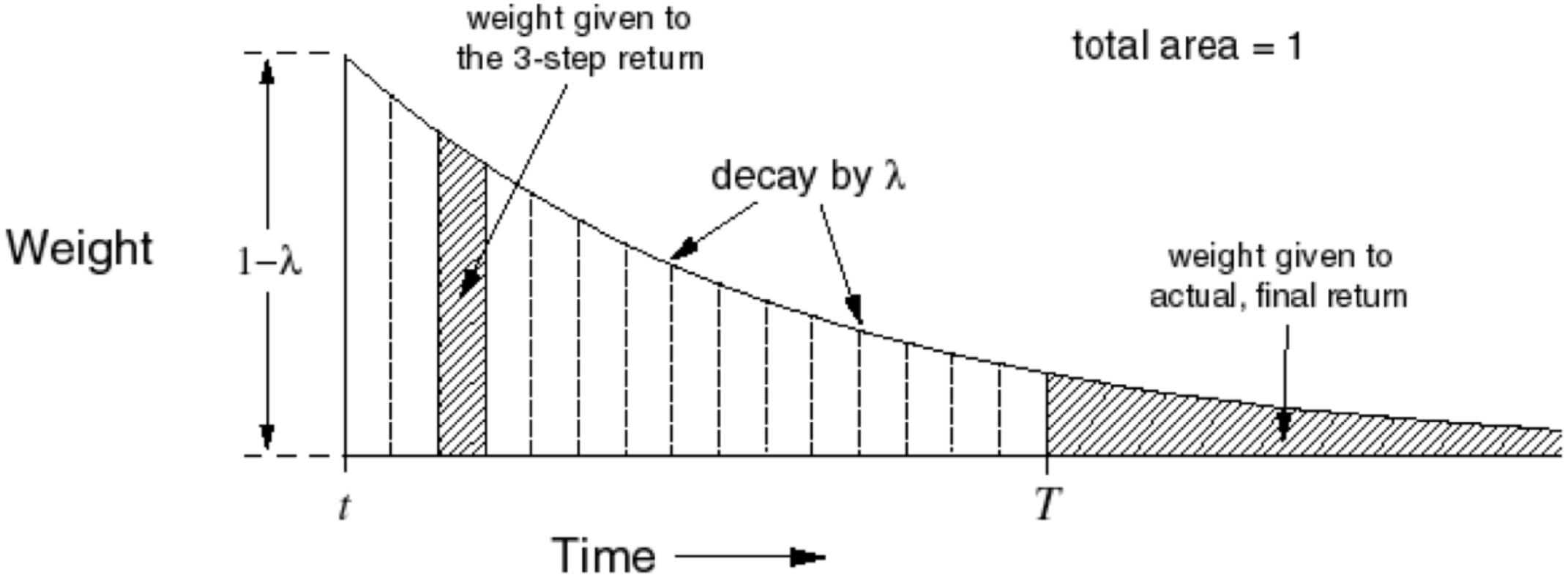


optimal policy


S_t
T
T
T
T
T
T
T
T
T
T


s_t
r_{t+1}
s_{t+1}
T
T
T
T
T
T
T
T
T
T


s_t
r_{t+1}
s_{t+1}
T
T
T
T
T
T
T
T
T
T

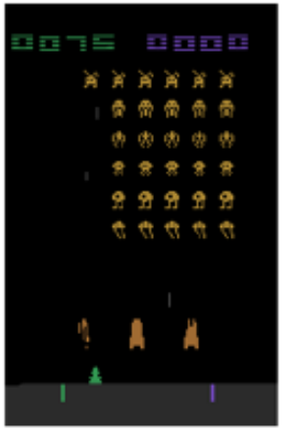

weight given to
the 3-step return
total area = 1
decay by λ
Weight
1−λ
weight given to
actual, final return
t
T
Time

Sarsa: An on-policy TD control algorithm

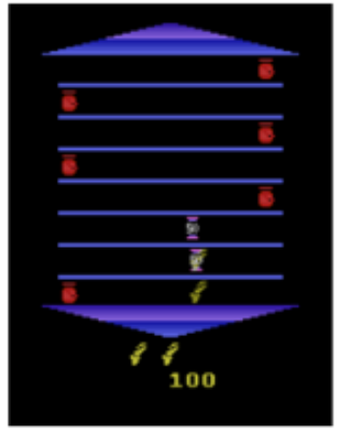
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Repeat (for each step of episode):
        Take action A, observe 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

Q-learning: An off-policy TD control algorithm

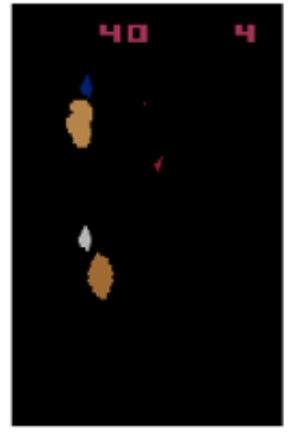
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, 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

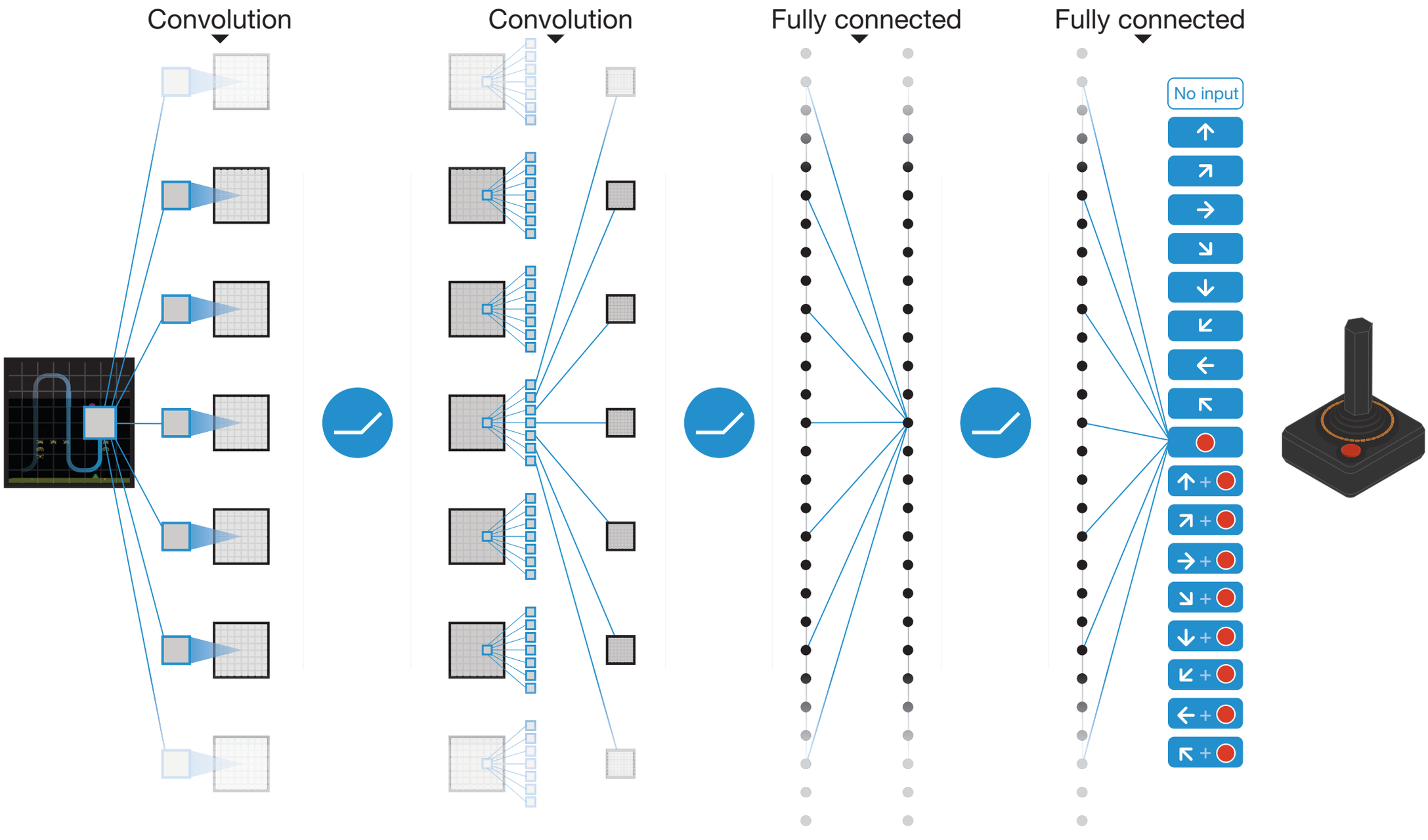


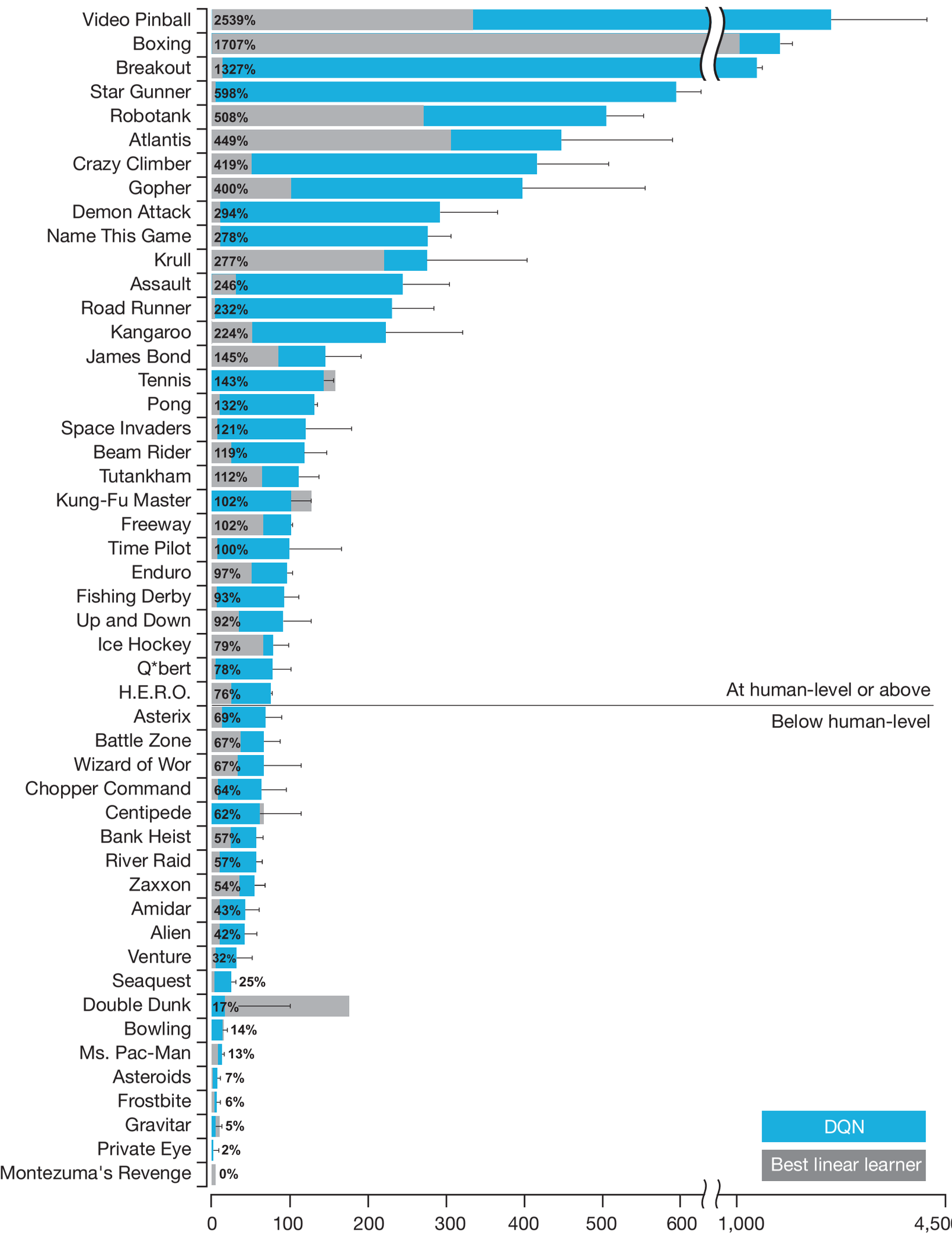

70
RD 1


100


2000
19:59
cce


40
4


Convolution
Convolution
Fully connected
Fully connected
No input


Video Pinball
Boxing
Breakout
Star Gunner
Robotank
Atlantis
Crazy Climber
Gopher
Demon Attack
Name This Game
Krull
Assault
Road Runner
Kangaroo
James Bond
Tennis
Pong
Space Invaders
Beam Rider
Tutankham
Kung-Fu Master
Freeway
Time Pilot
Enduro
Fishing Derby
Up and Down
Ice Hockey
Q*bert
H.E.R.O.
Asterix
Battle Zone
Wizard of Wor
Chopper Command
Centipede
Bank Heist
River Raid
Zaxxon
Amidar
Alien
Venture
Seaquest
Double Dunk
Bowling
Ms. Pac-Man
Asteroids
Frostbite
Gravitar
Private Eye
Montezuma's Revenge
2539%
1707%
1327%
598%
508%
449%
419%
400%
294%
278%
277%
246%
232%
224%
145%
143%
132%
121%
119%
112%
102%
102%
100%
97%
93%
92%
79%
78%
76%
69%
67%
67%
64%
62%
57%
57%
54%
43%
42%
32%
25%
17%
14%
13%
7%
6%
5%
2%
0%
At human-level or above
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%