UNIVERSITY OF TECHNOLOGY
IN THE EUROPEAN CAPITAL OF CULTURE
CHEMNITZ

Deep Reinforcement Learning

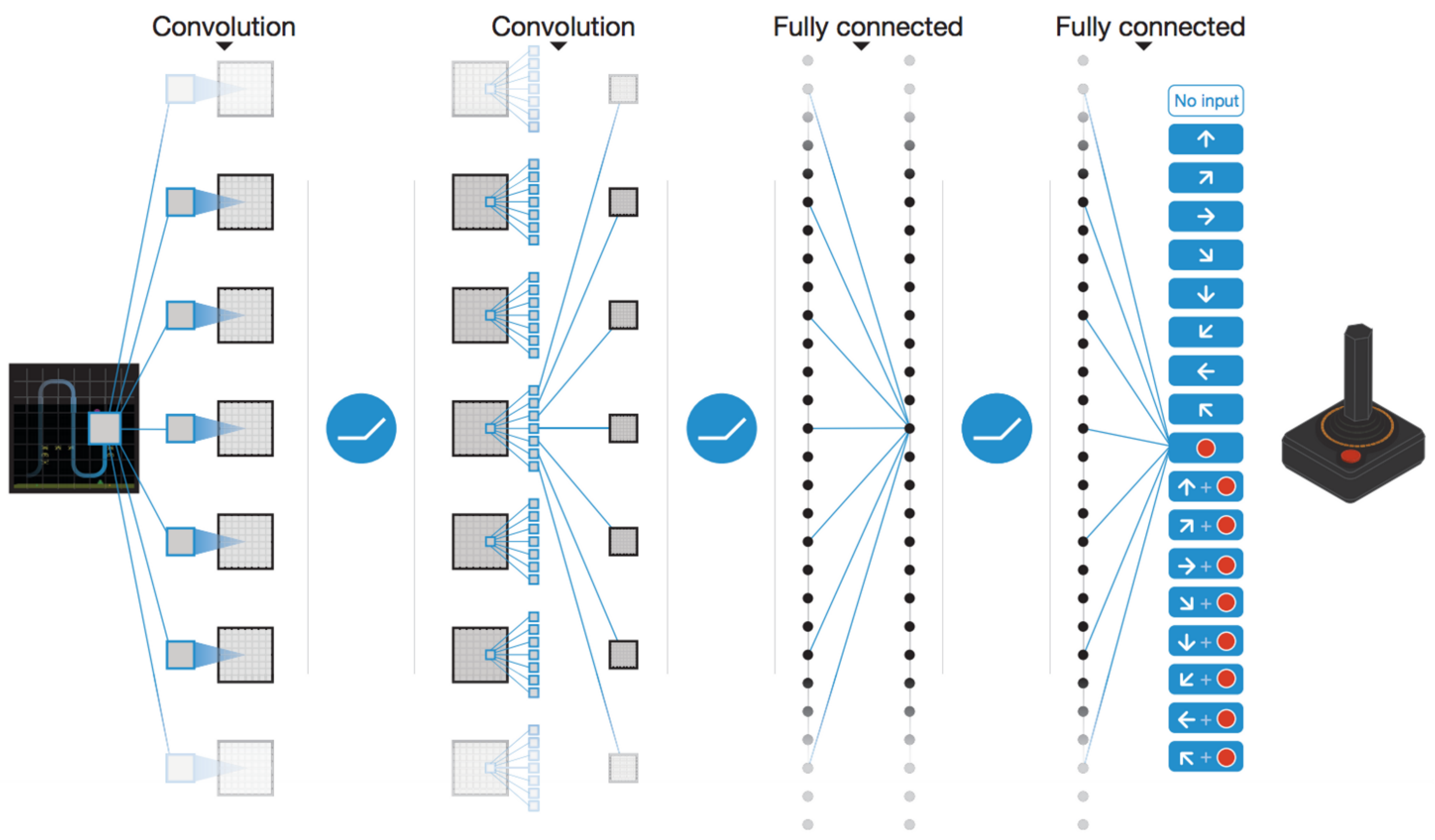
Policy gradient

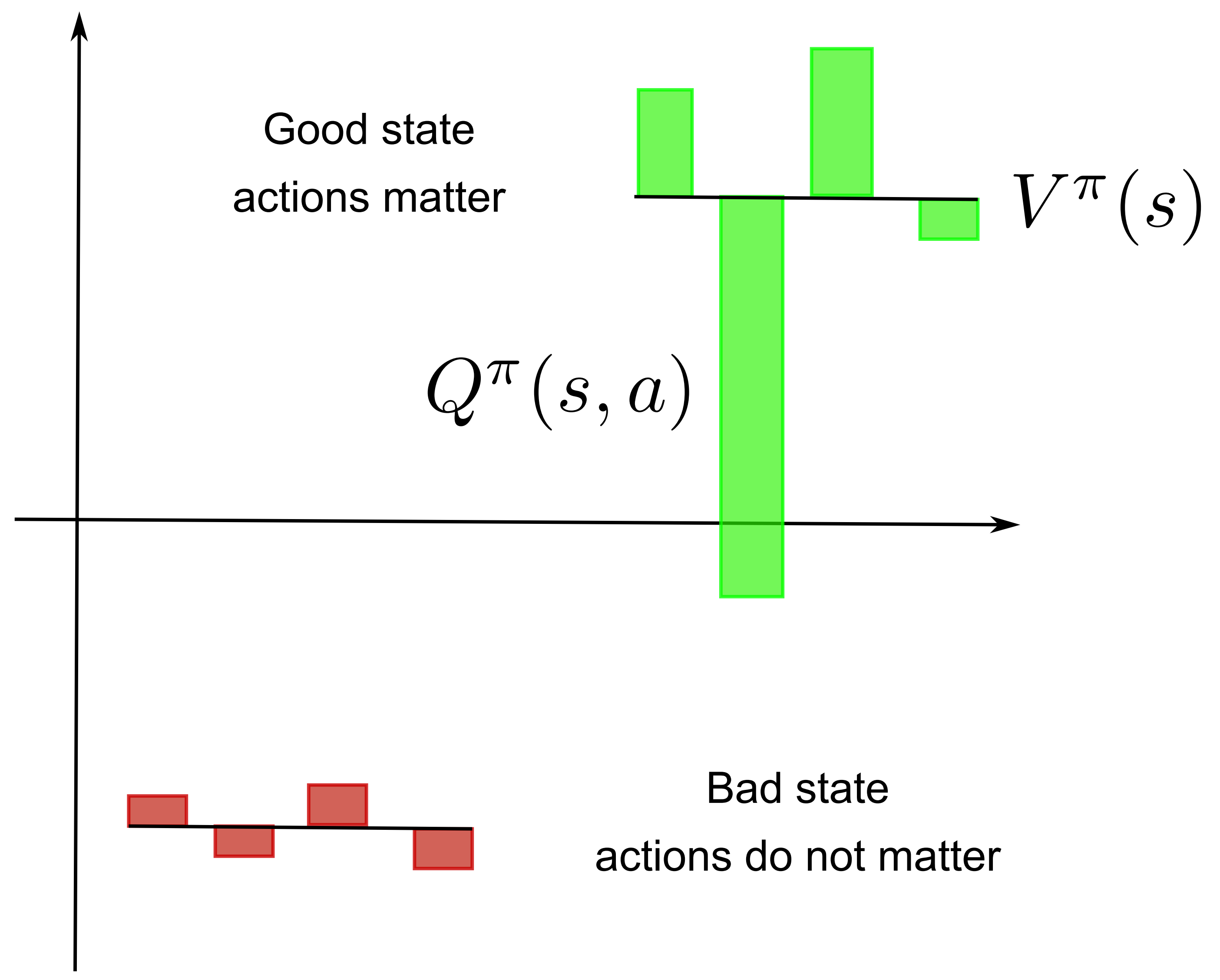
Julien Vitay

Professur für Künstliche Intelligenz - Fakultät für Informatik

1 - Policy Search

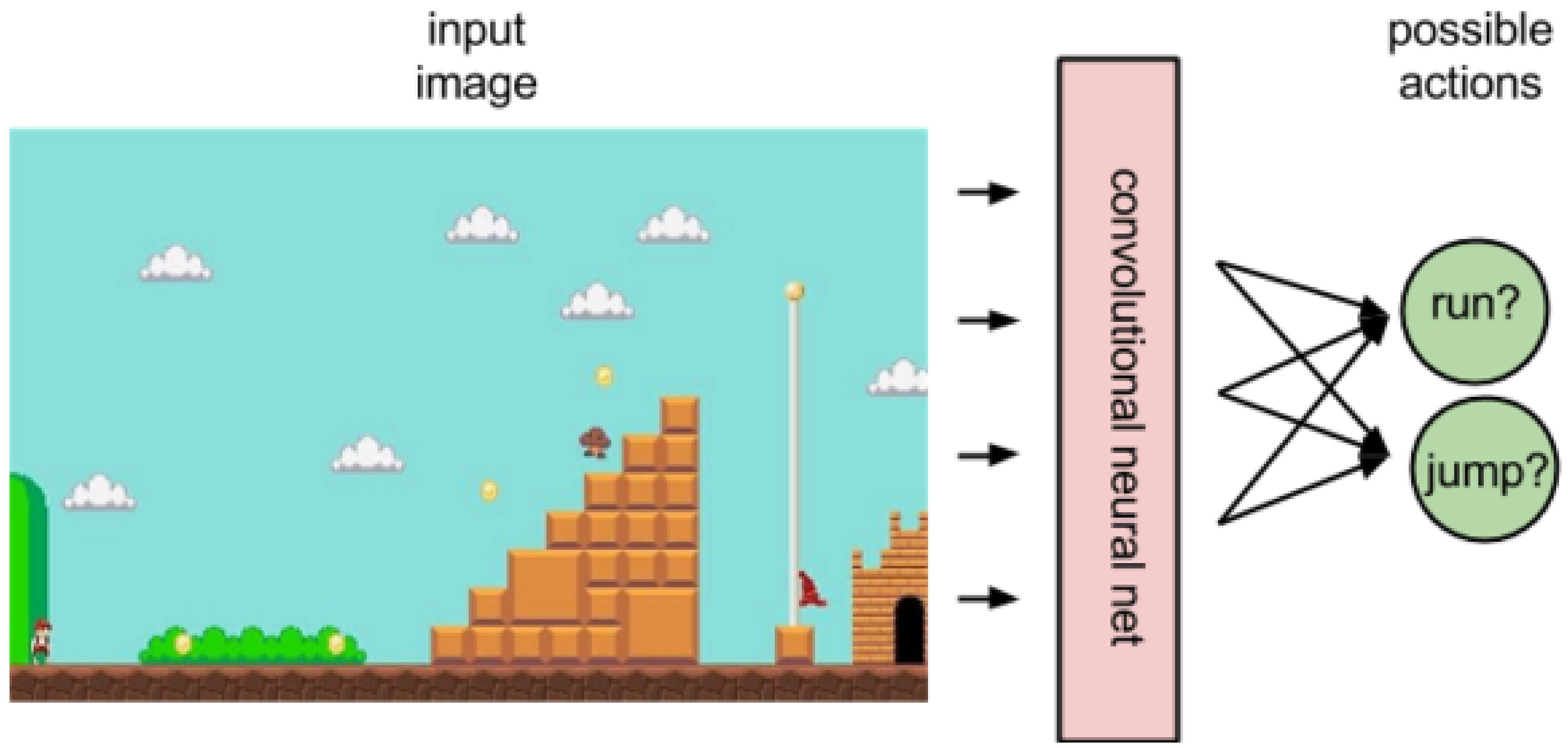
Policy search





- Learning directly the Q-values in value-based methods (DQN) suffers from many problems:
  - The Q-values are **unbounded**: they can take any value (positive or negative), so the output layer must be linear.
  - The Q-values have a **high variability**: some (s, a) pairs have very negative values, others have very positive values. Difficult to learn for a NN.
  - Works only for small **discrete action spaces**: need to iterate over all actions to find the greedy action.

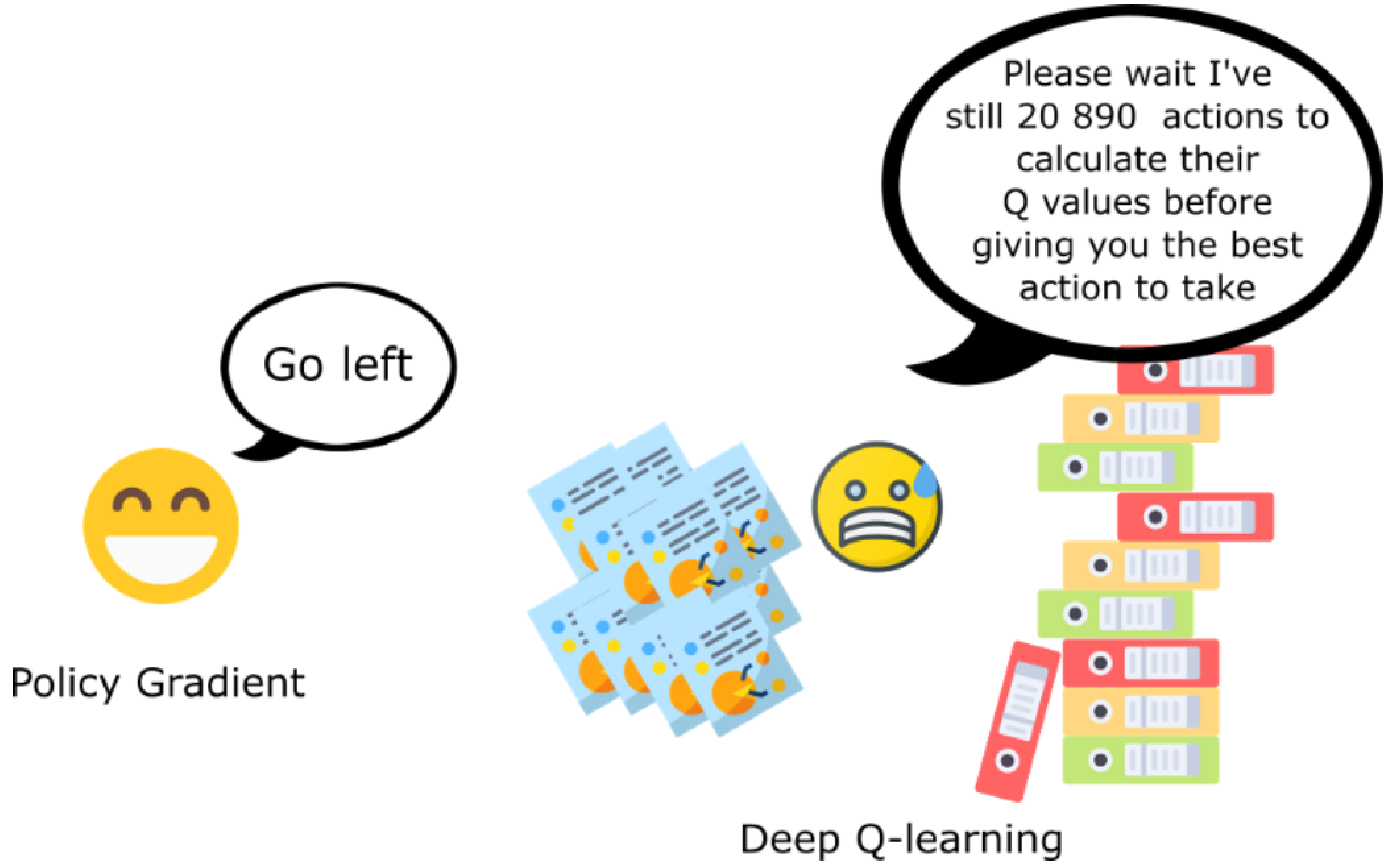
Policy search



- Instead of learning the Q-values, one could approximate directly the policy $\pi_\theta(s, a)$ with a neural network.
- $\pi_\theta(s, a)$ is called a **parameterized policy**: it depends directly on the parameters θ of the NN.
- For discrete action spaces, the output of the NN can be a **softmax** layer, directly giving the probability of selecting an action.
- For continuous action spaces, the output layer can directly control the effector (joint angles).

Policy search

- Parameterized policies can represent continuous policies and avoid the curse of dimensionality.


Go left
Policy Gradient
Please wait I've still 20 890 actions to calculate their Q values before giving you the best action to take
Deep Q-learning

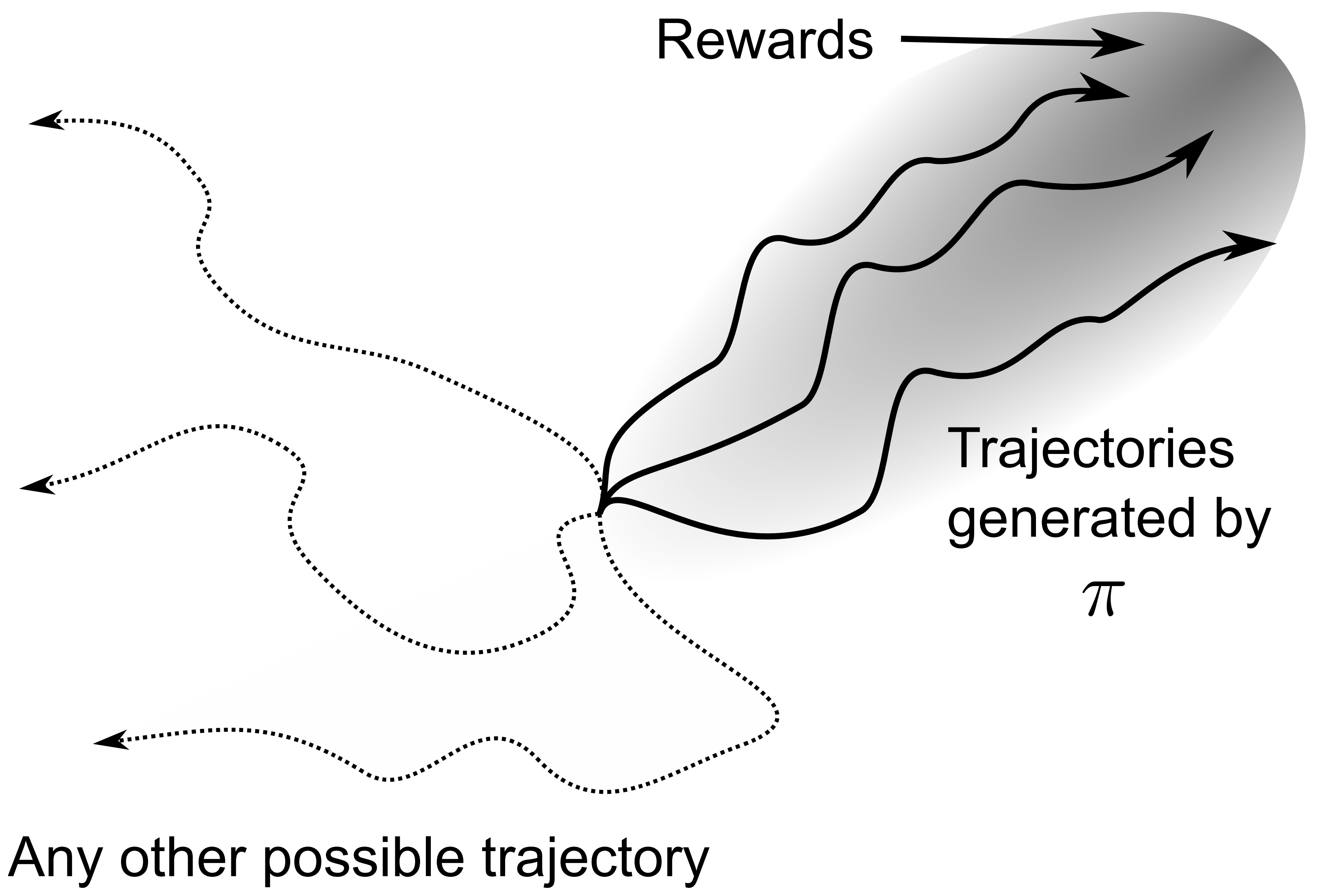
Source: https://www.freecodecamp.org/news/an-introduction-to-policy-gradients-with-cartpole-and-doom-495b5ef2207f/

Policy search

- **Policy search** methods aim at maximizing directly the expected return over all possible trajectories (episodes) $\tau = (s_0, a_0, \dots, s_T, a_T)$

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[R(\tau)] = \int_\tau \rho_\theta(\tau) \; R(\tau) \; d\tau$$

- All trajectories τ selected by the policy π_θ should be associated with a high expected return $R(\tau)$ in order to maximize this objective function.


Rewards
Trajectories generated by π
Any other possible trajectory

- $\rho_\theta(\tau)$ is the **likelihood** of the trajectory τ under the policy π_θ.
- This means that the optimal policy should only select actions that maximizes the expected return: exactly what we want.

Policy search

- Objective function to be maximized:

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[R(\tau)] = \int_\tau \rho_\theta(\tau) \, R(\tau) \, d\tau$$

- The objective function is however not **model-free**, as the likelihood of a trajectory does depend on the environments dynamics:

$$\rho_\theta(\tau) = p_\theta(s_0, a_0, \dots, s_T, a_T) = p_0(s_0) \, \prod_{t=0}^T \pi_\theta(s_t, a_t) \, p(s_{t+1} | s_t, a_t)$$

- The objective function is furthermore **not computable**:
  - An **infinity** of possible trajectories to integrate if the action space is continuous.
  - Even if we sample trajectories, we would need a huge number of them to correctly estimate the objective function (**sample complexity**) because of the huge **variance** of the returns.

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[R(\tau)] \approx \frac{1}{M} \, \sum_{i=1}^M R(\tau_i)$$

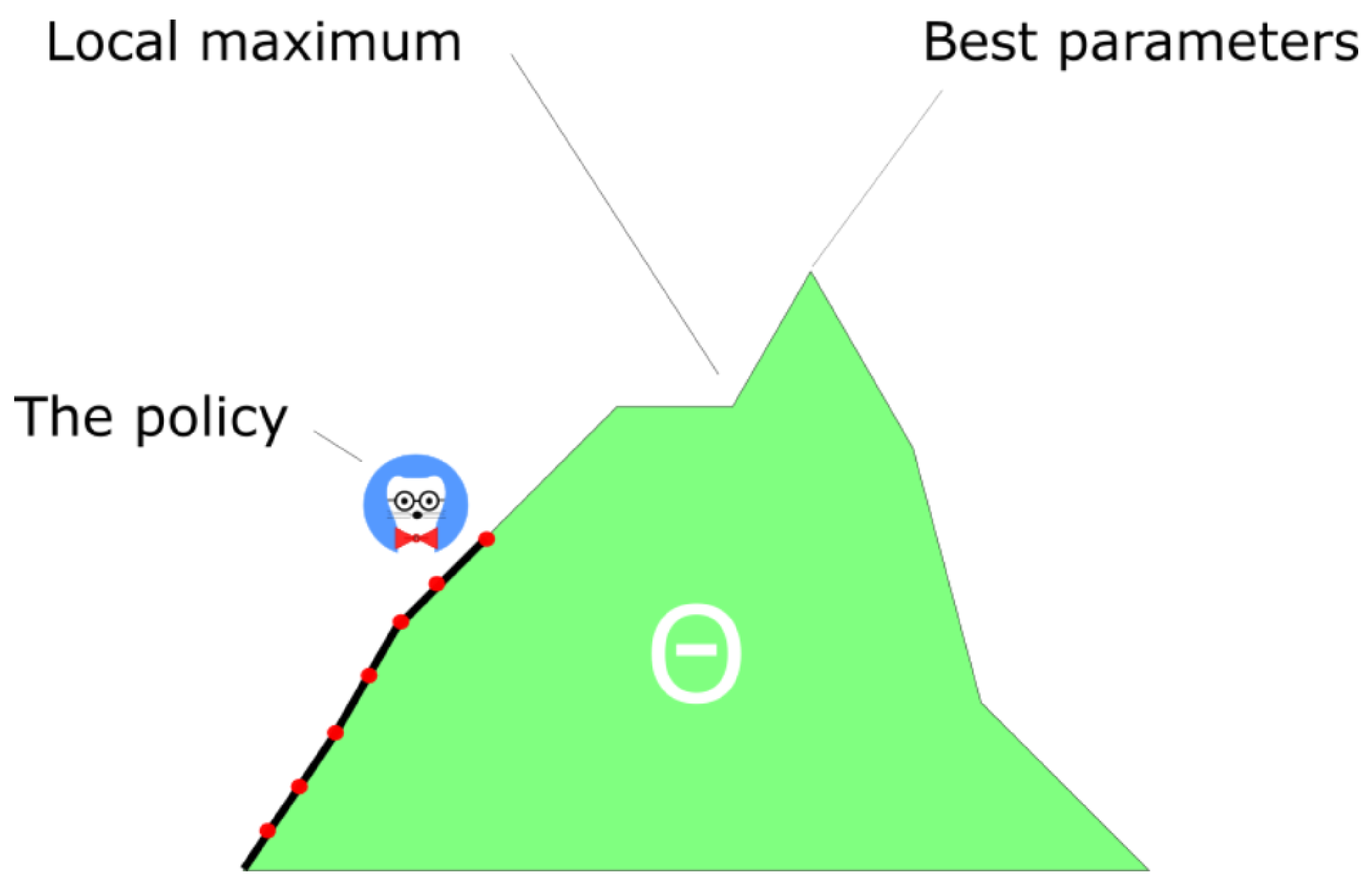
Policy gradient

- All we need to find is a computable gradient $\nabla_\theta \mathcal{J}(\theta)$ to apply gradient ascent and backpropagation.

$$\Delta \theta = \eta \, \nabla_\theta \mathcal{J}(\theta)$$

- **Policy Gradient** (PG) methods only try to estimate this gradient, but do not care about the objective function itself...

$$g = \nabla_\theta \mathcal{J}(\theta)$$



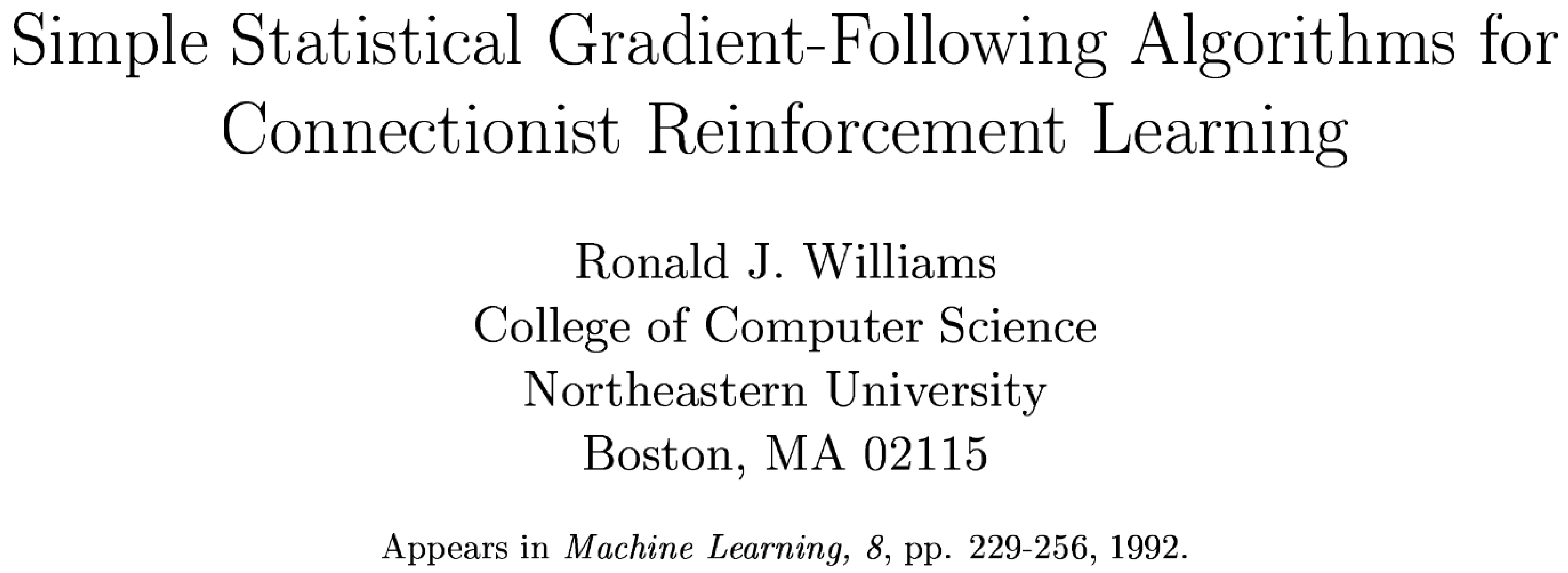
Source: https://www.freecodecamp.org/news/an-introduction-to-policy-gradients-with-cartpole-and-doom-495b5ef2207f/

- In particular, any function $\mathcal{J}'(\theta)$ whose gradient is locally the same (or has the same direction) will do:

$$\mathcal{J}'(\theta) = \alpha \, \mathcal{J}(\theta) + \beta \;\Rightarrow\; \nabla_\theta \mathcal{J}'(\theta) \propto \nabla_\theta \mathcal{J}(\theta) \;\Rightarrow\; \Delta \theta = \eta \, \nabla_\theta \mathcal{J}'(\theta)$$

- This is called **surrogate optimization**: we actually want to maximize $\mathcal{J}(\theta)$ but we cannot compute it.
- We instead create a surrogate objective $\mathcal{J}'(\theta)$ which is locally the same as $\mathcal{J}(\theta)$ and tractable.

2 - REINFORCE



Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning

Ronald J. Williams
College of Computer Science
Northeastern University
Boston, MA 02115

Appears in *Machine Learning, 8*, pp. 229-256, 1992.

REINFORCE

- The **REINFORCE** algorithm (Williams, 1992) proposes an unbiased estimate of the policy gradient:

$$\nabla_\theta \, \mathcal{J}(\theta) = \nabla_\theta \, \int_\tau \rho_\theta(\tau) \, R(\tau) \, d\tau = \int_\tau (\nabla_\theta \, \rho_\theta(\tau)) \, R(\tau) \, d\tau$$

by noting that the return of a trajectory does not depend on the weights θ (the agent only controls its actions, not the environment).

- We now use the **log-trick**, a simple identity based on the fact that:

$$\frac{d \log f(x)}{dx} = \frac{f'(x)}{f(x)}$$

or:

$$f'(x) = f(x) \times \frac{d \log f(x)}{dx}$$

to rewrite the gradient of the likelihood of a single trajectory:

$$\nabla_\theta \, \rho_\theta(\tau) = \rho_\theta(\tau) \times \nabla_\theta \log \rho_\theta(\tau)$$

REINFORCE

- The policy gradient becomes:

$$\nabla_\theta \, \mathcal{J}(\theta) = \int_\tau (\nabla_\theta \, \rho_\theta(\tau)) \, R(\tau) \, d\tau = \int_\tau \rho_\theta(\tau) \, \nabla_\theta \log \rho_\theta(\tau) \, R(\tau) \, d\tau$$

which now has the form of a mathematical expectation:

$$\nabla_\theta \, \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[\nabla_\theta \log \rho_\theta(\tau) \, R(\tau)]$$

- The policy gradient is, in expectation, the gradient of the **log-likelihood** of a trajectory multiplied by its return.

REINFORCE

- The advantage of REINFORCE is that it is **model-free**:

$$\rho_\theta(\tau) = p_\theta(s_0, a_0, \dots, s_T, a_T) = p_0(s_0) \prod_{t=0}^T \pi_\theta(s_t, a_t) p(s_{t+1} | s_t, a_t)$$

$$\log \rho_\theta(\tau) = \log p_0(s_0) + \sum_{t=0}^T \log \pi_\theta(s_t, a_t) + \sum_{t=0}^T \log p(s_{t+1} | s_t, a_t)$$

$$\nabla_\theta \log \rho_\theta(\tau) = \sum_{t=0}^T \nabla_\theta \log \pi_\theta(s_t, a_t)$$

- The transition dynamics $p(s_{t+1} | s_t, a_t)$ disappear from the gradient.
- The **Policy Gradient** does not depend on the dynamics of the environment:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[\sum_{t=0}^T \nabla_\theta \log \pi_\theta(s_t, a_t) \, R(\tau)]$$

REINFORCE algorithm

The REINFORCE algorithm is a policy-based variant of Monte-Carlo control:

- **while** not converged:
    - Sample M trajectories $\{\tau_i\}$ using the current policy π_θ and observe the returns $\{R(\tau_i)\}$.
    - Estimate the policy gradient as an average over the trajectories:

$$\nabla_\theta \mathcal{J}(\theta) \approx \frac{1}{M} \sum_{i=1}^M \sum_{t=0}^T \nabla_\theta \log \pi_\theta(s_t, a_t) \, R(\tau_i)$$

    - Update the policy using gradient ascent:

$$\theta \leftarrow \theta + \eta \, \nabla_\theta \mathcal{J}(\theta)$$

REINFORCE

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[\sum_{t=0}^T \nabla_\theta \log \pi_\theta(s_t, a_t) \, R(\tau)]$$

Advantages

- The policy gradient is **model-free**.
- Works with **partially observable** problems (POMDP): as the return is computed over complete trajectories, it does not matter whether the states are Markov or not.

Inconvenients

- Only for **episodic tasks**.
- The gradient has a **high variance**: returns may change a lot during learning.
- It has therefore a high **sample complexity**: we need to sample many episodes to correctly estimate the policy gradient.
- Strictly **on-policy**: trajectories must be frequently sampled and immediately used to update the policy.

REINFORCE with baseline

- To reduce the variance of the estimated gradient, a baseline is often subtracted from the return:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[\sum_{t=0}^T \nabla_\theta \log \pi_\theta(s_t, a_t) \, (R(\tau) - b)]$$

- As long as the baseline b is independent from θ, it does not introduce a bias:

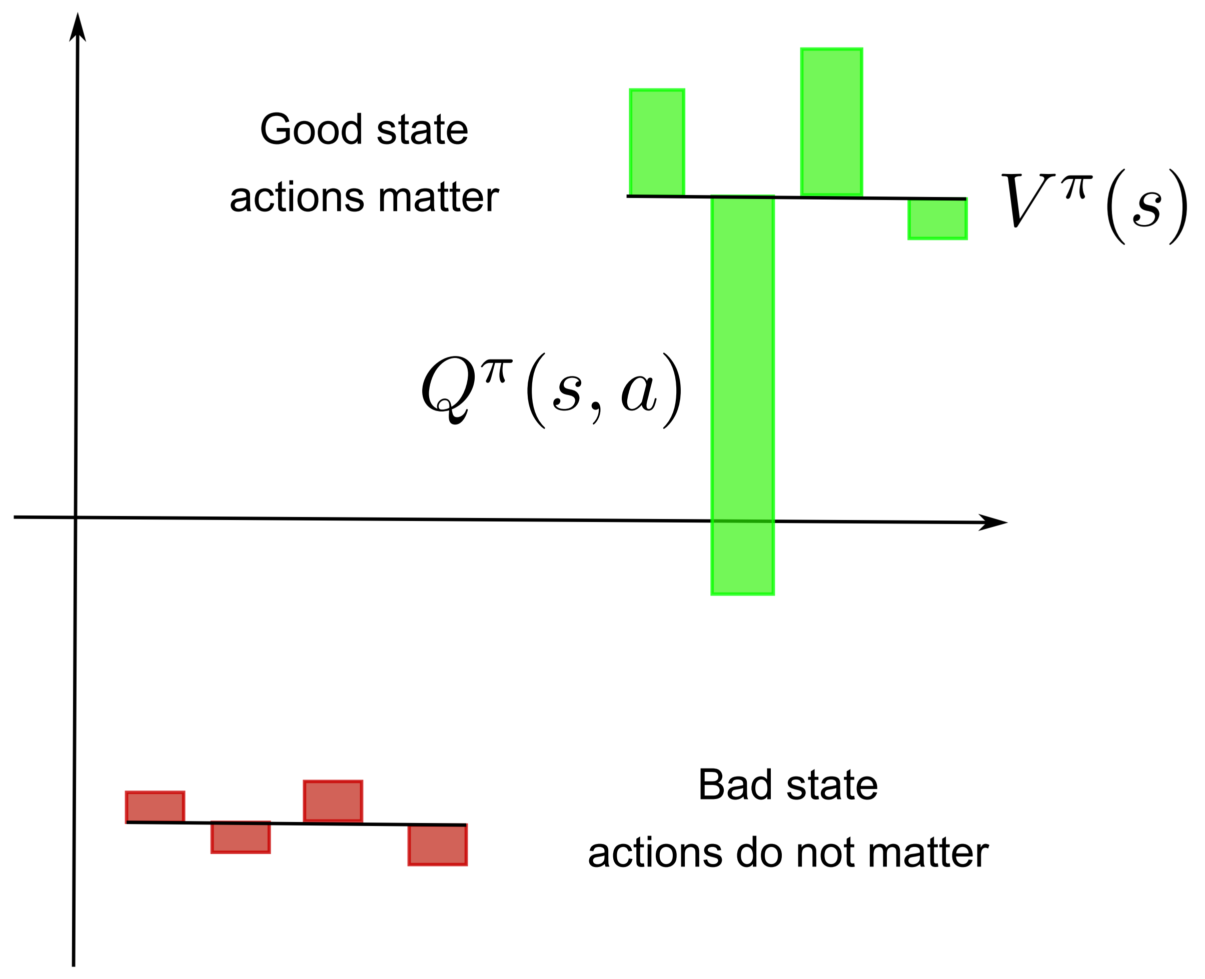
$$\begin{aligned}
\mathbb{E}_{\tau \sim \rho_\theta}[\nabla_\theta \log \rho_\theta(\tau) \, b] &= \int_\tau \rho_\theta(\tau) \nabla_\theta \log \rho_\theta(\tau) \, b \, d\tau \\
&= \int_\tau \nabla_\theta \rho_\theta(\tau) \, b \, d\tau \\
&= b \, \nabla_\theta \int_\tau \rho_\theta(\tau) \, d\tau \\
&= b \, \nabla_\theta 1 \\
&= 0
\end{aligned}$$

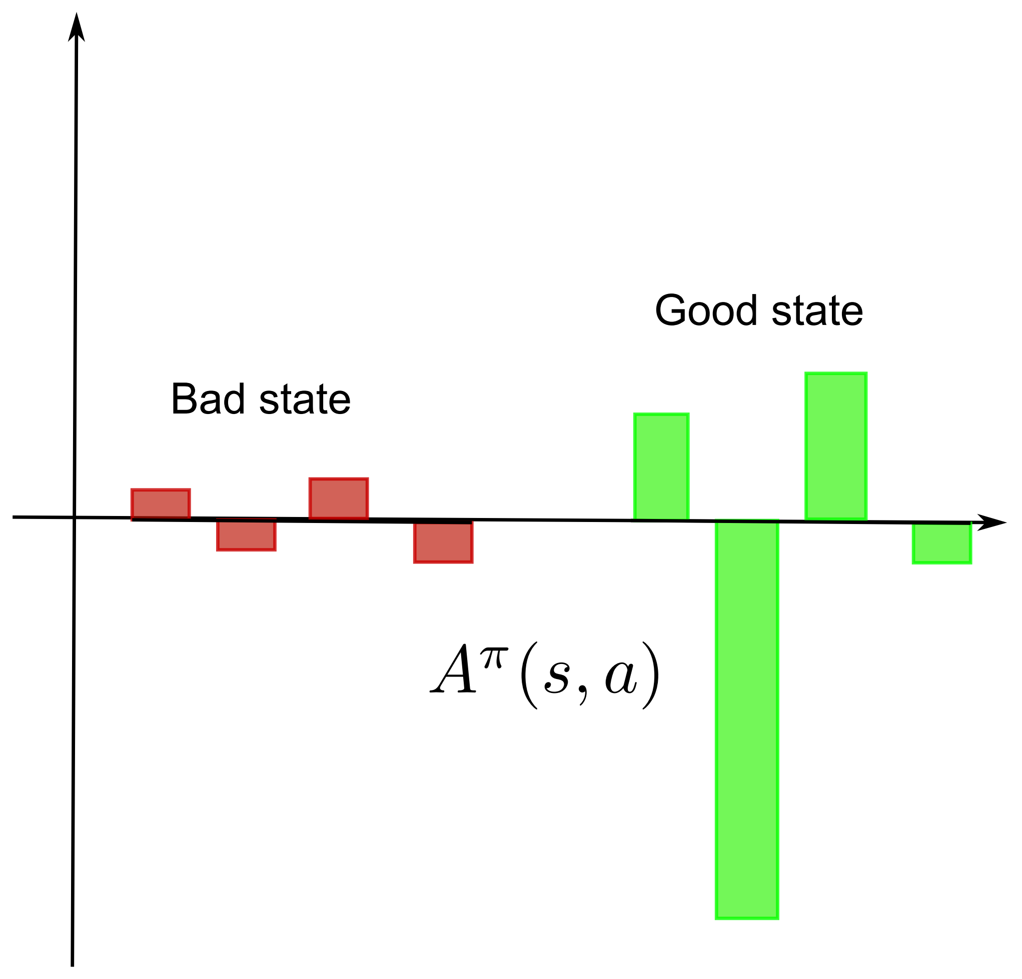
REINFORCE with baseline

- In practice, a baseline that works well is the value of the encountered states:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[\sum_{t=0}^T \nabla_\theta \log \pi_\theta(s_t, a_t) \, (R(\tau) - V^\pi(s_t))]$$

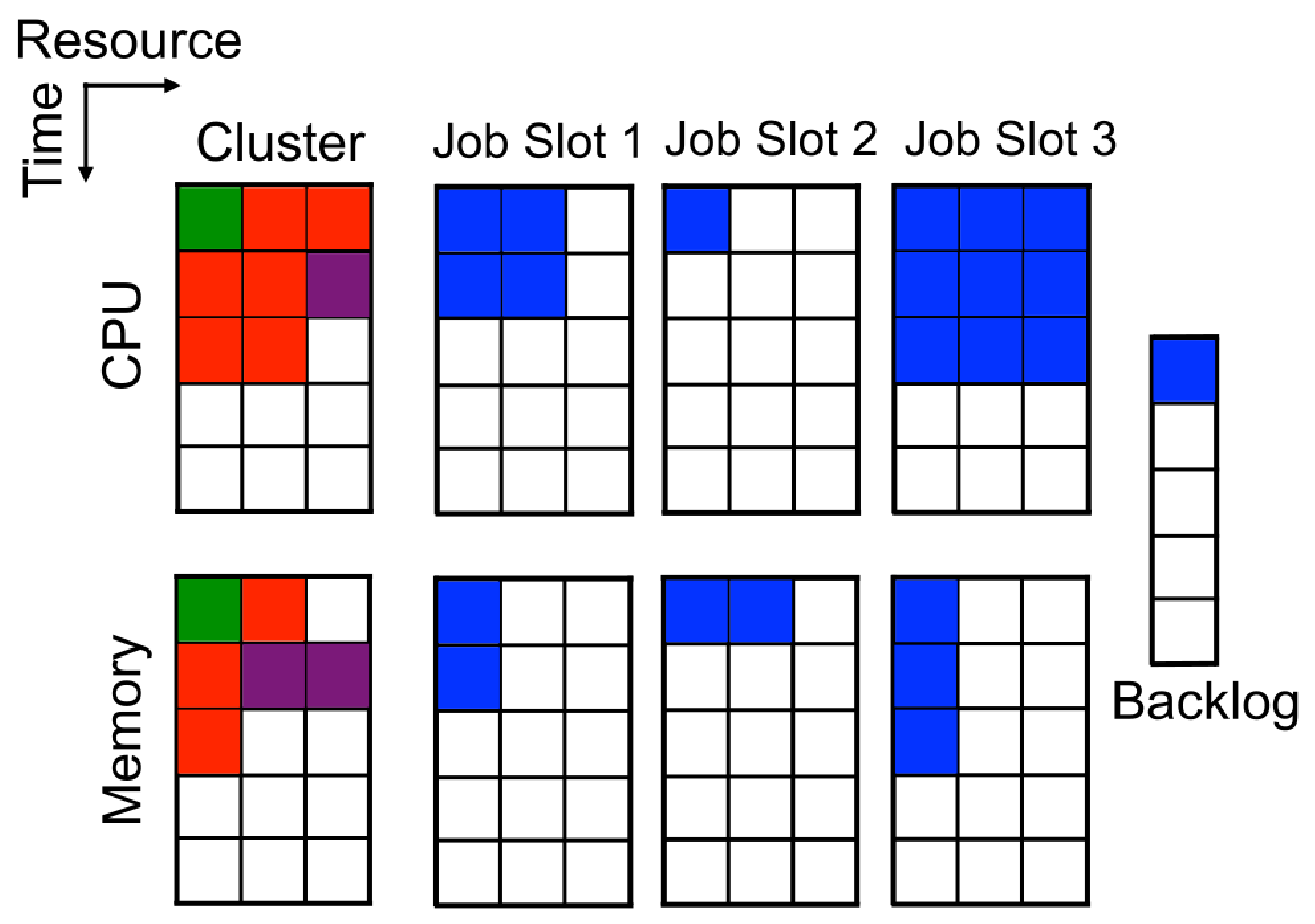
- $R(\tau) - V^\pi(s_t)$ becomes the **advantage** of the action a_t in s_t: how much return does it provide compared to what can be expected in s_t generally:





- As in **dueling networks**, it reduces the variance of the returns.
- Problem: the value of each state has to be learned separately (see actor-critic architectures).

Application of REINFORCE to resource management



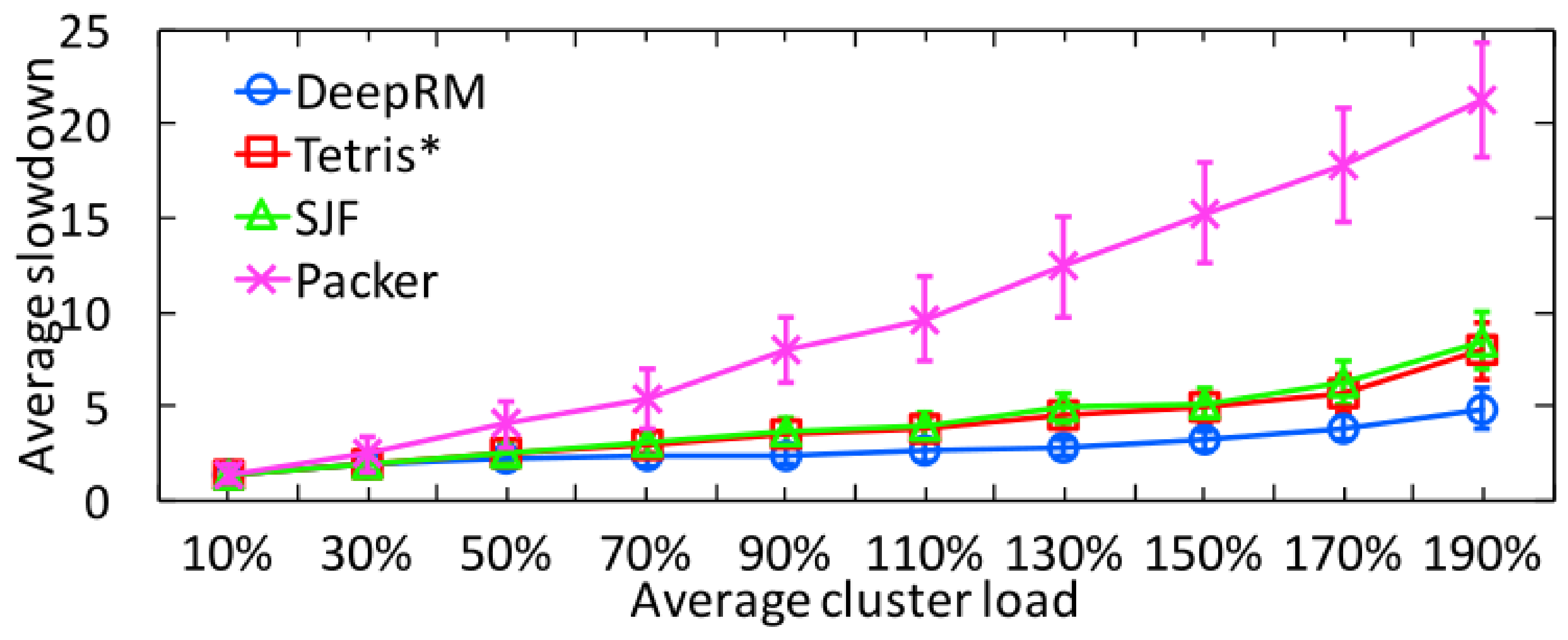
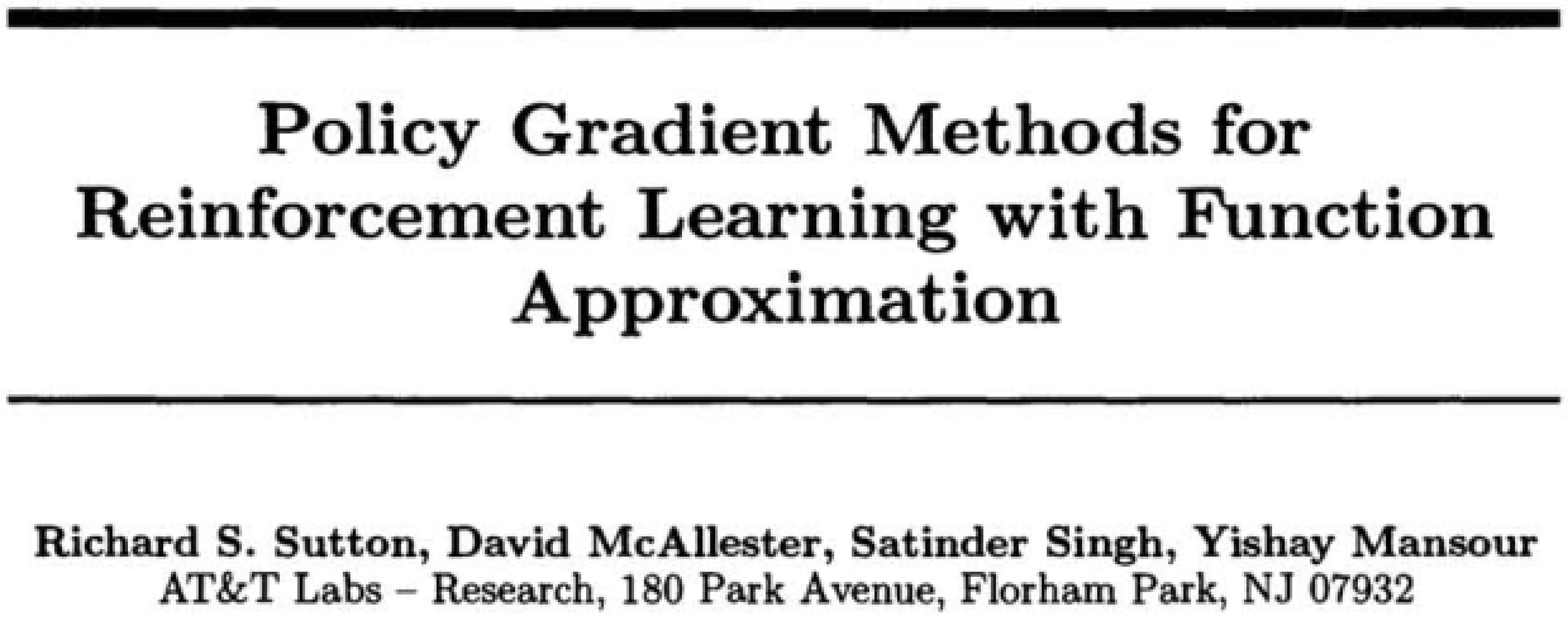


Figure 4: Job slowdown at different levels of load.

- REINFORCE with baseline can be used to allocate resources (CPU cores, memory, etc) when scheduling jobs on a cloud of compute servers.
- The policy is approximated by a shallow NN (one hidden layer with 20 neurons).
- The state space is the current occupancy of the cluster as well as the job waiting list.
- The action space is sending a job to a particular resource.
- The reward is the negative **job slowdown**: how much longer the job needs to complete compared to the optimal case.
- DeepRM outperforms all alternative job schedulers.

Mao et al. (2016) Resource Management with Deep Reinforcement Learning. HotNets '16 doi:10.1145/3005745.3005750.

3 - Policy Gradient Theorem



Policy Gradient Methods for Reinforcement Learning with Function Approximation

Richard S. Sutton, David McAllester, Satinder Singh, Yishay Mansour
AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932

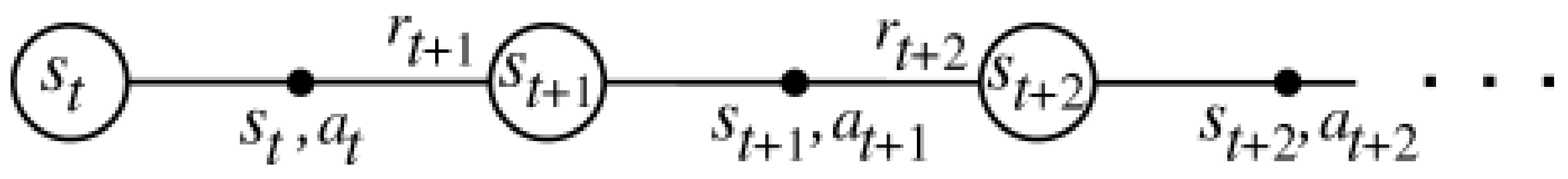
Policy Gradient

- The REINFORCE gradient estimate is the following:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[\sum_{t=0}^T \nabla_\theta \log \pi_\theta(s_t, a_t) \, R(\tau)] = \mathbb{E}_{\tau \sim \rho_\theta}[\sum_{t=0}^T (\nabla_\theta \log \pi_\theta(s_t, a_t)) \, (\sum_{t'=0}^T \gamma^{t'} \, r_{t'+1})]$$

- For each state-action pair (s_t, a_t) encountered during the episode, the gradient of the log-policy is multiplied by the complete return of the episode:

$$R(\tau) = \sum_{t'=0}^T \gamma^{t'} \, r_{t'+1}$$



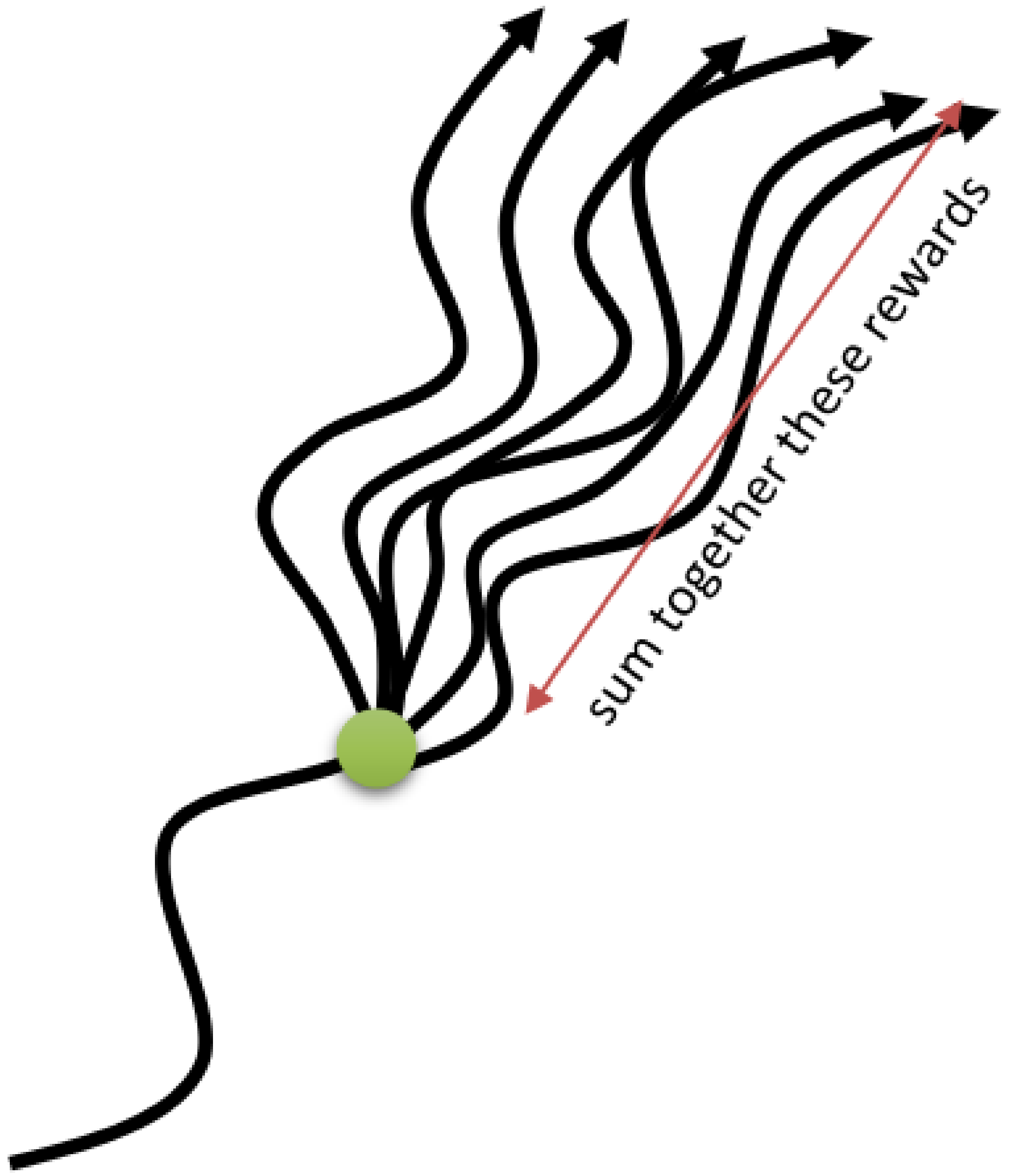
- The **causality principle** states that rewards obtained before time t are not caused by that action.
- The policy gradient can be rewritten as:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[\sum_{t=0}^T \nabla_\theta \log \pi_\theta(s_t, a_t) \, (\sum_{t'=t}^T \gamma^{t'-t} \, r_{t'+1})] = \mathbb{E}_{\tau \sim \rho_\theta}[\sum_{t=0}^T \nabla_\theta \log \pi_\theta(s_t, a_t) \, R_t]$$

Policy Gradient

- The return at time t (**reward-to-go**) multiplies the gradient of the log-likelihood of the policy (the **score**) for each transition in the episode:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[\sum_{t=0}^T \nabla_\theta \log \pi_\theta(s_t, a_t) \, R_t]$$



- As we have:

$$Q^\pi(s, a) = \mathbb{E}_\pi[R_t | s_t = s; a_t = a]$$

we can replace R_t with $Q^{\pi_\theta}(s_t, a_t)$ without introducing any bias:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[\sum_{t=0}^T \nabla_\theta \log \pi_\theta(s_t, a_t) \, Q^{\pi_\theta}(s_t, a_t)]$$

- This is true on average (no bias if the Q-value estimates are correct) and has a much lower variance!

Policy Gradient

- The policy gradient is defined over complete trajectories:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[\sum_{t=0}^T \nabla_\theta \log \pi_\theta(s_t, a_t) \, Q^{\pi_\theta}(s_t, a_t)]$$

- However, $\nabla_\theta \log \pi_\theta(s_t, a_t) \, Q^{\pi_\theta}(s_t, a_t)$ now only depends on (s_t, a_t), not the future nor the past.
- Each step of the episode is now independent from each other (if we have the Markov property).
- We can then **sample single transitions** instead of complete episodes:

$$\nabla_\theta \mathcal{J}(\theta) \propto \mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \, Q^{\pi_\theta}(s, a)]$$

- Note that this is not directly the gradient of $\mathcal{J}(\theta)$, as the value of $\mathcal{J}(\theta)$ changes (computed over single transitions instead of complete episodes, so it is smaller), but the gradients both go in the same direction!

Policy Gradient Theorem

> For any MDP, the policy gradient is:

$$g = \nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{s\sim\rho_\theta, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, Q^{\pi_\theta}(s,a)\right]$$

Policy Gradient Theorem with function approximation

- Better yet, (Sutton et al. 1999) showed that we can replace the true Q-value $Q^{\pi_\theta}(s, a)$ by an estimate $Q_\varphi(s, a)$ as long as this one is unbiased:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \, Q_\varphi(s, a)]$$

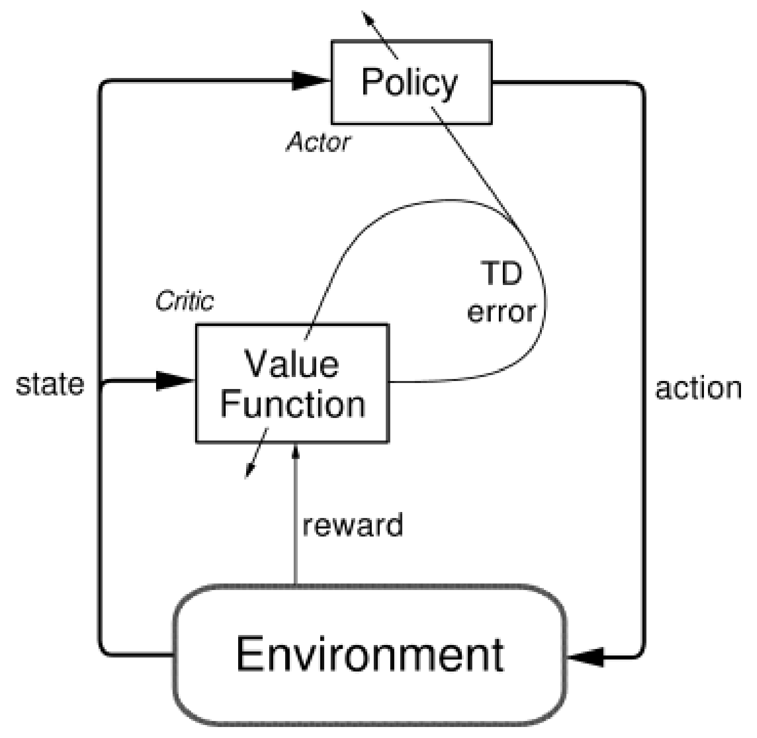
- We only need to have:

$$Q_\varphi(s, a) \approx Q^{\pi_\theta}(s, a) \; \forall s, a$$

- The approximated Q-values can for example minimize the **mean square error** with the true Q-values:

$$\mathcal{L}(\varphi) = \mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta}[(Q^{\pi_\theta}(s, a) - Q_\varphi(s, a))^2]$$

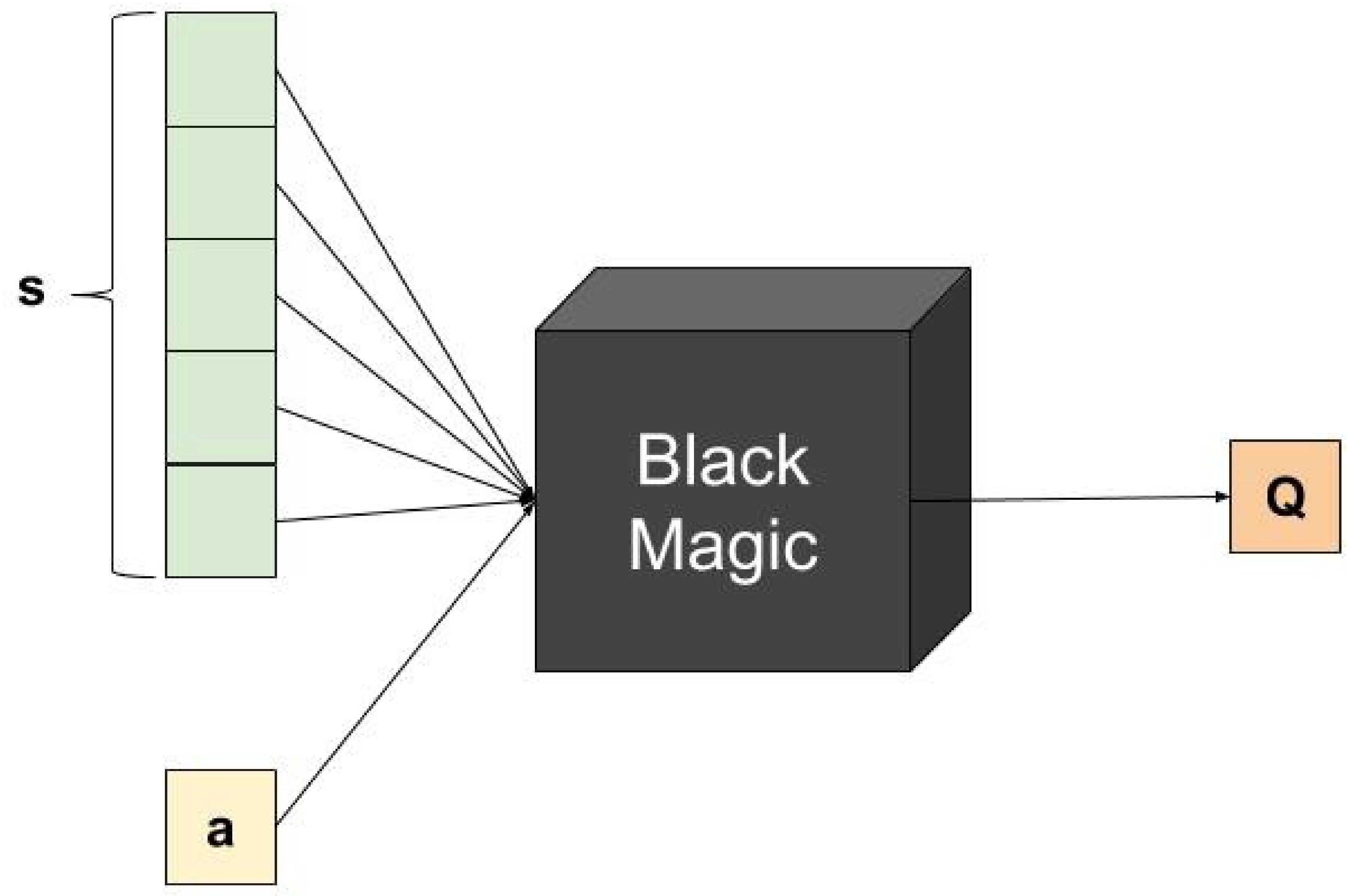
- We obtain an **actor-critic** architecture:
  - the **actor** $\pi_\theta(s, a)$ implements the policy and selects an action a in a state s.
  - the **critic** $Q_\varphi(s, a)$ estimates the value of that action and drives learning in the actor.



Function approximators to learn the Q-values

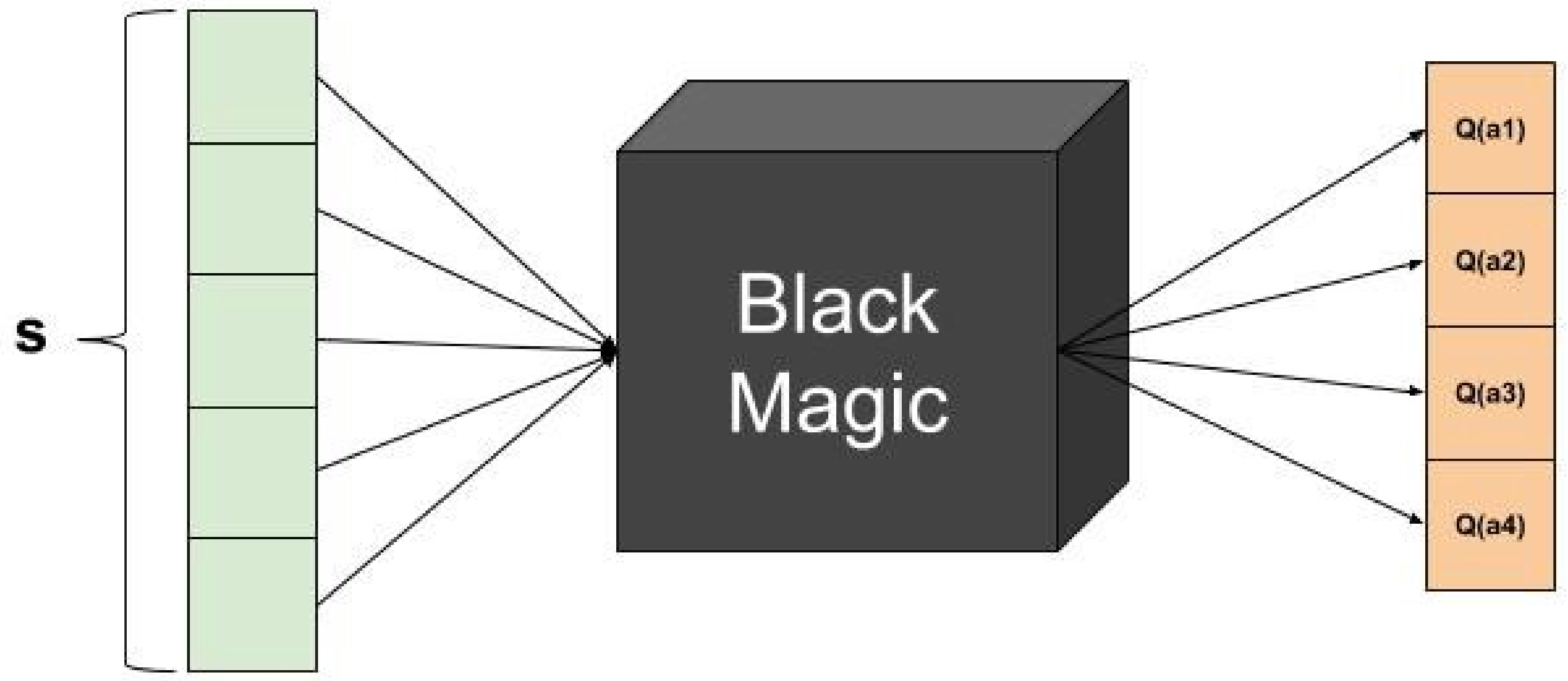
There are two possibilities to approximate Q-values $Q_\theta(s, a)$:

- The DNN approximates the Q-value of a single (s, a) pair.



- The action space can be continuous.

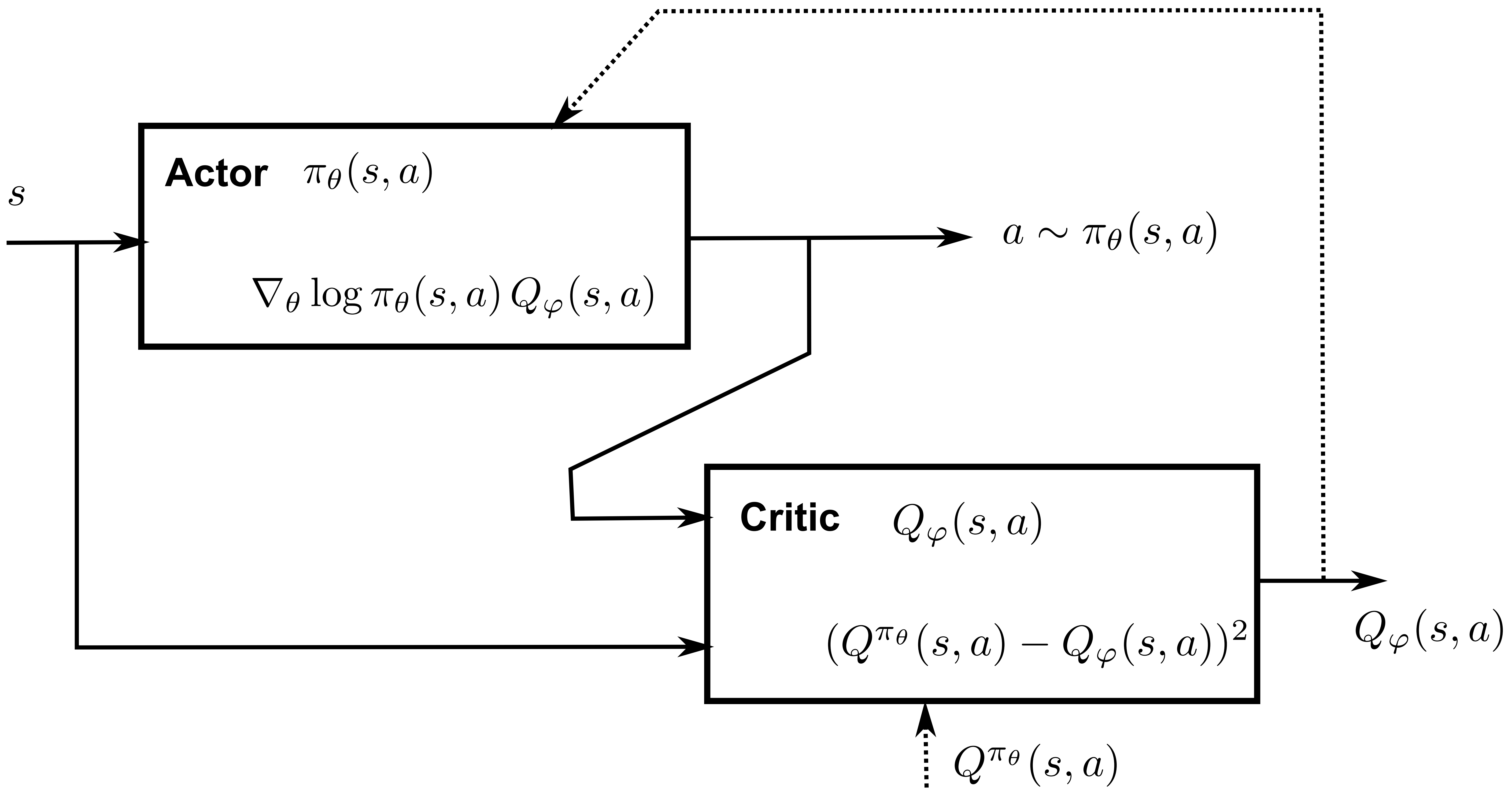
- The DNN approximates the Q-value of all actions a in a state s.



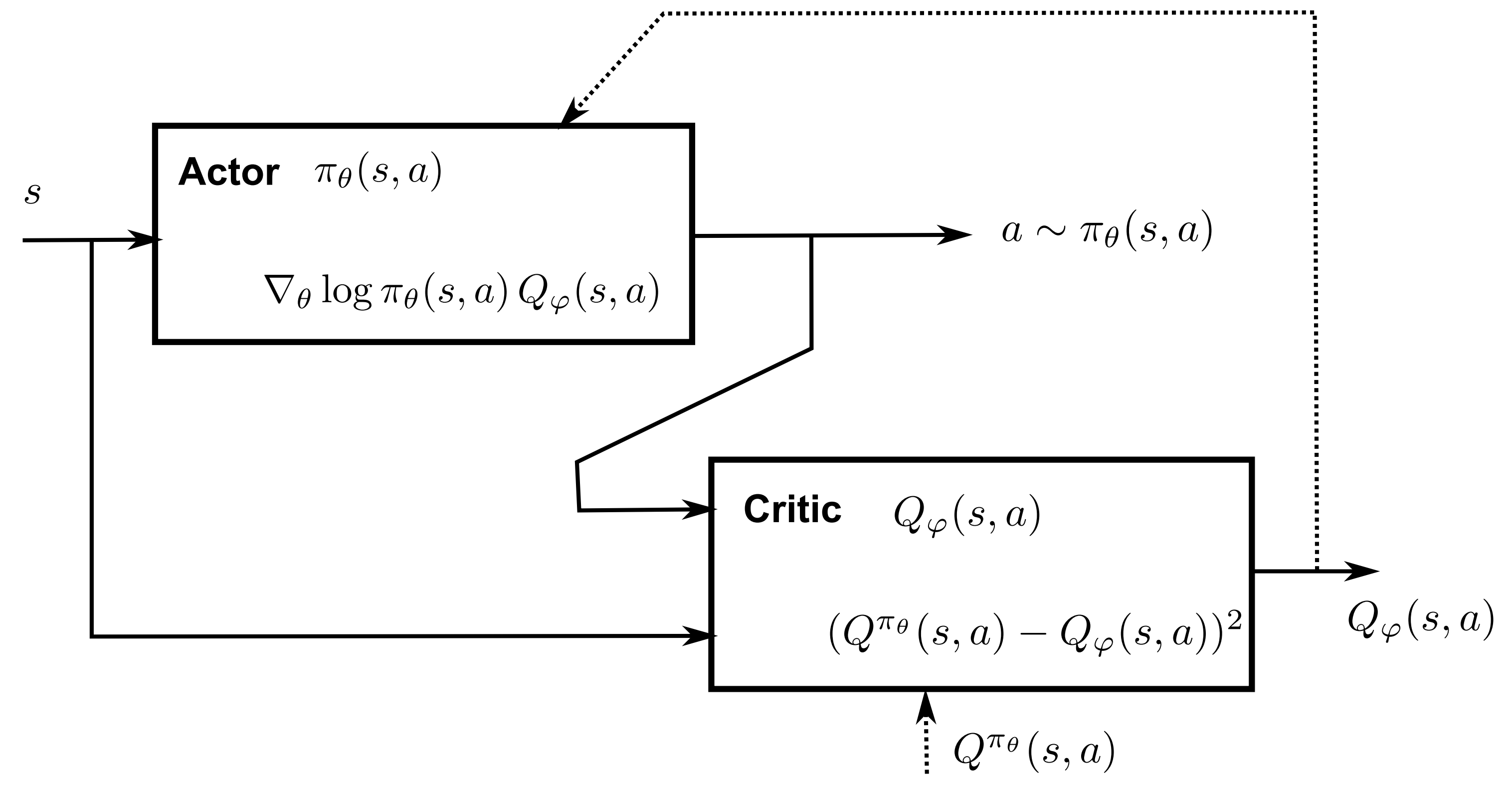
- The action space must be discrete (one neuron per action).

Source: https://towardsdatascience.com/qrash-course-deep-q-networks-from-the-ground-up-1bbda41d3677

Policy Gradient : Actor-critic


s
Actor $\pi_\theta(s, a)$
$\nabla_\theta \log \pi_\theta(s, a)\, Q_\varphi(s, a)$
$a \sim \pi_\theta(s, a)$
Critic $Q_\varphi(s, a)$
$(Q^{\pi_\theta}(s, a) - Q_\varphi(s, a))^2$
$Q_\varphi(s, a)$
$Q^{\pi_\theta}(s, a)$

Policy Gradient : Actor-critic


s
Actor $\pi_\theta(s, a)$
$\nabla_\theta \log \pi_\theta(s, a) \, Q_\varphi(s, a)$
$a \sim \pi_\theta(s, a)$
Critic $Q_\varphi(s, a)$
$(Q^{\pi_\theta}(s, a) - Q_\varphi(s, a))^2$
$Q_\varphi(s, a)$
$Q^{\pi_\theta}(s, a)$

- But how to train the critic? We do not know $Q^{\pi_\theta}(s, a)$. As always, we can estimate it through **sampling**:
    - **Monte-Carlo** critic: sampling the complete episode.

$$\mathcal{L}(\varphi) = \mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta} [(R(s, a) - Q_\varphi(s, a))^2]$$

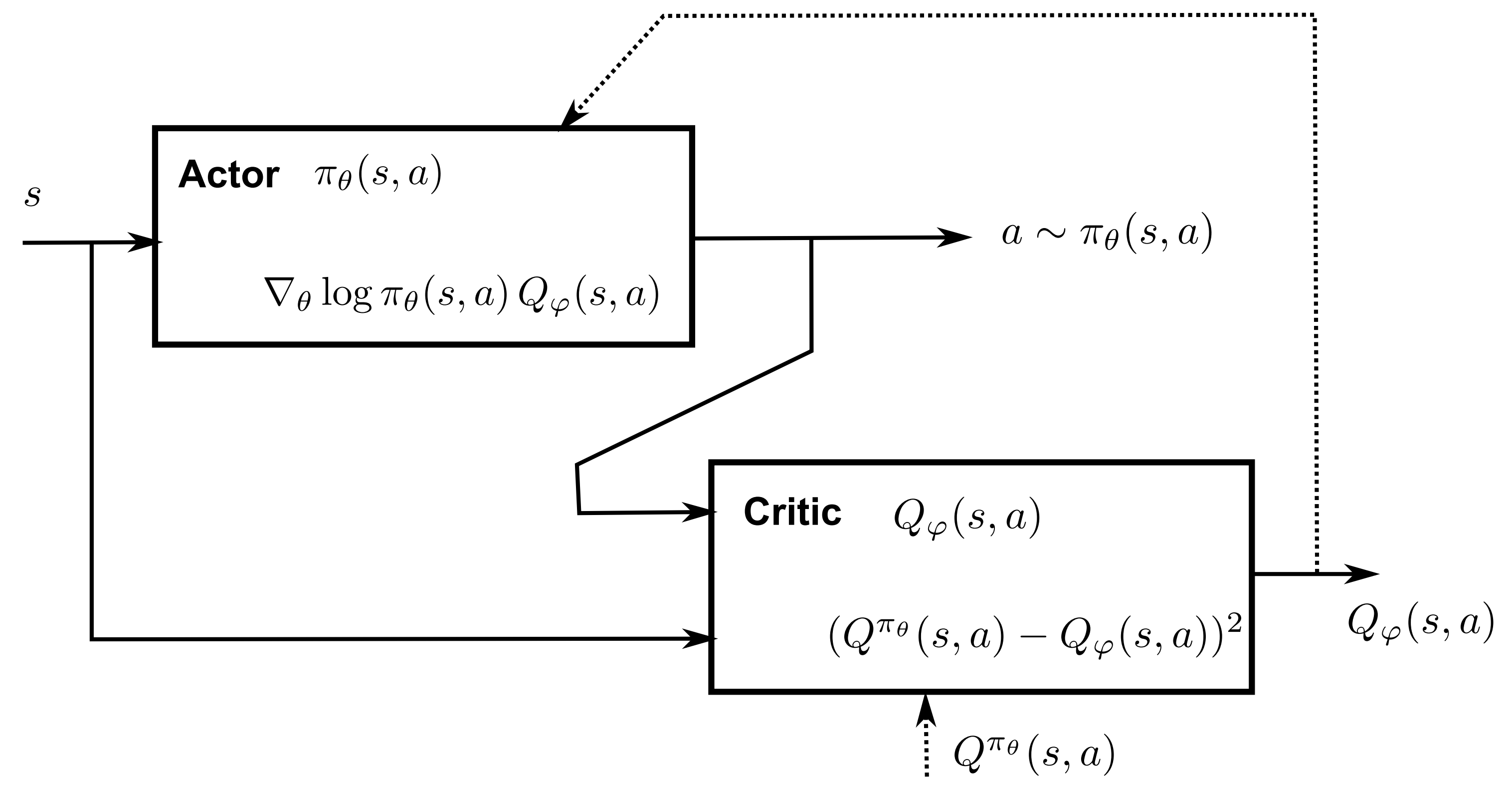
    - **SARSA** critic: sampling (s, a, r, s', a') transitions.

$$\mathcal{L}(\varphi) = \mathbb{E}_{s, s' \sim \rho_\theta, a, a' \sim \pi_\theta} [(r + \gamma \, Q_\varphi(s', a') - Q_\varphi(s, a))^2]$$

    - **Q-learning** critic: sampling (s, a, r, s') transitions.

$$\mathcal{L}(\varphi) = \mathbb{E}_{s, s' \sim \rho_\theta, a \sim \pi_\theta} [(r + \gamma \, \max_{a'} Q_\varphi(s', a') - Q_\varphi(s, a))^2]$$

Policy Gradient : Actor-critic



- The policy gradient (PG) theorem implies an **actor-critic** architecture.
- The **actor** learns using the PG theorem:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, Q_\varphi(s,a)]$$

- The **critic** learns using Q-learning:

$$\mathcal{L}(\varphi) = \mathbb{E}_{s, s' \sim \rho_\theta, a \sim \pi_\theta}[(r + \gamma \, \max_{a'} Q_\varphi(s', a') - Q_\varphi(s,a))^2]$$

Policy Gradient : reducing the variance

- As with REINFORCE, the PG actor suffers from the **high variance** of the Q-values.
- It is possible to use a **baseline** in the PG without introducing a bias:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \, (Q^{\pi_\theta}(s, a) - b)]$$

- In particular, the **advantage actor-critic** uses the value of a state as the baseline:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \, (Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s))]$$

$$= \mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \, A^{\pi_\theta}(s, a)]$$

- The critic can either:
  - learn to approximate both $Q^{\pi_\theta}(s, a)$ and $V^{\pi_\theta}(s)$ with two different NN (SAC).
  - replace one of them with a sampling estimate (A3C, DDPG)
  - learn the advantage $A^{\pi_\theta}(s, a)$ directly (GAE, PPO)

Many variants of the Policy Gradient

- **Policy Gradient methods** can take many forms :

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta (s_t, a_t) \, \psi_t]$$

where:

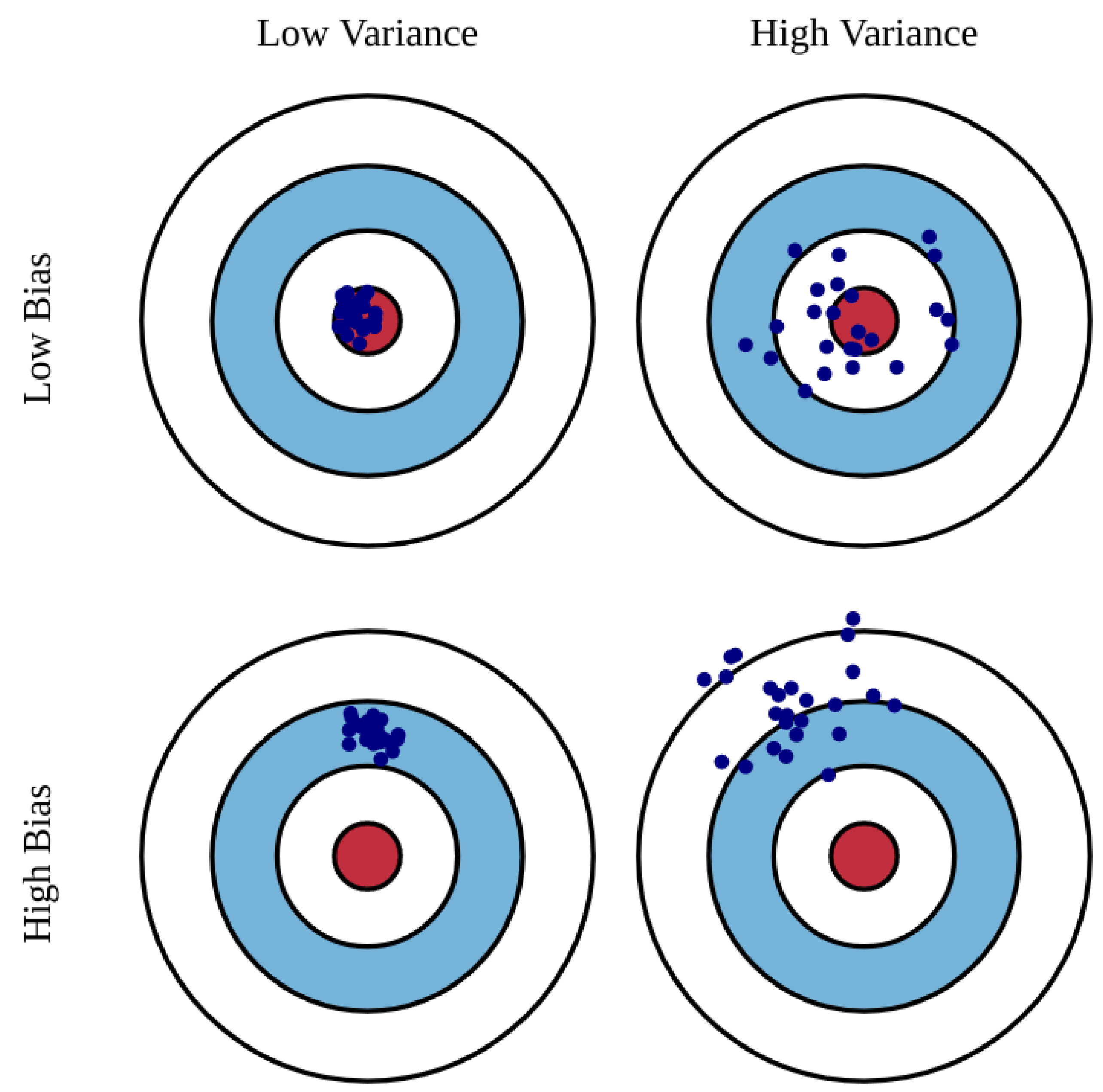
- $\psi_t = R_t$ is the *REINFORCE* algorithm (MC sampling).
- $\psi_t = R_t - b$ is the *REINFORCE with baseline* algorithm.
- $\psi_t = Q^\pi(s_t, a_t)$ is the *policy gradient theorem*.
- $\psi_t = A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi (s_t)$ is the *advantage actor-critic*.
- $\psi_t = r_{t+1} + \gamma \, V^\pi(s_{t+1}) - V^\pi (s_t)$ is the *TD actor-critic*.
- $\psi_t = \displaystyle\sum_{k=0}^{n-1} \gamma^{k} \, r_{t+k+1} + \gamma^n \, V^\pi(s_{t+n}) - V^\pi (s_t)$ is the *n-step advantage*.

and many others…

Bias and variance of Policy Gradient methods

- The different variants of PG deal with the bias/variance trade-off.

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t)\, \psi_t]$$



1. the more ψ_t relies on **sampled rewards** (e.g. R_t), the more the gradient will be correct on average (small bias), but the more it will vary (high variance).
   - This increases the sample complexity: we need to average more samples to correctly estimate the gradient.
2. the more ψ_t relies on **estimations** (e.g. the TD error), the more stable the gradient (small variance), but the more incorrect it is (high bias).
   - This can lead to suboptimal policies, i.e. local optima of the objective function.

- All the methods we will see in the rest of the course are attempts at finding the best trade-off.

4 - Generalized advantage estimation

Generalized advantage estimation (GAE)

- The **n-step advantage** at time t:

$$A_t^n = \sum_{k=0}^{n-1} \gamma^k \, r_{t+k+1} + \gamma^n \, V(s_{t+n}) - V(s_t)$$

can be written as function of the TD error of the next n transitions:

$$A_t^n = \sum_{l=0}^{n-1} \gamma^l \, \delta_{t+l}$$

Proof with $n = 2$:

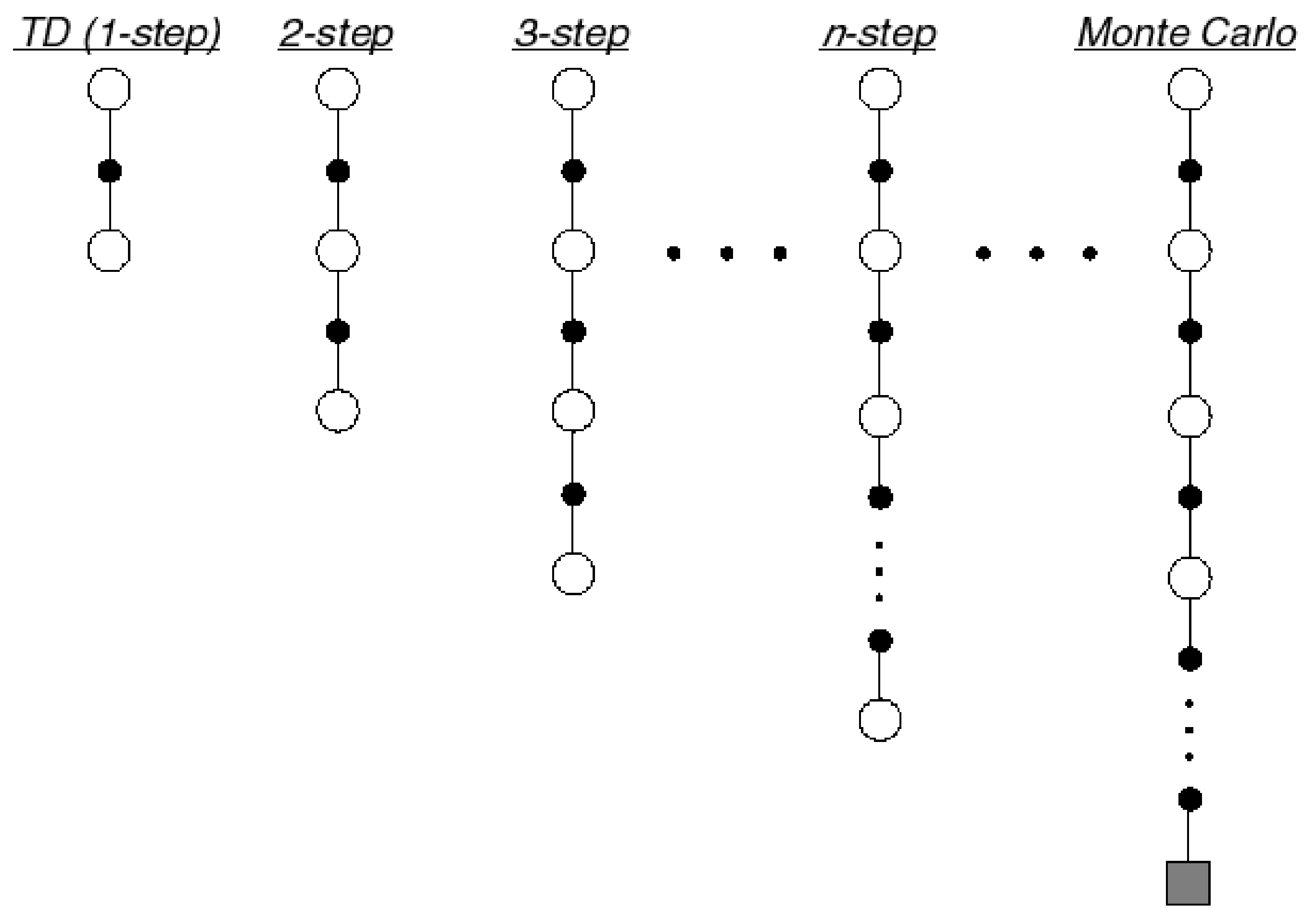
$$A_t^2 = r_{t+1} + \gamma \, r_{t+2} + \gamma^2 \, V(s_{t+2}) - V(s_t)$$

$$= (r_{t+1} - V(s_t)) + \gamma \, (r_{t+2} + \gamma \, V(s_{t+2}))$$

$$= (r_{t+1} + \gamma \, V(s_{t+1}) - V(s_t)) + \gamma \, (r_{t+2} + \gamma \, V(s_{t+2}) - V(s_{t+1}))$$

$$= \delta_t + \gamma \, \delta_{t+1}$$

Generalized advantage estimation (GAE)



- The **n-step advantage** realizes a bias/variance trade-off, but which value of n should we choose?

$$A_t^n = \sum_{k=0}^{n-1} \gamma^k \, r_{t+k+1} + \gamma^n \, V(s_{t+n}) - V(s_t)$$

- Schulman et al. (2015) proposed a **generalized advantage estimate** (GAE) $A_t^{\text{GAE}(\gamma, \lambda)}$ summing all possible n-step advantages with a discount parameter λ:

$$A_t^{\text{GAE}(\gamma, \lambda)} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^n \, A_t^n$$

- This is just a forward eligibility trace over distant n-step advantages: the 1-step advantage is more important the the 1000-step advantage (too much variance).
- We can show that the GAE can be expressed as a function of the future 1-step TD errors:

$$A_t^{\text{GAE}(\gamma, \lambda)} = \sum_{k=0}^{\infty} (\gamma \, \lambda)^k \, \delta_{t+k}$$

Generalized advantage estimation (GAE)

- **Generalized advantage estimate** (GAE) :

$$A_t^{\text{GAE}(\gamma, \lambda)} = (1 - \lambda) \sum_{n=1}^\infty \lambda^n \, A_t^n = \sum_{k=0}^\infty (\gamma \, \lambda)^k \, \delta_{t+k}$$

- The parameter λ controls the **bias-variance** trade-off.
- When $\lambda = 0$, the generalized advantage is the TD error:

$$A_t^{\text{GAE}(\gamma, 0)} = r_{t+1} + \gamma \, V(s_{t+1}) - V(s_t) = \delta_t$$

- When $\lambda = 1$, the generalized advantage is the MC advantage:

$$A_t^{\text{GAE}(\gamma, 1)} = \sum_{k=0}^\infty \gamma^k \, r_{t+k+1} - V(s_t) = R_t - V(s_t)$$

- Any value in between controls the bias-variance trade-off: from the high bias / low variance of TD to the small bias / high variance of MC.
- In practice, it leads to a better estimation than n-step advantages, but is more computationally expensive.

References

Mao, H., Alizadeh, M., Menache, I., and Kandula, S. (2016). Resource Management with Deep Reinforcement Learning. in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks - HotNets '16* (Atlanta, GA, USA: ACM Press), 50–56. doi:10.1145/3005745.3005750.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015). High-Dimensional Continuous Control Using Generalized Advantage Estimation. http://arxiv.org/abs/1506.02438.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. in *Proceedings of the 12th International Conference on Neural Information Processing Systems* (MIT Press), 1057–1063. https://dl.acm.org/citation.cfm?id=3009806.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8, 229–256.