Reinforcement Learning: Policy-based Methods

Benjamin Rosman

School of Computer Science and Applied Mathematics
University of the Witwatersrand
South Africa

African Masters of Machine Intelligence (Kigali, Rwanda)
January 13-17 2020

Further reading…

- Reinforcement Learning: An Introduction (Sutton and Barto)
  - Chapter 13
- CS 294-122 (Sergey Levine)
  - Lectures 4 and 5
- COMPM050/COMPGI13 (David Silver)
  - Lecture 7

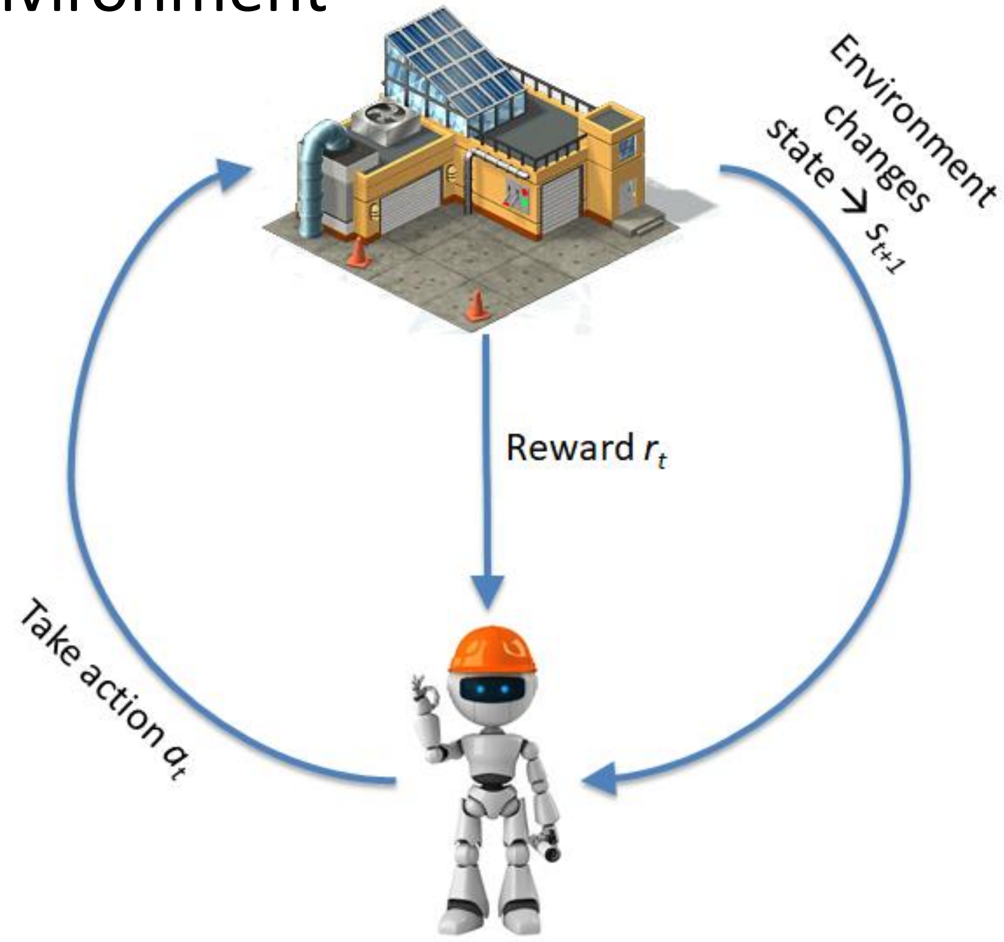
A quick recap

Decision maker (agent) exists within an environment

Agent takes **actions** based on the environment **state**

Environment **state** updates

Agent receives feedback as **rewards**

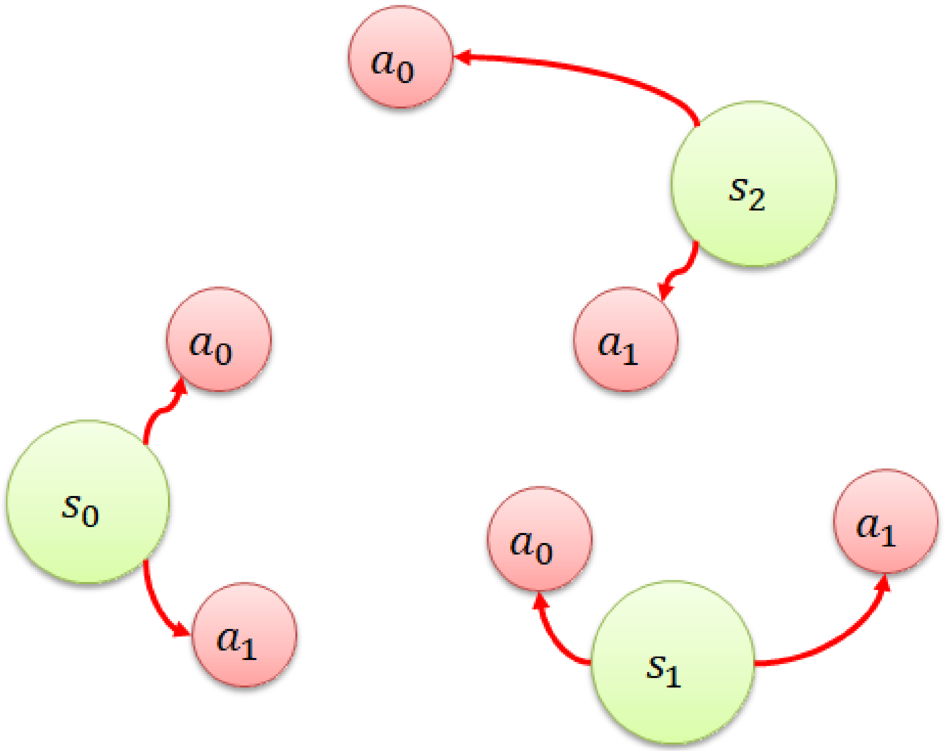


A model for decision making

Markov Decision Process (MDP)

M = $\langle \boldsymbol{S}, \boldsymbol{A}, \boldsymbol{T}, \boldsymbol{R}, \gamma \rangle$

- States: encode world configurations
- Actions: choices made by agent



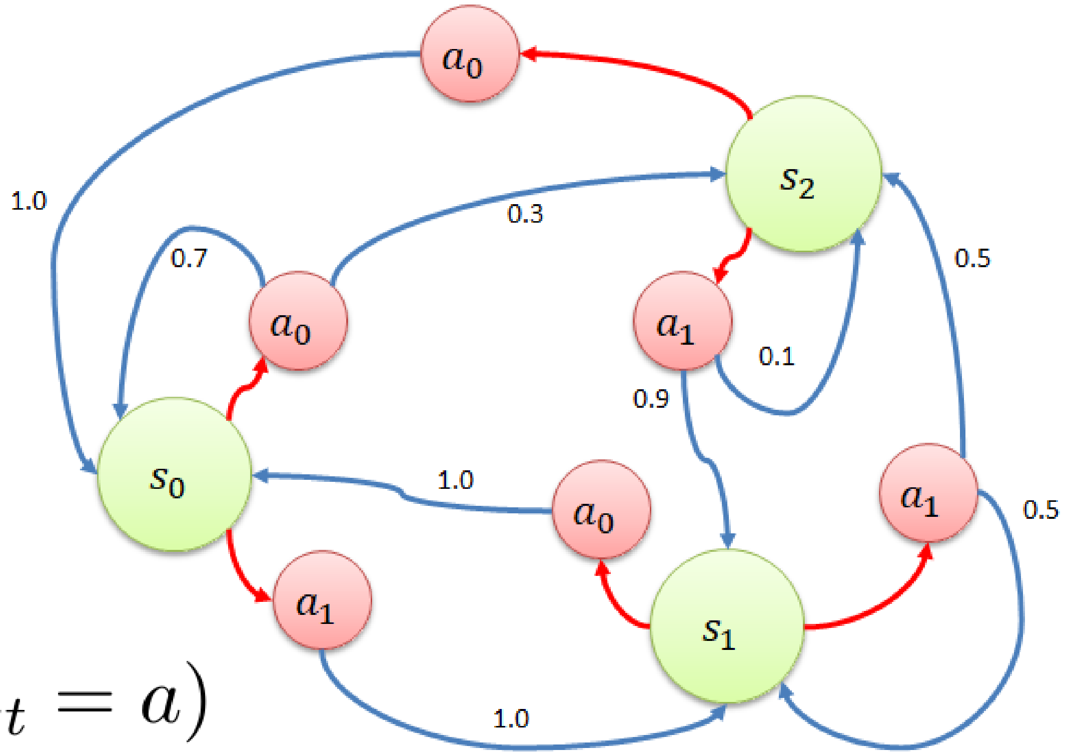
A model for decision making

Markov Decision Process (MDP)

M = ⟨***S***, ***A***, ***T***, ***R***, γ⟩

- Transition function: how the world evolves under actions

  Stochastic!



$$T(s, a, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$$

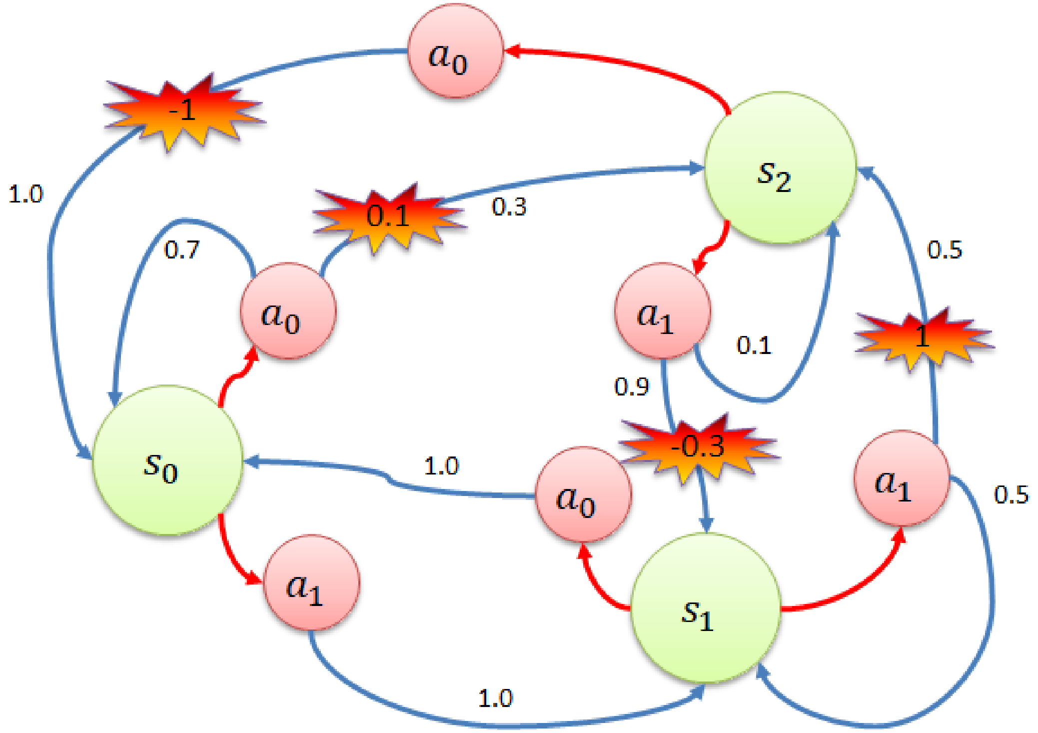
A model for decision making

Markov Decision Process (MDP)

M = ⟨***S***, ***A***, ***T***, ***R***, γ⟩

- Rewards: feedback signal to agent

$$R(s, a) = E[r_t | s_t = s, a_t = a]$$



A model for decision making

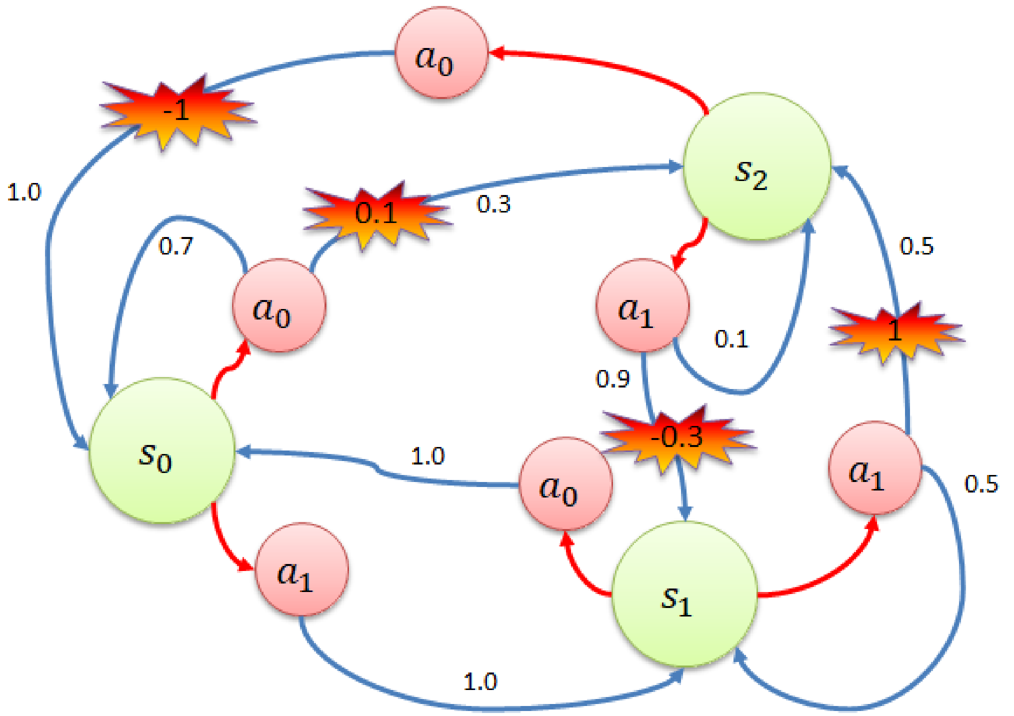
Markov Decision Process (MDP)

M = $\langle \boldsymbol{S}, \boldsymbol{A}, \boldsymbol{T}, \boldsymbol{R}, \gamma \rangle$

- $\gamma \in [0,1]$ discounting for future rewards

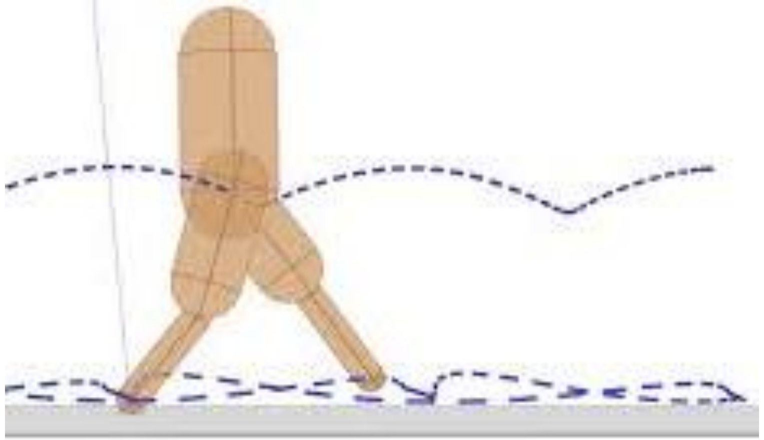
Particularly important for **continuing** tasks

The value of something now is usually greater than in the future.

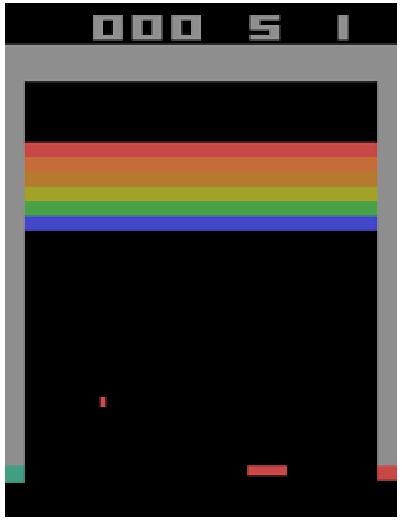


Example

- I want to get this robot to walk as far as possible:
  - What is S, A, T, R?



- I want to play this game as well as possible:
  - What is S, A, T, R?



Why can't we act greedily?

Cannot just rely on the **instantaneous** reward function
Tradeoff: don't just act myopically (short term)

Notion of **value** to codify the goodness of a state, considering a policy running into the future

- Represented as a **value function**

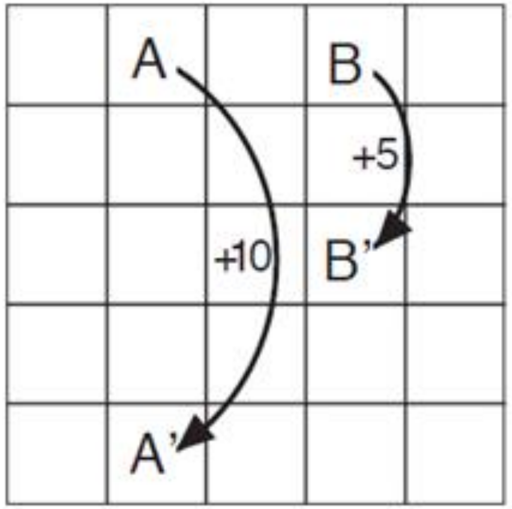
Warning: reward functions

- The reward tells the agent **what** we want it to achieve
  - Learning is about figuring out **how** to achieve it
- Be careful: the agent will literally do exactly what you ask it
  - It doesn’t “know what you mean”
- Where might this be a problem?

Value functions

Allow us to act greedily by considering estimates of the future

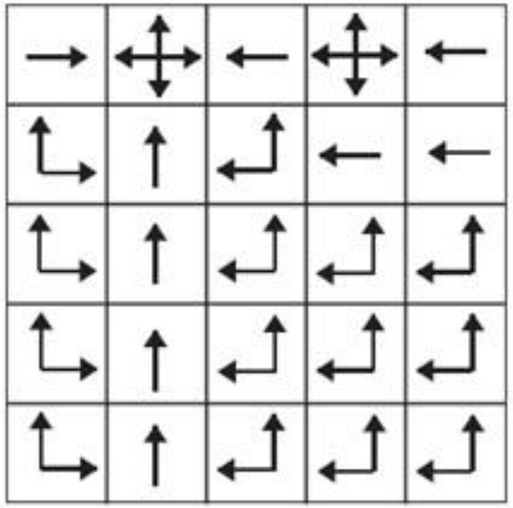
Optimal policy:



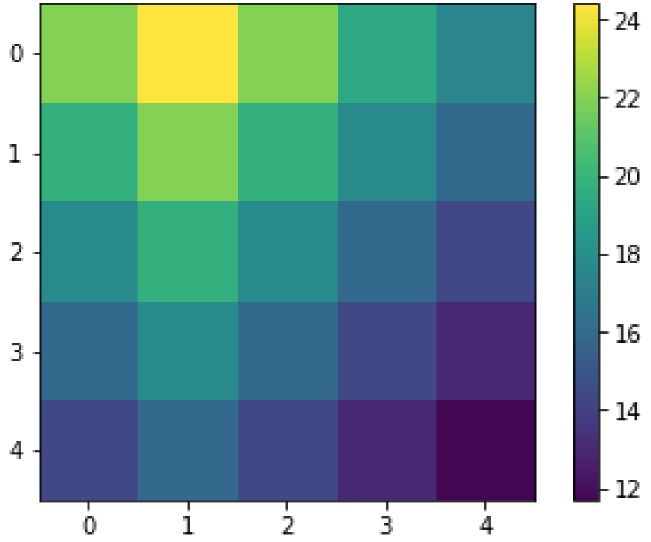
a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) v_*



c) π_*



NB: Equivalence between optimal policy and optimal value function

Value functions: recursion

$V(s) \Rightarrow$ expected return starting at s and following π

Suggests dependence on value of next state s'

Bellman Equation:

$$\boxed{V^{\pi}(s)} = \boxed{R(s, \pi(s))} + \gamma \boxed{\sum_{s'}} \boxed{T(s, \pi(s), s')} \boxed{V^{\pi}(s')}$$

value of s

immediate reward

for all possible next states

the probability of reaching that state with π

value of s’

Solution techniques

- If you know T and R:
  - Iteratively compute solution
  - Dynamic programming
  - Policy iteration and value iteration
- If you don't know T and R: (more common case)
  - Explore the environment and collect samples
  - Use samples to learn T and R
    - Model-based RL (more next time)
  - Use samples to learn V
    - Model-free RL
    - E.g. TD-learning, SARSA, Q-learning, …

Q-Learning

- Initialise $Q(s, a)$ for all s, a
- Repeat (for each episode):
    - Initialise s
    - Repeat (for each step of episode):
        - Choose a from s using policy from Q (ϵ-greedy)
        - Take action a, observe r, s'
        - $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
        - $s \leftarrow s'$
    - Until s is terminal

act

learn

take best next action
(so far)

ϵ-**Greedy** ($0 < \epsilon \leq 1$):

- With probability $1 - \epsilon$ exploit:
    - Choose the best action from Q
- With probability ϵ explore:
    - Randomly choose action

ε usually higher at beginning, decay later

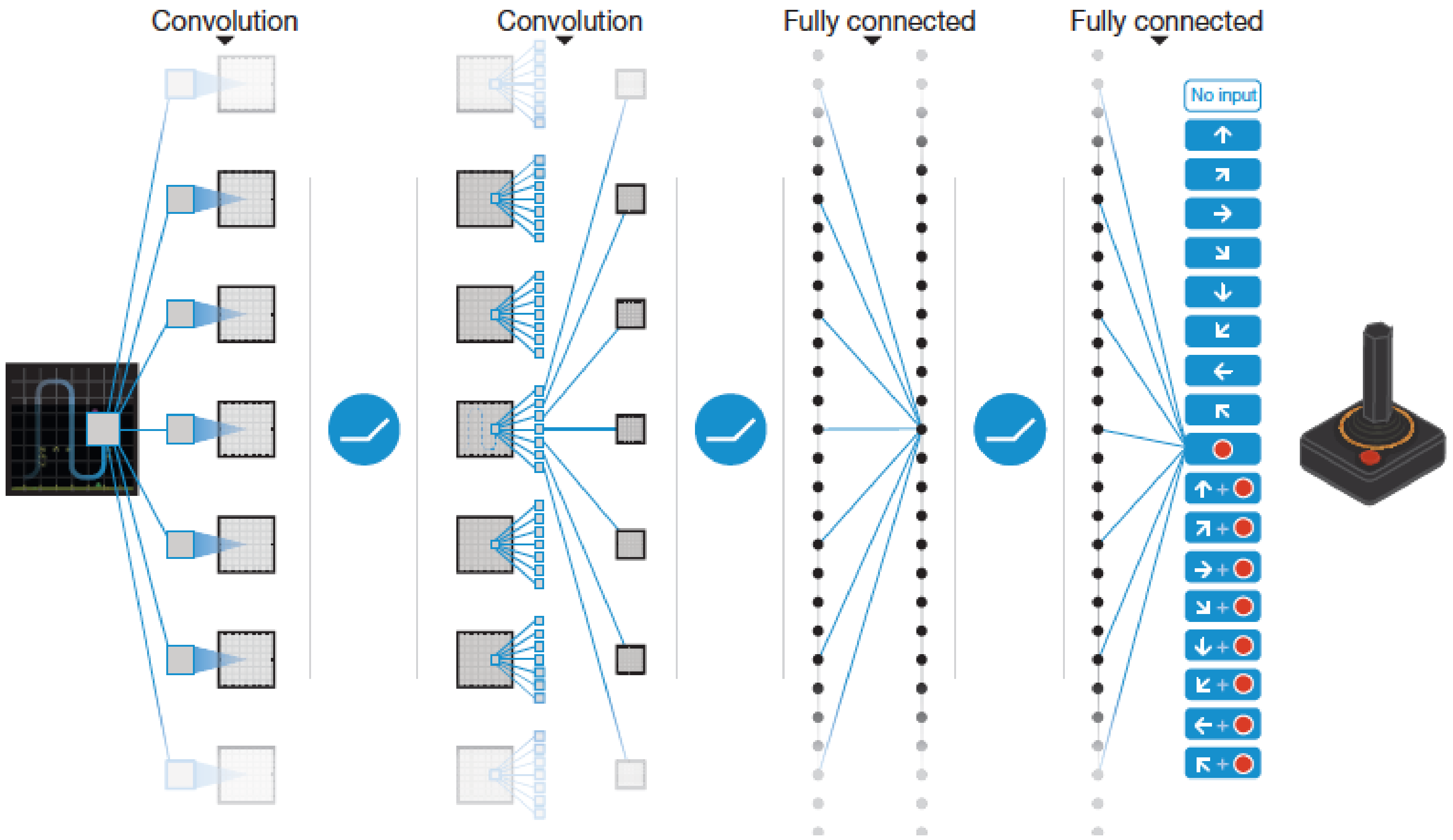
Function approximation

- This works well if you can tabulate the value function

State	a_1	…	a_m
$\boldsymbol{s_1}$	1.7	…	2.8
$\boldsymbol{s_2}$	-3.9	…	-3.1
…	…	…	…
$\boldsymbol{s_n}$	0.2	…	-0.1

- But if the state space is large/continuous
  - Approximate this with a learned function
  - Why?
    - Infinity? May not visit every state (many times)? Need to generalise?

Deep Q-Networks

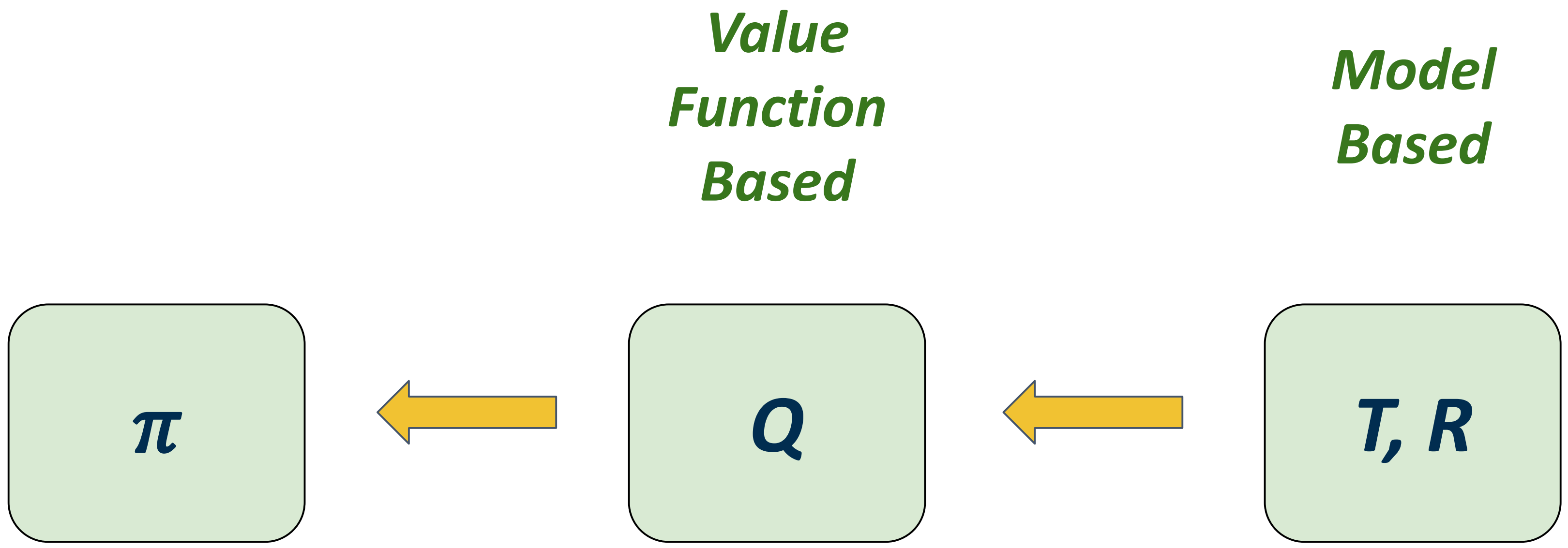


[Mnih et al., 2015]

DQN

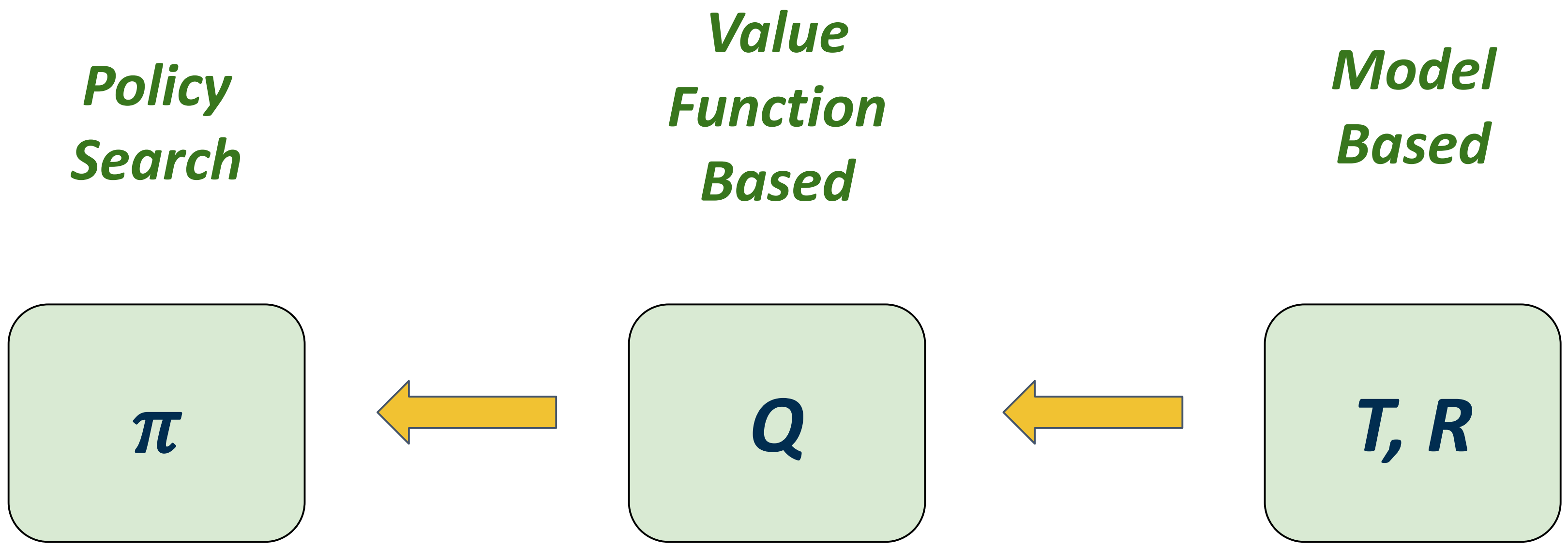
- Take action a_t according to ϵ-greedy policy
  - Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory D
- Sample mini-batch of transitions (s, a, r, s') from D
  - Experience replay: decorrelate samples
- Compute Q-learning targets w.r.t old fixed parameters w^-
  - Fixed Q-targets: avoid oscillations
- Optimise MSE between Q-network and Q-learning targets
  - $L_i(w_i) = \mathbb{E}_{s,a,r,s' \sim D_i}\left[\left(r + \gamma \max_{a'} Q(s', a', w_i^-) - Q(s, a, w_i)\right)^2\right]$
  - Using stochastic gradient descent on $\nabla_{w_i} L_i(w_i)$

RL approaches



Questions?

RL approaches



Policy Search

- Learn policy directly:

$$\pi_\theta(s,a) = \pi(s,a;\theta)$$

- Parameterise policy: learn parameters of policy

- Why?
  - When might it be easier to learn a policy than a value function?
  - Learning a Q-function can be complicated
    - Policy may be much simpler than learning a value for each state-action
  - Injecting information?
  - Stochasticity?

Policy Search

Trajectory τ

$$p(\tau|\theta) = p_\theta(s_1, a_1, s_2, a_2, \ldots, s_T, a_T)$$

$$= p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

- Objective function?
  - Maximise return given θ:
    - $J(\theta) = \mathbb{E}[\sum_t \gamma^t r_t | \pi_\theta]$
    - $\theta^* = argmax_\theta J(\theta)$
    - Always more efficient to follow the gradient!

Note:
The return R, cost J, utility U are often used interchangeably here

Hill Climbing

- What if you can't differentiate π?

- Sample-based optimisation:
  - Sample some θ values near your current best θ
  - Compute return
  - Approximate a gradient
    - Finite differences
  - Adjust your current best θ to give the highest value

- Other approaches, e.g. genetic algorithms

Aibo gait optimization

from Kohl and Stone, ICRA 2004

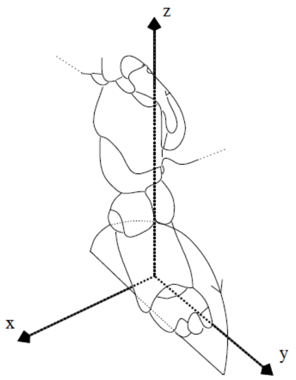


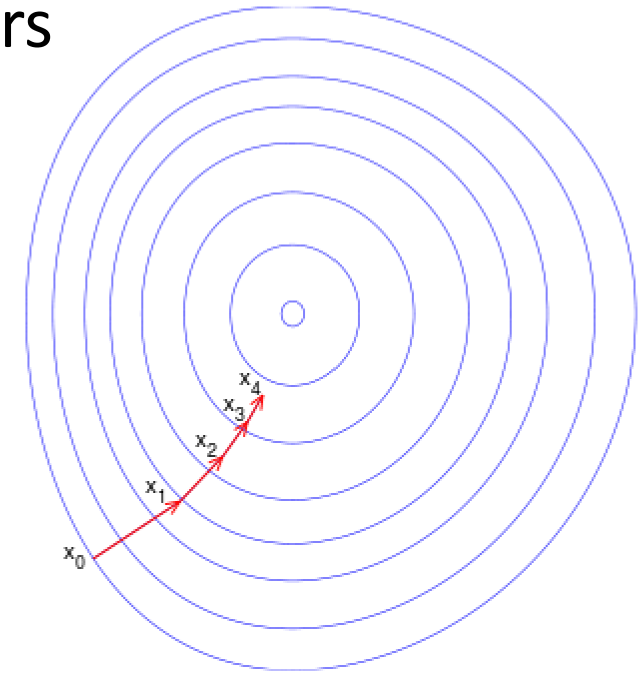
Fig. 2. The elliptical locus of the Aibo's foot. The half-ellipse is defined by length, height, and position in the x-y plane.

All told, the following set of 12 parameters define the Aibo's gait [10]:

- The front locus (3 parameters: height, x-pos., y-pos.)
- The rear locus (3 parameters)
- Locus length
- Locus skew multiplier in the x-y plane (for turning)
- The height of the front of the body
- The height of the rear of the body
- The time each foot takes to move through its locus
- The fraction of time each foot spends on the ground

Using gradients

- If we can differentiate π
  - Compute and ascend $\partial R / \partial \theta$
  - This is the gradient of return w.r.t policy parameters
  - $\theta_{t+1} = \theta_t + \Delta\theta_t$
    $= \theta_t + \alpha \nabla J(\theta_t)$
  - These are called **policy gradient methods**



Policy Gradient Theorem

- Why does this work?

- Relate the **gradient of performance with respect to the policy parameter** to the **gradient of the policy**

- Policy gradient theorem:
  - $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, Q^{\pi_\theta}(s,a)]$

- No explicit dependence on distribution of states (or model)

What is a good form for a parameterised function f?

- Simplest thing you can do?
  - *Linear value function approximation*
  - Use set of basis functions $\phi_1, \dots, \phi_n$
  - f is a linear function of them:

  - $\hat{f} = \boldsymbol{w} \cdot \Phi(s, a) = \sum_{i=1}^{n} w_i \phi_i(s, a)$
  - We'll want to learn parameters $\boldsymbol{w}$

- Neural network:
  - $f = f(s, a; \boldsymbol{w})$

Basis functions $\phi(x)$:
- Could be polynomial in state vars:
  - 1st order: $[1, x, y]$
  - 2nd order: $[1, x, y, x^2, y^2, xy]$
  - This is a Taylor expansion
- Others:
  - Fourier basis
  - Wavelet basis
  - …

REward Increment = Nonnegative Factor times Offset Reinforcement times Characteristic Eligibility

REINFORCE (Monte-Carlo Policy Gradient)

- REINFORCE: one particularly popular sample-based estimate of the gradient
  - Based on the policy gradient theorem, but approximate the $\mathbb{E}$ with sampled trajectories (Monte-Carlo samples)

The return R_t acts as an estimate of $Q^{\pi_\theta}(s, a)$

$$\Delta\theta_t = \alpha R_t \frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)} = \alpha R_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Log derivative trick: $\frac{\nabla_\theta x}{x} = \nabla_\theta \log x$

REINFORCE algorithm

- Initialise θ
- For each episode
  - Choose actions according to π_θ: $a \sim \pi_\theta(a|s)$
  - Gather samples $\{s_1, a_1, r_1, \dots, s_T, a_T, r_T\}$
  - For $t = 1\ to\ T$
    - $\theta \leftarrow \theta + \alpha R_t \nabla_\theta \log \pi_\theta(a_t|s_t)$

(that's it)

Deriving REINFORCE

- Cost: $$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right] \\ = \int_\tau r(\tau) p(\tau;\theta)\mathrm{d}\tau$$ $$\tau = (s_0, a_0, r_0, s_1, \ldots)$$

- Derivative: $$\nabla_\theta J(\theta) = \int_\tau r(\tau)\nabla_\theta p(\tau;\theta)\mathrm{d}\tau$$

Log derivative trick: $\frac{\nabla_\theta x}{x} = \nabla_\theta \log x$

- Transformation: $$\nabla_\theta p(\tau;\theta) = \boxed{p(\tau;\theta)}\frac{\nabla_\theta p(\tau;\theta)}{\boxed{p(\tau;\theta)}} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$$

- Substitute: $$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\mathrm{d}\tau \\ = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

Deriving REINFORCE

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} \left[r(\tau) \nabla_\theta \log p(\tau;\theta) \right]$$

- Recall: $p(\tau;\theta) = \prod_{t \geq 0} p(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t)$
- So: $\log p(\tau;\theta) = \sum_{t \geq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$
- Derivative: $\nabla_\theta \log p(\tau;\theta) = \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Note we lose dependence on dynamics

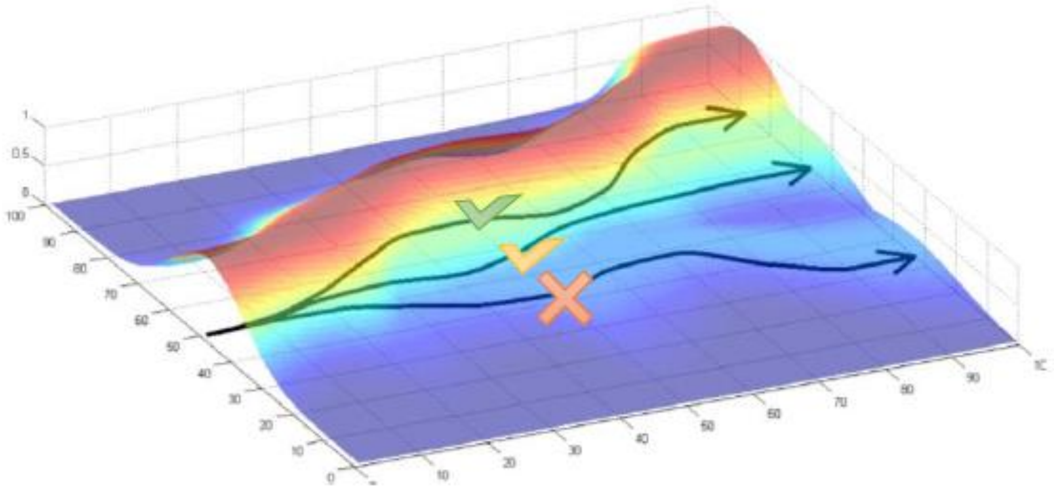
- And so estimate: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$

Interpretation

- Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \boxed{r(\tau)}\boxed{\nabla_\theta \log \pi_\theta(a_t|s_t)}$$

- Think of this as saying:
  - If $r(\tau)$ is high: push up probabilities of seen actions
  - If $r(\tau)$ is low: push down probabilities of seen actions

- Simple version of credit assignment



Variance

- This gradient estimator (MC) turns out to have a high variance

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

  - Why?
    - These are all samples of a run of a policy!
  - Slow convergence
- Correct with a baseline:
- $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} (r(\tau) - b(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t)$
- Because b could be 0, this is a generalisation of REINFORCE
  - Will converge asymptotically to a local minimum

Why can we use a baseline?

- $\nabla_\theta J(\theta) \approx \sum_{t\geq 0}(r(\tau) - b(s_t))\, \nabla_\theta \log \pi_\theta(a_t|s_t)$

- Intuition: can add or subtract b from r without biasing algorithm
  - As long as b not a function of a_t

- Mathematically (with some notational abuse):

$$\mathbb{E}_{\pi_\theta}\left[\sum_t b\nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$
$$= \int \left[\sum_t b\pi_\theta(a_t|s_t)\nabla_\theta \log \pi_\theta(a_t|s_t)\right] d\tau$$
$$= \int \mathrm{b}\nabla_\theta \pi_\theta(\tau)\, d\tau$$
$$= b\nabla_\theta \int \pi_\theta(\tau)\, d\tau = b\nabla_\theta 1 = 0$$

Keeps the gradient unbiased, i.e. doesn't change the expected value, but can change the variance!

Log derivative trick: $\frac{\nabla_\theta x}{x} = \nabla_\theta \log x$

Choice of baseline

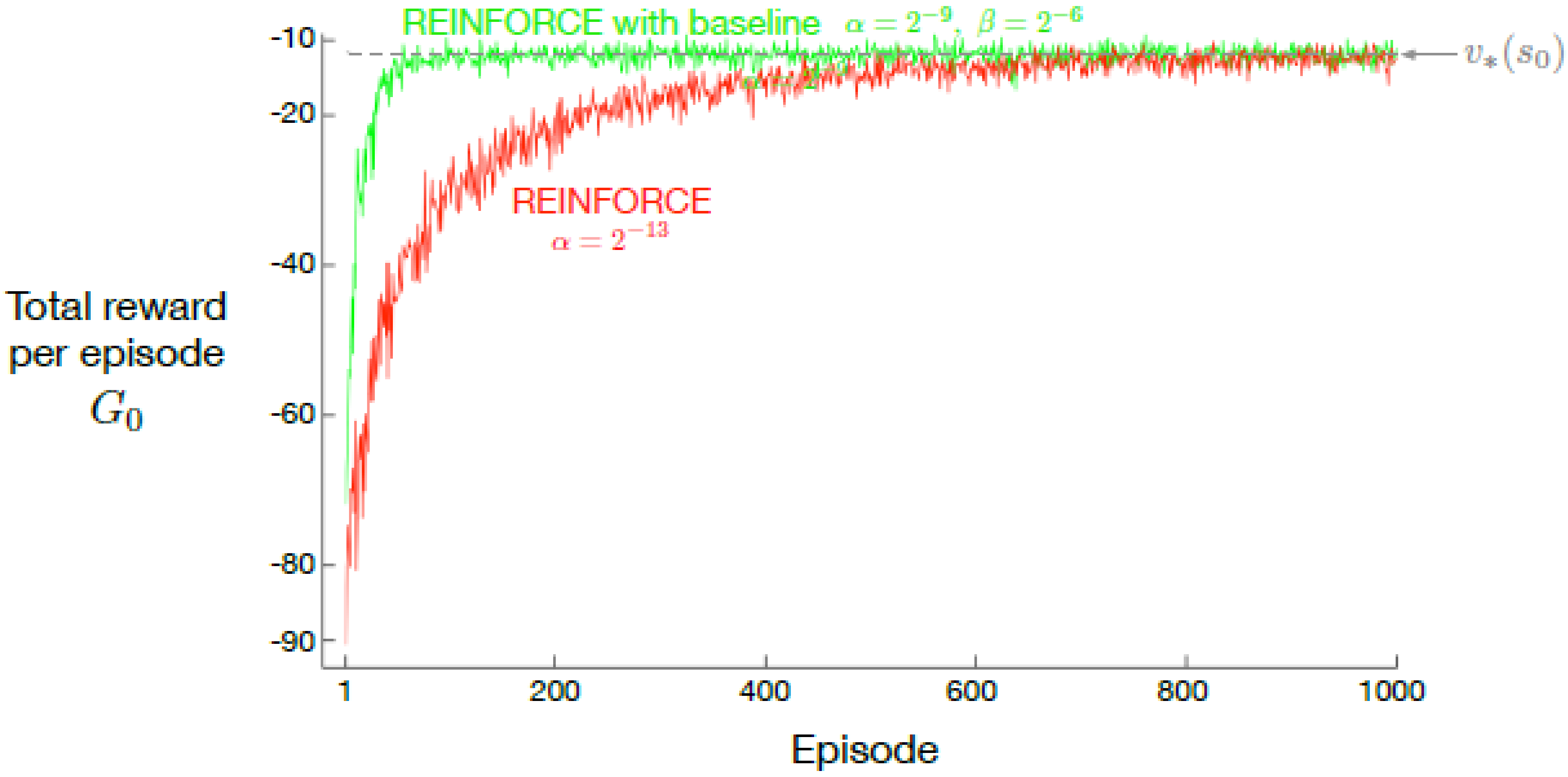
- What baseline to use?

- $b(s_t) = V(s_t)$
  - Change based on whether or not reward was better than expected

  - Term $r(\tau) - b(s_t)$ resembles advantage function:
    - $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$
    - Measures how much better a is than whatever π would have done

- Suggests we should be learning π and V!

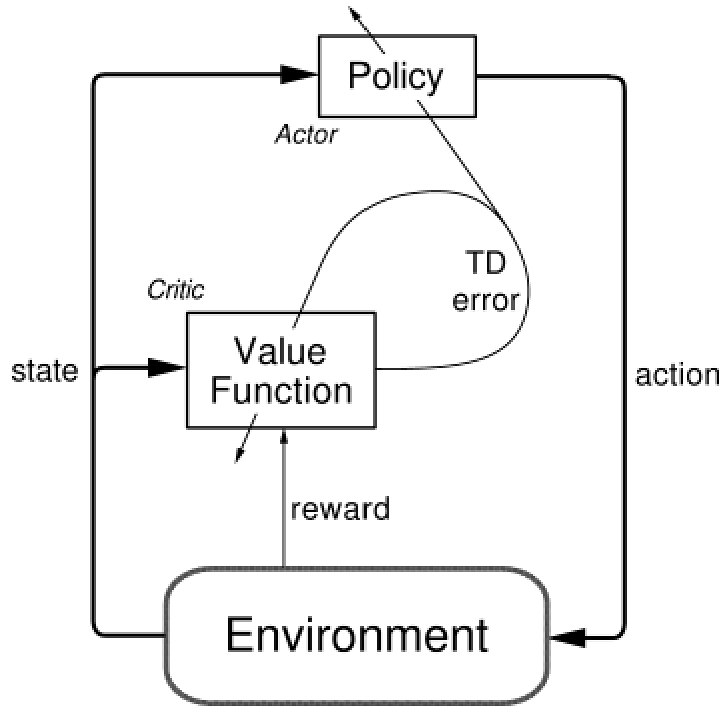
Learning with a baseline



Actor-Critic

- Combine ideas from policy and value function methods
  - Approximate both the policy and the value function
- Actor improvement
  - Policy parameterised by θ
- Critic evaluation
  - Value function parameterised by ω
  - Either $V(s;\omega)$ or $Q(s,a;\omega)$

- Keep track of two sets of parameters



Actor-Critic pseudocode

- Input: parameterised forms for $\pi_\theta(s|a)$ and $V_\omega(s)$

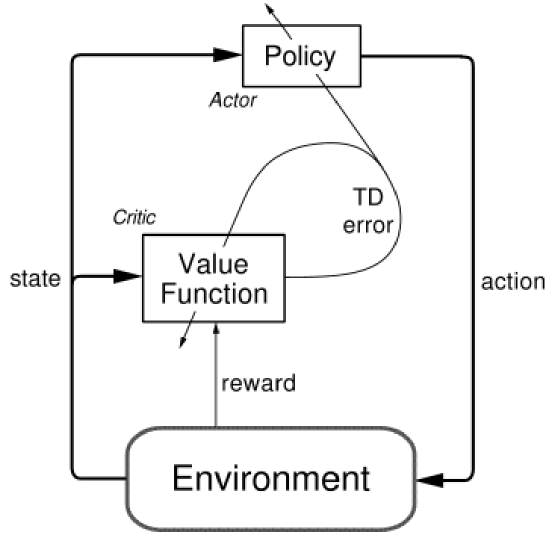
What forms could we use?

- Input: learning rates $\alpha_\omega > 0$ and $\alpha_\theta > 0$
- For each episode:
    - Initialise s
    - For each time step:
        - Choose $a \sim \pi_\theta(s|a)$
        - Take a, observe s', r
        - $\delta \leftarrow r + \gamma V_\omega(s') - V_\omega(s)$
        - $\omega \leftarrow \omega + \alpha_\omega \delta \nabla_\omega V_\omega(s)$
        - $\theta \leftarrow \theta + \alpha_\theta \delta \nabla_\theta \log \pi_\theta\,(a|s)$
        - $s \leftarrow s'$

Policy is stochastic: this is a random draw

Compute TD error

Update parameters by gradient ascent



Many different ways to do the updates

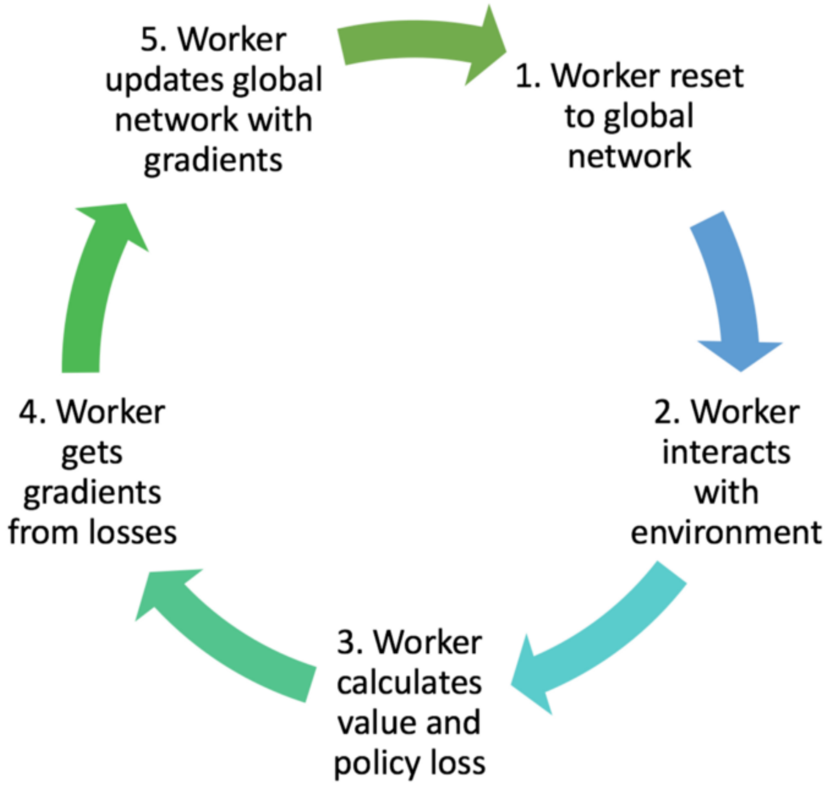
$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ v_t\right]$	REINFORCE
$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)\right]$	Q Actor-Critic
$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^w(s,a)\right]$	Advantage Actor-Critic
$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta\right]$	TD Actor-Critic
$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta e\right]$	TD(λ) Actor-Critic
$G_\theta^{-1}\nabla_\theta J(\theta) = w$	Natural Actor-Critic

Asynchronous Advantage Actor-Critic (A3C)

[Mnih et al., 2016]

- Actor-Critic can be easily parallelised
- Why is this useful?
  - Speed up exploration of state space

- Have multiple agents training with shared parameters



Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

```
// Assume global shared parameter vectors θ and θ_v and global shared counter T = 0
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter t ← 1
repeat
    Reset gradients: dθ ← 0 and dθ_v ← 0.
    Synchronize thread-specific parameters θ' = θ and θ'_v = θ_v
    t_start = t
    Get state s_t
    repeat
        Perform a_t according to policy π(a_t|s_t; θ')
        Receive reward r_t and new state s_{t+1}
        t ← t + 1
        T ← T + 1
    until terminal s_t or t − t_start == t_max
    R = { 0             for terminal s_t
        { V(s_t, θ'_v)  for non-terminal s_t // Bootstrap from last state
    for i ∈ {t − 1, ..., t_start} do
        R ← r_i + γR
        Accumulate gradients wrt θ': dθ ← dθ + ∇_θ' log π(a_i|s_i; θ')(R − V(s_i; θ'_v))
        Accumulate gradients wrt θ'_v: dθ_v ← dθ_v + ∂(R − V(s_i; θ'_v))^2/∂θ'_v
    end for
    Perform asynchronous update of θ using dθ and of θ_v using dθ_v.
until T > T_max
```

Spin up a new agent/thread

Use global parameters

Act

Update local parameters using advantage functions

Update global parameters

[Mnih et al., 2016]

Deep policy search

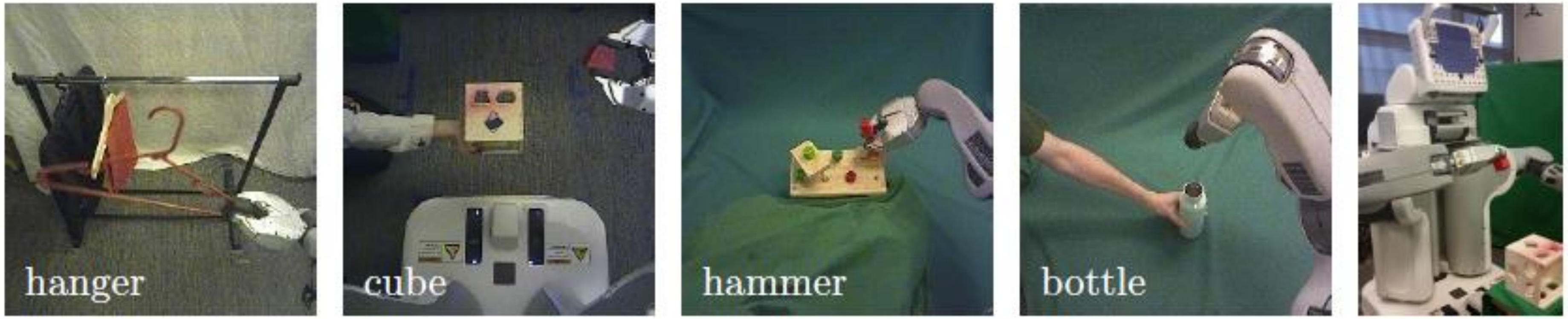
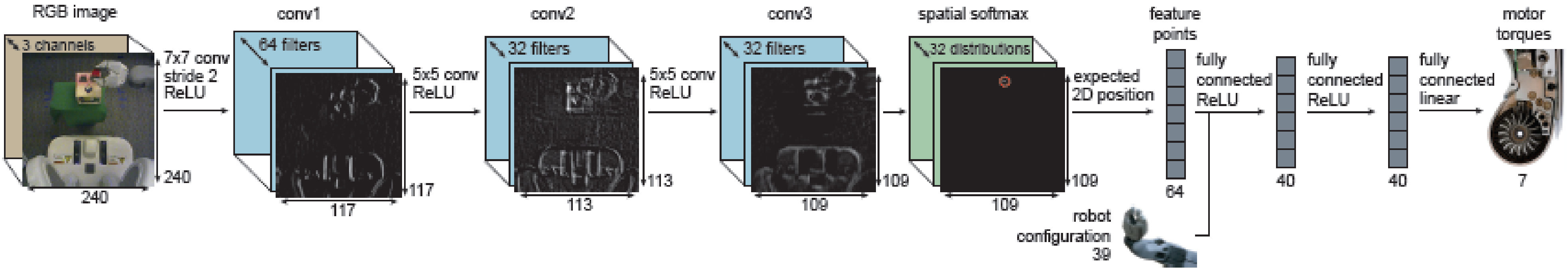
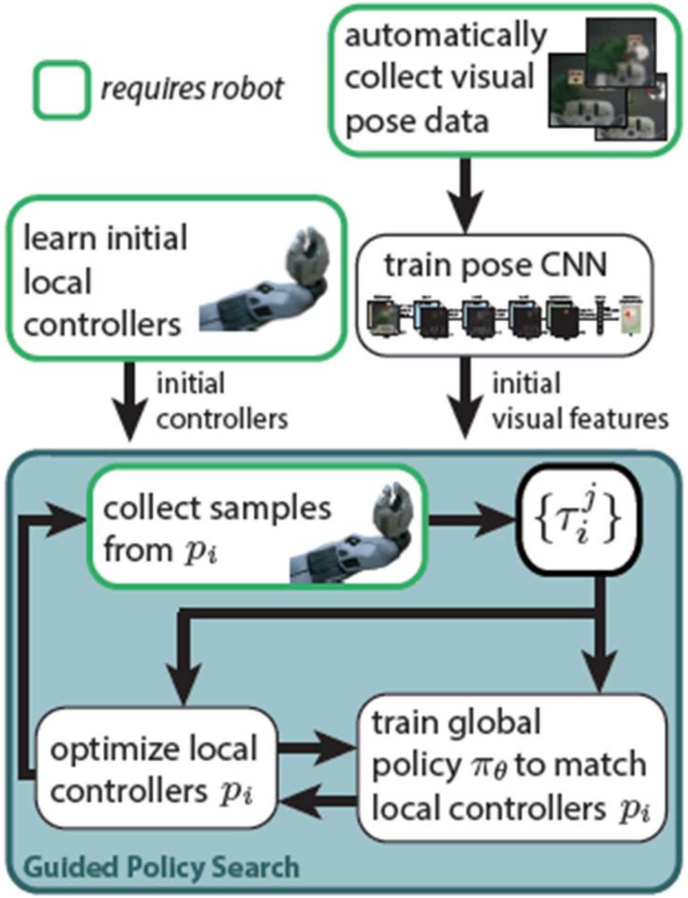


Figure 1: Our method learns visuomotor policies that directly use camera image observations (left) to set motor torques on a PR2 robot (right).



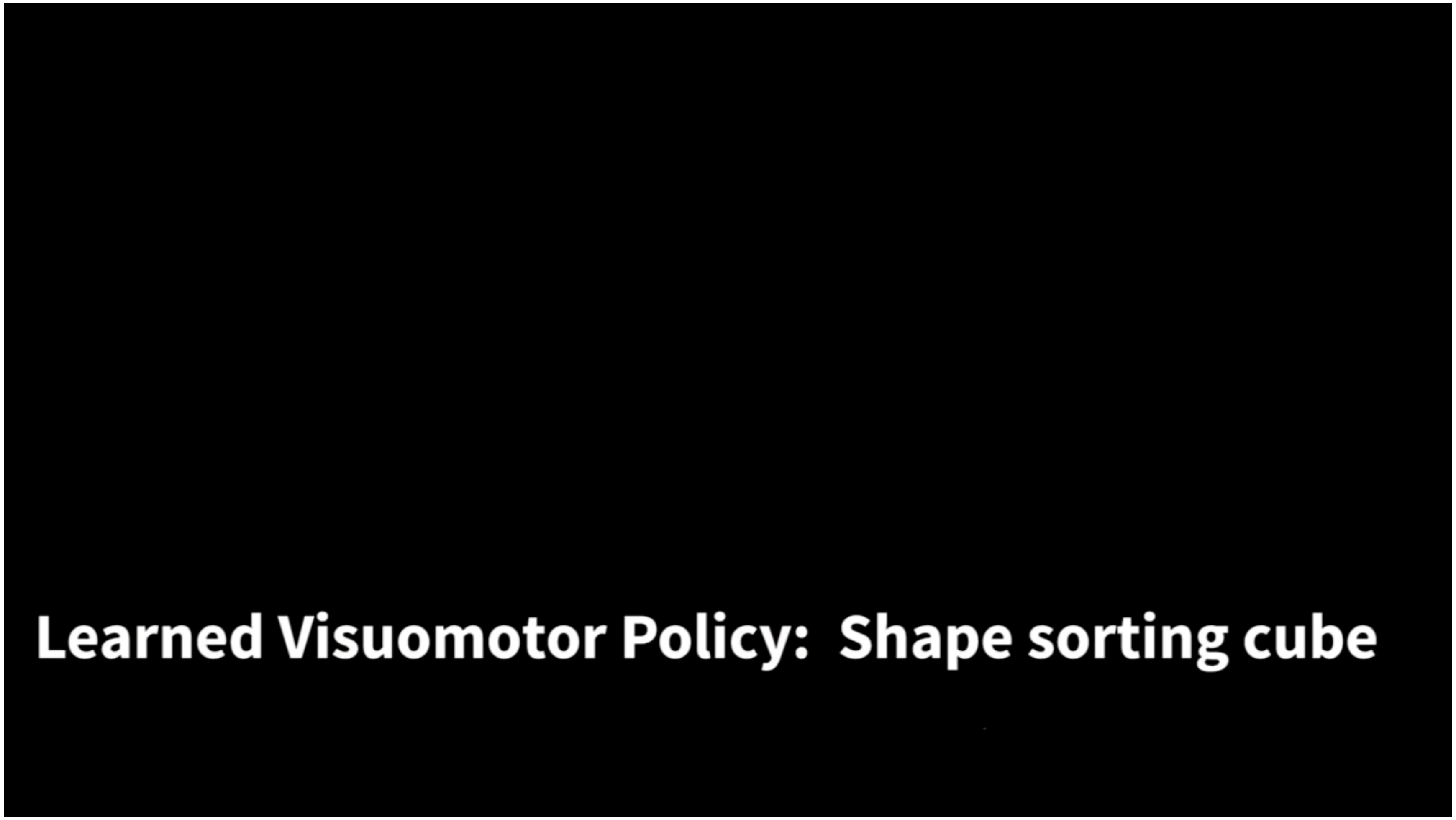
[Levine et al., 2016]

Deep policy search



[Levine et al., 2016]

Robotics



[Levine et al., 2016]

Conclusion

- Recap:
  - The RL setting, MDPs
  - Rewards and value functions
  - Q-learning
  - Function approximation → DQN
- Policy-based methods
  - Gradient free
    - Hill climbing
  - Gradient based
    - Policy Gradient Theorem
    - REINFORCE
    - Baselines
    - Actor-Critic (A3C)
  - Incorporate ideas from supervised learning, deep learning, etc.