CS 4610/5335 – Lecture 21
Learning for decision making

Lawson L.S. Wong
Northeastern University
4/20/22

Announcements

Project presentations:

- 4/27 for CS 4610 (including majority CS 4610 teams)
- 5/2 and 5/4 for CS 5335

- ~15 minutes per team
- We do not expect everything to be completed yet
- Tell an interesting story (the process)

More on the presentations and project reports will be posted by end of Friday

Outline

Imitation learning – behavior cloning

Reinforcement learning

Imitation learning – inverse reinforcement learning
(time permitting)

Technical introduction to RL

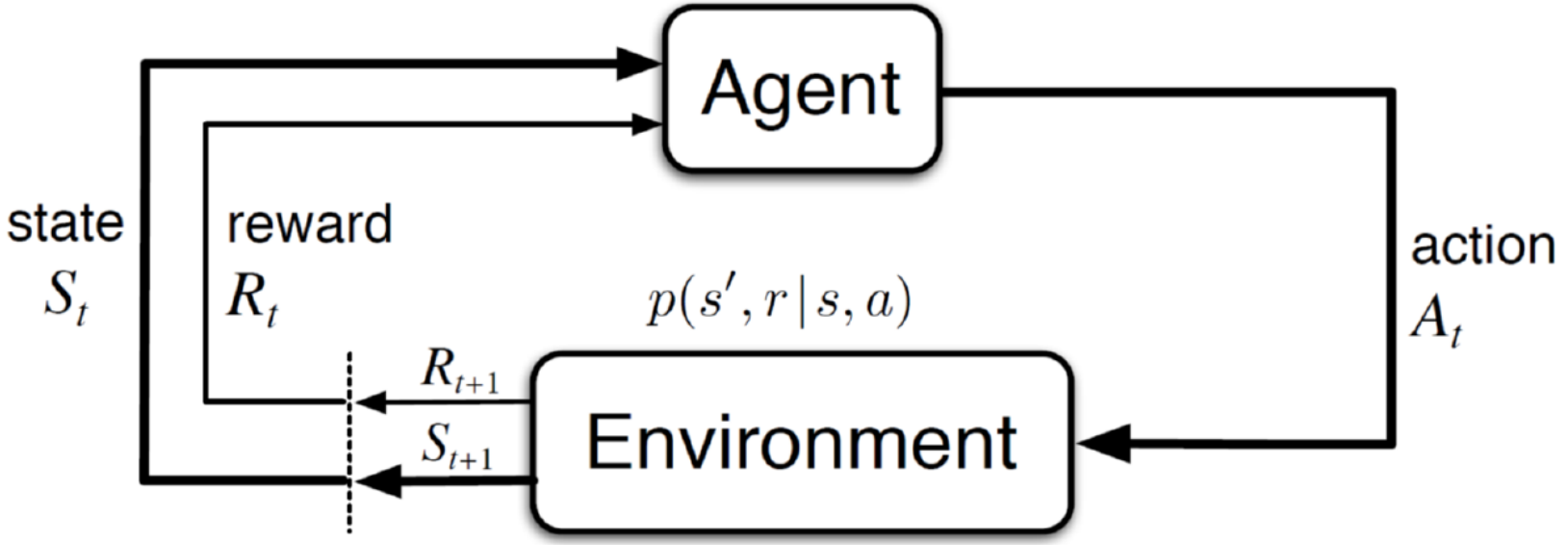


Figure 3.1: The agent–environment interaction in a Markov decision process.

$$\text{MDP } \mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$$

$$= \text{(state space, action space, transition fn., reward fn., discount factor)}$$

Transition / reward functions shown as $p(s', r|s, a)$ above

Objective: Find **policy** $\pi : \mathcal{S} \to \mathcal{A}$

Sutton & Barto, Reinforcement Learning: An Introduction

An Algorithmic Perspective on Imitation Learning

Takayuki Osa
University of Tokyo
osa@edu.k.u-tokyo.ac.jp

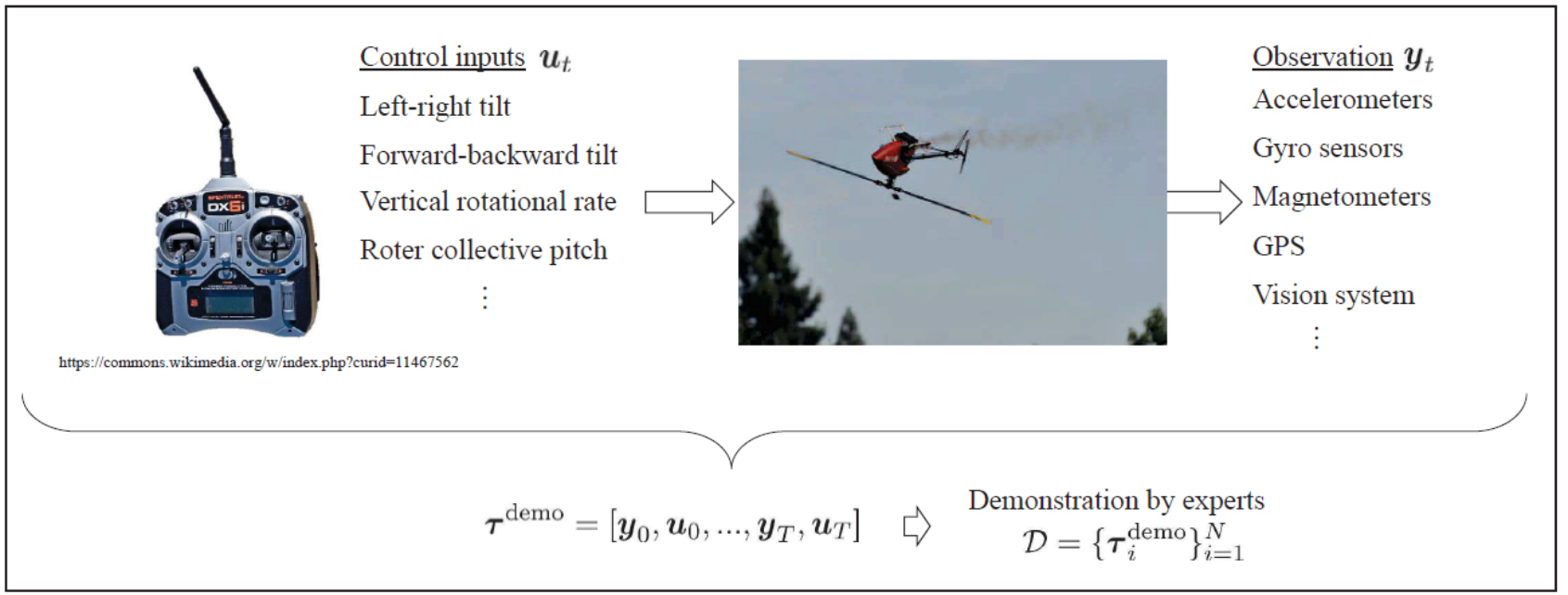
Joni Pajarinen
Technische Universität Darmstadt
pajarinen@ias.tu-darmstadt.de

Gerhard Neumann
University of Lincoln
gneumann@lincoln.ac.uk

J. Andrew Bagnell
Carnegie Mellon University
dbagnell2@andrew.cmu.edu

Pieter Abbeel
University of California, Berkeley
pabbeel@cs.berkeley.edu

Jan Peters
Technische Universität Darmstadt
mail@jan-peters.net



(a) Learning of acrobatic RC helicopter maneuvers [Abbeel et al., 2010]. The trajectories for acrobatic flights are learned from a human expert's demonstrations. To control the system with highly nonlinear dynamics, iterative learning control was used.

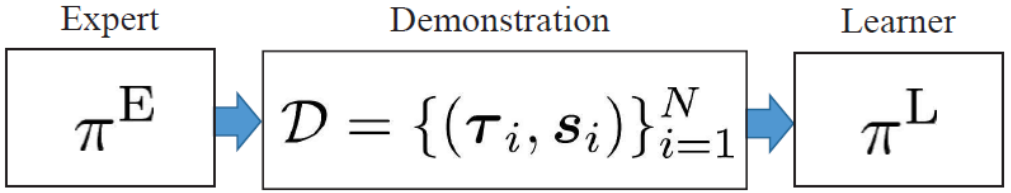


Figure 2.2: Diagram of general imitation learning. The learner cannot directly observe the expert's policy in many problems. Instead, a set of trajectories induced by the expert's policy is available in imitation learning. The learner estimates the policy that reproduces the expert's behavior using the given demonstrations. Please note that the process of querying the demonstration and updating the learner's policy can be interactive.

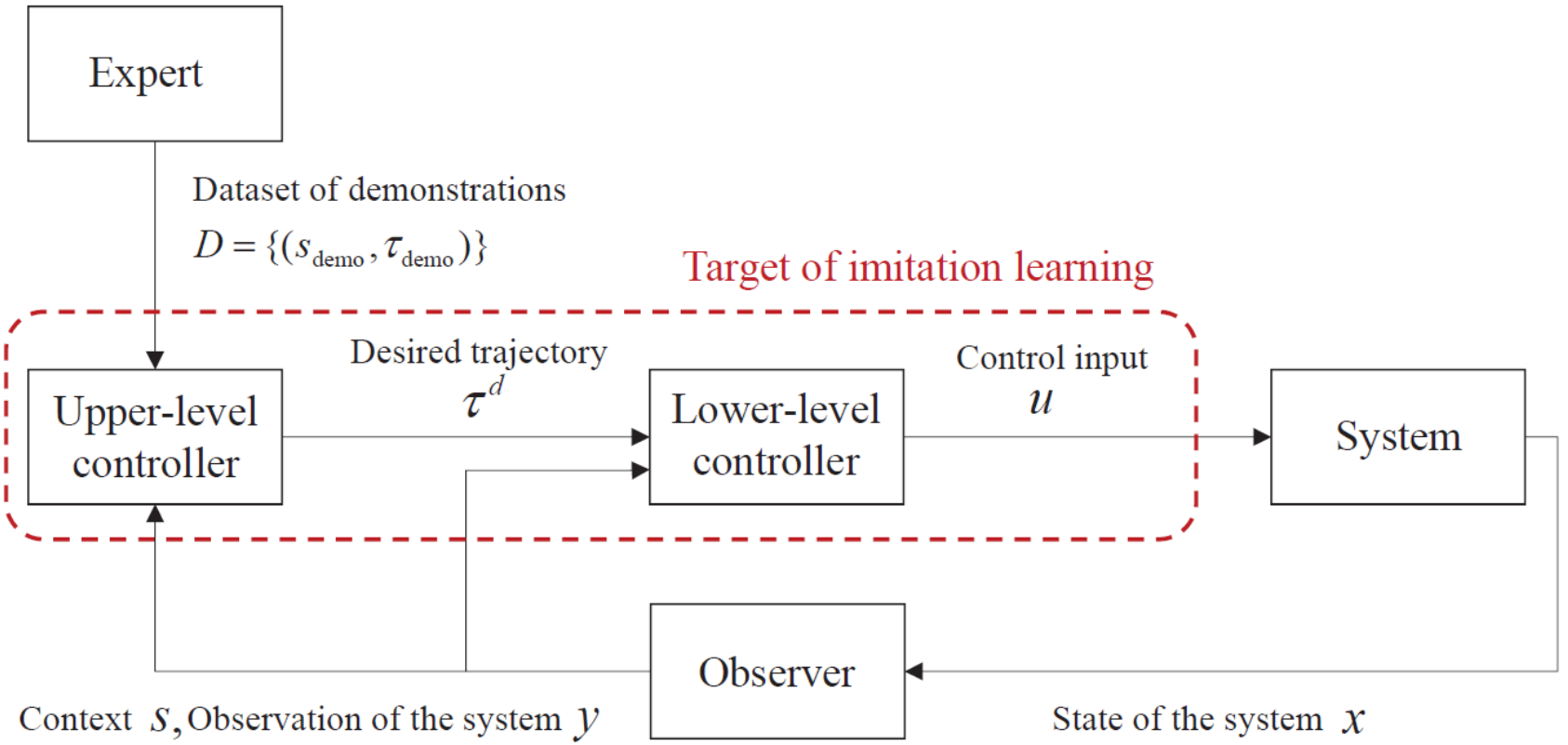


Figure 3.1: Control diagram of a robotic system with imitation learning. An expert demonstrates the desired behavior generating a dataset D. Based on D and observations about the current context and system state an upper-level controller generates the desired trajectory $\boldsymbol{\tau}^d$. A lower-level feedback controller tries to follow $\boldsymbol{\tau}^d$ using observation feedback to generate a control $\boldsymbol{u}$ which causes a change to the system state $\boldsymbol{x}$ and a new observation. In imitation learning, the controllers are tuned to imitate the expert demonstrations.

Algorithm 1 Abstract of behavioral cloning

Collect a set of trajectories demonstrated by the expert $\mathcal{D}$
Select a policy representation $\pi_{\boldsymbol{\theta}}$
Select an objective function $\mathcal{L}$
Optimize $\mathcal{L}$ w.r.t. the policy parameter $\boldsymbol{\theta}$ using $\mathcal{D}$
return optimized policy parameters $\boldsymbol{\theta}$

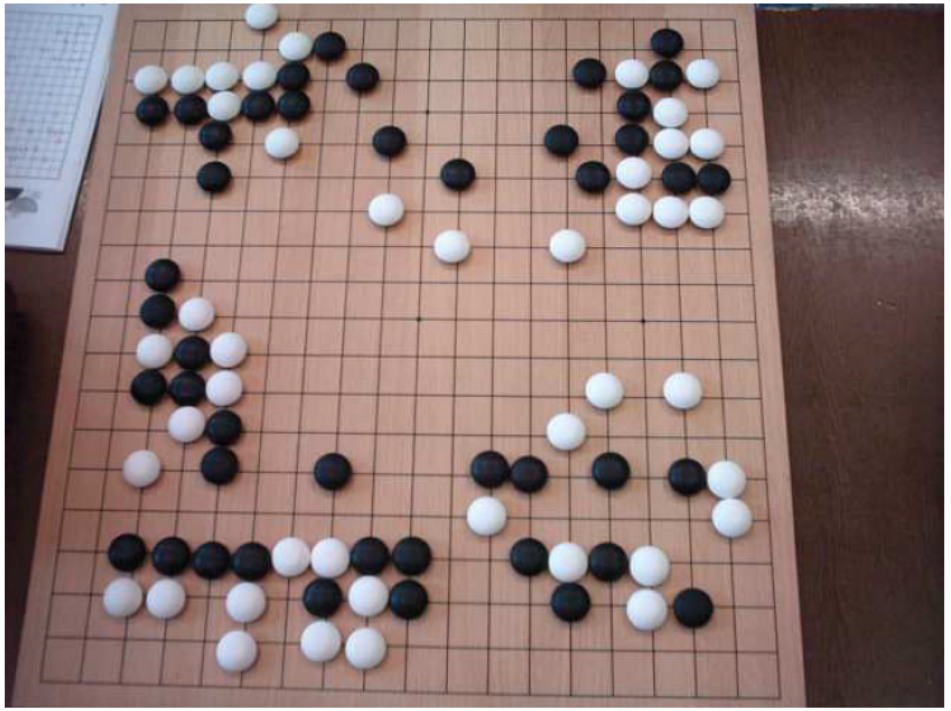


Figure 3.2: The game of Go is played on a 19x19 board. Even though the total number of possible board configurations exceeds 10^{170} and thus the training data can not cover all possible plays, the simple imitation learning approach in [Silver et al., 2016] was able to learn a competitive policy from demonstrations and improve the policy using self-play. [Figure from `https://commons.wikimedia.org/wiki/File:Tuchola_026.jpg`. CC license.]

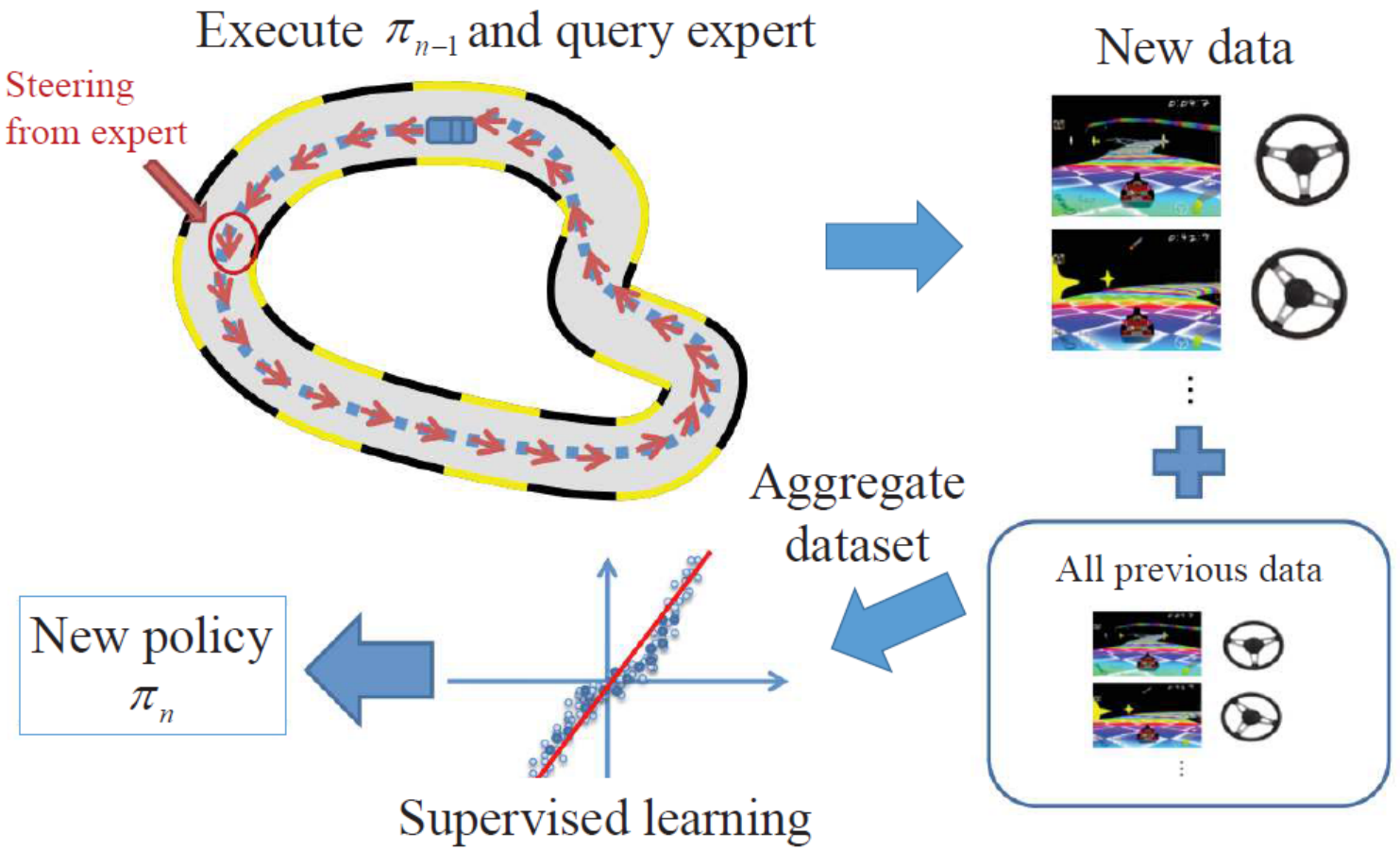


Figure 3.3: An overview of DAGGER from [Bagnell, 2015]. In each iteration, DAGGER generates new examples using the current policy with corrections (labels) provided by the experts, adds the new demonstrations to a demonstration dataset and computes a new policy to optimize performance in aggregate over that dataset. The figure illustrates a single iteration of DAGGER . The basic version of DAGGER initializes the demonstration dataset from a single set of expert demonstrations and then interleaves policy optimization and data generation to grow the dataset. More generally, there is nothing special about aggregating data– any method, like gradient descent or weighted majority that is sufficiently stable in its policy generation and does well on average over the iterations (or more broadly, all *no-regret* algorithm run over each iterations dataset) will achieve the same guarantees, and maybe strongly preferred for computational reasons.

Algorithm 3 DAGGER [Ross et al., 2011]

Input: initial dataset of demonstrations $\mathcal{D} = \{(\boldsymbol{x}, \boldsymbol{u})\}$, $\{\beta_i\}$ such that $\frac{1}{N}\sum_{i=1}^{N}\beta_i \to 0$

Initialize: π_1^{L}

for $i = 1$ **to** N **do**

Let $\pi_i = \beta_i \pi^{\mathrm{E}} + (1 - \beta_i)\pi_i^{\mathrm{L}}$.

Sample trajectories $\boldsymbol{\tau} = [\boldsymbol{x}_0, \boldsymbol{u}_0, ..., \boldsymbol{x}_T, \boldsymbol{u}_T]$ using π_i

Get dataset $\mathcal{D}_i$ of visited states by π_i and actions given by expert.

Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$

Train the policy π_{i+1}^{L} on $\mathcal{D}$.

end for

return best π_i^{L} on validation.

Outline

Imitation learning – behavior cloning

Reinforcement learning

Imitation learning – inverse reinforcement learning
(time permitting)

Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

Deep Q-Networks

LETTER

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

Here we use recent advances in training deep neural networks[9–11] to develop a novel artificial agent, termed a deep Q-network, that can learn successful policies directly from high-dimensional sensory inputs using end-to-end reinforcement learning. We tested this agent on the challenging domain of classic Atari 2600 games[12]. We demonstrate that the deep Q-network agent, receiving only the pixels and the game score as inputs, was able to surpass the performance of all previous algorithms and achieve a level comparable to that of a professional human games tester across a set of 49 games, using the same algorithm, network architecture and hyperparameters. This work

Deep Q-Networks

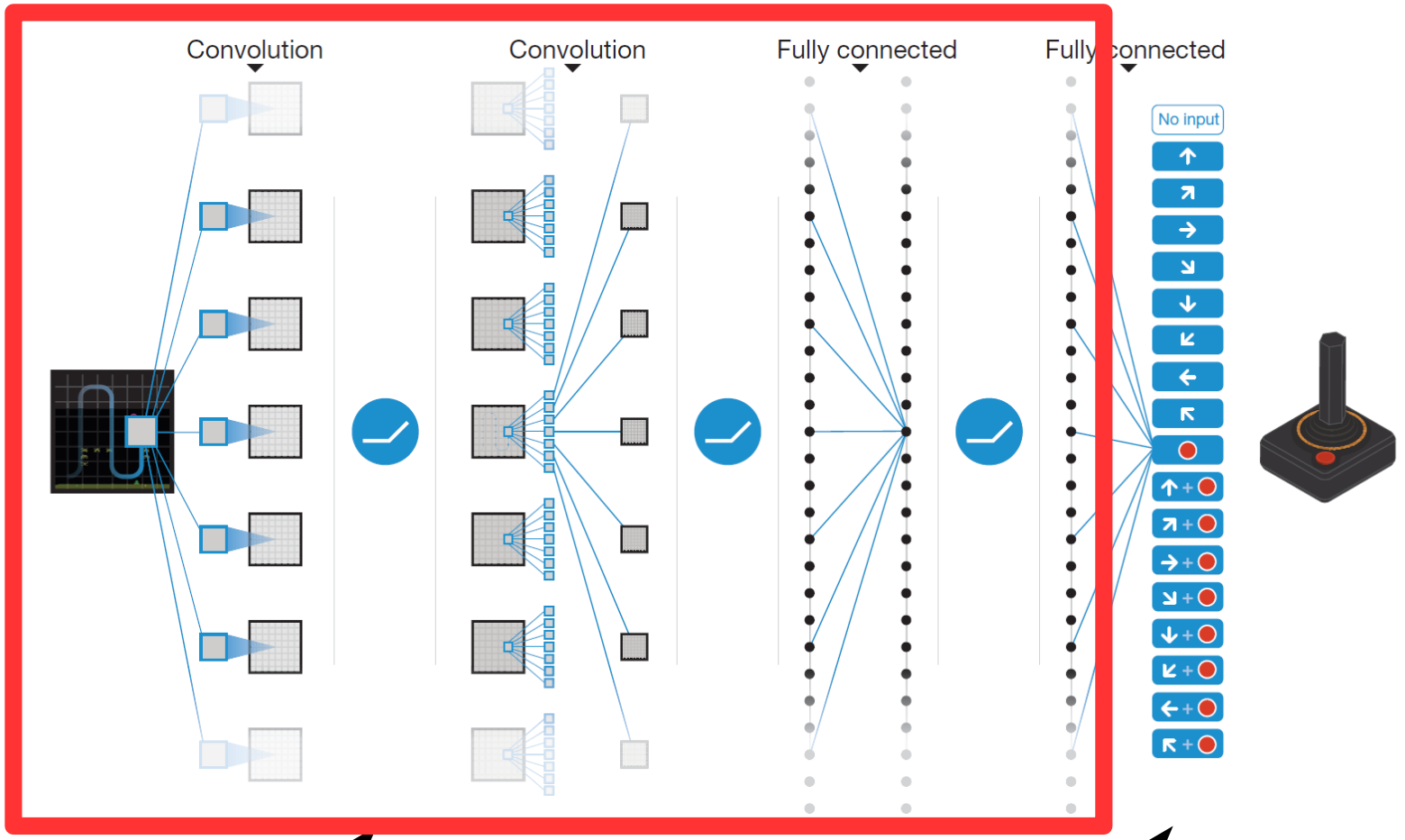


Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an $84 \times 84 \times 4$ image produced by the preprocessing map ϕ, followed by three convolutional layers (note: snaking blue line symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is, $\max(0,x)$).

This replaces hand-crafted features with a convolutional neural network (learned features)

This is Q-learning with linear function approximation

Deep Q-Networks: Objective

More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \middle| s_t = s,\ a_t = a,\ \pi\right],$$

which is the maximum sum of rewards r_t discounted by γ at each time-step t, achievable by a behaviour policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a) (see Methods)[19].

Deep Q-Networks: Objective

More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \,|\, s_t = s,\, a_t = a,\, \pi\right],$$

which is the maximum sum of rewards r_t discounted by γ at each time-step t, achievable by a behaviour policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a) (see Methods)[19].

The basic idea behind many reinforcement learning algorithms is to estimate the action-value function by using the Bellman equation as an iterative update, $Q_{i+1}(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$. Such value iteration algorithms converge to the optimal action-value function, $Q_i \rightarrow Q^*$ as $i \rightarrow \infty$.

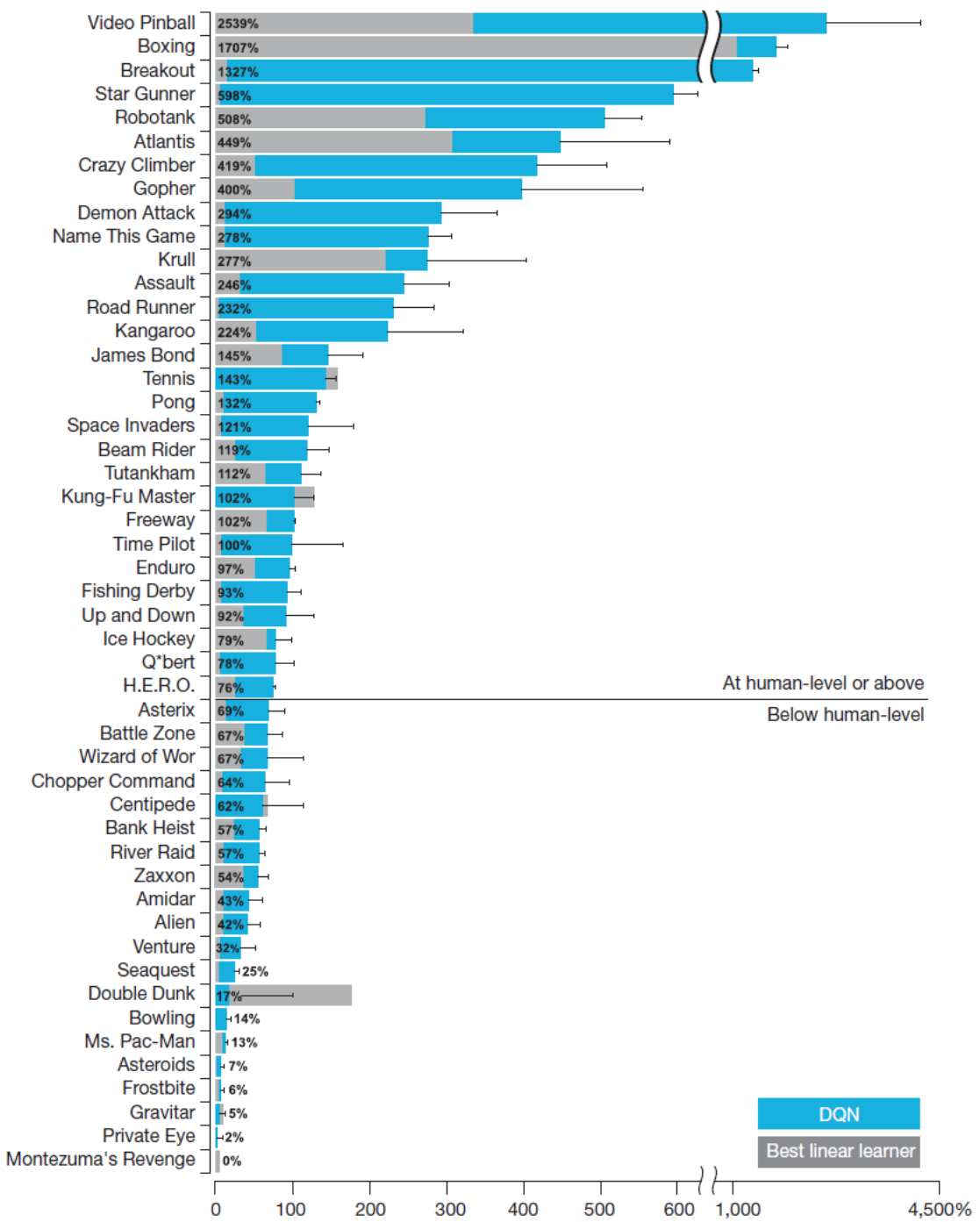
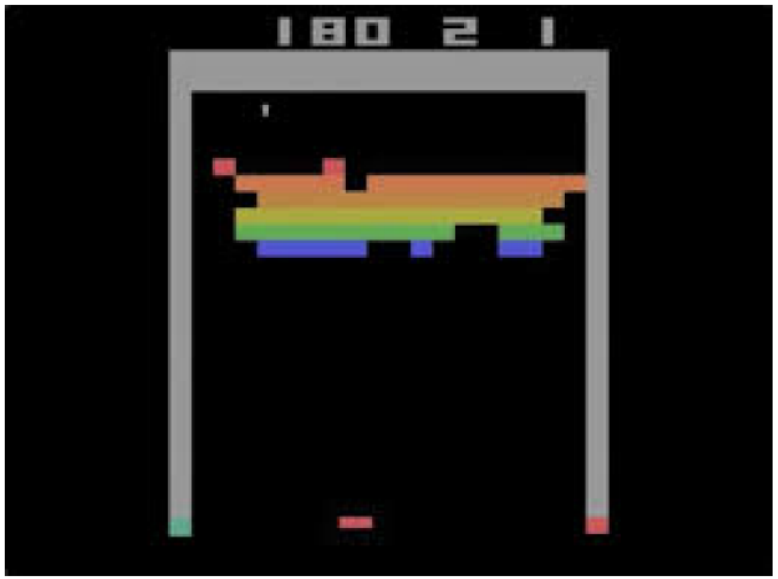


Figure 3 | Comparison of the DQN agent with the best reinforcement learning methods[15] in the literature. The performance of DQN is normalized with respect to a professional human games tester (that is, 100% level) and random play (that is, 0% level). Note that the normalized performance of DQN, expressed as a percentage, is calculated as: 100 × (DQN score − random play score)/(human score − random play score). It can be seen that DQN outperforms competing methods (also see Extended Data Table 2) in almost all the games, and performs at a level that is broadly comparable with or superior to a professional human games tester (that is, operationalized as a level of 75% or above) in the majority of games. Audio output was disabled for both human players and agents. Error bars indicate s.d. across the 30 evaluation episodes, starting with different initial conditions.

Do we really need to learn a value function?

Sometimes, the policy might be quite simple

Breakout: Do not die (ensure paddle always underneath ball)



In contrast, value function might be quite complicated (depends on remaining bricks, etc.)

“Yak shaving”: Do not make the solution more complicated than the original problem

Policy search

Optimize the policy directly with respect to some objective

Consider a parameterized family of policies:

$$\pi(a|s,\boldsymbol{\theta}) = \Pr\{A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$$

$$\boldsymbol{\theta} \in \mathbb{R}^{d'} \text{ for the policy's parameter vector}$$

Define a scalar performance measure $J(\boldsymbol{\theta})$:

Typically: $$J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0)$$

Goal: Find $\boldsymbol{\theta}$ that maximizes $J(\boldsymbol{\theta})$

End-to-End Training of Deep Visuomotor Policies

Sergey Levine[†] SVLEVINE@EECS.BERKELEY.EDU
Chelsea Finn[†] CBFINN@EECS.BERKELEY.EDU
Trevor Darrell TREVOR@EECS.BERKELEY.EDU
Pieter Abbeel PABBEEL@EECS.BERKELEY.EDU
Division of Computer Science
University of California
Berkeley, CA 94720-1776, USA
[†]These authors contributed equally.

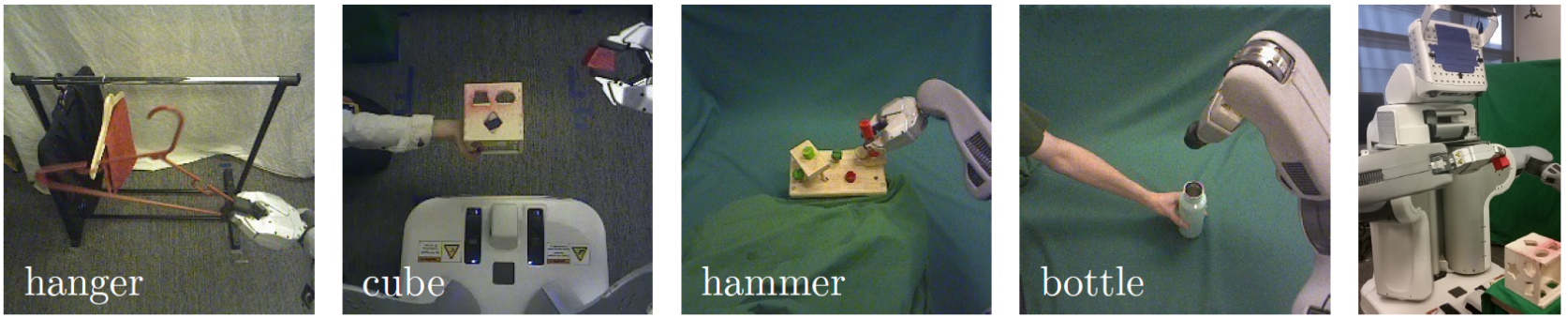


Figure 1: Our method learns visuomotor policies that directly use camera image observations (left) to set motor torques on a PR2 robot (right).

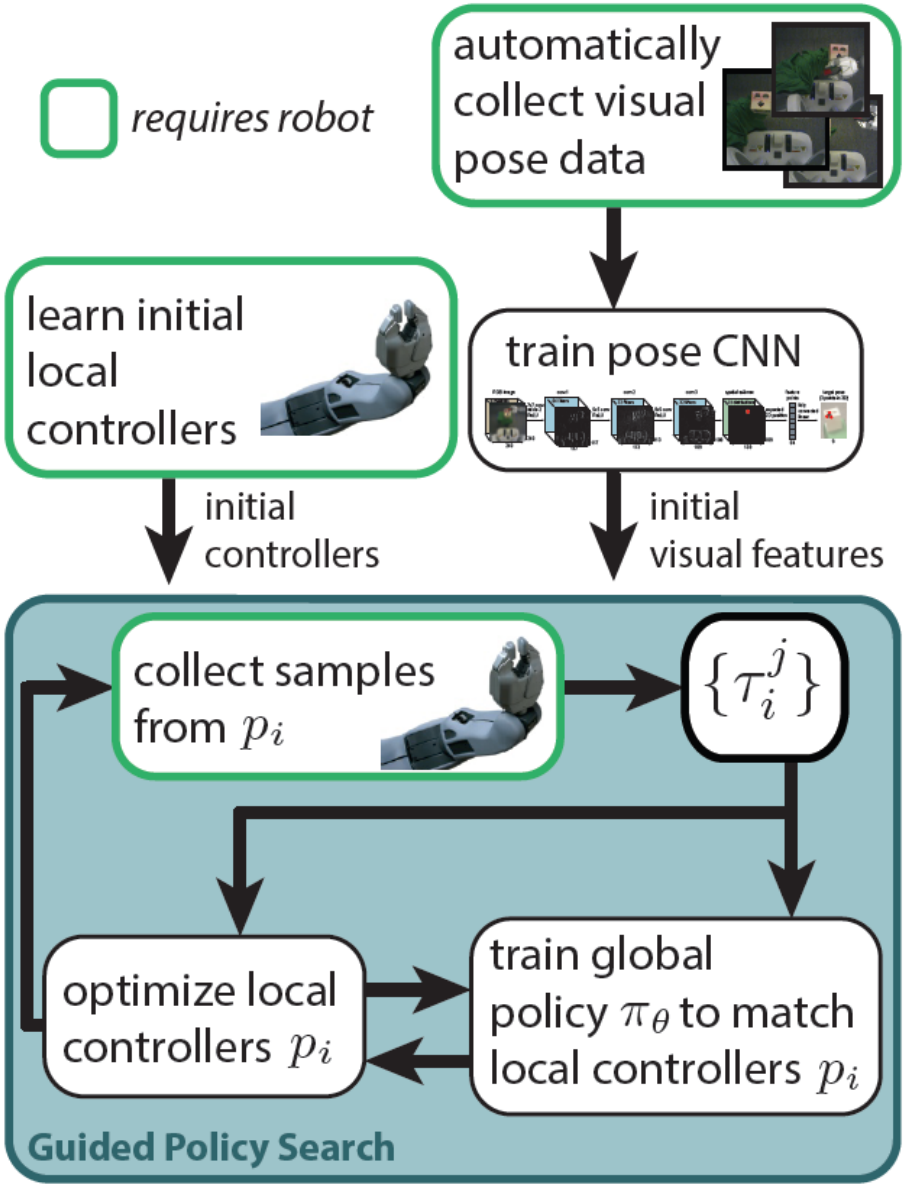


Figure 3: Diagram of our approach, including the main guided policy search phase and initialization phases.

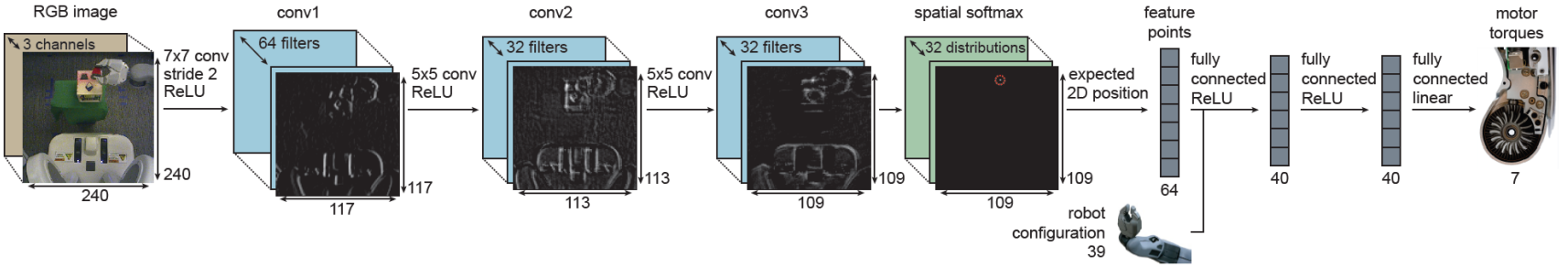
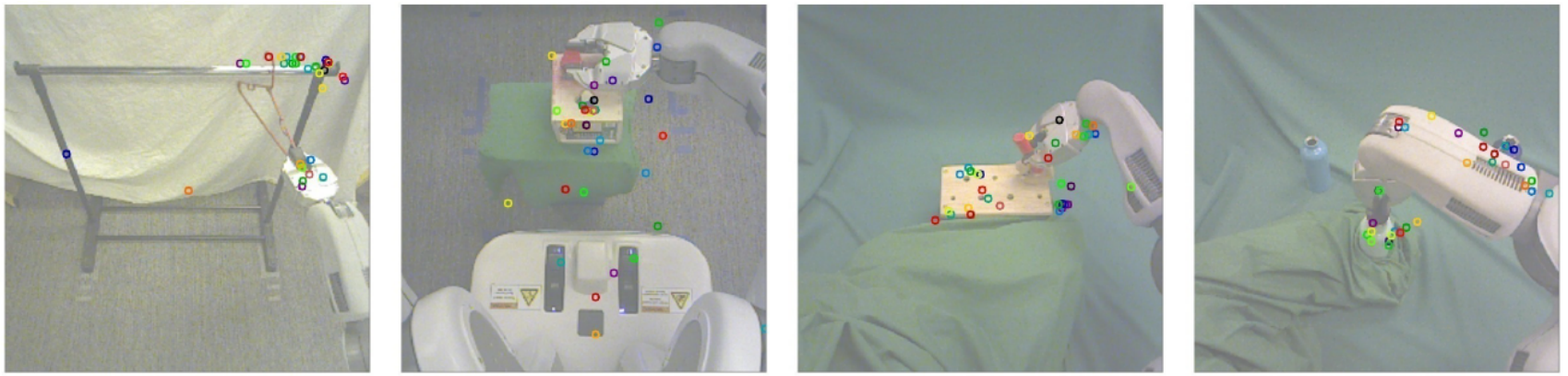
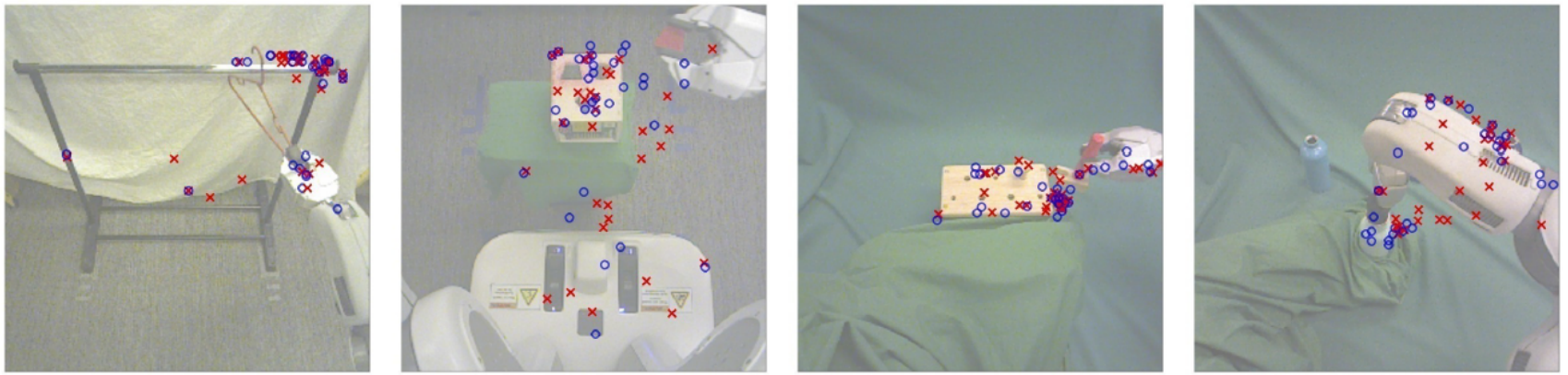


Figure 2: Visuomotor policy architecture. The network contains three convolutional layers, followed by a spatial softmax and an expected position layer that converts pixel-wise features to feature points, which are better suited for spatial computations. The points are concatenated with the robot configuration, then passed through three fully connected layers to produce the torques.



(a) hanger (b) cube (c) hammer (d) bottle

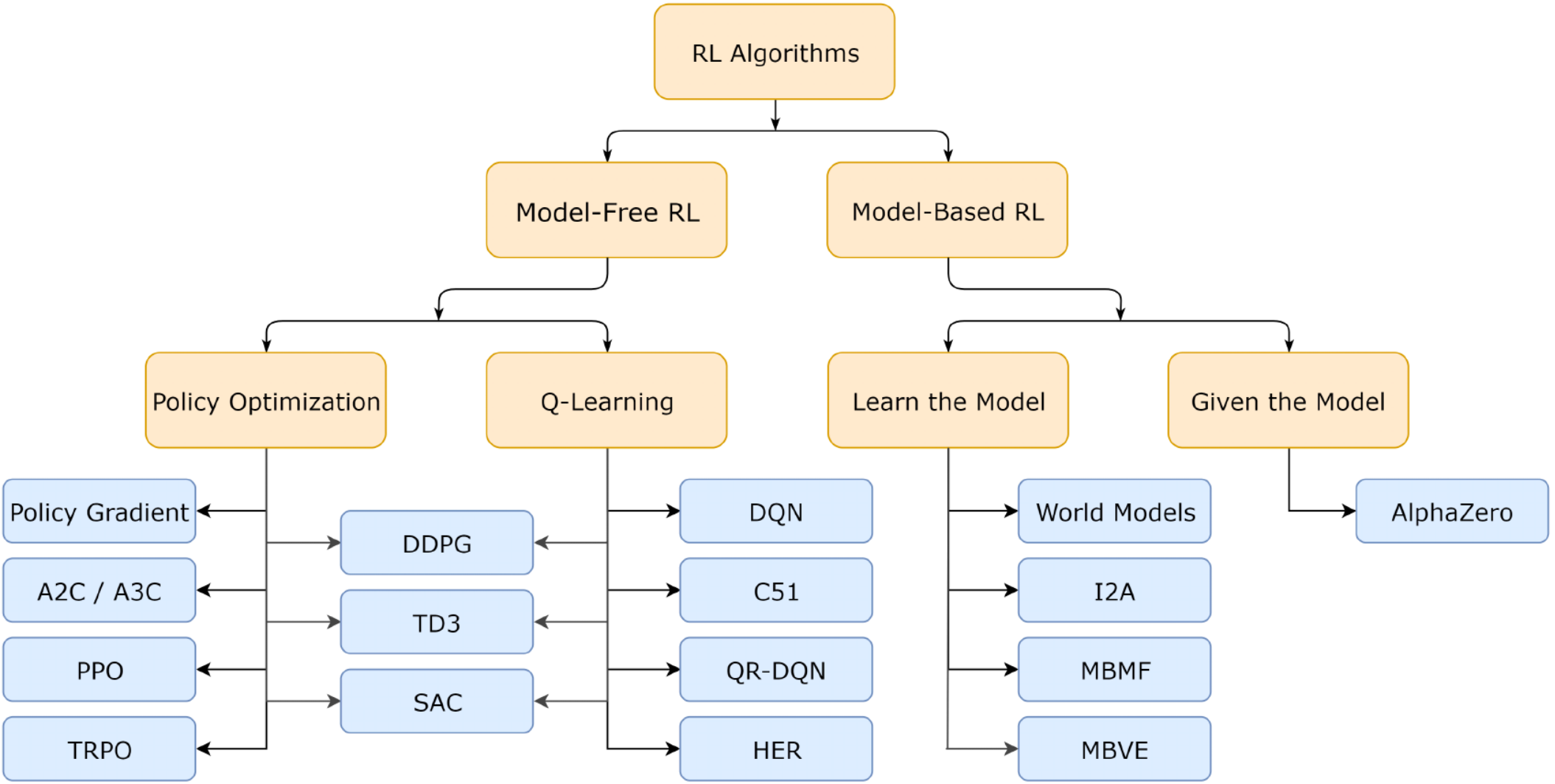
Figure 10: Feature points tracked by the policy during task execution for each of the four tasks. Each feature point is displayed in a different random color, with consistent coloring across images. The policy finds features on the target object and the robot gripper and arm. In the bottle cap task, note that the policy correctly ignores the distractor bottle in the background, even though it was not present during training.



(a) hanger (b) cube (c) hammer (d) bottle

Figure 11: Feature points learned for each task. For each input image, the feature points produced by the policy are shown in blue, while the feature points of the pose prediction network are shown in red. The end-to-end trained policy tends to discover more feature points on the target object and the robot arm than the pose prediction network.

A selection of “modern” RL algorithms

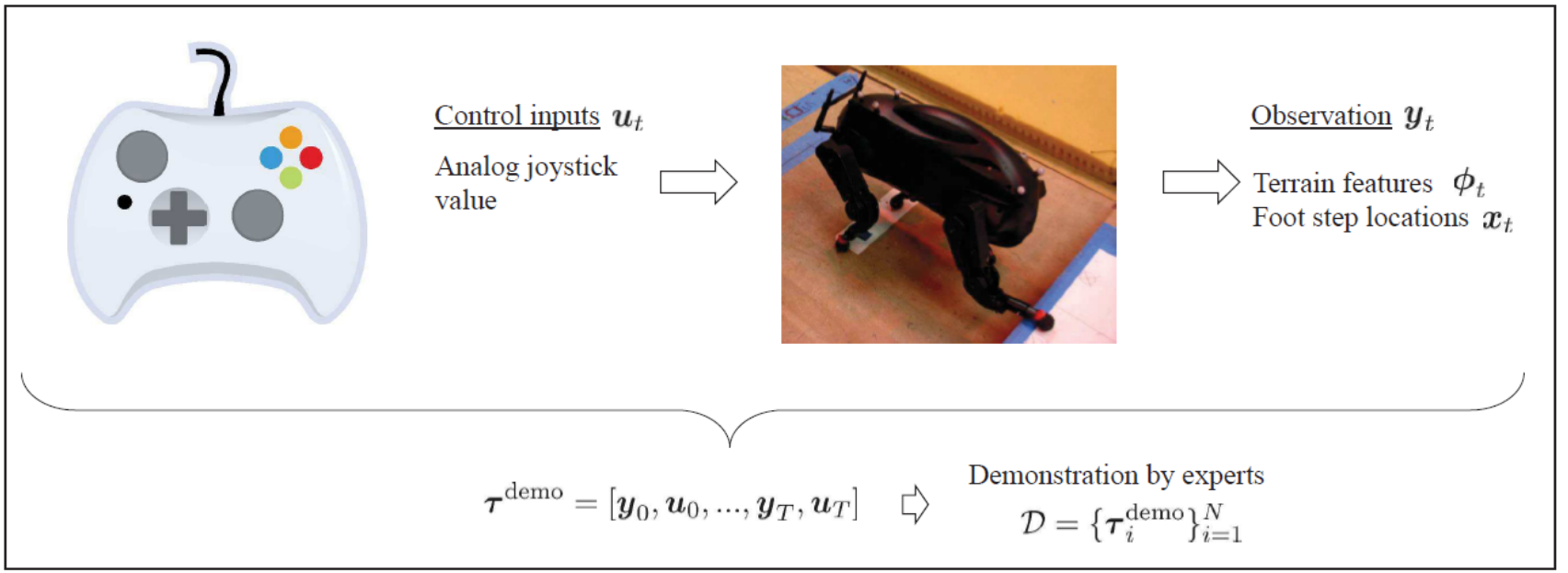


Outline

Imitation learning – behavior cloning

Reinforcement learning

Imitation learning – inverse reinforcement learning
(time permitting)



(c) Learning quadruped robot locomotion [Zucker et al., 2011]. The footstep planning was addressed as an optimization of the reward/cost function, which was recovered from the expert demonstrations. Learning the reward/cost function allows the footstep planning strategy to be generalized to different terrains.

4.1 Problem Statement

Russell defines the problem of IRL [Russell, 1998] as follows:

> **Given** 1) measurements of an agent's behavior over time, in a variety of circumstances, 2) measurements of the sensory inputs to that agent; 3) a model of the physical environment (including the agent's body).
> **Determine** the reward function that the agent is optimizing.

A common assumption in IRL is that the demonstrator utilizes a Markov decision process (MDP) for decision making. Formally, an MDP is a tuple $(\mathcal{X}, \mathcal{U}, \mathcal{P}, \gamma, D, R)$. $\mathcal{X}$ is a finite set of states; $\mathcal{U}$ is a set of control inputs; $\mathcal{P}$ is a set of state transitions probabilities; $\gamma \in [1, 0)$ is a discount factor; D is the initial-state distribution from which the initial state $\boldsymbol{x}_0$ is drawn; and $R : \mathcal{X} \mapsto \mathbb{R}$ is the reward function.

The goal of IRL is to recover the unknown reward function $R(\boldsymbol{\tau})$ from the expert's trajectories.

Algorithm 14 Abstract version of feature matching inverse reinforcement learning

Input: Expert trajectories $\mathcal{D} = \{\boldsymbol{\tau}_i\}_{i=1}^{N}$

Initialize the reward function and policy parameters $\boldsymbol{w}, \boldsymbol{\theta}$

repeat

Evaluate the state-action visitation frequency $\boldsymbol{\mu}$ of the current policy π_θ

Evaluate the objective function $\mathcal{L}$ and its derivative $\nabla_{\boldsymbol{w}}\mathcal{L}$ by comparing $\boldsymbol{\mu}$ and the state-action distribution implied by $\mathcal{D}$

Update the reward function parameter $\boldsymbol{w}$

Update the policy parameter $\boldsymbol{\theta}$ with a reinforcement learning method

until

return optimized policy parameters $\boldsymbol{\theta}$ and reward function parameter $\boldsymbol{w}$

Table 2.2: Categorization of existing imitation learning methods with distinction between model-free and model-based methods. Model-free methods are dominant in behavioral cloning, and model-based methods are dominant in inverse reinforcement learning. Recent studies on IRL have proposed model-free methods.

	Model-free	**Model-based**
Behavioral Cloning	Widrow and Smith [1964], Chambers and Michie [1969], Pomerleau [1988], Schaal et al. [2004], Schaal [1999], Ijspeert et al. [2013], Calinon et al. [2007], Khansari-Zadeh and Billard [2011], Paraschos et al. [2013], Osa et al. [2014], Ross and Bagnell [2010], Ross et al. [2011], Takano and Nakamura [2015], Maeda et al. [2016], Deniša et al. [2016], Ho and Ermon [2016]	Ude et al. [2004], Englert et al. [2013], van den Berg et al. [2010]
Inverse Reinforcement Learning	Boularias et al. [2011] Kalakrishnan et al. [2013]	Abbeel and Ng [2004], Ratliff et al. [2006b], Silver et al. [2010], Ziebart et al. [2008], Ziebart [2010], Levine et al. [2011], Levine and Koltun [2012], Hadfield-Menell et al. [2016], Finn et al. [2016b]

Table 2.1: Advantages and disadvantages of model-based and model-free methods in imitation learning. Model-free methods learn a policy without knowledge on the system dynamics, and the system dynamics is encoded only implicitly in policies. Model-based methods learn a policy that explicitly satisfies the system dynamics by leveraging the system dynamics. However, learning/estimating the system dynamics can be challenging.

	Model-free	**Model-based**
Advantages	A policy can be learned without learning/estimating the system dynamics.	The learning process can be data-efficient. A learned policy satisfies the system dynamics.
Disadvantages	The prediction of future states is difficult. The system dynamics is only implicitly considered in the resulting policy.	Model learning can be difficult. Computationally expensive.

Feedback

Piazza thread: 4/20 Lec 21 Feedback

Please post your answers to the following anonymously.

1. What did you like today?

2. What was unclear?

3. How is your project going?

4. Any additional feedback / comments?