

Robot Learning: Imitation, RL, and Generative Policies

Compare data sources and learning objectives for imitation, reinforcement learning, generative policies and sim-to-real transfer.

Gregory Dudek
COMP 598

-

Focus & Outline

1.What is reinforcement learning (RL)

2.How it's useful

- Informal definition
- Formal definition
- Approaches, use cases
- Issues and shortcomings
- Recent advances, tricks & insights

Easy Reference: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

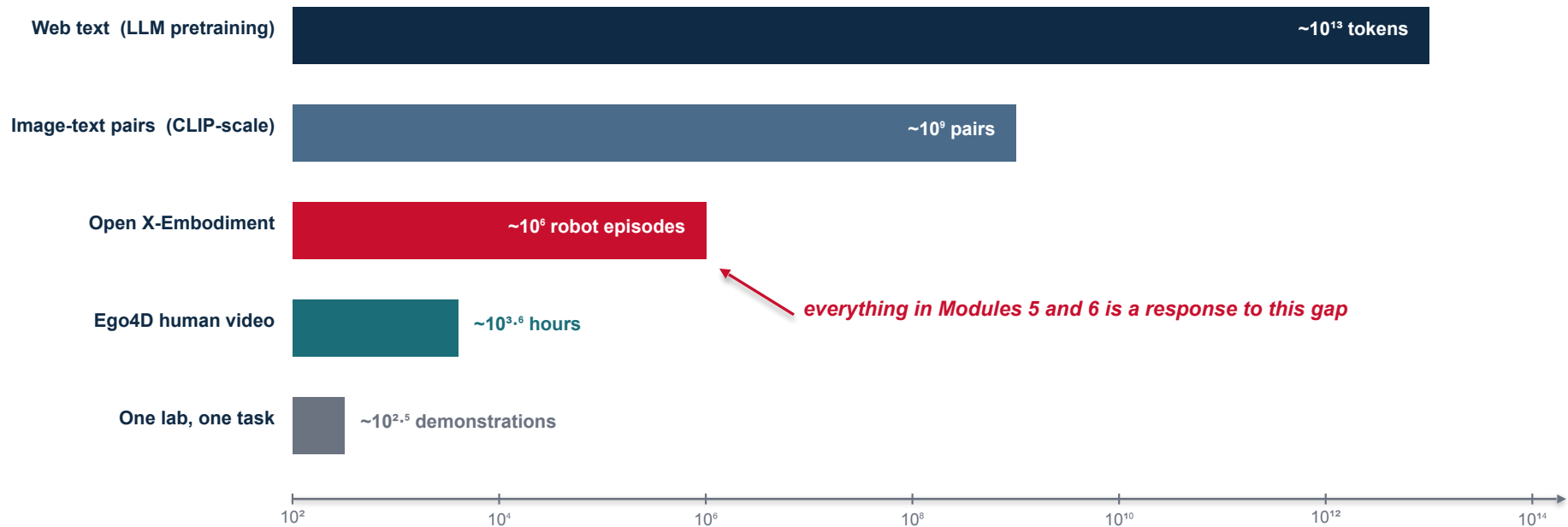
The data problem, stated plainly

- Language models train on $\sim 10^{13}$ tokens scraped for free. Open X-Embodiment is $\sim 10^6$ episodes, collected by dozens of labs over years, at roughly human-labour cost per episode.
- Robot data does not transfer freely across embodiments, cameras, controllers, or even lab lighting.
- Evaluation is as expensive as training data: a real-robot A/B test is hours of human supervision for tens of samples.

Responses to the data constraint

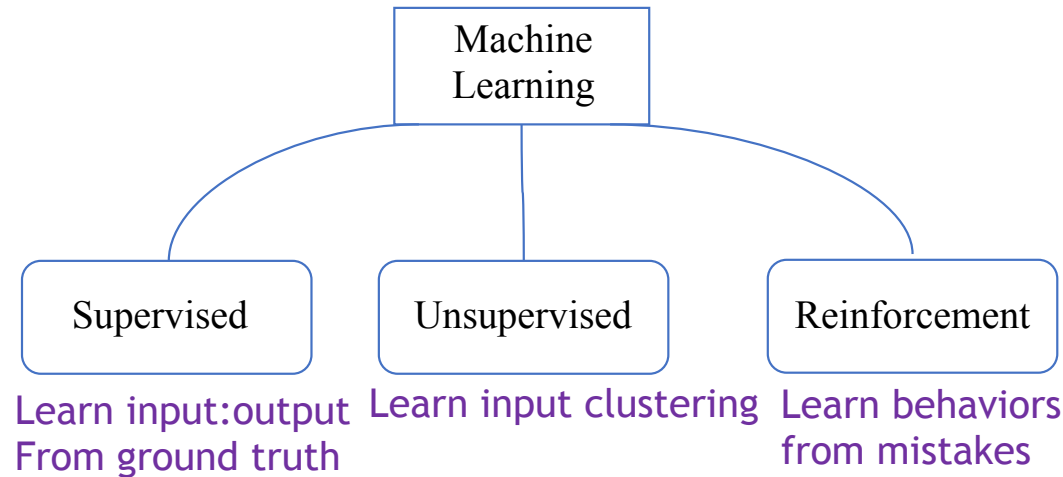
- Imitation — squeeze the most from few demonstrations.
- Simulation and domain randomisation — manufacture data that is cheap but wrong.
- World models — learn the simulator instead of writing it.
- Human video and latent actions — use data collected for other reasons.

The data asymmetry



Units differ deliberately — the comparison is of exponents, not equivalents. Robot action data sits 6–7 orders of magnitude behind the corpora that made the language half work.

RL versus supervised learning



- **Supervised learning:**

- Given a set of labelled data.
- Learn a function to map input features to output labels.

- **Reinforcement learning**

- Collect the training by interacting with the environment.
- Learn a control policy to map the input states to output control actions.
- Optimization is performed over *trajectories* through state space (time dependency).

Supervised Learning

- Supervised Learning
 - Have access to labeled training data (x, y) where $x \sim p(x)$
 - Learn a function $f : X \rightarrow Y$ from training data such that $f(x')$ correctly predicts y' for unseen pairs (x', y') where $x' \sim p(x)$

Supervised Learning: A classic example

- Supervised Learning
 - Have access to labeled training data (x, y) where $x \sim p(x)$
 - Learn a function $f: X \rightarrow Y$ from training data such that $f(x')$ correctly predicts y' for unseen pairs (x', y') where $x' \sim p(x)$



ImageNet Challenge(s)

1000 object classes
1.2M training examples
100K test examples

- classification
- localization
- scene parsing
- detection in video

Behaviour cloning: why it works, and exactly how it fails

- Behaviour cloning (BC) is supervised learning: regress the expert's action from the state, over demonstration pairs. No reward, no exploration, no simulator. That simplicity is why it dominates.

Compounding covariate shift — the central pathology

- Train on the expert's state distribution; deploy on your own. A small error moves you off-distribution, where the policy is worse, which moves you further off.
- Error compounds quadratically in the horizon, not linearly — so long-horizon tasks fail for structural reasons, not for lack of capacity.
- **DAgger** — query the expert on the states the learner visits. Fixes the theory; requires an interactive expert, which is expensive and often infeasible.
- Mitigations in practice: action chunking (fewer decision points), diverse and deliberately 'recoverable' demonstrations, and injected noise during collection.

$$\min_{\theta} \mathbb{E}_{(s, a^*) \sim D} \|\pi_{\theta}(s) - a^*\|^2$$

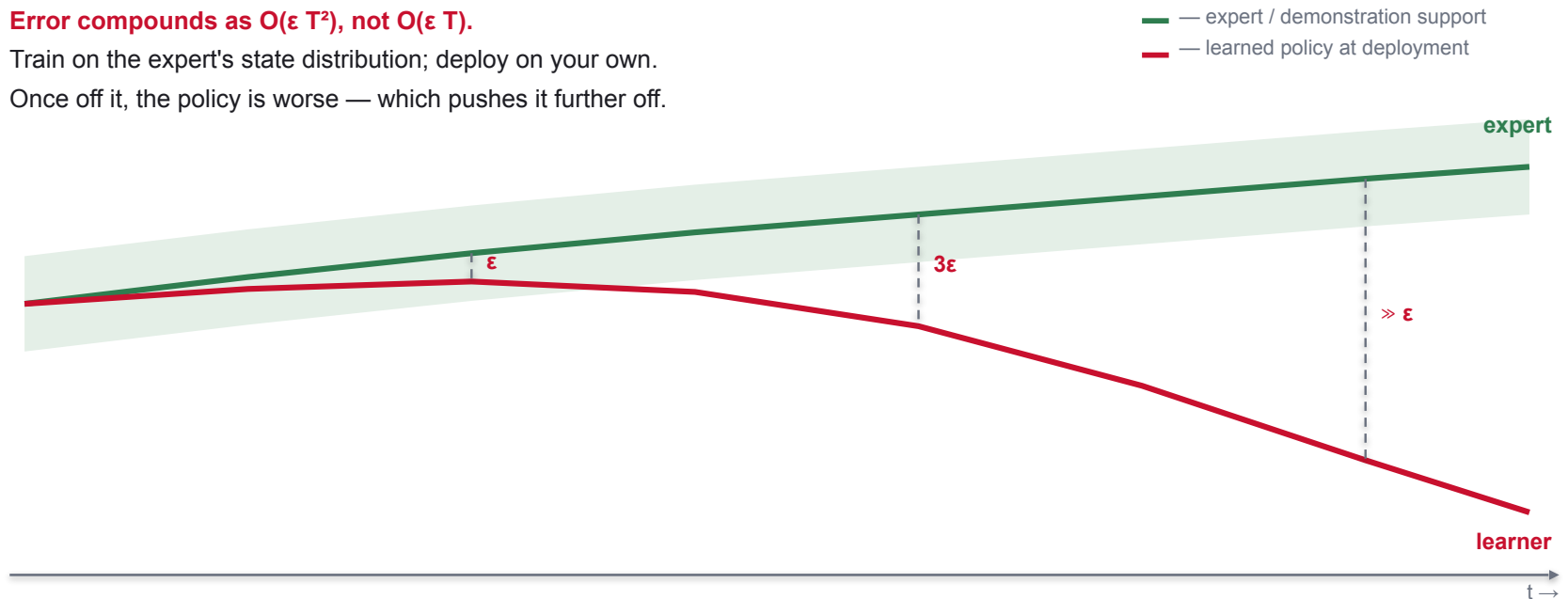
$$\mathbb{E}[\text{cost}] = \mathcal{O}(\varepsilon T^2) \quad \text{not} \quad \mathcal{O}(\varepsilon T)$$

Compounding error in behaviour cloning

Error compounds as $O(\epsilon T^2)$, not $O(\epsilon T)$.

Train on the expert's state distribution; deploy on your own.

Once off it, the policy is worse — which pushes it further off.



DAgger fixes this by querying the expert on the states the learner visits. Action chunking mitigates it by cutting the number of decision points.

Where demonstrations come from

Collection modalities

- **Teleoperation** — VR controllers, leader-follower arms (ALOHA), 3D mice; highest fidelity, lowest throughput
- **Kinesthetic teaching** — physically guide the arm; easy, no force data, awkward for dynamic tasks
- **Handheld grippers (UMI-style)** — collect in the wild without a robot; needs careful camera/pose calibration
- **Human video** — enormous and free, but no action labels and a different embodiment
- **Play data** — unstructured interaction, relabelled in hindsight

The gaps you inherit

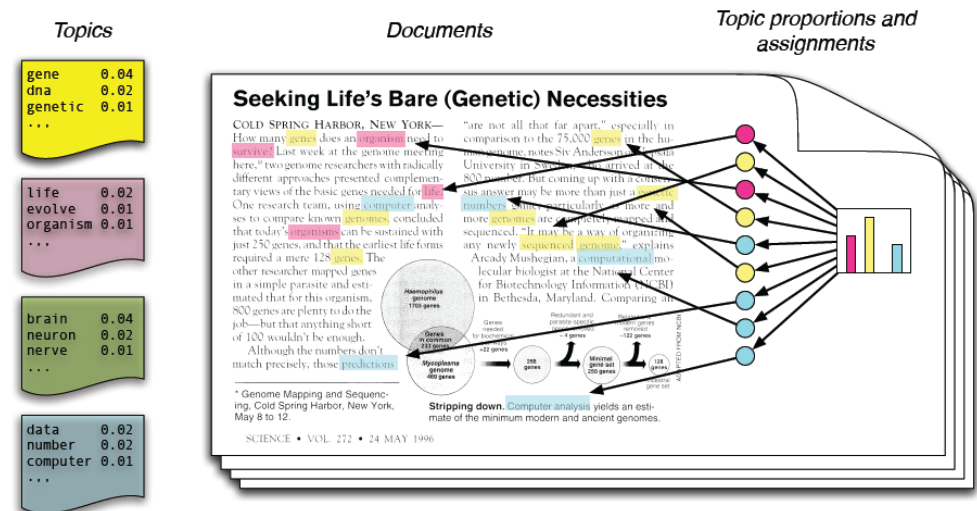
- **Embodiment gap** — human hands are not parallel grippers
- **Observation gap** — the demonstrator sees the scene directly; the policy sees two 224×224 crops
- **Causal confusion** — the policy latches onto a spurious cue (the demonstrator's hand, a clock, the previous action) that predicts the action without causing it

Diagnostic worth teaching

- If removing an input improves performance, you had causal confusion, not a feature.

Unsupervised learning

- Unsupervised Learning
 - Have access to unlabeled data x
 - Learn “hidden” structure of the data. For example:
 - learn distribution of data $p(x)$ (density estimation)
 - learn factors explaining the data
 - cluster analysis
 - dimensionality reduction



David Blei, 2011

Other Paradigms

- Semi-supervised Learning
 - Have access to many unlabeled data x , and only a few labeled data x'

Other Paradigms

- Imitation Learning
 - Expert/teacher/demonstrator shows good actions/controllers
 - Agent needs to optimize its actions without veering too far off the expert's trajectories.

Other Paradigms

- Imitation Learning
 - Expert/teacher/demonstrator shows good actions/controllers
 - Agent needs to optimize its actions without veering too far off the expert's trajectories.
- Transfer Learning
 - Policies learnt in one domain can be morphed and applied to another domain.
 - E.g. learn policies in a simulator and apply them in the real-world after some refinement/transformation

Other Paradigms: RL

- Reinforcement Learning
 - Agent interacts with environment by applying actions
 - Environment responds by rewarding or punishing the agent
 - Agent has to optimize actions in order to maximize expected cumulative rewards (minimize punishment) in the future.

Different than supervised learning because the agent's actions potentially change the distribution of the data being observed. Also, in RL, time-consecutive observations can be dependent.

Reinforcement learning ...

- A general framework for inferring **optimal behavior for an AI agent from experience** with the environment.
- Captures of notion of **benefit that accrues over time**, and where the rewards may be deferred.
- Responsible for many of the “biggest wins” on AI.
- In the simplest case, considers discrete state spaces where the states -- things we can do, and rewards -- can all be expressed in a big table.
- Can be generalized in numerous ways, but the generalizations are successively more heuristic.

Reinforcement learning, when you can afford it

The settings

- **Online RL** — needs a simulator or a very patient, very robust robot; standard for locomotion, rare for manipulation
- **Offline RL** — learn from a fixed dataset; the killer problem is value overestimation on out-of-distribution actions, addressed by conservatism (CQL — conservative Q-learning) or by implicit methods (IQL)
- **Offline-to-online** — pretrain offline, fine-tune with a small amount of real interaction; the pragmatic middle, and increasingly used to polish VLAs
- **RL on top of a BC policy** — residual policies and RL fine-tuning of a pretrained VLA

Reward is the hard part

- Hand-designed dense rewards are the classic source of reward hacking
- Sparse task-completion reward is unbiased and nearly unlearnable without demonstrations
- Learned rewards: inverse RL, preference learning (RLHF-style) from human comparisons
- **VLM-as-reward** — ask a vision-language model whether the goal was achieved; cheap, scalable, and exploitable exactly where it is wrong

The trap

- A learned reward is an adversarial target. The policy optimises the reward model's errors.

Sim-to-real: what transfers

- **The reality gap has three parts** — dynamics (friction, contact, actuator lag), perception (rendering, lighting, textures), and task distribution (which scenes and objects exist at all).
- **Domain randomisation** — randomise physics and appearance until reality is one sample from the training distribution. Cheap and remarkably effective for locomotion; weaker for contact-rich manipulation.
- **System identification / real-to-sim** — measure the real robot and fit the simulator to it. More sample-efficient, less general.
- **Photorealistic real2sim** — reconstruct actual scenes with 3D Gaussian splatting or NeRF and train or evaluate in them.
- **Synthetic data from generative models** — video and world models as data engines (M6); physically-plausible augmentation of existing demonstrations.

The uncomfortable empirical fact

— Simulation success and real success correlate weakly for manipulation. Benchmarks such as SimplerEnv exist specifically to improve that correlation — and their existence is the admission.

What kind of generalisation are you claiming?

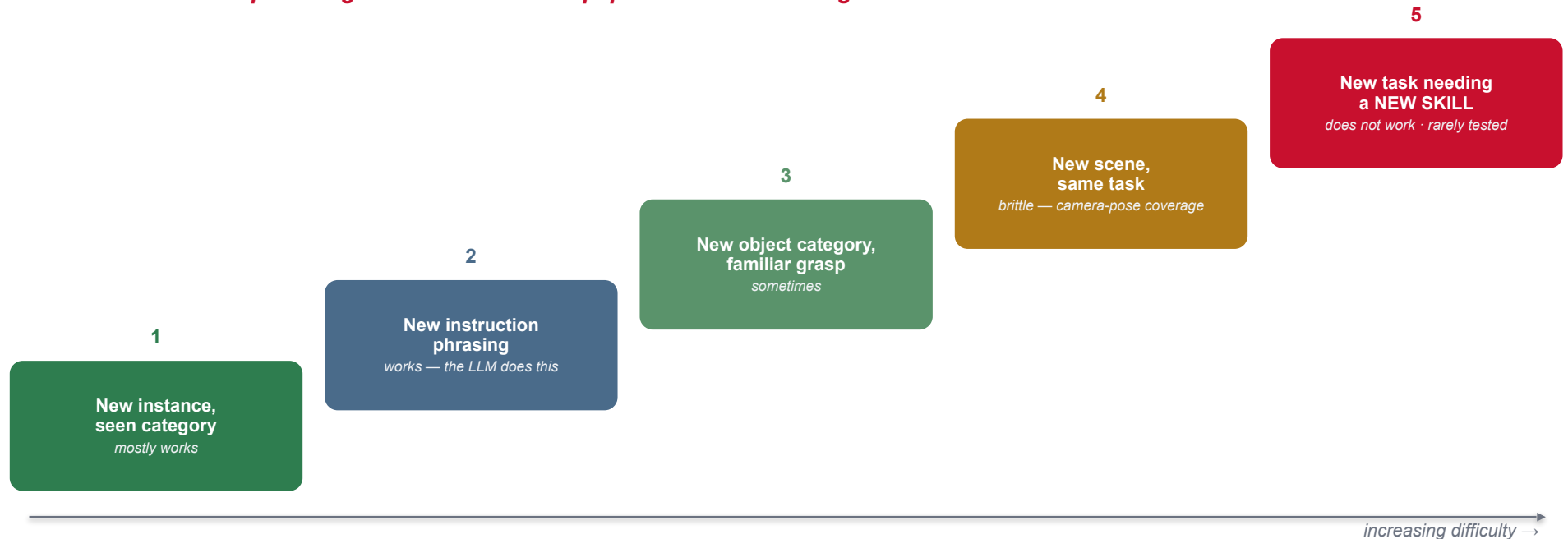
- **New instance of a seen object category** — mostly works.
- New object category with a familiar shape/grasp — sometimes works, driven by the pretrained vision backbone rather than the robot data.
- **New scene, same task** — works if the camera pose distribution was covered; fails abruptly if not.
- **New instruction phrasing** — works, courtesy of the language model. Frequently reported as if it were behavioural generalisation.
- **New task requiring a new skill** — rarely works and is rarely tested.

Rubric for reading results tables

- For every 'generalisation' number, identify which of these five it is. Most papers report the first and fourth, and title the paper after the fifth.

Which generalisation are you claiming?

Most results tables report rungs 1 and 2 — and the paper is titled after rung 5.

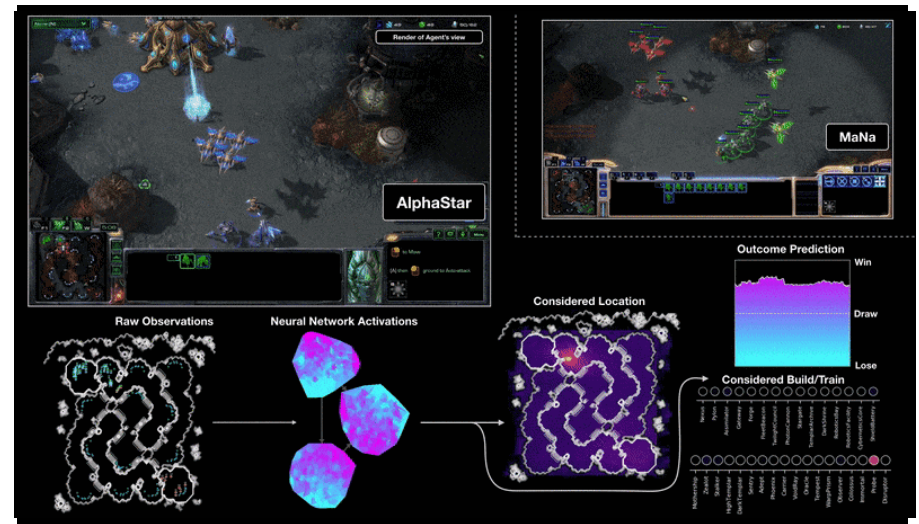
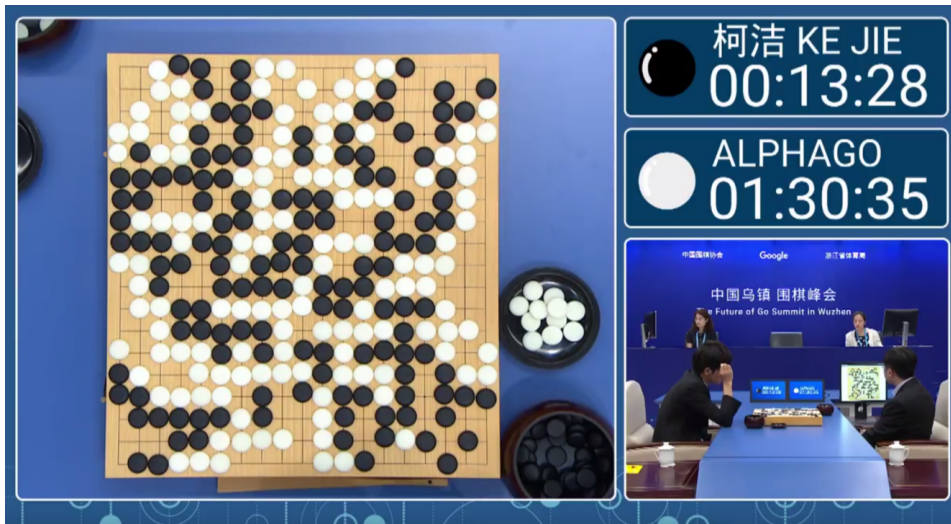


LIBERO's four suites (Spatial · Object · Goal · Long) are a partial operationalisation of this ladder — and recent audits find even those splits are partly shortcut-solvable.

Reinforcement Learning use cases

- **Control/robotics:** high-dimensional control problems. Google reported cutting its energy consumption by 50% using Deep Mind's RL technologies.
- **Health:** epilepsy treatment, medication, treatment planning all have a feedback control “flavor” have have been addressed using RL.
- **Finance:** optimization of trades is a natural fit for RL technologies, with major companies such as JPMorgan saying they are investing in “trading robots”.
- **Text mining:** RL technologies are used for summarization, with Salesforce and ServiceNow investing heavily in text-processing RL methods.

RL agents can outperform humans!



AlphaGo 2016, predecessor of AlphaStar

Try the film "ALPHAGO" on Netflix! <https://www.netflix.com/title/80190844>

Also Atari game playing, IBM Watson's wagering strategy, and both Go playing programs AlphaGo and AlphaGo Zero

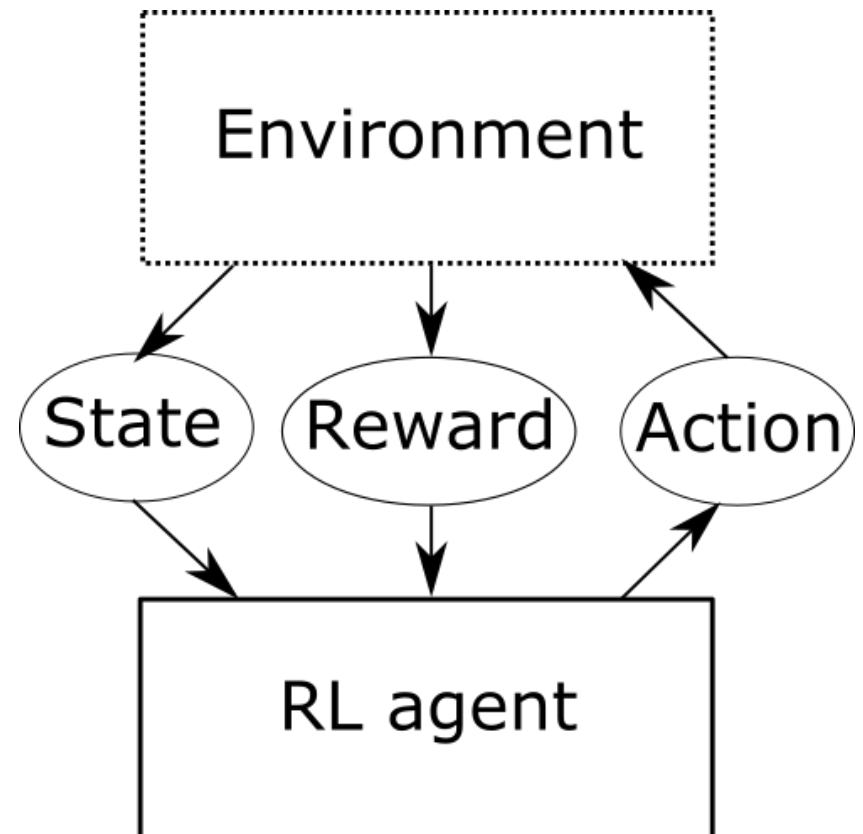
Reinforcement Learning

Oh, and by the way, we cannot pragmatically compute this analytically!

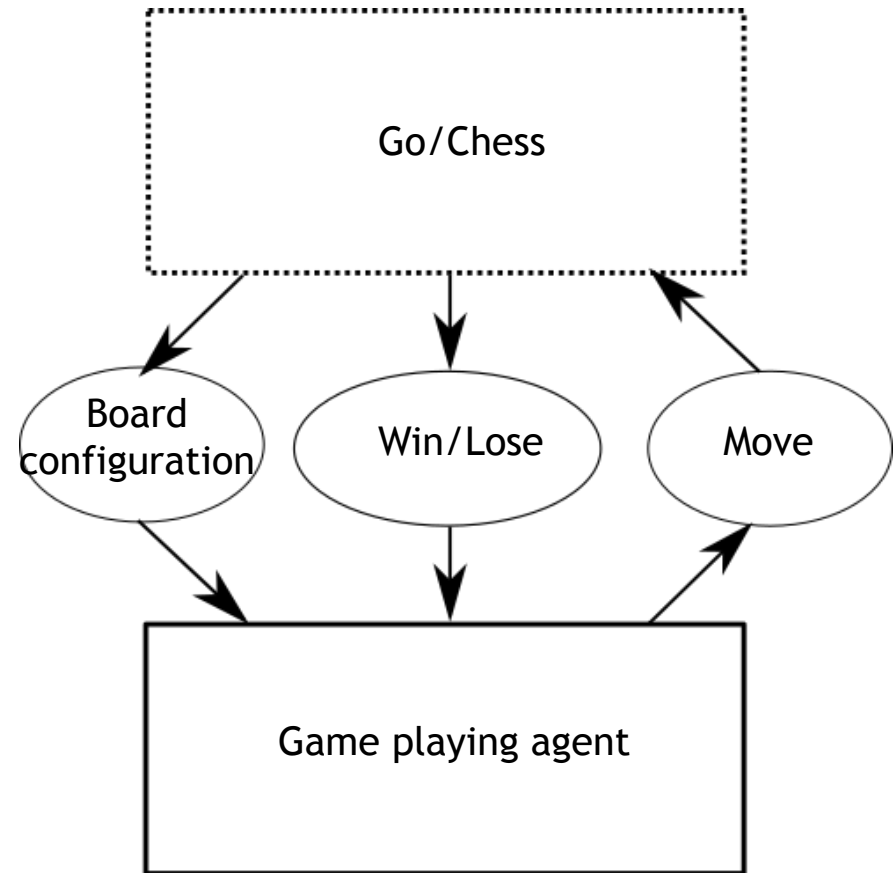
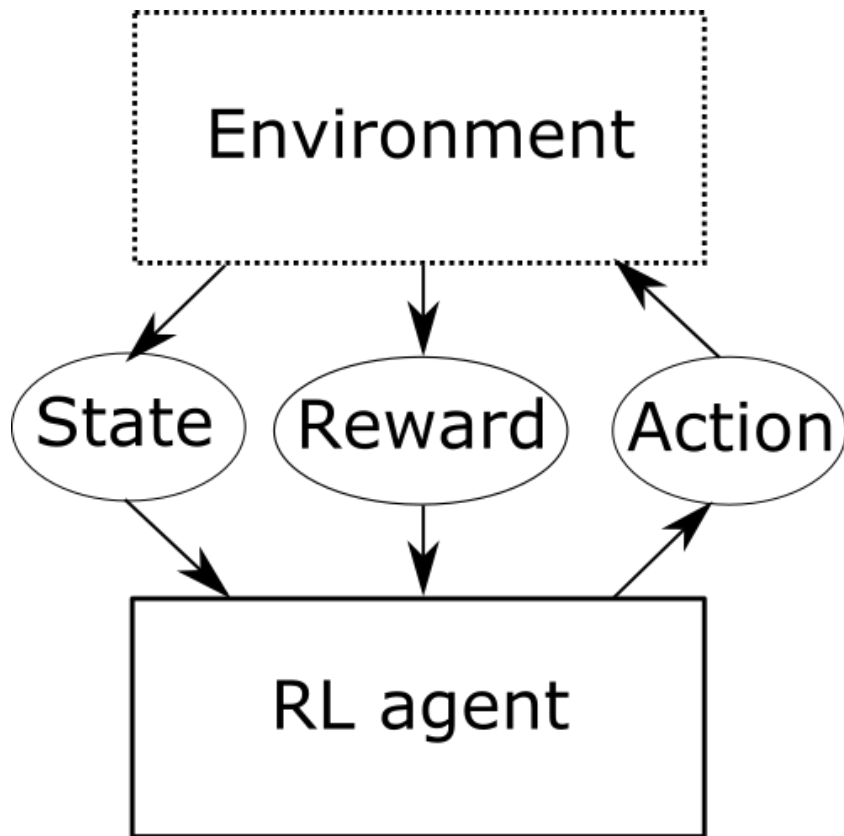
- RL is learning via *successive approximation (or trial-and-error, or other metaphors)*
- Instead of **presupposing equations** for how “things work”, we assume the environment can give us these:
 - What is the state-by-state reward for winning ping pong?
 - This is difficult to write analytically, but we can easily check the scoreboard during the game
 - What is the state transition for a robot swimming with flippers?
 - We can guess it or compute complex fluid dynamics, but we can more easily just observe its motions over time and use Machine Learning to derive a model
- We will switch from guaranteed, analytic reasoning to data-driven methods. Modeled on human developmental learning.
 - Historically, resisted by people who like classical clean methods.
 - They have been swamped by indisputably amazing results.

The reinforcement learning problem

- An agent interacts with the environment using discrete actions (continuous actions space is a generalization)
- State changes produce a "reward" that can be positive, negative or zero.
- Our goal is to get as much reward as possible over time.

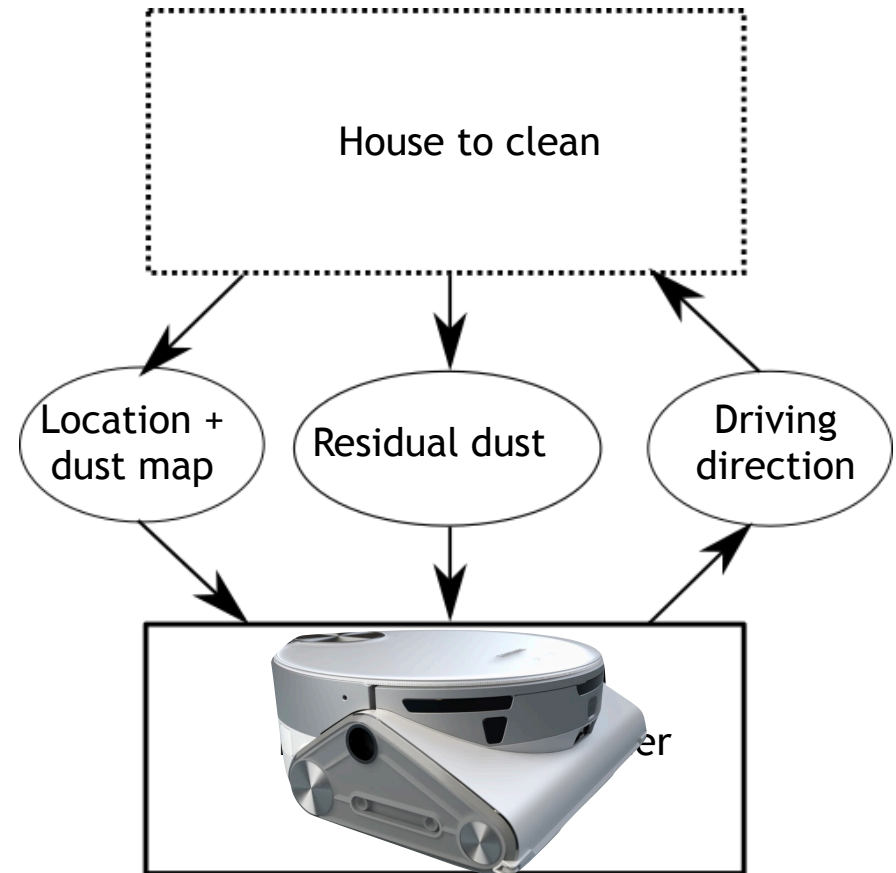


An example: chess



An example: the robot vacuum cleaner

- An agent interacts with the environment using discrete actions (continuous actions space is a generalization)
 - State changes produce a "reward" that can be positive, negative or zero.
 - Our goal is to get as much reward as possible over time.
- Warning: this example breaks some key assumptions, for example the discrete state space and stationarity!**



RL combines prediction and control

- In classical control, we use system inputs to drive the system to a "desired state":
 - minimizing
 - delay,
 - overshoot,
 - steady-state error
 - maximizing
 - stability
- Try to achieve "optimality".

Does PID also combine **prediction** and **control**

- One could say so.
- The PID expression can be viewed as a differential estimator for the error in the next time instant!
 - Think about the Taylor series, a linear estimator:
 $f(x+h) = f(x) + h \cdot df/dx$
 - We don't estimate the next state, but the next error.
 - or, *we estimate the next control input based on the estimate of the next state.*
- The error estimate directly leads to a control input

Traditional methods for control

- Optimal control also gives us a good language for optimizing behavior:
 - E.g. LQ regulation gives guarantees of optimal performance under very strict assumptions
 - Extensions to extend this idea by linearizing, etc.
- Typically we solve optimal control constrained optimization assuming knowledge of both **dynamics**, f , and **cost**, g .
 - **Are these known?!**
 - If yes, why and how?**
 - If not, why not?**

$$\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}, \mathbf{u})$$

$$g(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{x}_t^T Q \mathbf{x}_t + \mathbf{u}_t^T R \mathbf{u}_t$$

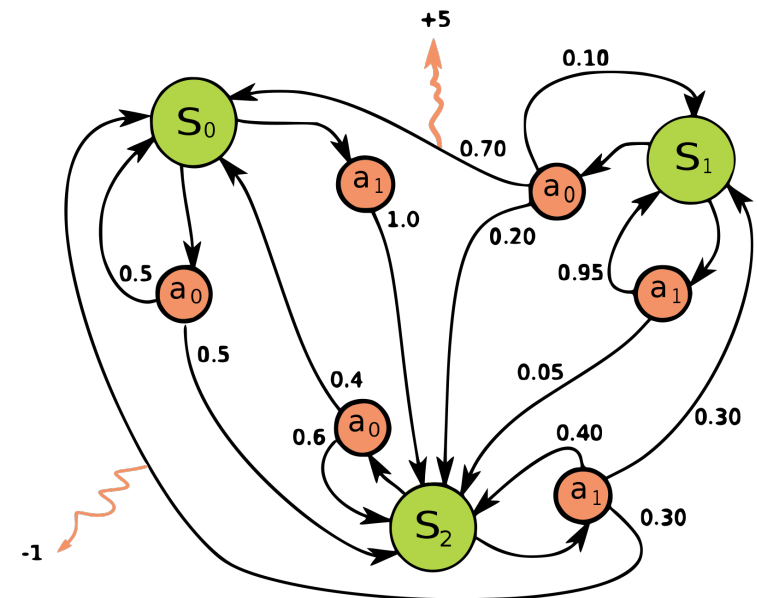
$$\operatorname{argmin}_{u_0, \dots, u_N} \sum_{t=0}^{t=N} g(\mathbf{x}_t, \mathbf{u}_t)$$

One might argue no, since real systems are imperfect, interact with an ever-changing world, and even wear down (and thus change themselves).

Definitions: Markov Decision Process (MDP)

- A state space: $\vec{s} \in \mathcal{S}$
- An action space: $\vec{a} \in \mathcal{A}$
- A transition model: $s_{t+1} = T(s_t, a_t)$
- A scalar **reward** function: $r(\vec{s}, \vec{a})$
- A behavior **policy**: $\vec{a} = \pi(\vec{s})$ with optimal policy π^*

(Not quite complete, lacks discount factor)



And MDP with 3 states (green), 2 non-deterministic actions (brown), and 2 rewards (wavy orange lines)

Definitions: Markov Decision Process (MDP)

- A state space: $\vec{s} \in \mathcal{S}$
- An action space: $\vec{a} \in \mathcal{A}$
- A transition model: $\vec{s}_{t+1} = T(\vec{s}_t, \vec{a}_t)$
- A scalar **reward** function: $r(\vec{s}, \vec{a})$
- A behavior **policy**: $\vec{a} = \pi(\vec{s})$ with optimal policy π^*

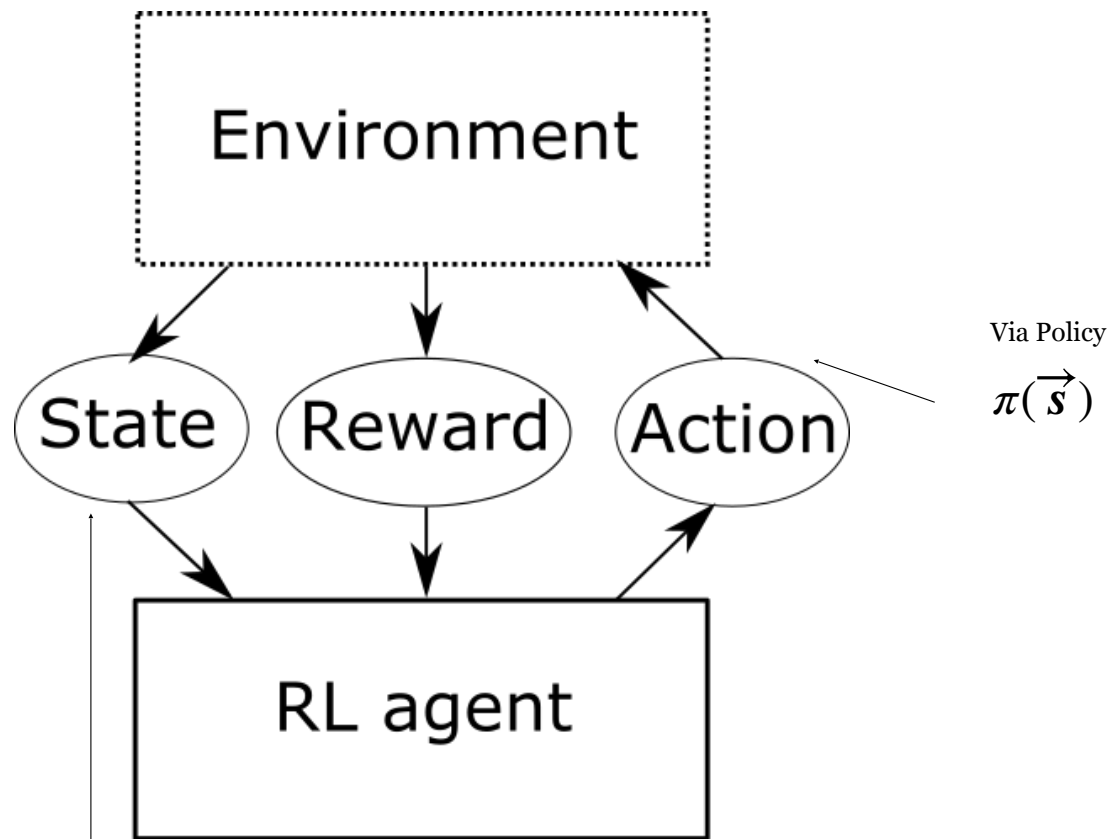
These are identical to the states (x), controls (u), motion model $f(x,u)$ and cost $g(x,u)$ seen in control theory, but AI people reinvented some ideas and changed the notation

Transition model $\sim$ "dynamics" or "rules of the game"

We accumulate reward over time (from state to state)

(Not quite complete, lacks discount factor)

Reinforcement Learning

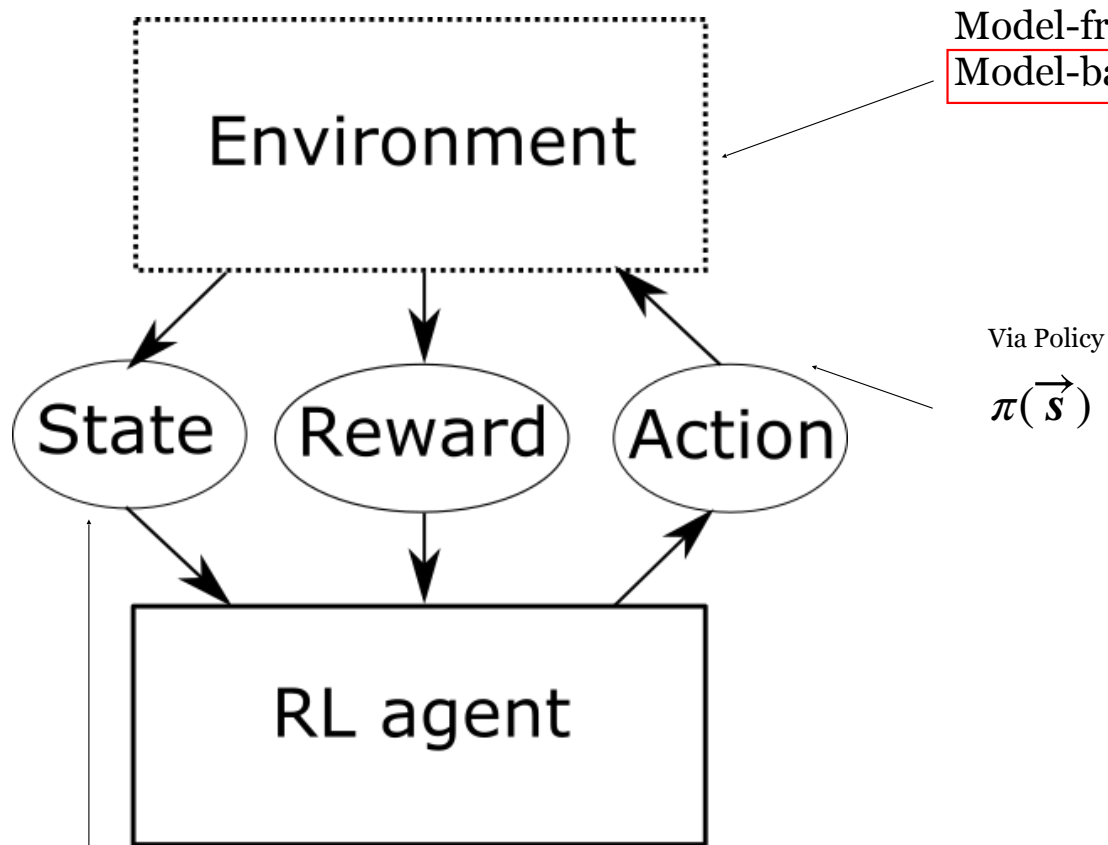


Agent's state is observed directly at each time step.

The policy is often assumed to be determined by some **set** of parameters θ

What these are depends on how the policy is formulated.

Reinforcement Learning



Model-free RL: reward and dynamics are unknown
Model-based RL: reward and dynamics are known

Very useful for driving, robotics, games in terms of efficient learning.

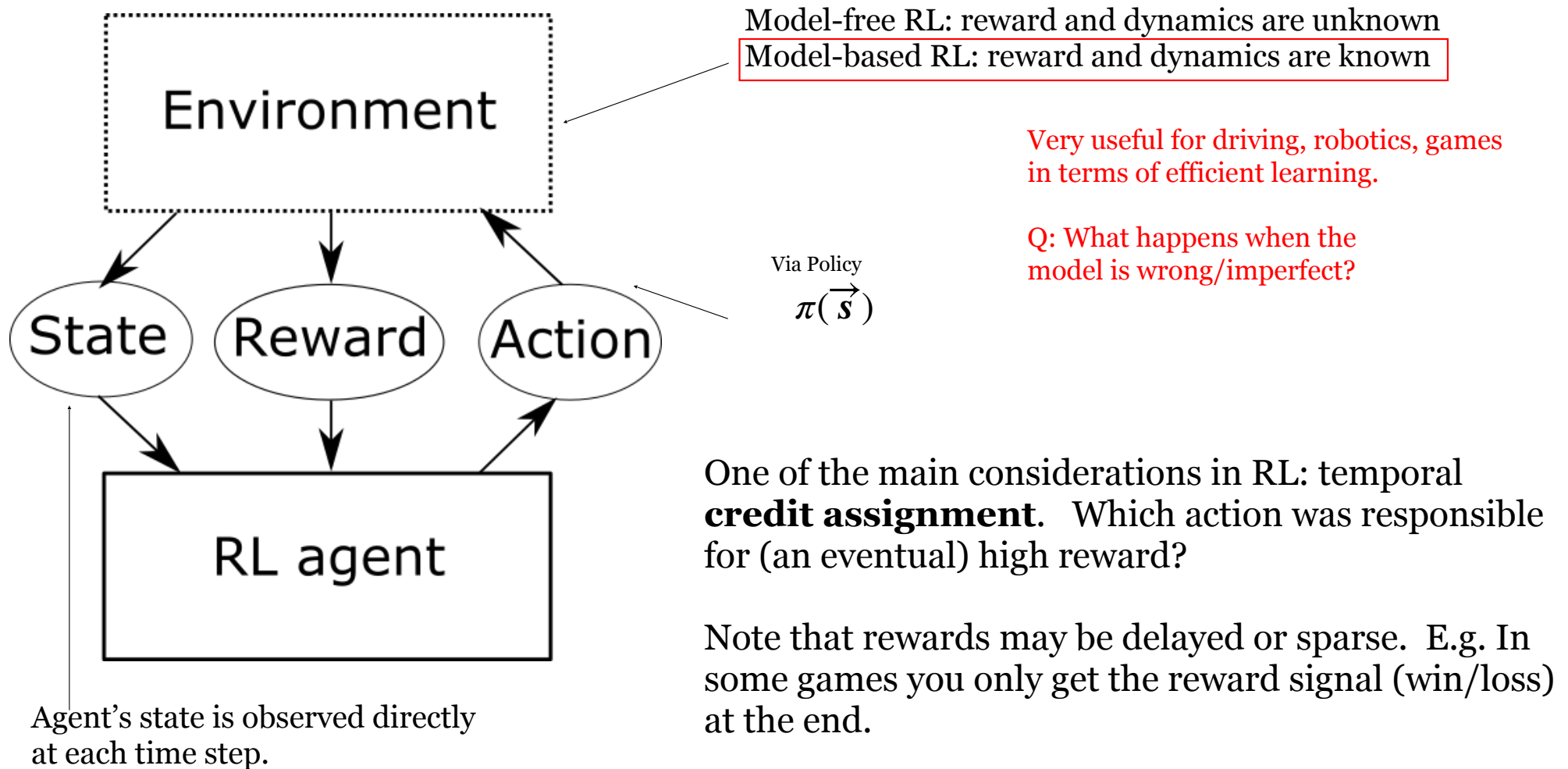
Q: What happens when the model is wrong/imperfect?

The policy is assumed to be determined by some **set** of parameters θ

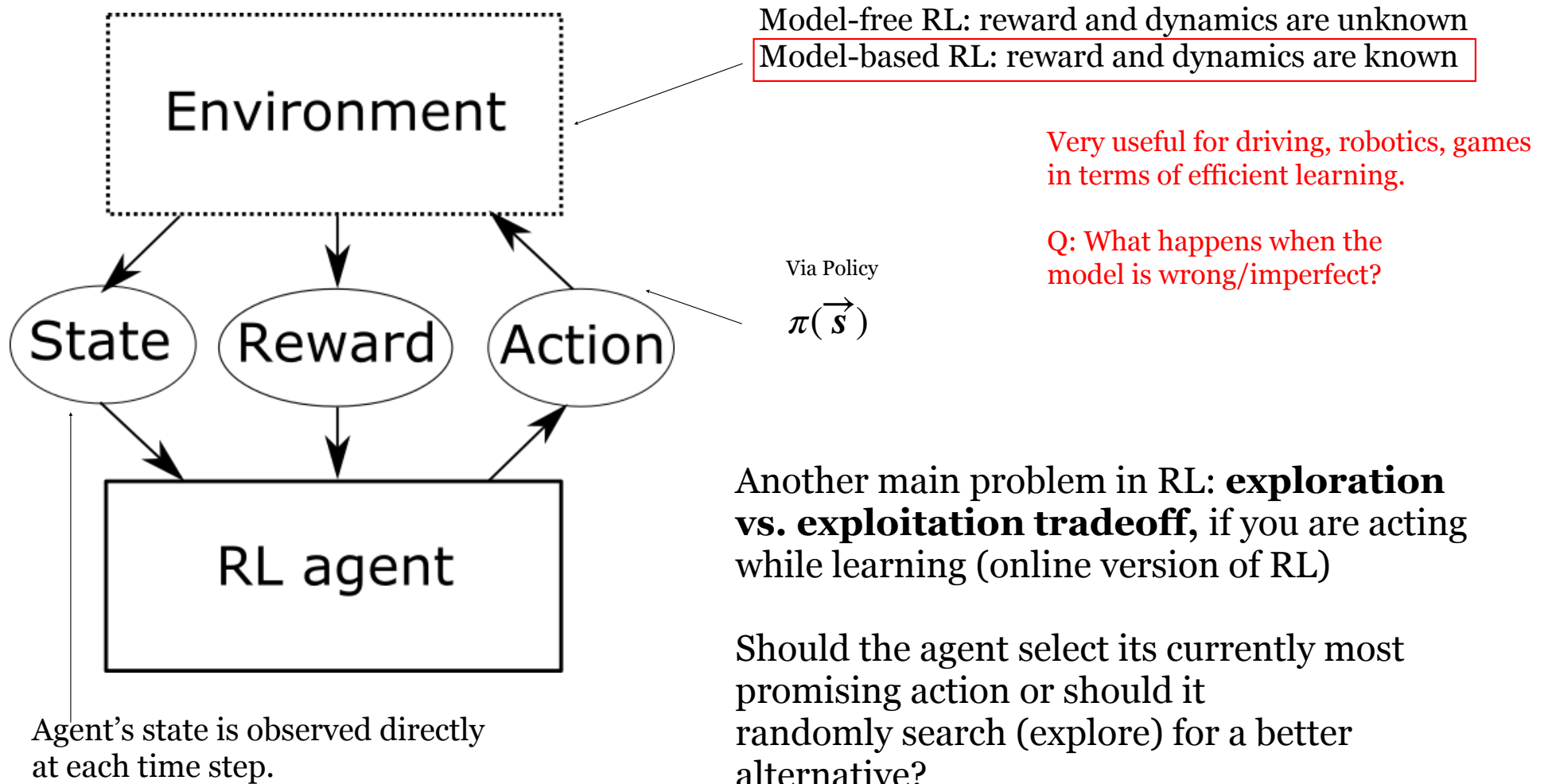
What these are depends on how the policy is formulated.

Agent's state is observed directly at each time step.

Reinforcement Learning



Reinforcement Learning



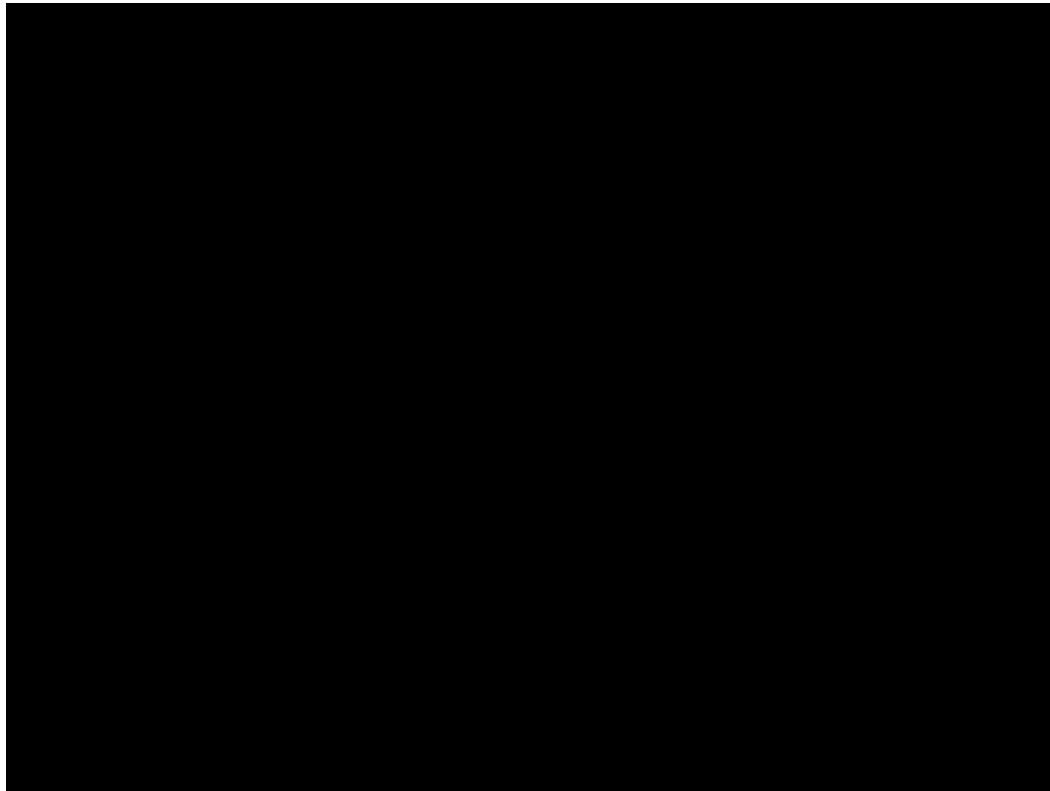
Examples

- Playing a **game**: the system has **perfect information about the states and action, the current state, and the result of transitions** (deterministic).
 - Playing video games can add a need to use an image to compute the state, but it's often simple.
 - Some games have a large state space (e.g. go).
- Swinging an inverted pendulum: state is low dimensional, and can be measured accurately.
 - Physical systems usually make the effect of actions somewhat uncertain



Deisenroth, Fox, Rasmussen, 2011

Breakout, Atari



DQN, Volodymyr Mnih et al., 2013

Total reward

- Given some policy, we want to determine how much total reward we are likely to accumulate
 - Use the sum of rewards
 - Use the **weighted** sum of rewards
- These ideas go back to Ivan Pavlov (1890's)

Revisiting the formalism

Definitions: Markov Decision Process (MDP)

- A state space: $\vec{s} \in S$
- An action space: $\vec{a} \in A$
- A {probabilistic,deterministic}
transition model: $p(\vec{s}' \mid \vec{s}, \vec{a})$
- A scalar reward function: $r(\vec{s}, \vec{a})$
- A behavior policy: $\vec{a} = \pi(\vec{s})$ with optimal policy π^*
- Discount factor $\gamma \in [0,1]$ that expresses that a reward tomorrow is less valuable than a reward today
- Distinction between reward and value of a state on the way to something better!

Total reward:

$$\sum_t \gamma^t r(s_t, a_t)$$

For a reward 10 steps away, the discount is compounded 10 times.

Two major problem variants:

- Known transitions, but uncertain probabilities (and/or rewards).
- Unknown transitions.

Markov Decision Processes (MDP)

Frameworks for optimal decision-making under uncertainty

Discrete states x_t
Discrete actions u_t

Uncertain dynamics $p(x_t|x_{t-1}, u_{t-1})$
State is directly observed at each step, without uncertainty

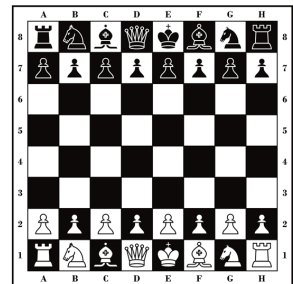
Agent receives instantaneous reward $R(x_t, u_t, x_{t+1}) = R_t$ after each transition.

Goal: find a policy $p(u_t|x_t) = \pi^*(u_t|x_t)$ that maximizes cumulative expected reward

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E} \left[\sum_{t=0}^H \gamma^t R_t \mid \pi \right]$$

Time horizon for optimization.
Can be infinite.

Discount factor in $[0,1)$ specifies
later rewards are less valued than
earlier ones.



Markov Decision Processes (MDP)

Frameworks for optimal decision-making under uncertainty

Discrete states x_t
Discrete actions u_t

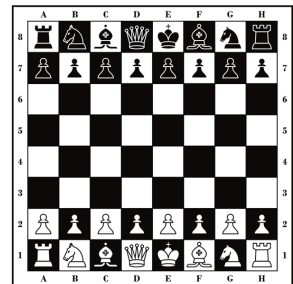
Uncertain dynamics $p(x_t|x_{t-1}, u_{t-1})$
State is directly observed at each step, without uncertainty

Agent receives instantaneous reward $R(x_t, u_t, x_{t+1}) = R_t$ after each transition.

Goal: find a policy $p(u_t|x_t) = \pi^*(u_t|x_t)$ that maximizes cumulative expected reward

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E} \left[\sum_{t=0}^H \gamma^t R_t \mid \pi \right]$$

With respect to future states



Partially-Observable Markov Decision Processes (POMDP)

Frameworks for optimal decision-making under uncertainty

Like MDP, but state is not directly observed. Instead, observation model $p(z_t|x_t)$

Agent knows action-observation history $h_t = (u_0, z_0, u_1, z_1, \dots, u_t, z_t)$

Maintains and updates belief $b_t(x_t|h_t)$ using Bayes' filter

Receives instantaneous reward $R(x_t, u_t, x_{t+1}) = R_t$ after each transition, but when planning it can only weigh the reward by the probability of being at a state (since it does not observe it), so we define

$$r_t(h_t, u_t) = \sum_{x_t} b_t(x_t|h_t) R_t$$



Partially-Observable Markov Decision Processes (POMDP)

Frameworks for optimal decision-making under uncertainty

Like MDP, but state is not directly observed. Instead, observation model $p(z_t|x_t)$

Agent knows action-observation history $h_t = (u_0, z_0, u_1, z_1, \dots, u_t, z_t)$

Maintains and updates belief $b_t(x_t|h_t)$ using Bayes' filter

Receives instantaneous reward $R(x_t, u_t, x_{t+1}) = R_t$ after each transition, but when planning it can only weigh the reward by the probability of being at a state (since it does not observe it), so we define

$$r_t(h_t, u_t) = \sum_{x_t} b_t(x_t|h_t) R_t$$

Goal: find a policy $p(u_t|h_t) = \pi^*(u_t|h_t)$ that maximizes cumulative expected reward

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E} \left[\sum_{t=0}^H \gamma^t r_t \mid \pi, b_0 \right]$$



Partially-Observable Markov Decision Processes (POMDP)

Frameworks for optimal decision-making under uncertainty

Like MDP, but state is not directly observed. Instead, observation model $p(z_t|x_t)$

Agent knows action-observation history $h_t = (u_0, z_0, u_1, z_1, \dots, u_t, z_t)$

Maintains and updates belief $b_t(x_t|h_t)$ using Bayes' filter

Receives instantaneous reward $R(x_t, u_t, x_{t+1}) = R_t$ after each transition, but when planning it can only weigh the reward by the probability of being at a state (since it does not observe it), so we define

$$r_t(h_t, u_t) = \sum_{x_t} b_t(x_t|h_t) R_t$$

Goal: find a policy $p(u_t|h_t) = \pi^*(u_t|h_t)$ that maximizes cumulative expected reward

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E} \left[\sum_{t=0}^H \gamma^t r_t \mid \pi, b_0 \right]$$

With respect to future states
and future observations



Decomposition by Value Functions

- We eventually want to solve the control over all history.

$$J(\pi) = \mathbb{E}_{s_0 \sim p(s_0)} \sum_{t=0:\infty} \gamma^t r(s_t, a_t)$$

Total expected discounted reward as a function of the action-selection policy being used

- We can break this up to divide-and-conquer

$$V^\pi(s) = r(s_t, \pi(s_t)) + \mathbb{E}_{p(s_{t+1} | s_t, \pi(s_t))} \left[\sum_{k=1:\infty} \gamma^k r(s_{t+k}, a_{t+k}) \right]$$

$$V^\pi(s) = r(s_t, \pi(s_t)) + \gamma \mathbb{E}_{p(s_{t+1} | s_t, \pi(s_t))} [V^\pi(s_{t+1})]$$

Bellman Equation

- Remember our overall goal is to find the **best policy**,
 $\pi^* = \operatorname{argmax}_{\pi} J(\pi)$
- The best policy at every state maximizes the value forever, which must also optimize the value at the next state:

$$V^*(s) = \operatorname{argmax}_a r(s_t, a_t) + \gamma \mathbb{E}_{p(s_{t+1} | s_t, a_t)} [V^*(s_{t+1})]$$

For all $s_t \in S$

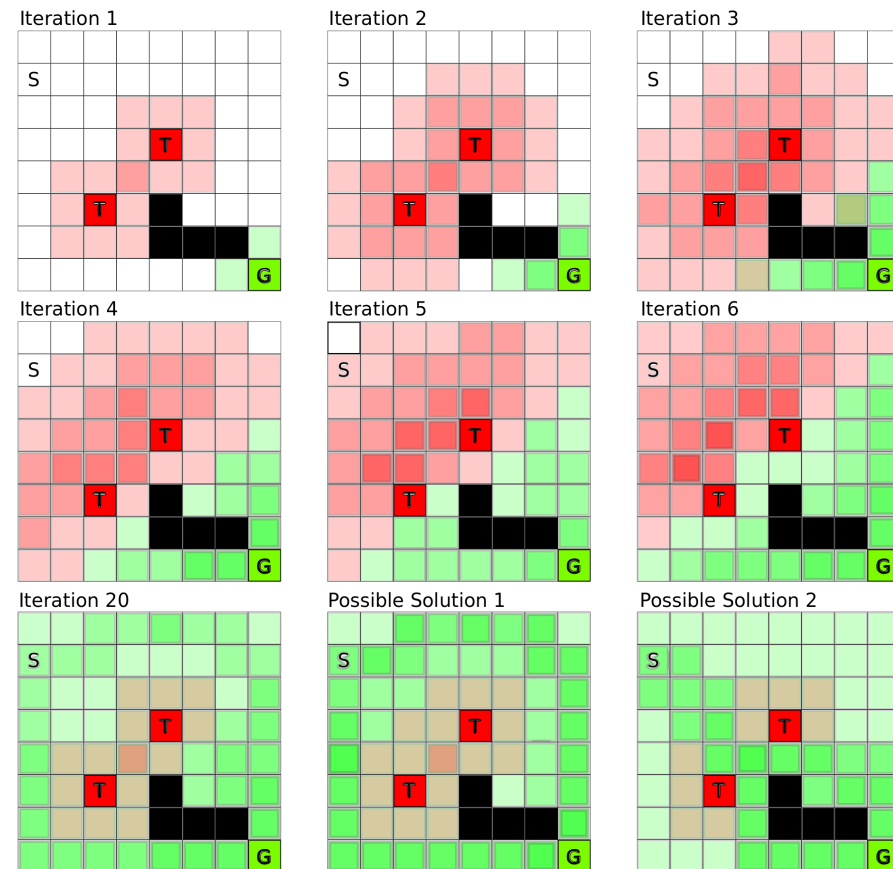
With For all $s \in S$, with $s_t = s$

This is a Dynamic Programming recursion. It was Bellman's objective when he invented the concept!

Hence it's called the "the Bellman Equation(s)".

Value Iteration

- The DP recursion leads directly to an algorithm! Keep a table of “value”: expected discounted reward from here on
- Value Iteration is just dynamic programming that uses *neighboring value estimate* to capture “everything after the next step”
- Assuming known transition probability and rewards.



Evolving value map from Start to Goal, with Traps.

Image credit: <https://devblogs.nvidia.com/deep-learning-nutshell-reinforcement-learning/>

Value Iteration

Initialize V arbitrarily, e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat

$\Delta \leftarrow 0$

For each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$

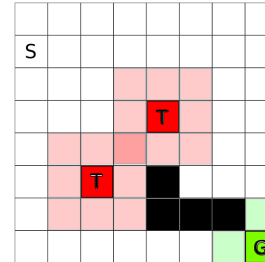
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

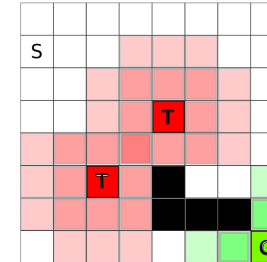
Output a deterministic policy, π , such that

$\pi(s) = \arg \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$

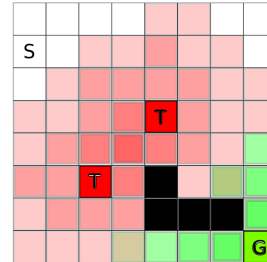
Iteration 1



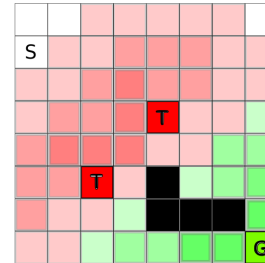
Iteration 2



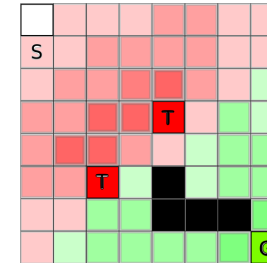
Iteration 3



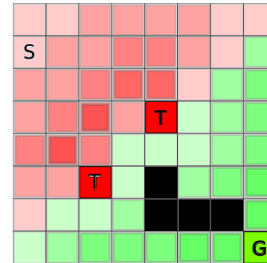
Iteration 4



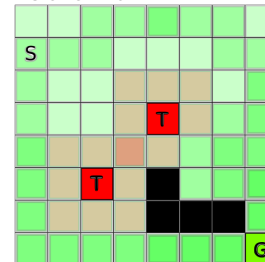
Iteration 5



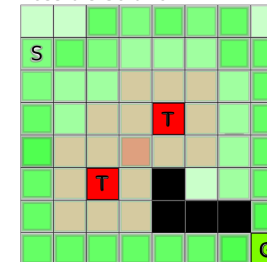
Iteration 6



Iteration 20



Possible Solution 1



Possible Solution 2

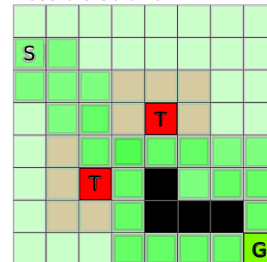


Image credit: <https://devblogs.nvidia.com/deep-learning-nutshell-reinforcement-learning/>

Value Iteration

- Consider that VI is very similar to path planning search
- Differences:
 - can handle probabilistic transitions
 - can handle negative rewards
 - needs discounting
- Proven to converge by “contraction”. Initially wrong $V(s)$ values get closer to correct in each round .

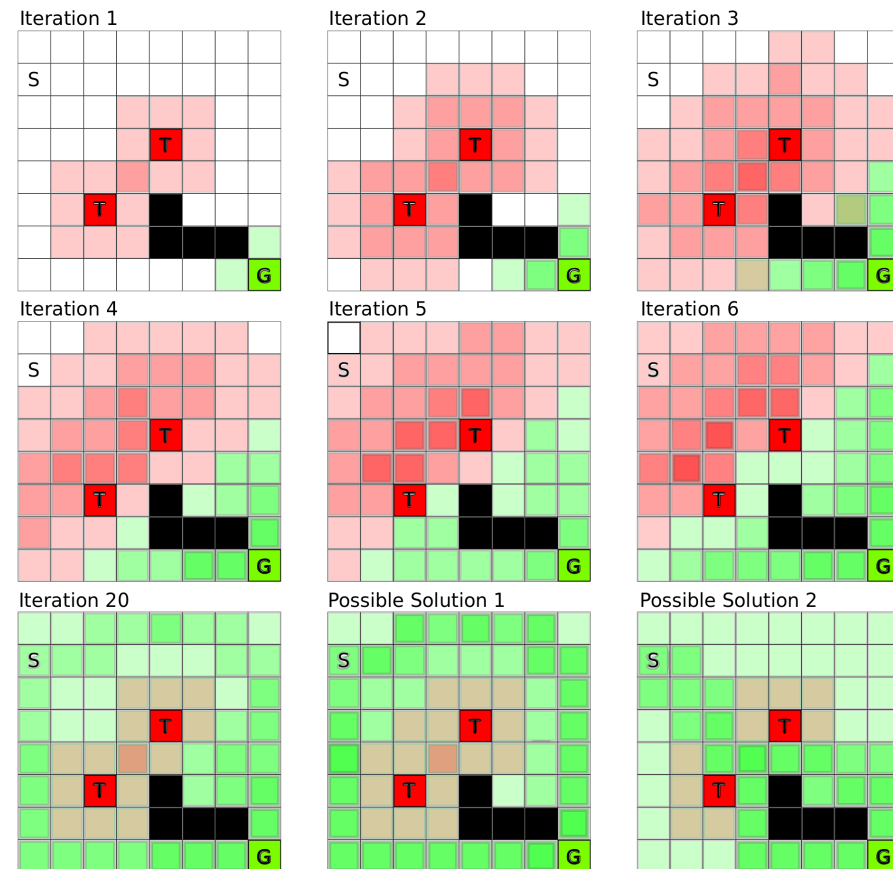


Image credit: <https://devblogs.nvidia.com/deep-learning-nutshell-reinforcement-learning/>

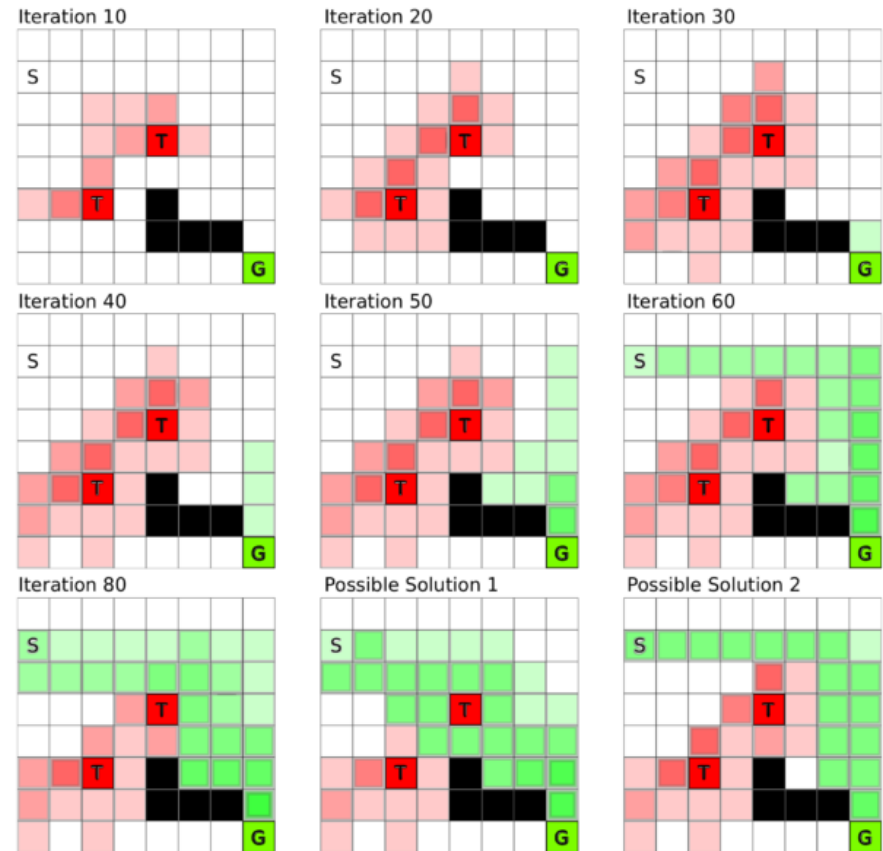
Q-Learning: Online RL

- If we don't know the models of transition and reward?
- We must experience the world through exploration:
 - s, a, r, s' : being somewhere, taking an action, and observing where you end up plus reward

- Use s, a, r, s' to update Q values

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

- Arrow denotes an adjustment to the current value and can be weighted by a constant other than 1.
- [Watkins 1989] proves convergence with infinite data



Full story:

Take action an A, observe R, S' then update

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

Scale Q-learning

- Recall: gather s, a, r, s' by running exploration, update Q values

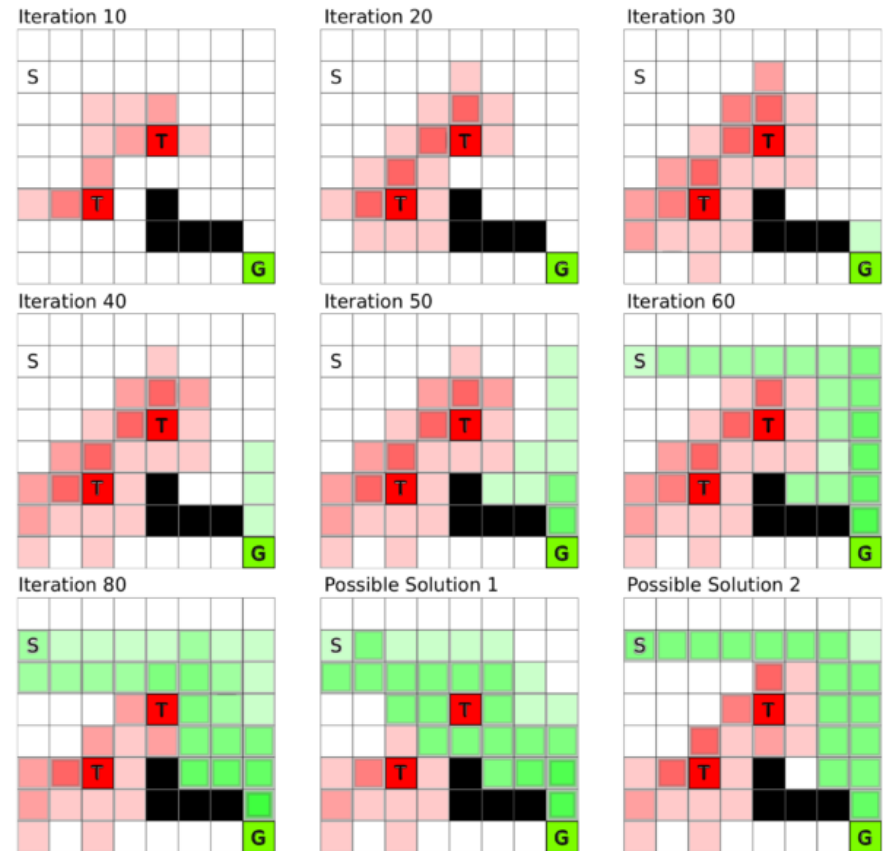
$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

measurable reward for next step

+

estimated utility for subsequent steps

- For the grid on the right, use a table/array to hold the Q values



Credit: <https://devblogs.nvidia.com/deep-learning-nutshell-reinforcement-learning/>

Table driven Q learning

- Store (state, action) outcomes in a table
 - Store rewards too, as they are measured.
- Update the table as we explore more of the state space (possibly over-and-over)
- One entry for each action in each state

Breakpoint

- More on RL
- Loose ends.
- Recap.
- Special cases
- Extensions.
- [IROS report](#).

Recall:

Markov Decision Process (MDP)

- A state space: $\vec{s} \in S$
- An action space: $\vec{a} \in A$
- A {probabilistic, deterministic} transition model: $p(\vec{s}' \mid \vec{s}, \vec{a})$
- A scalar **reward** function: $r(\vec{s}, \vec{a})$
- A behavior **policy**: $\vec{a} = \pi(\vec{s})$ with optimal policy π^*
- Discount factor $\gamma \in [0,1]$ that expresses that a reward tomorrow is less valuable than a reward today

Key ideas:

Markov property.

States, actions, policies.

SEEKING A POLICY TO MAXIMIZE

Total reward:

$$\sum_t \gamma^t r(s_t, a_t)$$

Major problem variants:

- Known transitions, but uncertain probabilities (and/or rewards).
- Unknown transitions.

What if we need to estimate our state and are not sure?

(When can this happen?)

Markov Decision Processes (MDP)

Frameworks for optimal decision-making under uncertainty

Discrete states x_t
Discrete actions u_t

Uncertain dynamics $p(x_t|x_{t-1}, u_{t-1})$
State is directly observed at each step, without uncertainty

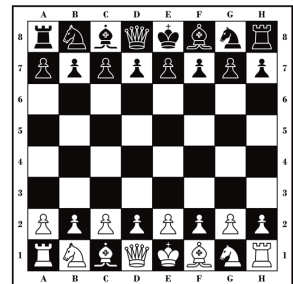
Agent receives instantaneous reward $R(x_t, u_t, x_{t+1}) = R_t$ after each transition.

Goal: find a policy $p(u_t|x_t) = \pi^*(u_t|x_t)$ that maximizes cumulative expected reward

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E} \left[\sum_{t=0}^H \gamma^t R_t \mid \pi \right]$$

Time horizon for optimization.
Can be infinite.

Discount factor in $[0,1)$ specifies
later rewards are less valued than
earlier ones.



Markov Decision Processes (MDP)

Frameworks for optimal decision-making under uncertainty

Discrete states x_t
Discrete actions u_t

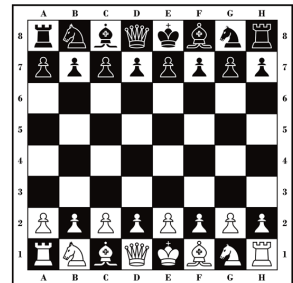
Uncertain dynamics $p(x_t|x_{t-1}, u_{t-1})$
State is directly observed at each step, without uncertainty

Agent receives instantaneous reward $R(x_t, u_t, x_{t+1}) = R_t$ after each transition.

Goal: find a policy $p(u_t|x_t) = \pi^*(u_t|x_t)$ that maximizes cumulative expected reward

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E} \left[\sum_{t=0}^H \gamma^t R_t \mid \pi \right]$$

With respect to future states



Partially-Observable Markov Decision Processes (POMDP)

Frameworks for optimal decision-making under uncertainty

Like MDP, but state is not directly observed. Instead, observation model $p(z_t|x_t)$

Agent knows action-observation history $h_t = (u_0, z_0, u_1, z_1, \dots, u_t, z_t)$

Maintains and updates belief $b_t(x_t|h_t)$ using Bayes' filter

Receives instantaneous reward $R(x_t, u_t, x_{t+1}) = R_t$ after each transition, but when planning it can only weigh the reward by the probability of being at a state (since it does not observe it), so we define

$$r_t(h_t, u_t) = \sum_{x_t} b_t(x_t|h_t) R_t$$



Partially-Observable Markov Decision Processes (POMDP)

Frameworks for optimal decision-making under uncertainty

Like MDP, but state is not directly observed. Instead, observation model $p(z_t|x_t)$

Agent knows action-observation history $h_t = (u_0, z_0, u_1, z_1, \dots, u_t, z_t)$

Maintains and updates belief $b_t(x_t|h_t)$ using Bayes' filter

Receives instantaneous reward $R(x_t, u_t, x_{t+1}) = R_t$ after each transition, but when planning it can only weigh the reward by the probability of being at a state (since it does not observe it), so we define

$$r_t(h_t, u_t) = \sum_{x_t} b_t(x_t|h_t) R_t$$

Goal: find a policy $p(u_t|h_t) = \pi^*(u_t|h_t)$ that maximizes cumulative expected reward

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E} \left[\sum_{t=0}^H \gamma^t r_t \mid \pi, b_0 \right]$$



Partially-Observable Markov Decision Processes (POMDP)

Frameworks for optimal decision-making under uncertainty

Like MDP, but state is not directly observed. Instead, observation model $p(z_t|x_t)$

Agent knows action-observation history $h_t = (u_0, z_0, u_1, z_1, \dots, u_t, z_t)$

Maintains and updates belief $b_t(x_t|h_t)$ using Bayes' filter

Receives instantaneous reward $R(x_t, u_t, x_{t+1}) = R_t$ after each transition, but when planning it can only weigh the reward by the probability of being at a state (since it does not observe it), so we define

$$r_t(h_t, u_t) = \sum_{x_t} b_t(x_t|h_t) R_t$$

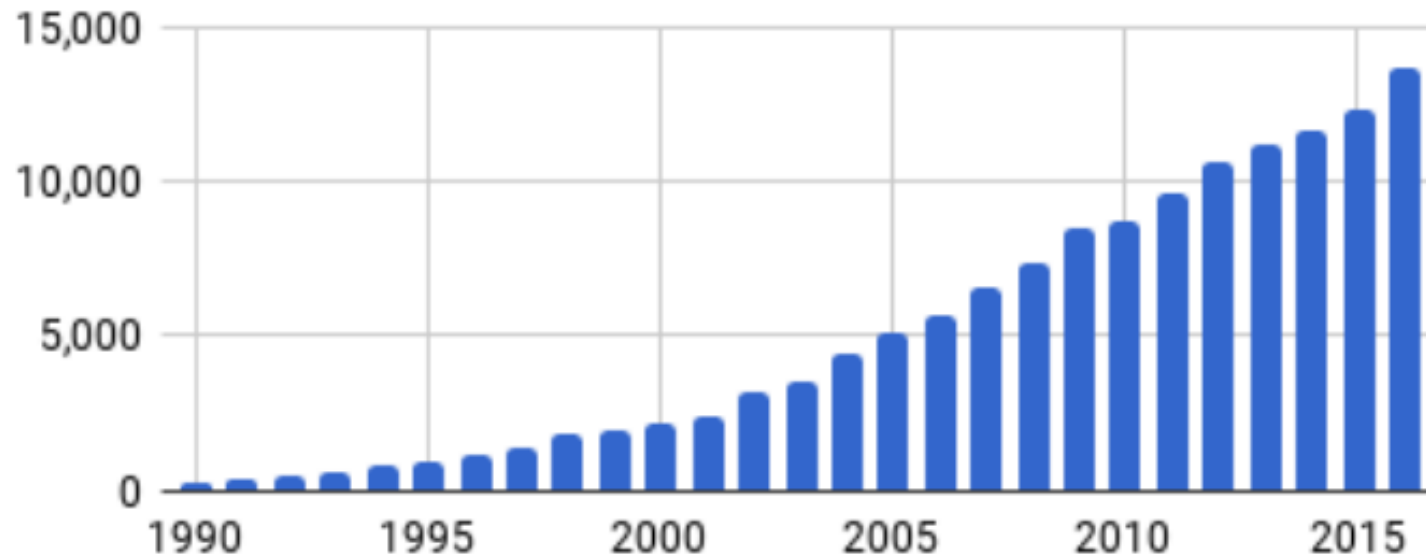
Goal: find a policy $p(u_t|h_t) = \pi^*(u_t|h_t)$ that maximizes cumulative expected reward

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E} \left[\sum_{t=0}^H \gamma^t r_t \mid \pi, b_0 \right]$$

With respect to future states
and future observations



Lots of progress: obviously some problems remain unsolved



Growth of published reinforcement learning papers. Shown are the number of RL-related publications (y-axis) per year (x-axis) scraped from Google Scholar searches.

Scaling RL to big (real) problems

- Store (state, action) outcomes in a table
 - The table can be very large if there are many states, actions, or if the actions space is continuous.
- A classic idea is to use an interpolation function to represent the mapping.
- What's the most capable large-scale general-purpose interpolation method today?
 - Deep networks!
 - Combine Deep Networks and Q-learning: deep-Q, DNQ

Deep Q Learning: The Engine behind Go, SC2, etc.

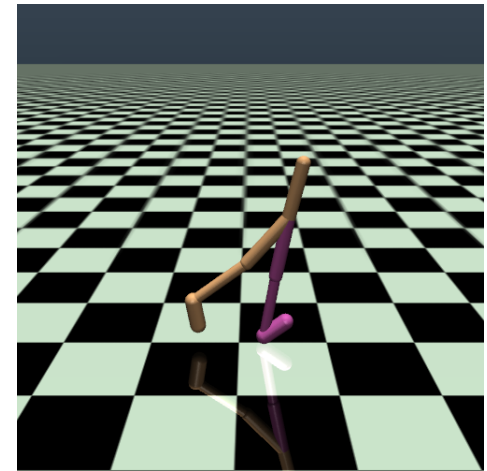
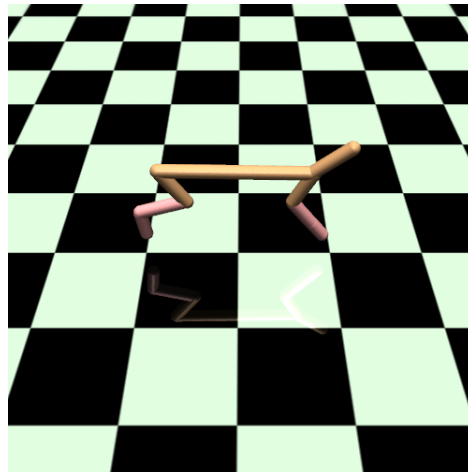
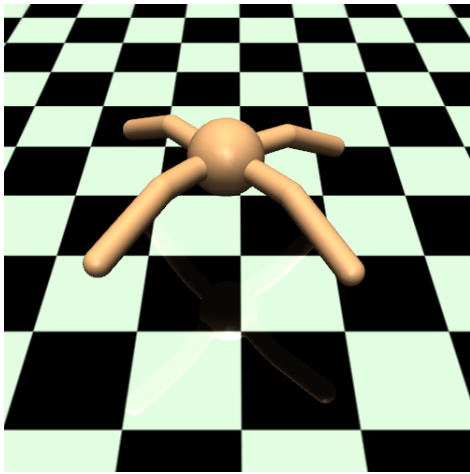
- We learn a deep neural network to predict the action-value function

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

- This is a recursive equation, so similar to our DP table, for every data point observed, we update estimates based on neighbors. Over time, the estimate (hopefully) converges to accurately capture "how good it is to be at state S and take action A ".
 - We can then apply control to the agent by taking the max over actions for the current state read by the sensors.
-

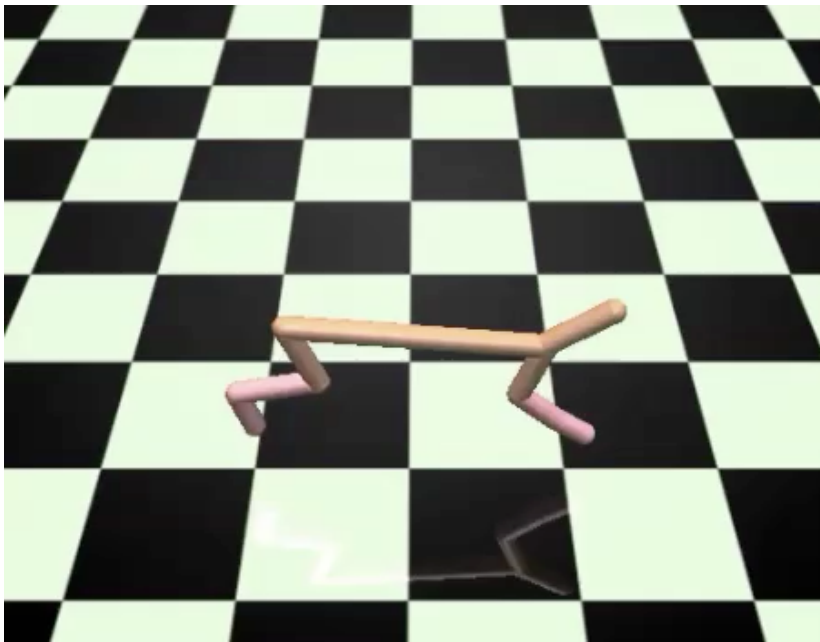
How to measure Deep RL for robots?

- Standard “robot-like” evaluation is OpenAI Gym

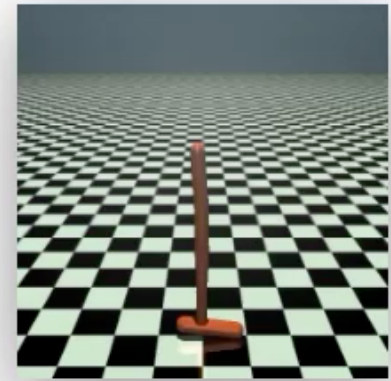


- Simple interface for a suite of benchmark tasks to measure progress

Learning Progress

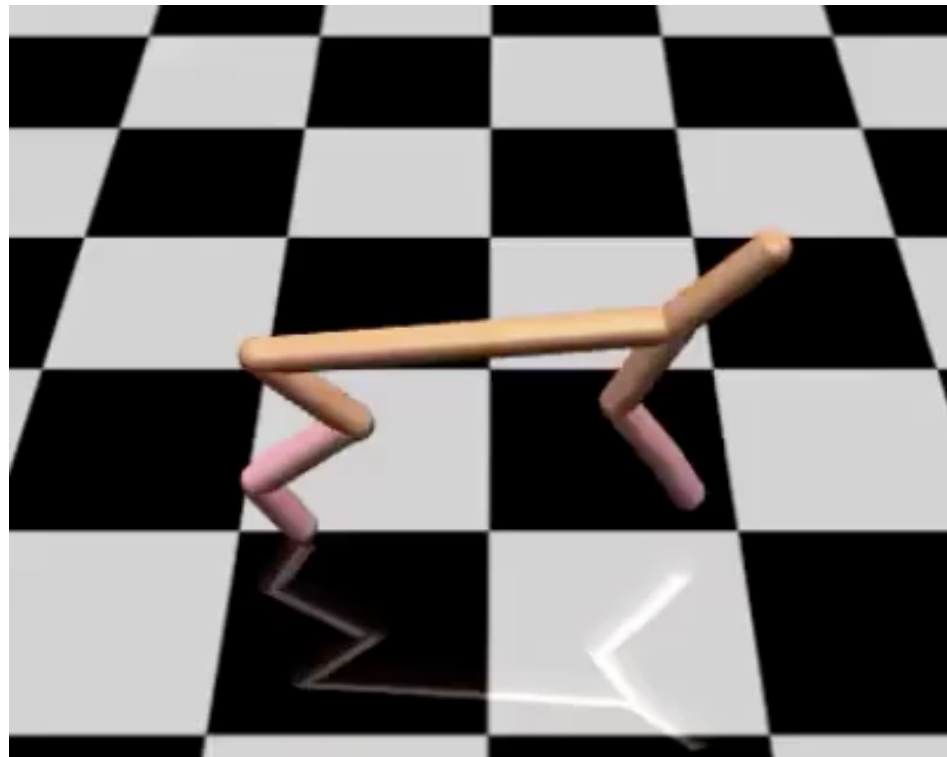


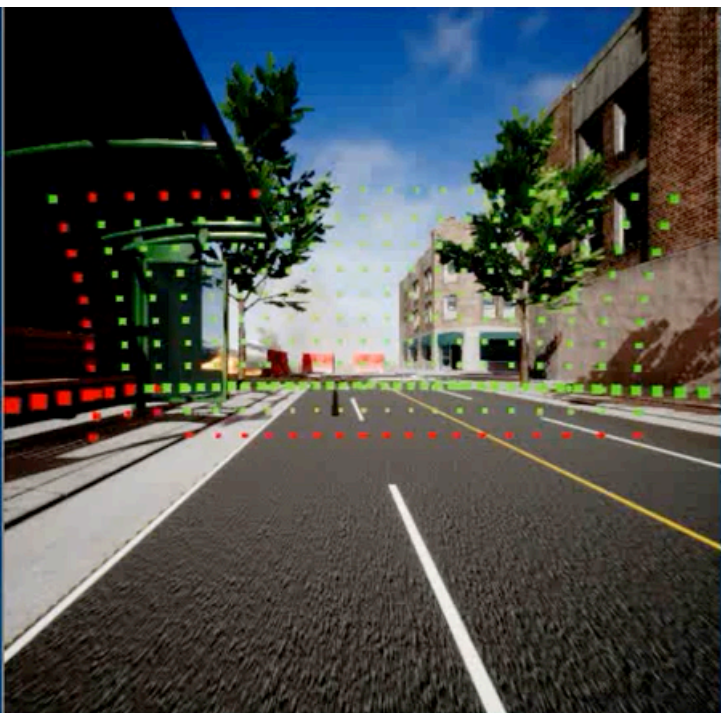
HalfCheetah-v2
Make a 2D cheetah robot
run.



Hopper-v2
Make a 2D robot hop.

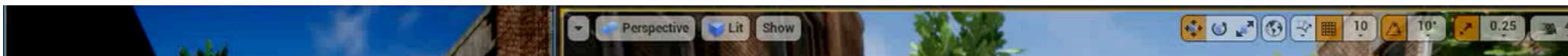
What kind of crazy behaviors can happen?



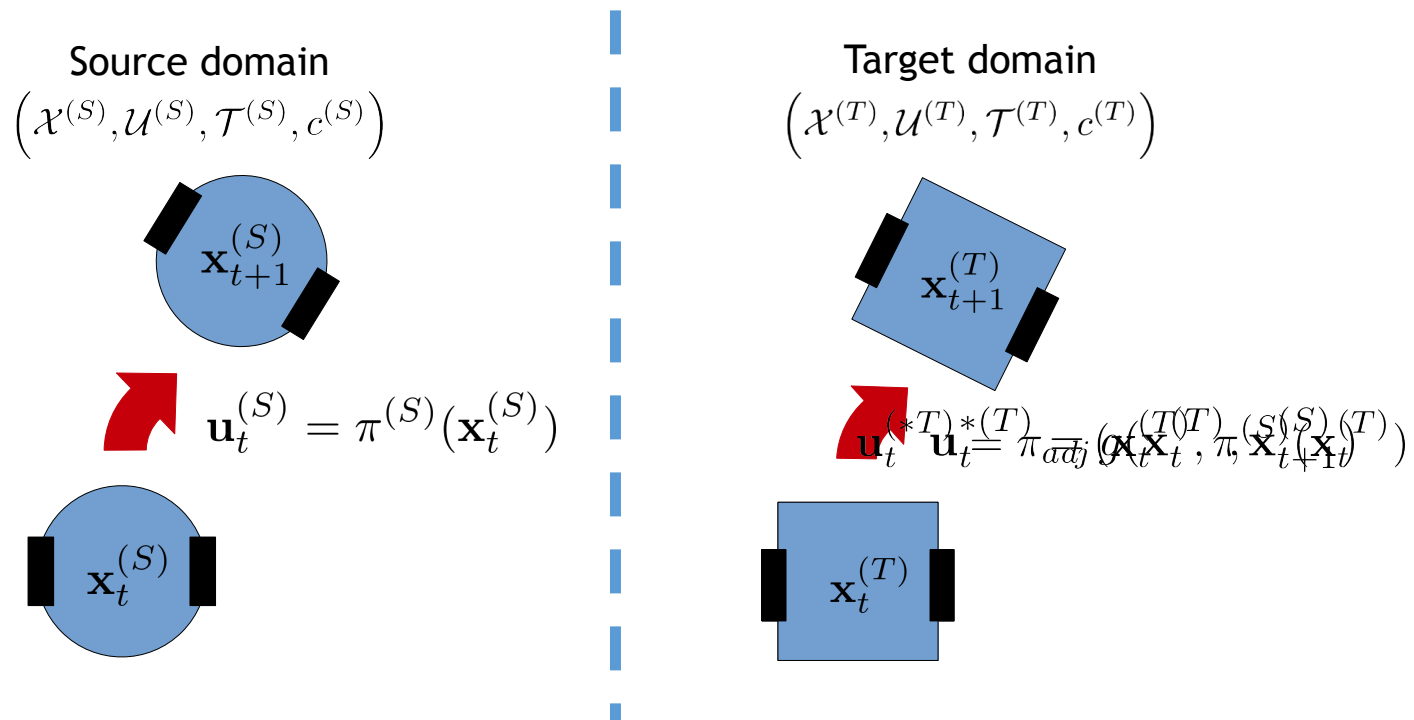


Is this enough skill to drive a real car?

- Surely not.
- So many real phenomena are missing or inaccurate.
- Can we transfer learned results directly from a simulation to the real world? Can we learn in simulation, and transfer to the real world: This problem/challenge is *sim2real*.
- Open issue: is sim2real an actual domain of study, or just a matter of making better simulators?



Sim2Real: Learning policy adjustments



3. Use **inverse dynamics** g to find the controls that match the source behavior and fit policy adjustment. π_{adj}

Inverse Optimal Control (synonym for Inverse Reinforcement Learning, IRL)

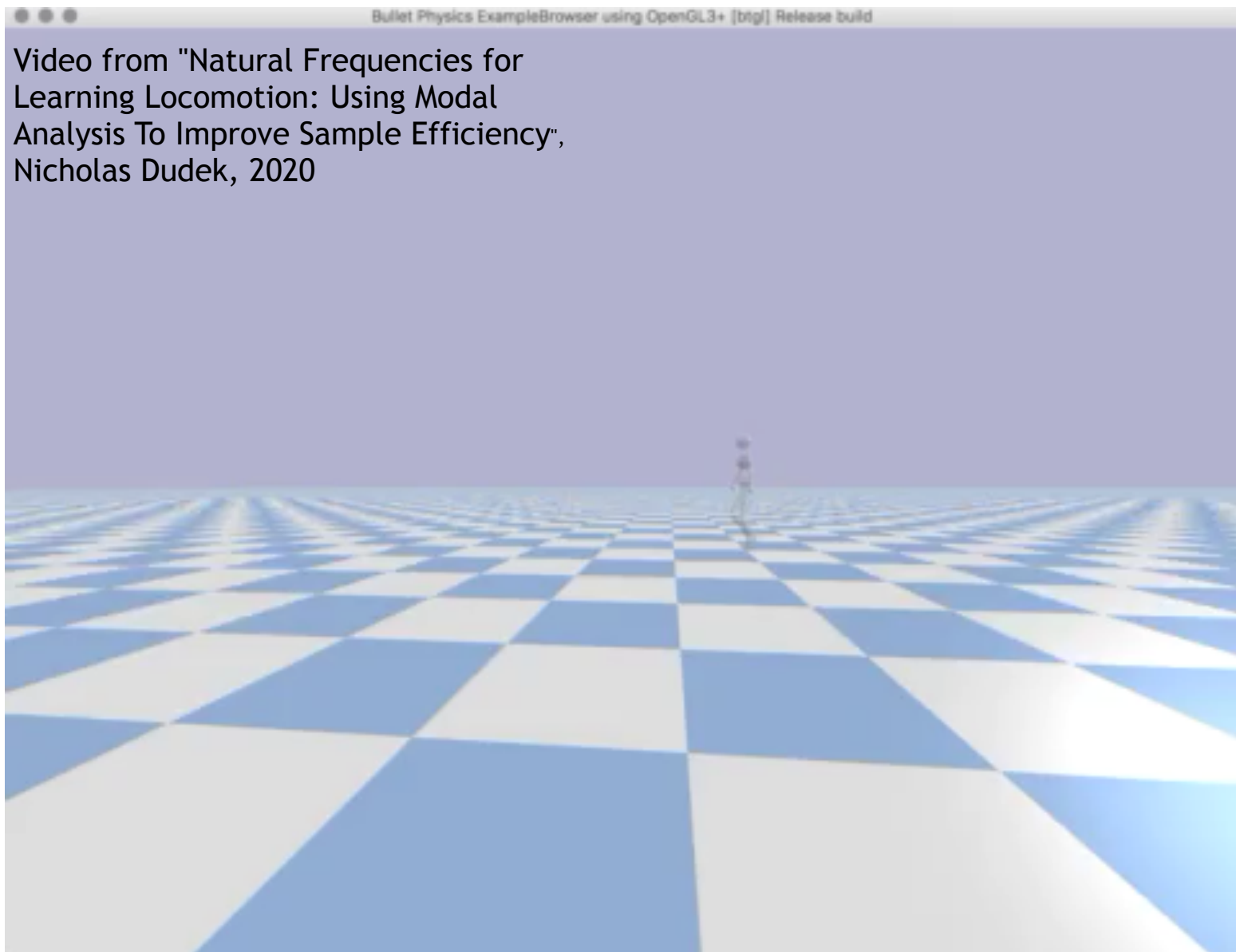
- Rather than learning $a=f(x)$ directly, instead infer the user's *cost function*, $r(x)$
- Now, we can use the tools from optimal control, RL etc in order to come up with the policy that fits. Notable differences in learning:
 - The ability to think deliberatively (plan for the future)
 - Optimization over time rather than assuming independence
- What about robustness to missing data? What $r(x)$ do we expect for unseen x values?

Components of the Method

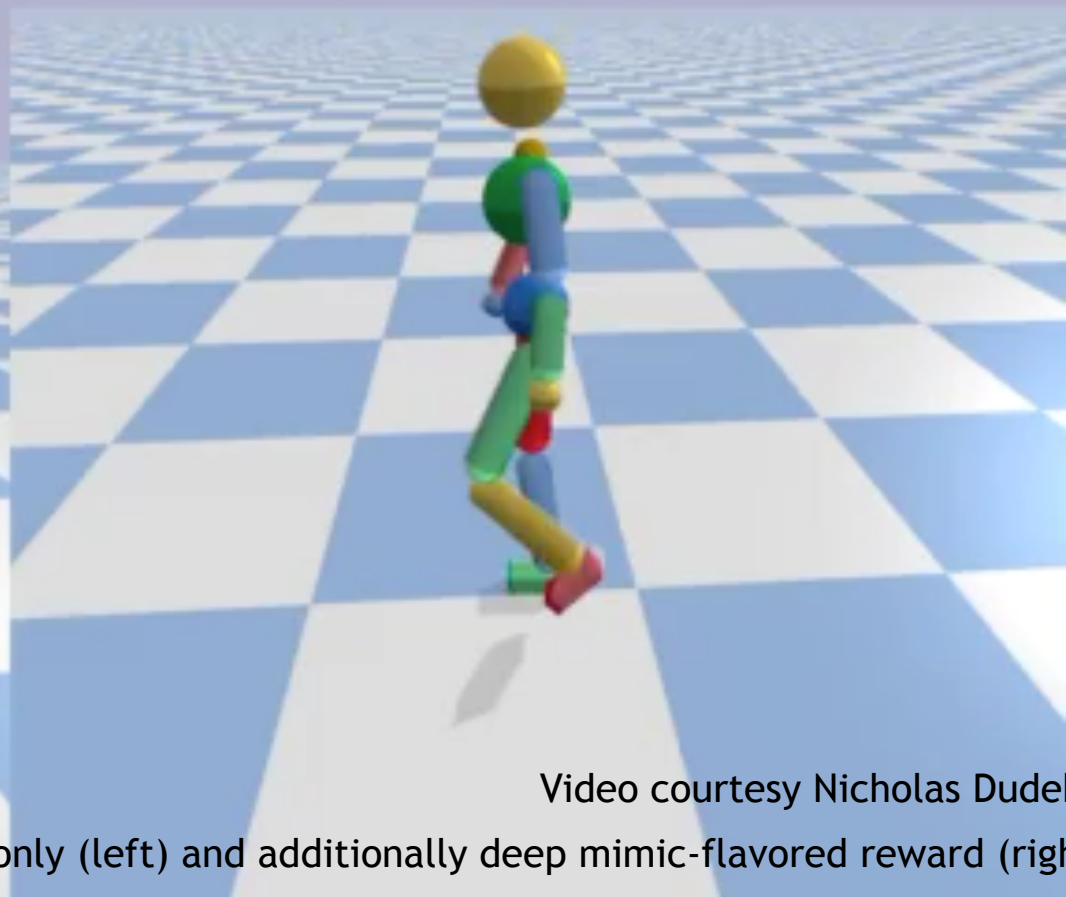
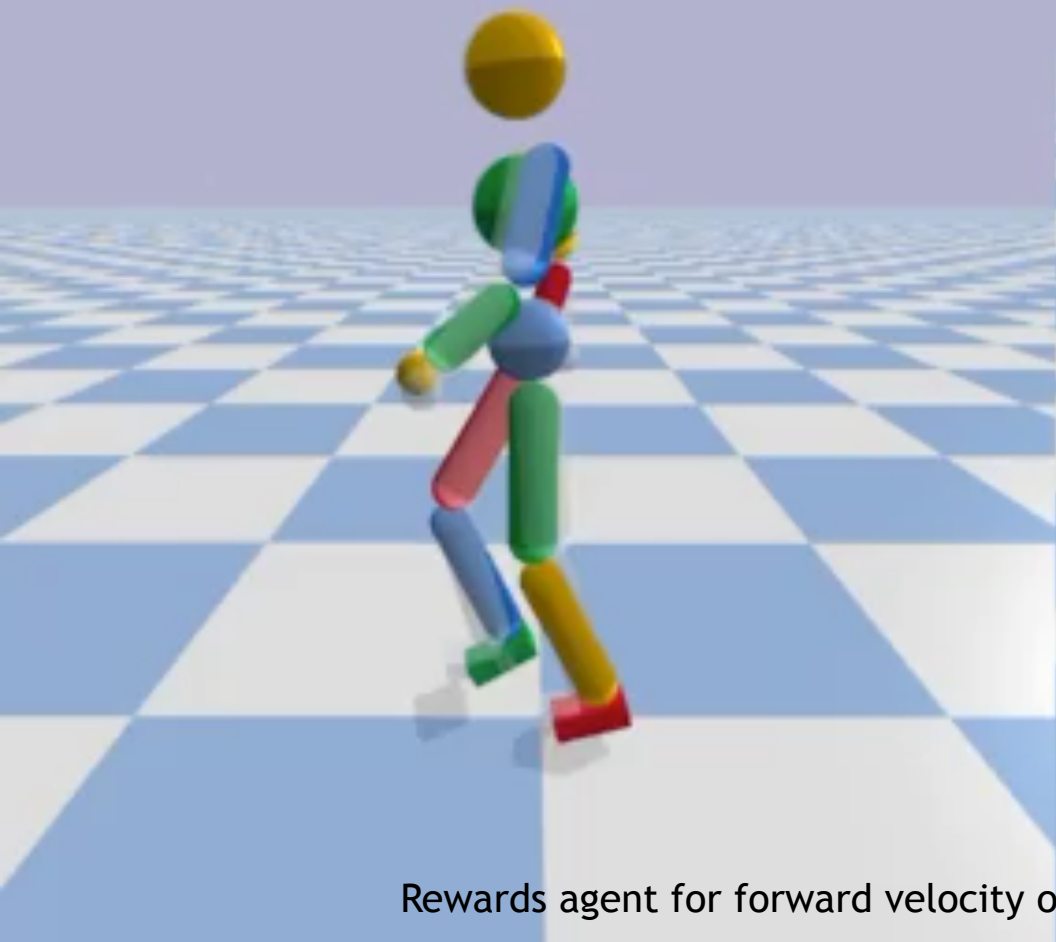
- Learn a task specification from demonstrations
- Learn a dynamics model mixing human-derived parameterizations and model fitting with demonstration/live data
- Trajectory optimization to generate autonomous control
- Experimental validation with some of the most "edge of envelope" behaviors seen to date

Deep Mimic

- One idea is to add two different kinds of control/reward
 - One based on actual goals
 - Another based on imitation (mimicry)



Video from "Natural Frequencies for Learning Locomotion: Using Modal Analysis To Improve Sample Efficiency", Nicholas Dudek, 2020



Video courtesy Nicholas Dudek

Rewards agent for forward velocity only (left) and additionally deep mimic-flavored reward (right)

Discrete vs Continuous Value Functions

- For discrete problems, there may be many actions, but at least we can track the outcome of each
- In continuous control problems, there are truly infinite outcomes at each stage.
- One approach is to directly optimize the policy as a continuous function



Continuous State/Action: Actor Critic

- We replace the max operator with a parameterized policy π_θ :

$$Q(s, a) \leftarrow r + \gamma Q'(s', a'), \quad a' \sim \pi_\theta'(s')$$

- The policy can be updated following the ***Policy Gradient***:

$$\nabla_\theta J(\pi) = \nabla_\theta Q(s, a) \big|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)$$

$$\theta = \theta + \alpha \nabla_\theta J(\pi_\theta)$$

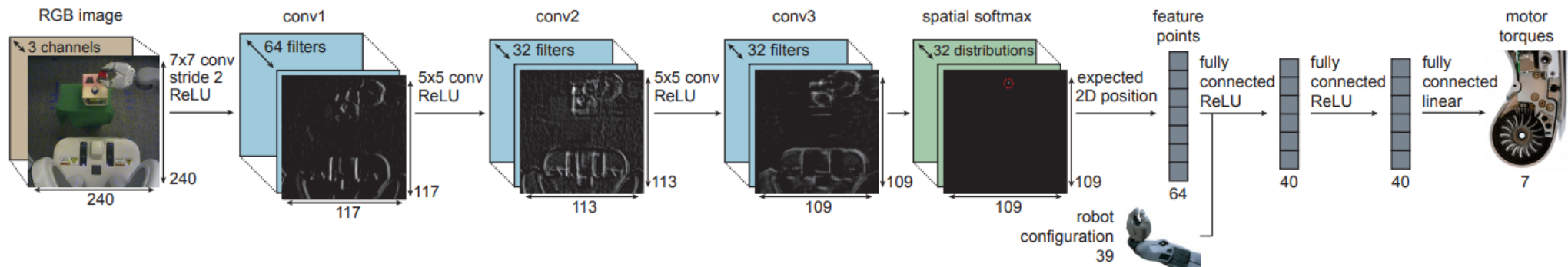
Actor Critic Methods $\nabla_{\theta} J(\pi) = \nabla_{\theta} Q(s, a) \big|_{a=\pi_{\theta}(s)} \nabla_{\theta} \pi_{\theta}(s)$

- Built from 2 interacting pieces:
 - Critic: learns Q (action-value) function from data.
 - Note that this is policy-dependent, so will always be changing during learning.
 - Actor: sets parameters of policy π_{θ} to maximize current critic
- Difference from Q learning:
 - No “max” operation. Therefore, can work in continuous state/ actions.

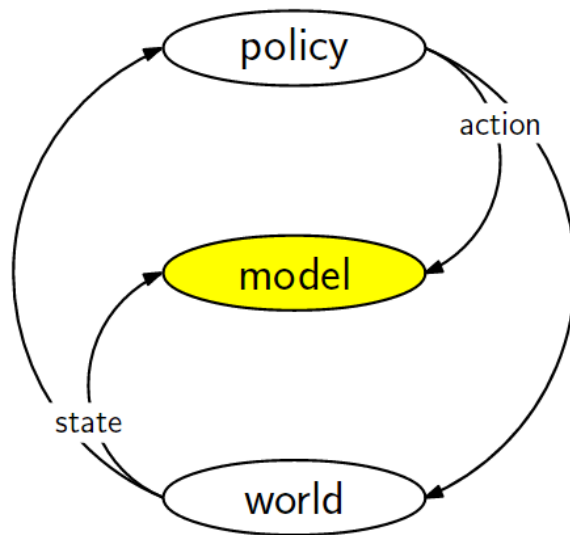
f

End-to-End Reinforcement Learning

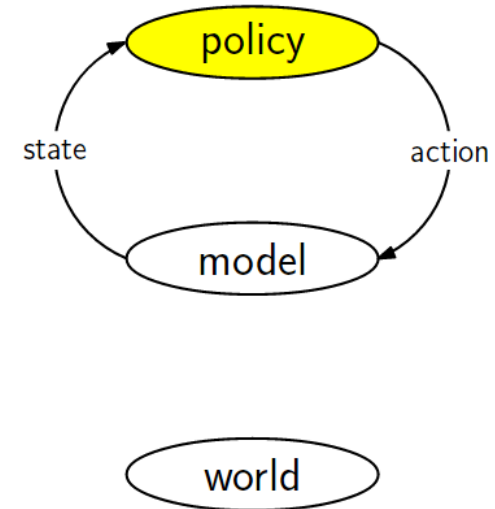
- We defined “state” by hand, but how does this scale to huge variety of new tasks? (e.g., peg in hole vs coat hanger vs water-bottle cap and more)
- End-to-end means the policy uses an image or other sensor directly
 - Both perception and control are taken care of “end-to-end”



Model-based Control Learning



interaction



simulation

Interact with the world to improve the model. Use the model to improve the control policy in simulation.

When to use RL?

When to use RL?

1. Data comes as trajectory.
2. Decisions are sequential.
3. Some state feedback (can be delayed, or noisy)
4. Sense that there will be a gain in terms of actions chosen over parts of trajectory (subtle point).

Traditionally presupposes discrete time.

Agent sees time-dependent training signal after each action.

Learning combined with planning. Effects of actions must be learned.

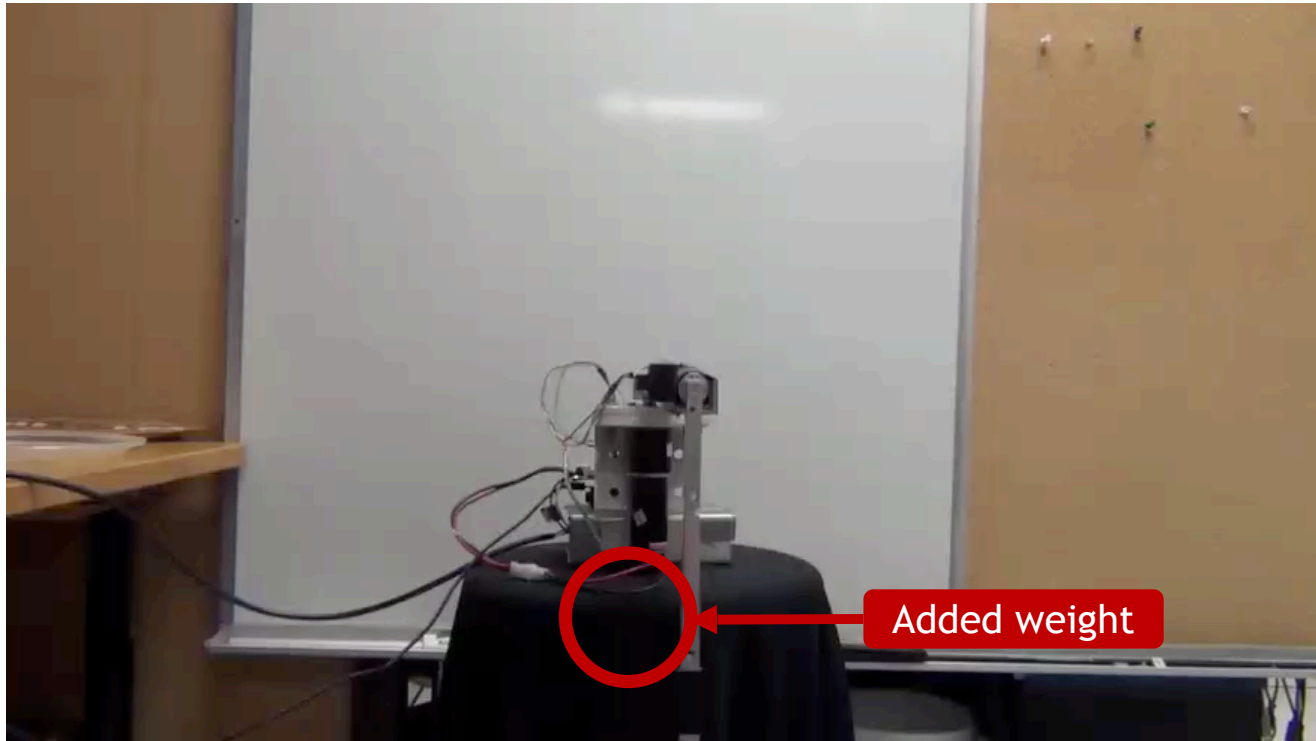
Data distribution changes with choice of actions. Not IID data!

Need access to the environment: action-result loop is closed through the environment.

Exchange of environments is difficult. That's why games are so popular as a testbed.

To reduce training overhead when the world/dynamics change, various tricks such as "domain randomization" can be used, where we learn from many different examples to build a "better learner".

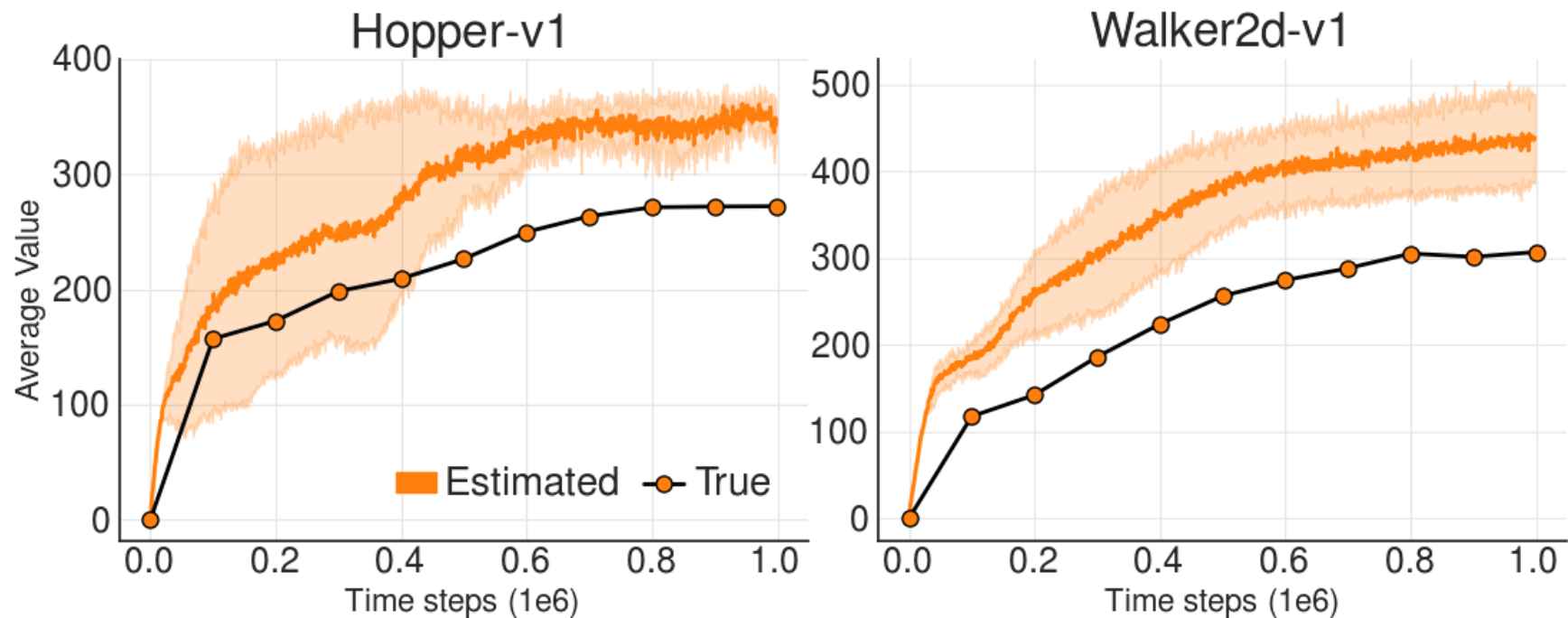
Transfer between cart-pole of varied mass



Juan Camilo Gamboa Higuera, David Meger and Gregory Dudek. Adapting Robot Behaviours Through Policy Adjustments. ICRA 2017.

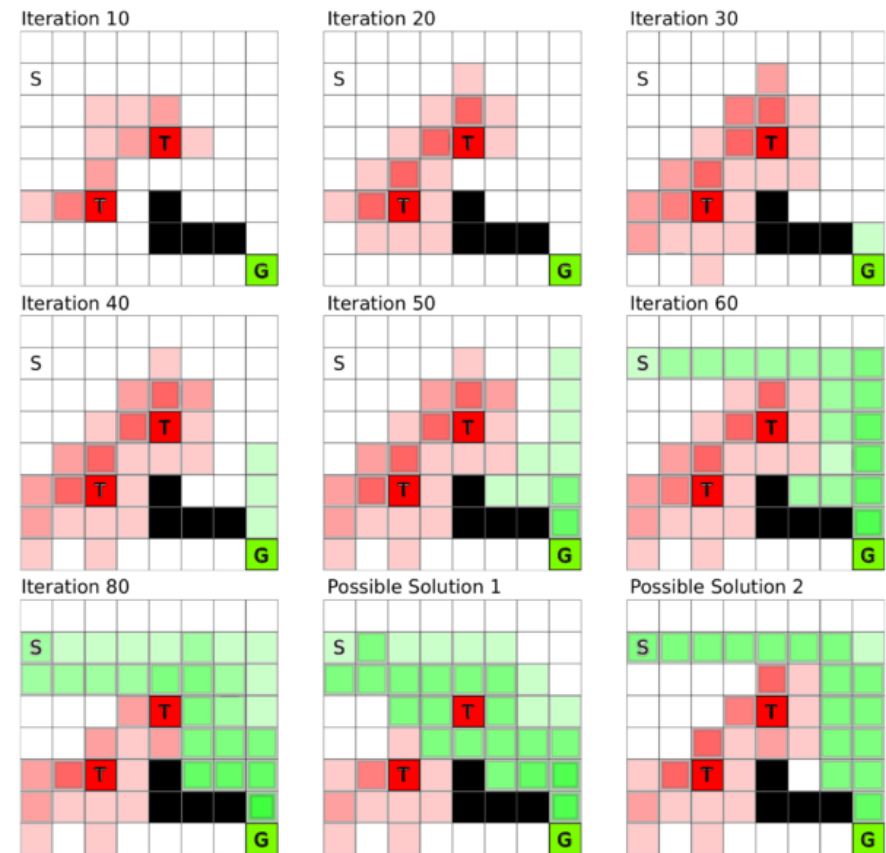
Improving DRL stability and performance
by diving deep on value learning

Empirical observation: values are overestimated, this is problematic



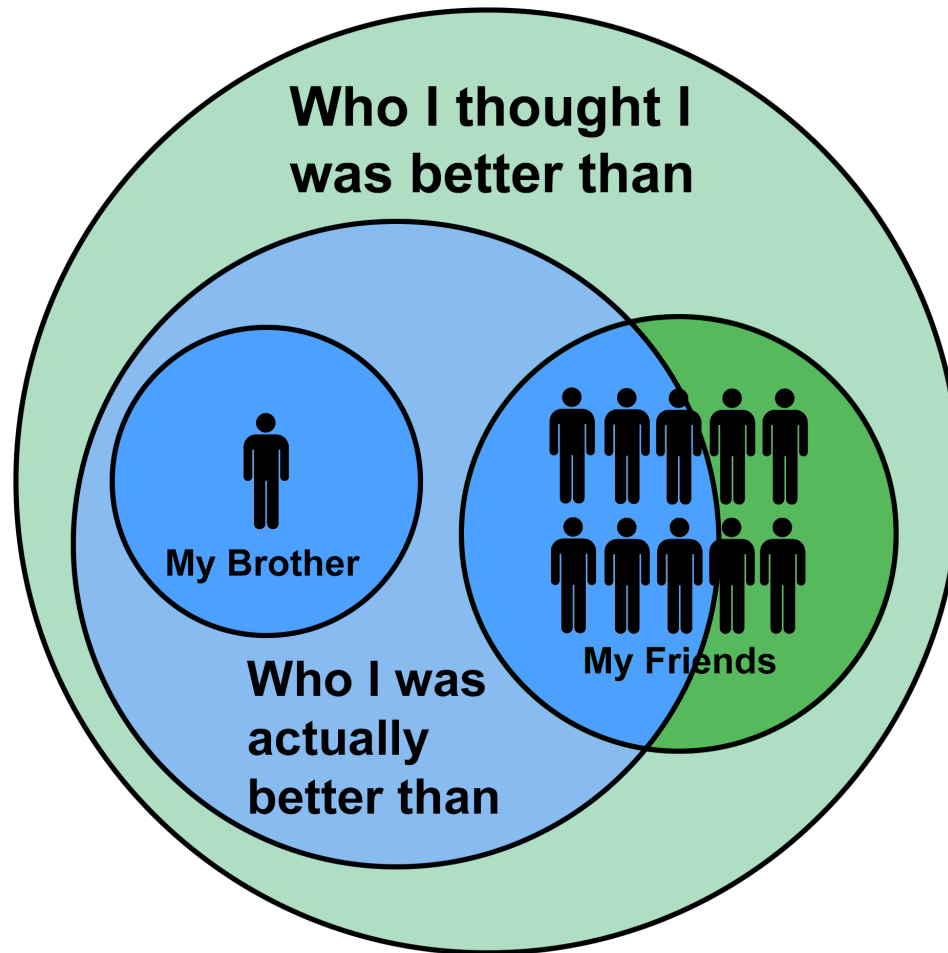
Why are Q-values
Overestimated, despite proofs of
performance?

- There is a proof of the convergence of Q-Learning with infinite data... however...
- How fast?
- How reliability when the Deep-net is involved?



Credit: <https://devblogs.nvidia.com/deep-learning-nutshell-reinforcement-learning/>

People also
over-estimate
their
competence



Why exams are a questionable methodology.

Goodheart's Law

***When a measure becomes a target,
it ceases to be a good measure.***

High interest rate means growing economy?
Most lines of code means best programmer?
Highest RL value estimate means best
action?

Only if we do not
allow
“gaming the system”

Actually from Marilyn Strathern who rephrased & popularized Goodheart. Economist Charles Goodhart actually said “Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes.” [*Problems of Monetary Management: the U.K. Experience*]

- Stare at the update rule of Q-learning with discrete actions.
Remember Q is a neural network that starts with random weights:

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$



- With errors ϵ overestimation is likely to occur

$$E \left[\max_{a'} (Q(s', a') + \epsilon) \right] > E \left[\max_{a'} Q(s', a') \right]$$

An extremum operator does not "mix well" when there is noise.

Actor Critic is also expected to over-estimate

- Recall the Policy Gradient learning updates:

$$Q(s, a) \leftarrow r + \gamma Q'(s', a'), \quad a' \sim \pi'(s')$$

$$\nabla J(\pi) = \nabla Q(s, a) \big|_{a=\pi(s)} \nabla \pi(s)$$

$$\pi = \pi + \alpha \nabla J(\pi)$$

- The gradient is expected to choose the an action that over-estimates when Q has noise

Double Q-Learning: Known for discrete MDP

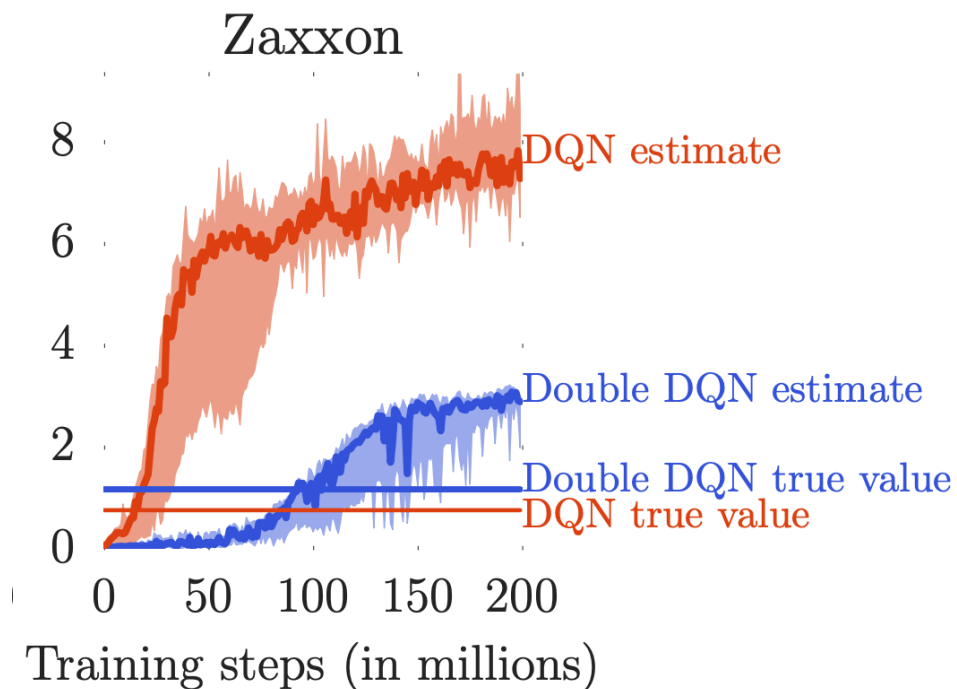
- A strategy for reducing overestimation bias is to use two independent estimates of the value function (van Hasselt, 2010):

$$Q_1 \leftarrow r + \gamma Q'_2(s', \pi'_1)$$

$$Q_2 \leftarrow r + \gamma Q'_1(s', \pi'_2)$$

- In Double DQN, the target network Q'_1 is used as one of the estimates (van Hasselt et al., 2015):

$$Q_1 \leftarrow r + \gamma Q'_1(s', \pi_1)$$



Clipped Double Q-learning

- A novel learning target, Clipped Double Q-learning:

$$Q_1 \leftarrow r + \gamma \min(Q'_1(s', \pi'_1), Q'_2(s', \pi'_1))$$

- If Q_1 generally overestimates, we can upper-bound the less biased Q_2 by the biased Q_1
 - This eliminates the situation where $Q_2 > Q_1 > Q_{\text{true}}$
-

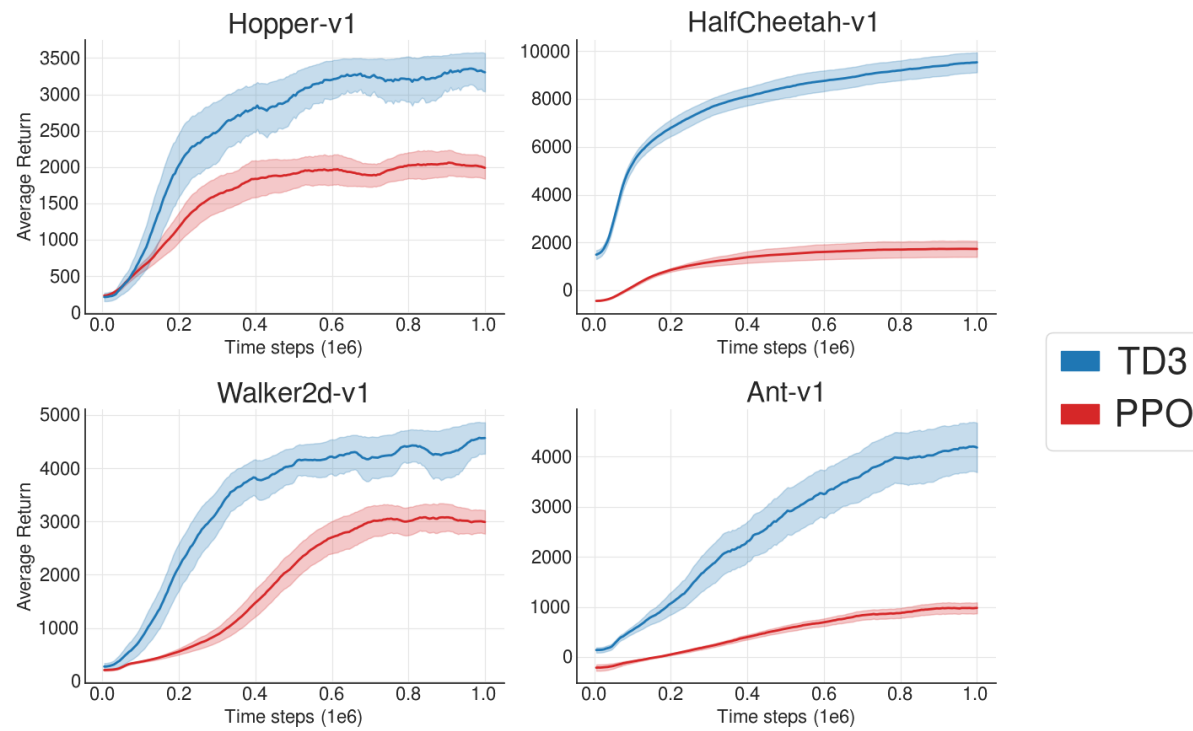
Twin Delayed DDPG (TD3)

Clipped update and several other modifications are added to DDPG to form **Twin Delayed Deep Deterministic policy gradient (TD3)**, developed by Fujimoto & Meger (McGill)

$$Q_1, Q_2 \leftarrow r + \gamma \min(Q_1(s', a'), Q_2(s', a')), \quad a' \sim \text{clip}(\pi(s') + \epsilon, -c, c)$$

- **Informal notion, ask your friend to help you estimate your own strength in a video game**

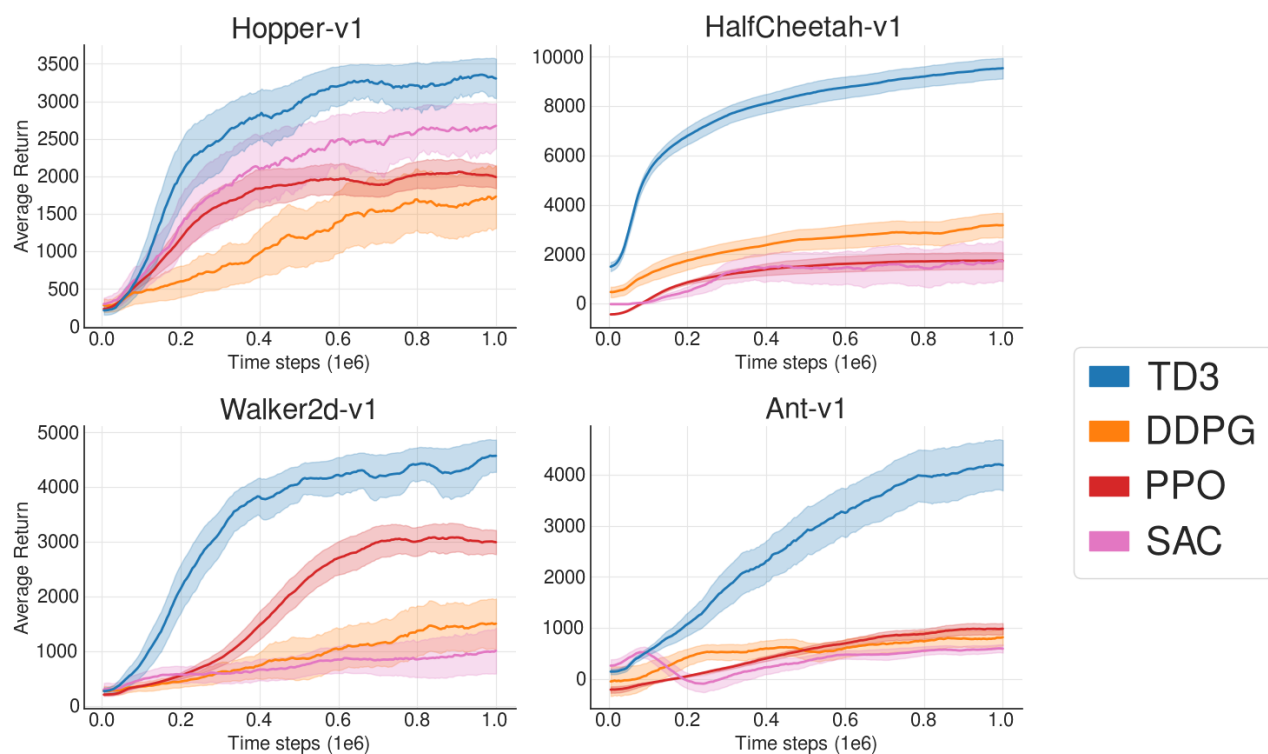
TD3 vs PPO



Results

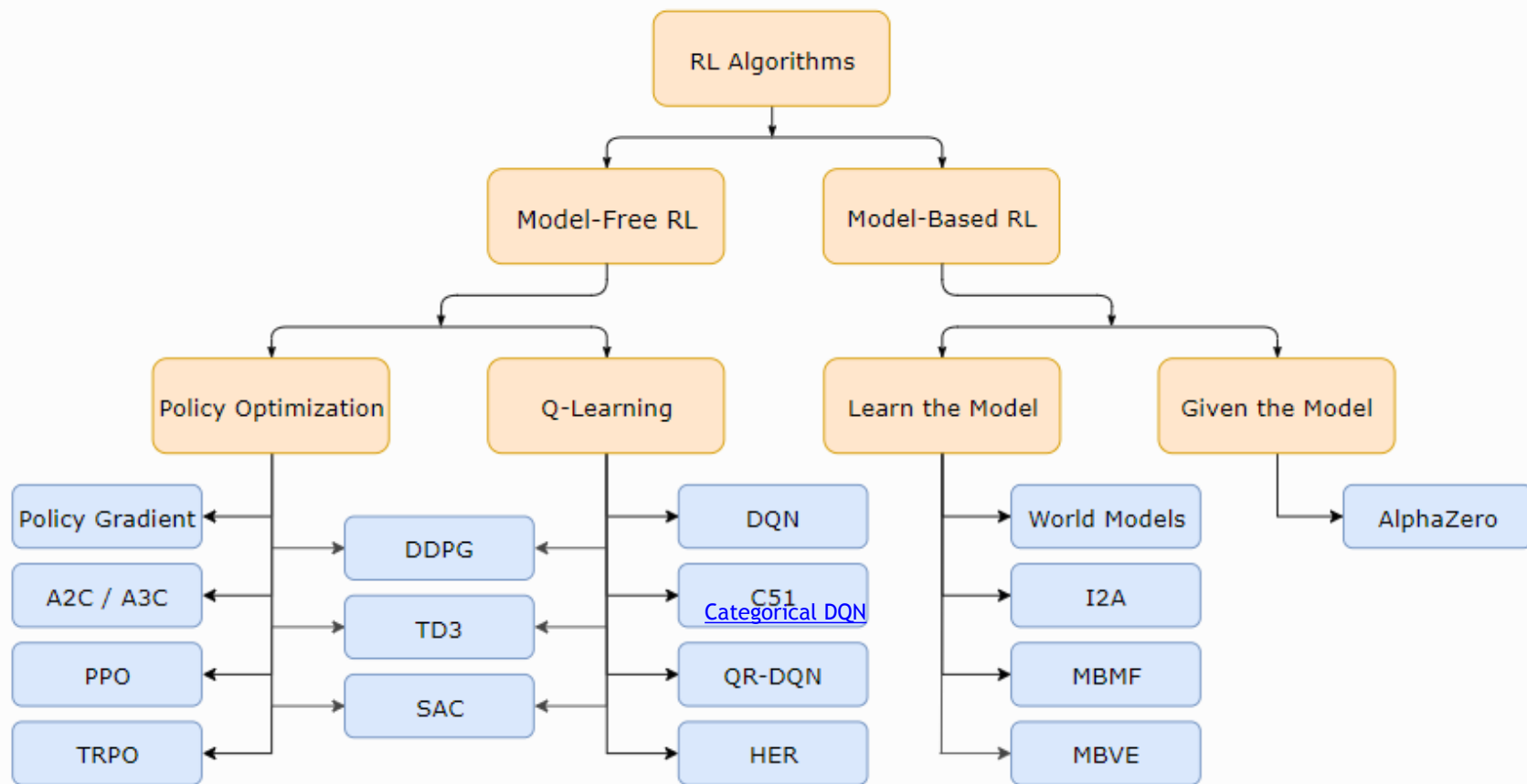
TD3: Scott Fujimoto, Herke van Hoof and David Meger.
Addressing Function Approximation Error in
Actor-Critic Methods. ICML 2018.

- <https://github.com/sfujim/TD3>
- Under 200 lines of PyTorch code

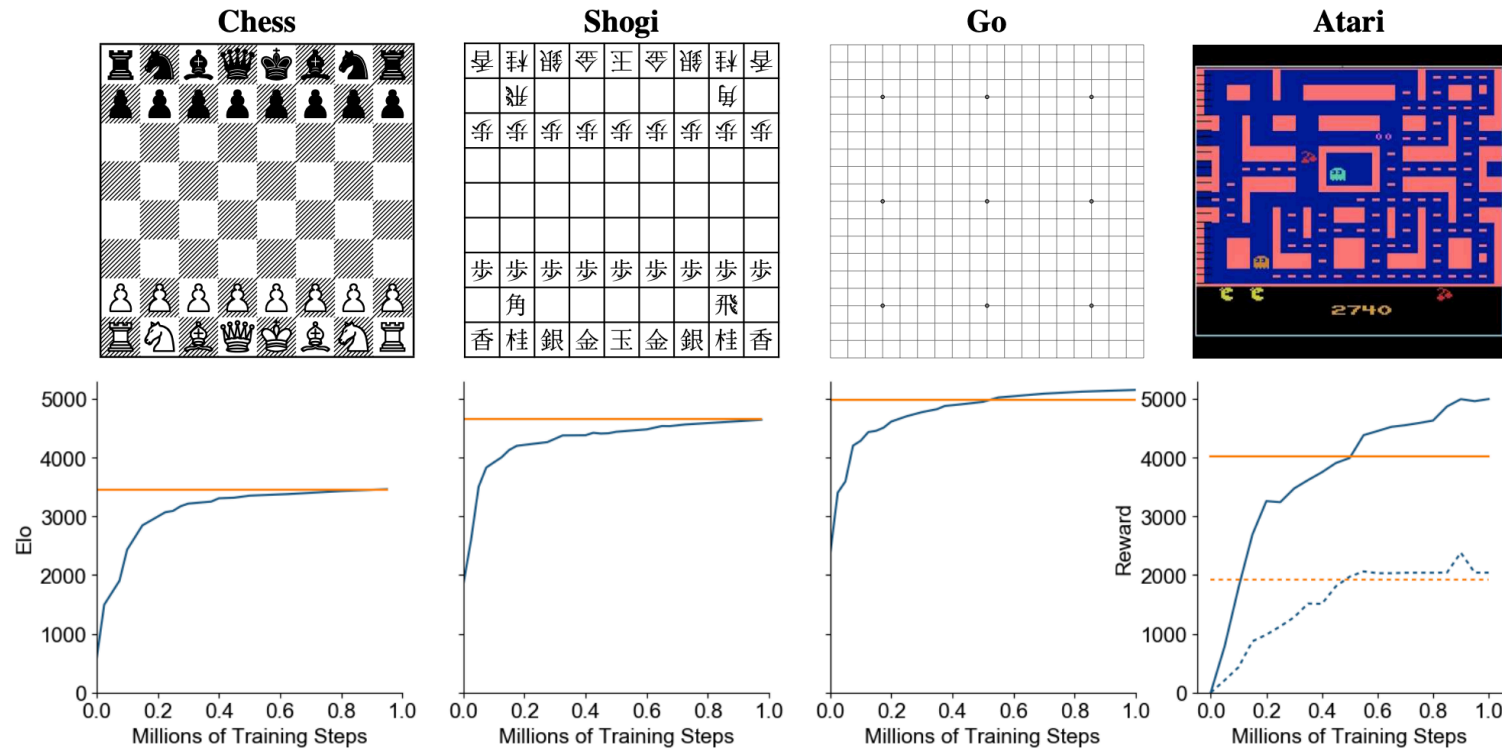


Wrap Up, parting thoughts &
examples

Algorithm Family



A recent landmark example: *muZero*

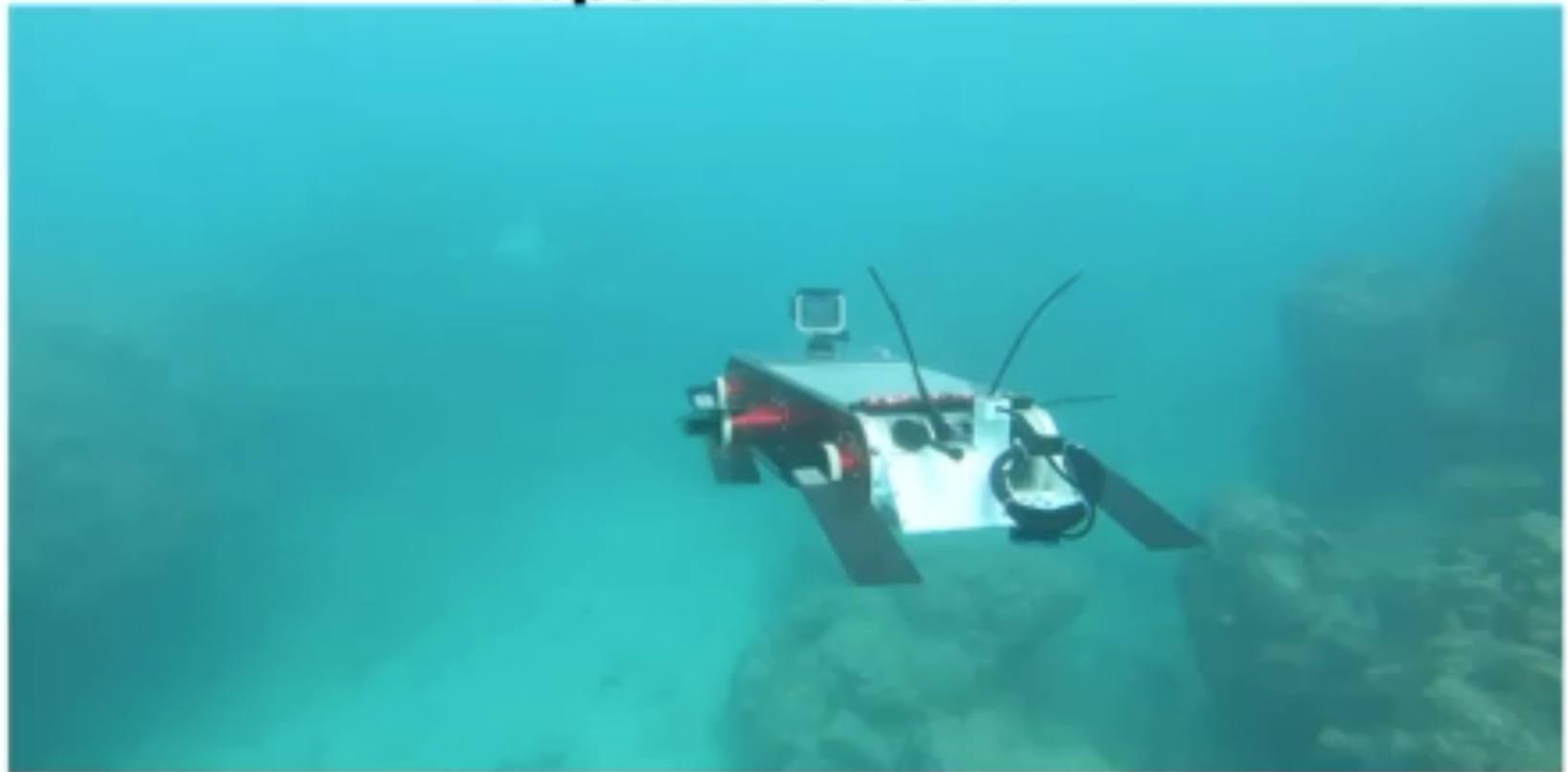


Predict aspects of the future that are *directly* relevant for planning. Model receives the observation as an input and **transforms it into a hidden state**. The hidden state is then updated iteratively by a recurrent process that receives the previous hidden state and a hypothetical next action.

It predicts the policy (e.g. the move to play), the value [function] (e.g. the predicted winner), and immediate reward (e.g. the points scored by playing a move).

Vision-Based, Goal-Conditioned Policies for Underwater Navigation in the Presence of Obstacles

Paper ID: 1324



Break-down and Names of Related Learning Tasks

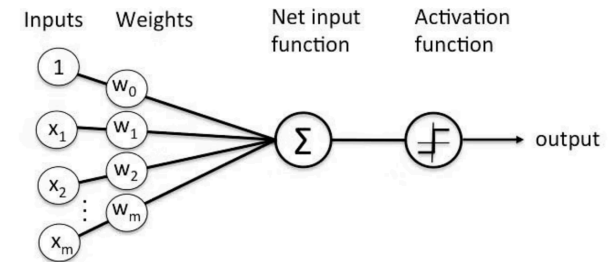
- **Optimal Control**: find the policy that minimizes cost
 - Known dynamics model, known reward function
 - Compute off-line so the first “real run” of the policy is perfect
 - E.g., LQR, Value Iteration
- **Reinforcement Learning**: gather data using trial-and-error
 - Model-based RL, where we first learn dynamics and/or reward models and then apply optimal control on those (imperfect) models
 - Model-free RL, where we directly learn cost-to-go and policy, never estimating the dynamics alone
 - E.g., Q-Learning, Deep Q Networks
- **Learning from demonstrations**: base learning on data collected by humans or other experts
 - Imitation learning: mimic expert actions directly
 - Example: Behavior Cloning
 - Inverse Reinforcement Learning (IRL): estimate reward function from data, then apply RL/opt. control
 - Example: Apprenticeship Learning (helicopter)
- **Transfer Learning**: adapt a policy that works on one “source” system to a similar “target”
 - Special case Sim2Real: where the source system is just a software simulation and target the real robot

More terms to remember

- *Kinodynamic Planning*
- *Infinite Horizon*
- *Receding Horizon*
- *Learning from demonstrations*
- *Transfer Learning*
- *Inverse Reinforcement Learning*
- *Deep-Q Network: DQN*
- *Out of distribution (learning)*

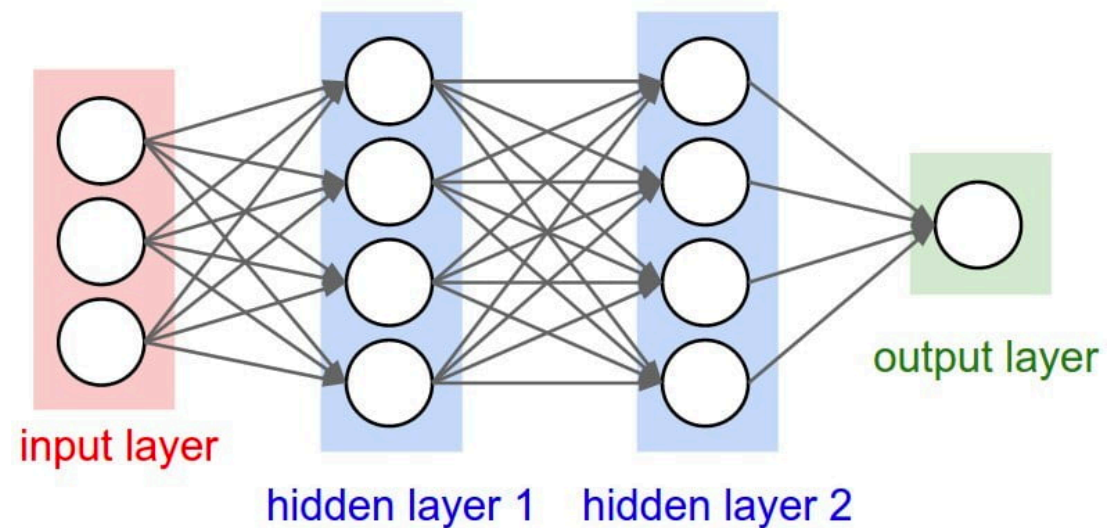
Deep Neural Networks (cartoon overview)

- Layers of artificial "neurons" that map input to output
- Between the input and output we have "hidden layers"
 - Often we have one or more narrow layers
- The hidden layers are often called the "**latent space**"
 - This especially applies to the narrow layer(s)
- The latent space can be regarded as a compressed representation of the domain
- A special class of possible layer is convolutional: the same local computation at each location
 - This leads to the term "convolutional neural network" (CNN)

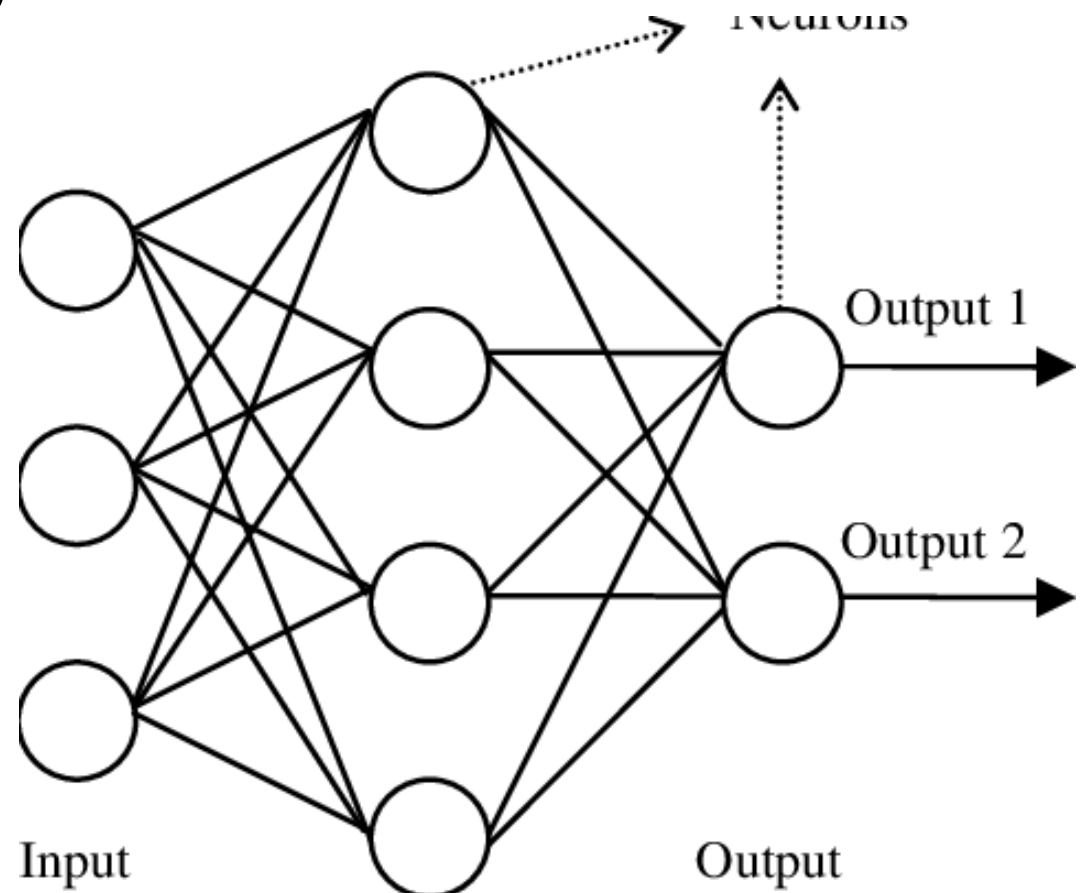
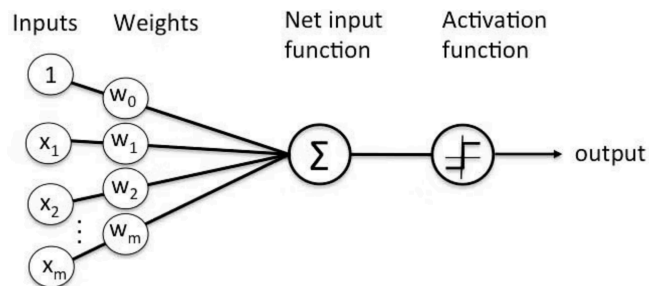


Not-so-deep network

- Each circle is an "artificial neuron" (this term not so popular now)
- Black lines or weights
- Circles compute a function of their weighted inputs

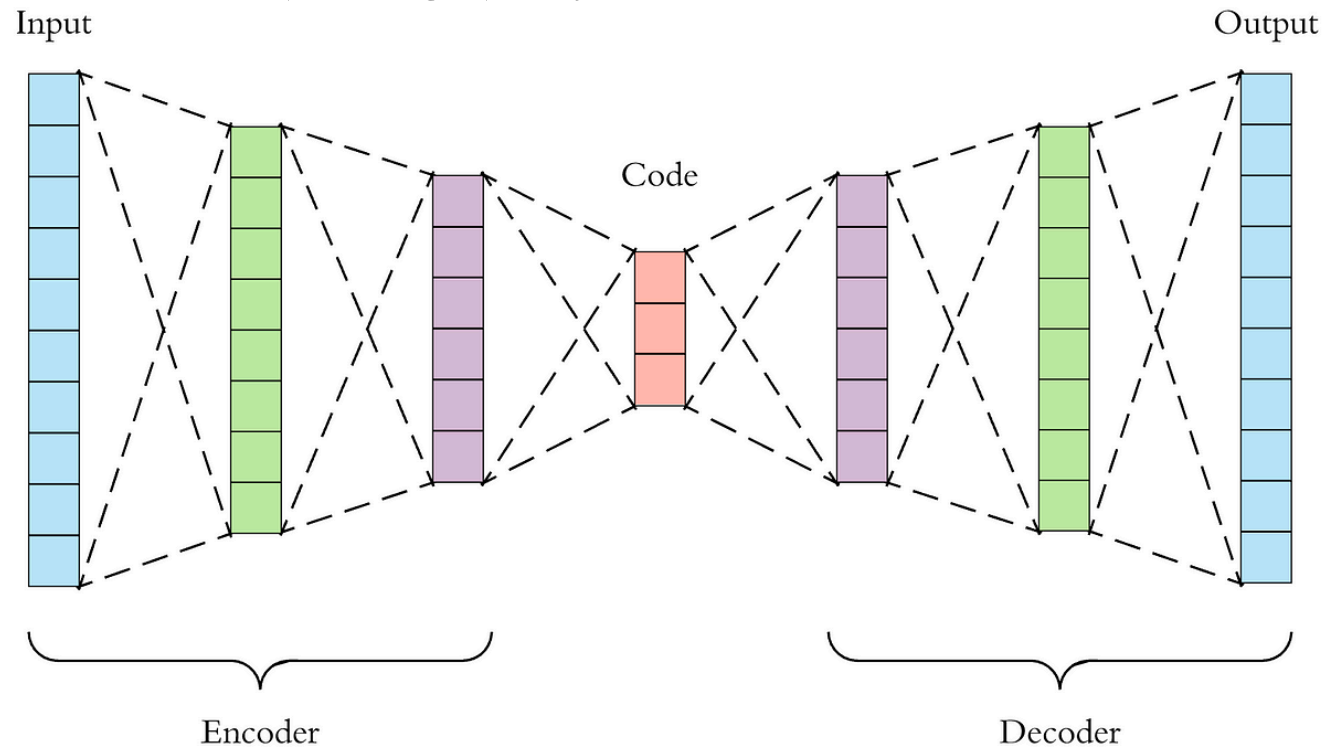


Simple multi-layer nets



The auto encoder

- Maps input space back to input space
- The latent (space) is the middle (orange) layer, AKA bottleneck
- Can find hidden structure in a data set
- Decoder side alone can be used to synthesis!



AI for Robotics in 2023

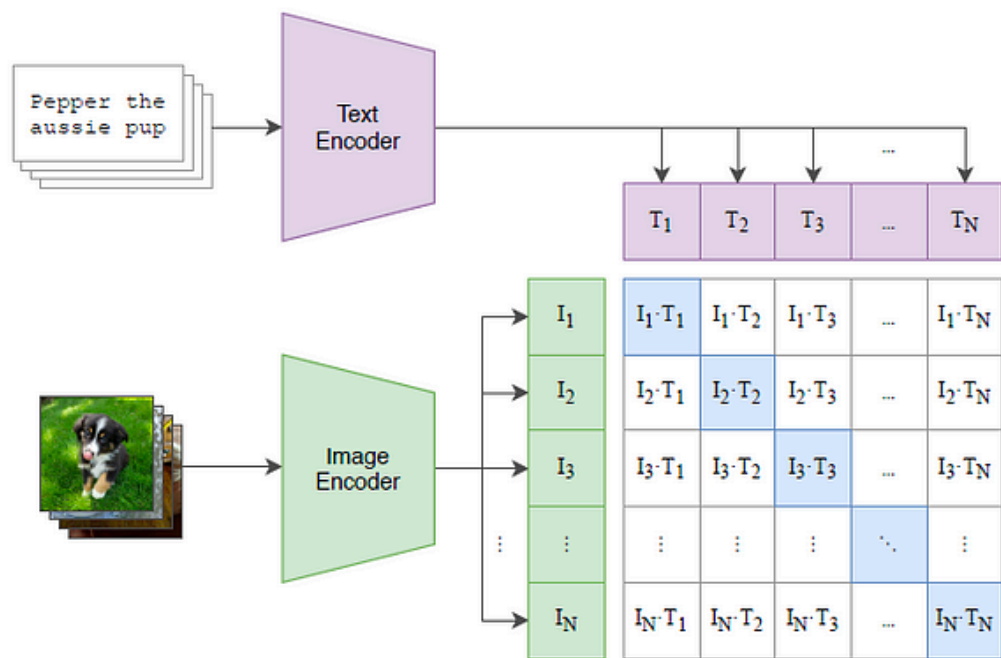
- Large Language Models are set to have a huge impact
- There are not notable branches:
 - LLM's themselves: language-to-language
 - VLM's: vision-language-models
- LLMs capture language
 - They predict words, given prior words
- VLMs use a shared latent space that takes input from both language and pictorial datas
 - They predict images given words, or words and images

Actual systems

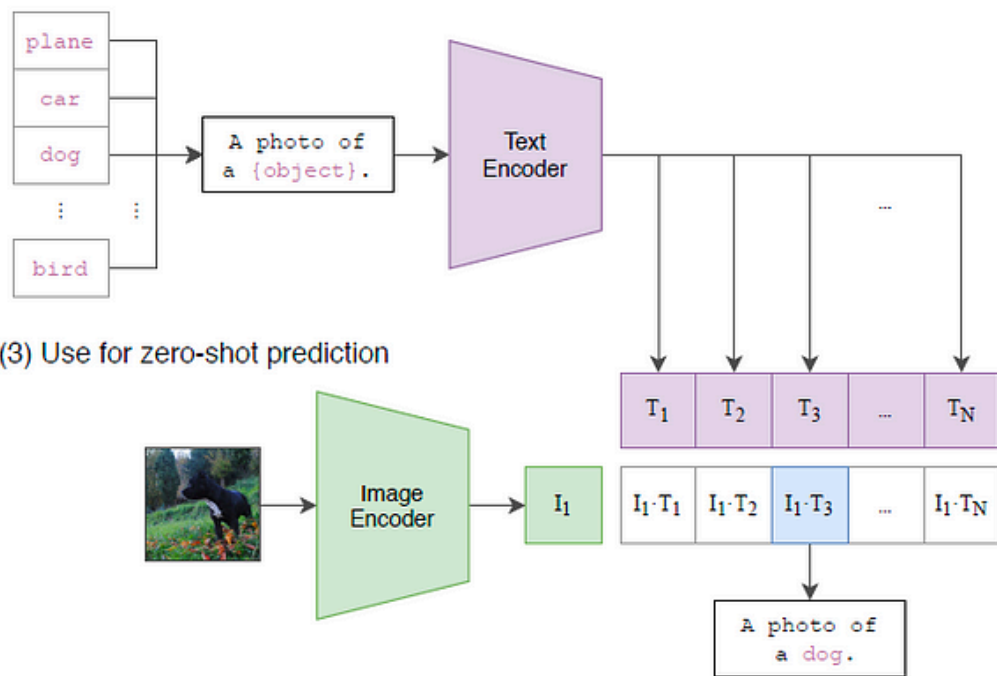
- (on white board)

- Contrastive Language-Image Pre-Training (CLIP) uses a pre-training task of predicting which caption to learn image representations from scratch on a dataset of 400 million (image, text) pairs.
- English is used to access/activate learned visual concepts (or describe new ones) enabling zero-shot transfer of the model to downstream tasks.
- Read: **Learning Transferable Visual Models From Natural Language Supervision**
- <https://proceedings.mlr.press/v139/radford21a.html>

(1) Contrastive pre-training



(2) Create dataset classifier from label text



Kostas Alexis: Resilient Autonomy in Perceptually-degraded Environments

Suggested for Nov 26th

Required viewing before the end of term: <http://y1d.com/w>



Resilient Autonomy in Perceptually-degraded Environments

Kostas Alexis

Autonomous Robots Lab

Norwegian University of Science and Technology | NTNU





Further reading (free download)

Richard Sutton and Andrew Barto (2018),
Reinforcement Learning. MIT Press.

<http://incompleteideas.net/book/RLbook2020.pdf>

How to Choose a Reinforcement-Learning Algorithm
Fabian Bongratz, ... , Daniel Cremers

<https://arxiv.org/pdf/2407.20917>

Reinforcement learning, when you can afford it

The settings

- **Online RL** — needs a simulator or a very patient, very robust robot; standard for locomotion, rare for manipulation
- **Offline RL** — learn from a fixed dataset; the killer problem is value overestimation on out-of-distribution actions, addressed by conservatism (CQL — conservative Q-learning) or by implicit methods (IQL)
- **Offline-to-online** — pretrain offline, fine-tune with a small amount of real interaction; the pragmatic middle, and increasingly used to polish VLAs
- **RL on top of a BC policy** — residual policies and RL fine-tuning of a pretrained VLA

Reward is the hard part

- Hand-designed dense rewards are the classic source of reward hacking
- Sparse task-completion reward is unbiased and nearly unlearnable without demonstrations
- Learned rewards: inverse RL, preference learning (RLHF-style) from human comparisons
- **VLM-as-reward** — ask a vision-language model whether the goal was achieved; cheap, scalable, and exploitable exactly where it is wrong

The trap

- A learned reward is an adversarial target. The policy optimises the reward model's errors.

Sim-to-real: what transfers

- **The reality gap has three parts** — dynamics (friction, contact, actuator lag), perception (rendering, lighting, textures), and task distribution (which scenes and objects exist at all).
- **Domain randomisation** — randomise physics and appearance until reality is one sample from the training distribution. Cheap and remarkably effective for locomotion; weaker for contact-rich manipulation.
- **System identification / real-to-sim** — measure the real robot and fit the simulator to it. More sample-efficient, less general.
- **Photorealistic real2sim** — reconstruct actual scenes with 3D Gaussian splatting or NeRF and train or evaluate in them.
- **Synthetic data from generative models** — video and world models as data engines (M6); physically-plausible augmentation of existing demonstrations.

The uncomfortable empirical fact

— Simulation success and real success correlate weakly for manipulation. Benchmarks such as SimplerEnv exist specifically to improve that correlation — and their existence is the admission.

What kind of generalisation are you claiming?

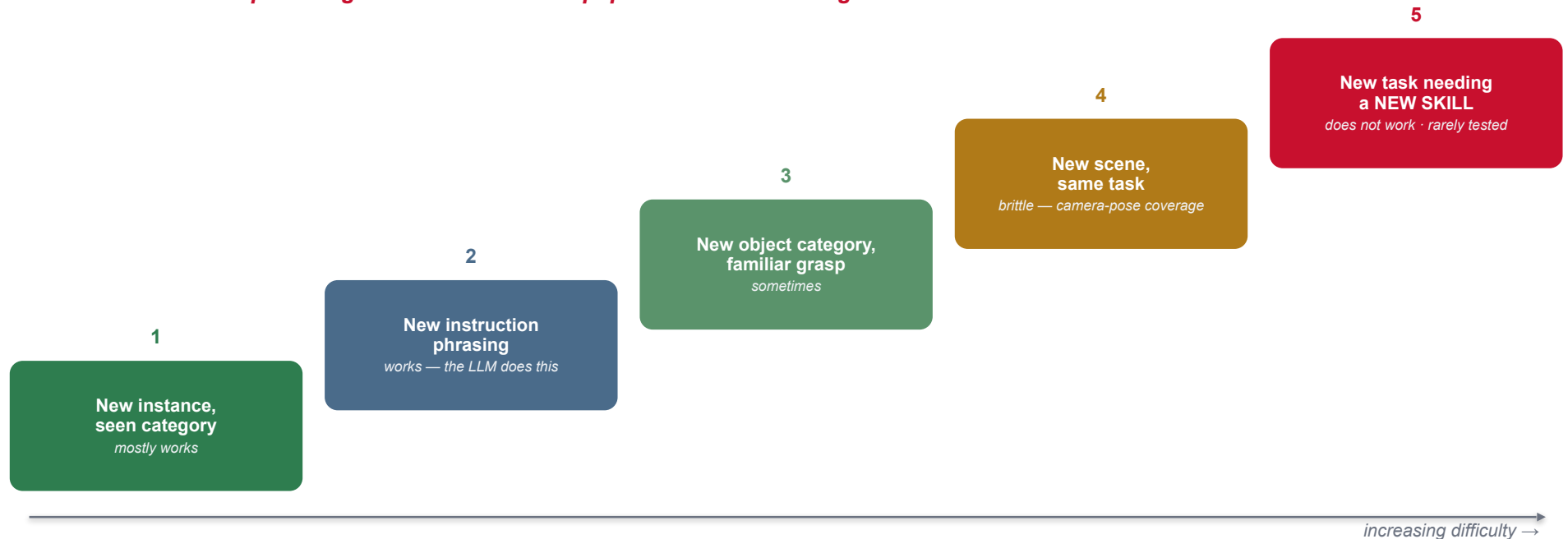
- **New instance of a seen object category** — mostly works.
- New object category with a familiar shape/grasp — sometimes works, driven by the pretrained vision backbone rather than the robot data.
- **New scene, same task** — works if the camera pose distribution was covered; fails abruptly if not.
- **New instruction phrasing** — works, courtesy of the language model. Frequently reported as if it were behavioural generalisation.
- **New task requiring a new skill** — rarely works and is rarely tested.

Rubric for reading results tables

- For every 'generalisation' number, identify which of these five it is. Most papers report the first and fourth, and title the paper after the fifth.

Which generalisation are you claiming?

Most results tables report rungs 1 and 2 — and the paper is titled after rung 5.



LIBERO's four suites (Spatial · Object · Goal · Long) are a partial operationalisation of this ladder — and recent audits find even those splits are partly shortcut-solvable.

TAKEAWAY

Imitation learning works when demonstrations cover the states the robot will visit.

- For current manipulation tasks, this is often cheaper than designing a reward and running RL.
- Long horizons increase distribution shift; hierarchical language plans can reduce the horizon handled by each policy.

Conclusions

- Reinforcement Learning is a key approach for linking actions and success states to produce intelligent behavior
- RL has already been used to achieve super-human performance for some problems
- Using RL depends on many subtle observations and adaptations
 - The world is complicated, one cannot expect a "once size fits all" solutions
 - Learning from demonstration, or from observation, seem very promising
- Data sparsity and human feedback are big pain points