CIS 419/519

Reinforcement Learning: ML For Sequential Decision Making

Lecture 20

Mar 29, 2023

Instructor: Dinesh Jayaraman

Based on slides from Sergey Levine, Dan Klein, Eric Eaton

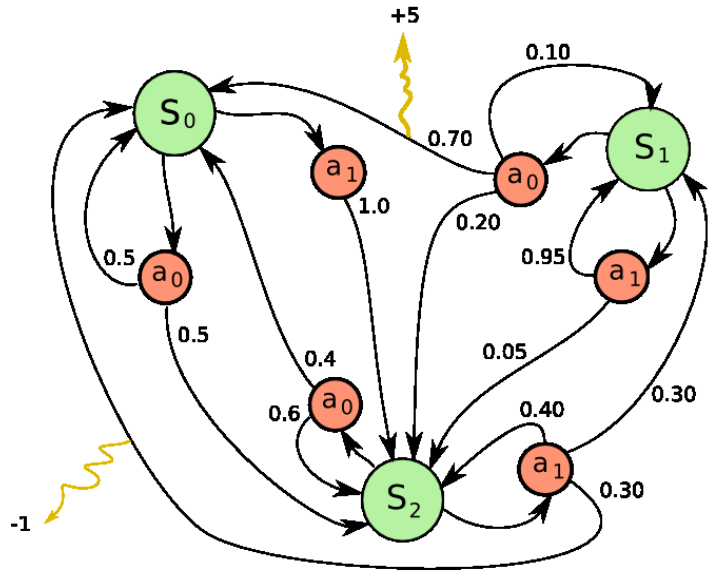
Recap: The Markov Decision Process Framework for RL

An MDP (S, A, P, R, γ) is defined by:

- Set of states $s \in S$
- Set of actions $a \in A$
- Transition function $P(s' \mid s, a)$
  - Probability $P(s' \mid s, a)$ that a from s leads to s'
  - Also "dynamics model" / just "model"
- Reward function $r_t = R(s, a, s')$
- Discount factor $\gamma < 1$, expressing how much we care about the future (vs. immediate rewards)
- "utility" = *discounted* future reward sum $\sum_t \gamma^t r_{t+1}$
- Goal: maximize *expected* utility

Unknown to agent

Example



In RL , we assume no knowledge of the true functions $P(\cdot)$ or $R(\cdot)$

Image: https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690

Recap: RL vs SL

SL: Find $h(x): X \rightarrow Y$, that minimizes a loss L over training (x, y) pairs

RL: Find $\pi(s): S \rightarrow A$ that maximizes expected utility

Supervised Learning

- Target labels for h are directly available in the training data
- Train to map (regress/classify) from x to y in the training data

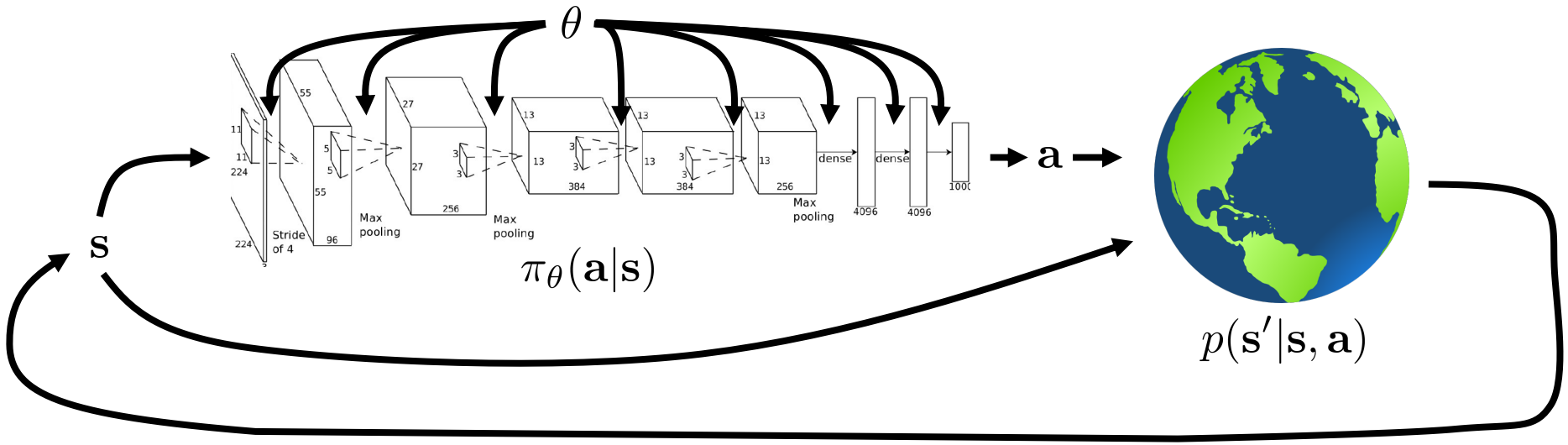
Reinforcement Learning

- Optimal action labels a for states s are not given to us. No predefined solutions!
- Train by trying various action sequences in an environment, and observing which ones produce good rewards over time.

Key Problems Specific to RL:

- **Credit assignment:** Which actions in a sequence were the good/bad ones?
- **Exploration vs Exploitation:** Yes, trial-and-error, but smartly pick what to try?

Recap: The goal of RL



Trajectory distribution

$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

Optimal policy parameters

$$\theta^* = \arg\max_\theta \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\sum_{t=1}^{T} \gamma^t r(s_t, a_t) \right]$$

Goal of RL is to learn a **policy** $\pi(s): S \rightarrow A$ for acting in the environment

Can we solve reinforcement learning with gradient descent?

Policy Gradients

Recall: Behavioral Cloning for Imitation Learning

The BC gradient w.r.t. policy parameters θ looked like:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\right)$$

Demonstration data

Expert actions

“Policy Gradient” Methods

Update policy parameters with the gradients of the expected utility in an episode by following policy $\pi_{\boldsymbol{\theta}}$

$$\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_{old} + \alpha \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left[\sum_{t'=t} r_t \right]$$

$\pi_{\boldsymbol{\theta}}$ induces a trajectory distribution, which induces a reward distribution.

We will show, the gradient $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}} [\sum_t r_t]$ works out to:

$$\frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \sum_{t'=t}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

Note: we will focus for now on “finite-horizon” settings, i.e., T is finite.

Note: we are ignoring discount factors for now, all formulae will easily generalize

Compare to the BC Gradient

BC Gradient: Demonstration data Expert actions

$$\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\right)$$

RL "Policy Gradient": ~~Demonstration data~~ ~~Expert actions~~

$$\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\left[\sum_{t'=t}^{T} r(s_{i,t}, a_{i,t})\right]\right)$$

trajectories

time

Likelihood gradient: "Change policy to make these actions more likely".

Critic: "how good was this trajectory?" --- credit assignment

Recall: we start out with no data at all … so where does this data come from?

Ans: We generate our own data during learning … this is trial-and-error learning!

Preparing: Approximating Expectations

The formula for the expectation of a function y of a random variable x is:

$$\mathbb{E}_X[y(x)] = \int P(x)y(x)dx$$

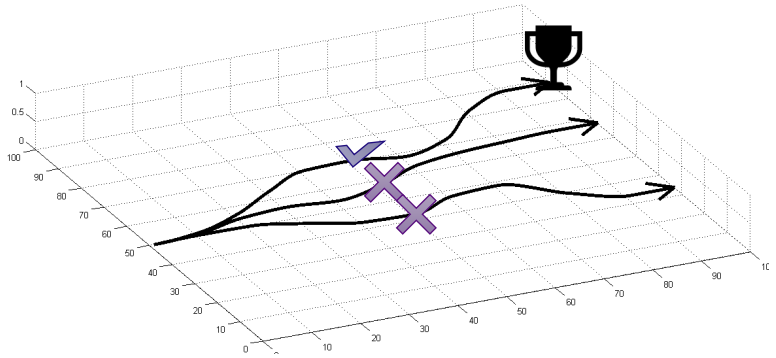
And this can be approximated as the "sample mean":

$$\int P(x)y(x)dx \approx \frac{1}{N} \sum_{\text{samples } x \sim P(x)} y(x)$$

Lesson: If you can make an integral look like an expectation, you may be able to approximate it easily.

Evaluating the RL objective

$$\theta^\star = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]}_{J(\theta)}$$



$$J(\theta) = E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

sum over samples from π_θ

(since expectations can be approximated by sample means)

Of course, we must go beyond just evaluating the objective to optimizing it.

Policy Gradients Derivation Slide 1/2

$$\theta^\star = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]}_{J(\theta)}$$

a convenient identity (log-gradient trick)

$$\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) = \pi_\theta(\tau)\frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \nabla_\theta \pi_\theta(\tau)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\underbrace{r(\tau)}_{\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)}] = \int \pi_\theta(\tau) r(\tau) d\tau$$

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau = \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

Policy Gradients Derivation Slide 2/2

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

$$\underbrace{\pi_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

log of both sides

$$\log \pi_\theta(\tau) = \log p(\mathbf{s}_1) + \sum_{t=1}^{T}\log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\nabla_\theta \left[\log p(\mathbf{s}_1) + \sum_{t=1}^{T}\log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\right]$$

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

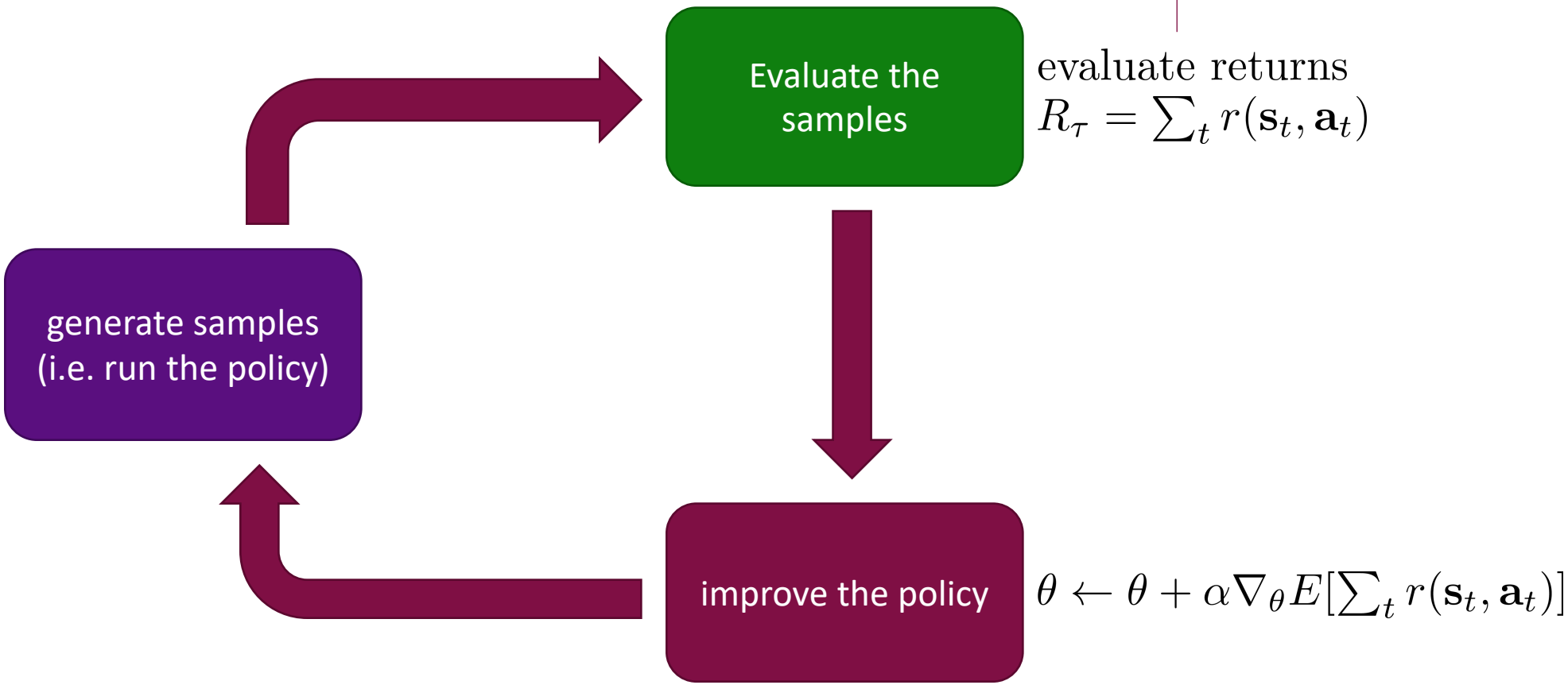
And now, we have an expectation with all computable terms!

Approximated with sample average!

The basic policy gradients algorithm: REINFORCE

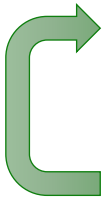
REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



Reward function need not be differentiable!

REINFORCE algorithm:



1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

In supervised learning, when we optimized an objective using gradient descent, we needed the objective to be differentiable w.r.t. to the parameters θ.

In RL, this is not true any more. See how the update term involves no derivative of the reward function!

2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$

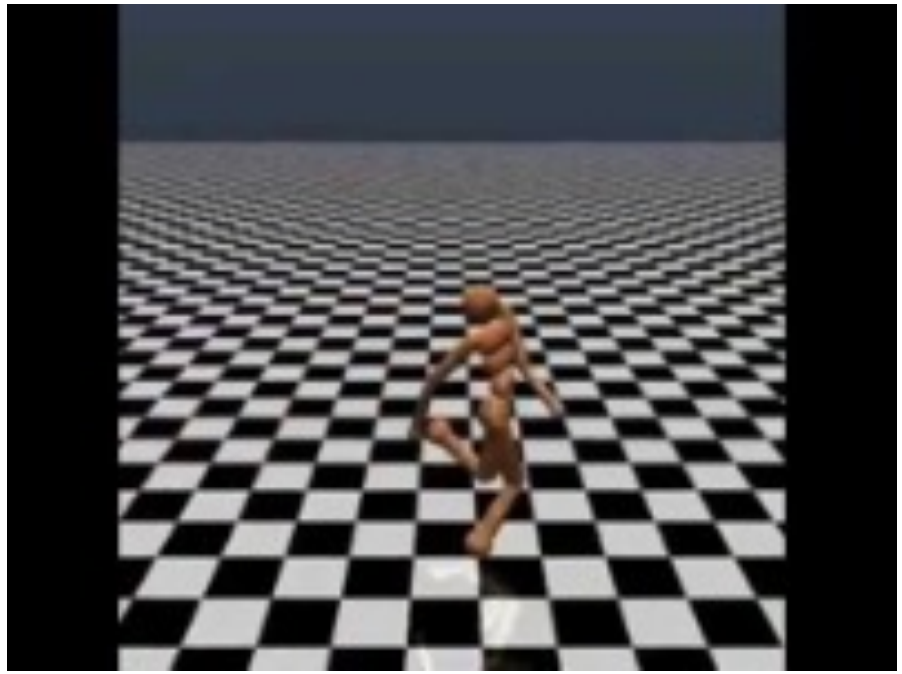
Example: Gaussian policies

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

example: $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = \mathcal{N}(f_{\text{neural network}}(\mathbf{s}_t); \Sigma)$

$$\log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2}\|f(\mathbf{s}_t) - \mathbf{a}_t\|_\Sigma^2 + \text{const}$$

$$\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2}\Sigma^{-1}(f(\mathbf{s}_t) - \mathbf{a}_t)\frac{df}{d\theta}$$



REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

What did we just do?

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

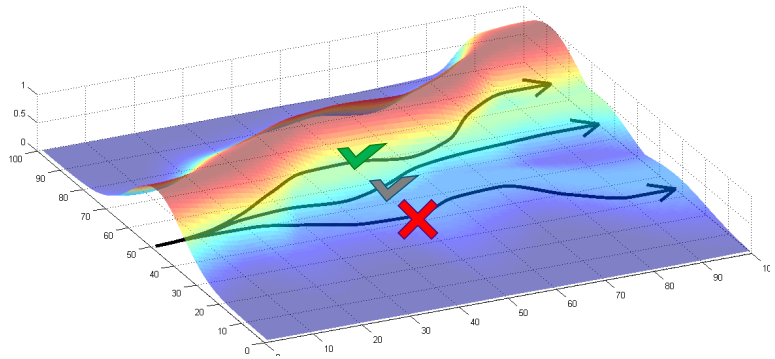
$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \underbrace{\nabla_\theta \log \pi_\theta(\tau_i)}_{\sum_{t=1}^{T} \nabla_\theta \log_\theta \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} r(\tau_i)$$

maximum likelihood: $\nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau_i)$

good stuff is made more likely

bad stuff is made less likely

simply formalizes the notion of “trial and error”!



REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) \right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) \right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Causal policy gradient

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

Causality: policy at time t' cannot affect reward at time t when $t < t'$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \underbrace{\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)}$$

"reward to go"

often denoted $\hat{Q}_{i,t}$

“On-Policy” Learning

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

- The policy gradient increases the likelihood of those past actions that yielded good eventual utility ***when later actions were generated from the current policy.***

- This means you can only ever compute the policy gradient update on data that is generated *from the current policy.*
  - “On-policy” learning.
  - Expensive in terms of amount of experience required in the environment, because old experience, generated from old policies, is no longer relevant. Need to keep generating fresh new experiences.

REINFORCE is On-Policy

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

this is trouble...

- Neural networks change only a little bit with each gradient step
- On-policy learning can be extremely inefficient!
- Off-policy variants using “importance sampling” are possible.

can’t just skip this!

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

Whither Exploration?

- **Exploration in RL**: Which actions to execute in the world to most efficiently learn an optimal policy?
  - But with on-policy RL, do we really have a choice? Remember, our updates can only be computed from trajectories sampled from the current policy π_θ at each stage of training!
- **Two solutions:**
  - π_θ is inherently stochastic, because it is probabilistic, so it does automatically perform different actions each time it is executed, and therefore induces some exploration.
  - Explicitly add an **"exploration bonus"** to the reward, e.g. entropy

$$r_t \leftarrow r_t + \lambda H\big(\pi_\theta(a_t|s_t)\big)$$

which incentivizes more uncertain policies, inducing more exploration. $\lambda \to 0$ during training.

"Policy Gradient" with Discount Factor γ

With discount factor set to 1, the policy gradient we have seen is:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\left[\sum_{t'=t}^{T} r(s_{i,t},a_{i,t})\right]\right)$$

With non-trivial discount factors, the policy gradient simply changes to:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\left[\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{i,t},a_{i,t})\right]\right)$$

Policy gradient with automatic differentiation

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \underline{\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

How can we compute policy gradients with automatic differentiation?

We need a graph such that its gradient is the policy gradient!

Just implement "pseudo-loss" as a weighted maximum likelihood:

$$\tilde{J}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

cross entropy (discrete) or squared error (Gaussian)

Policy gradient with automatic differentiation

Pseudocode example (with discrete actions):

Maximum likelihood as in behavior cloning:

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# Build the graph:
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = softmax_cross_entropy_with_logits(labels=actions, logits=logits)
loss = reduce_mean(negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

Policy gradient with automatic differentiation

Pseudocode example (with discrete actions):

Policy gradient:

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# q_values – (N*T) x 1 tensor of estimated state-action values
# Build the graph:
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = softmax_cross_entropy_with_logits(labels=actions, logits=logits)
weighted_negative_likelihoods = multiply(negative_likelihoods, q_values)
loss = reduce_mean(weighted_negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

$$\tilde{J}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

q_values

Policy gradient in practice

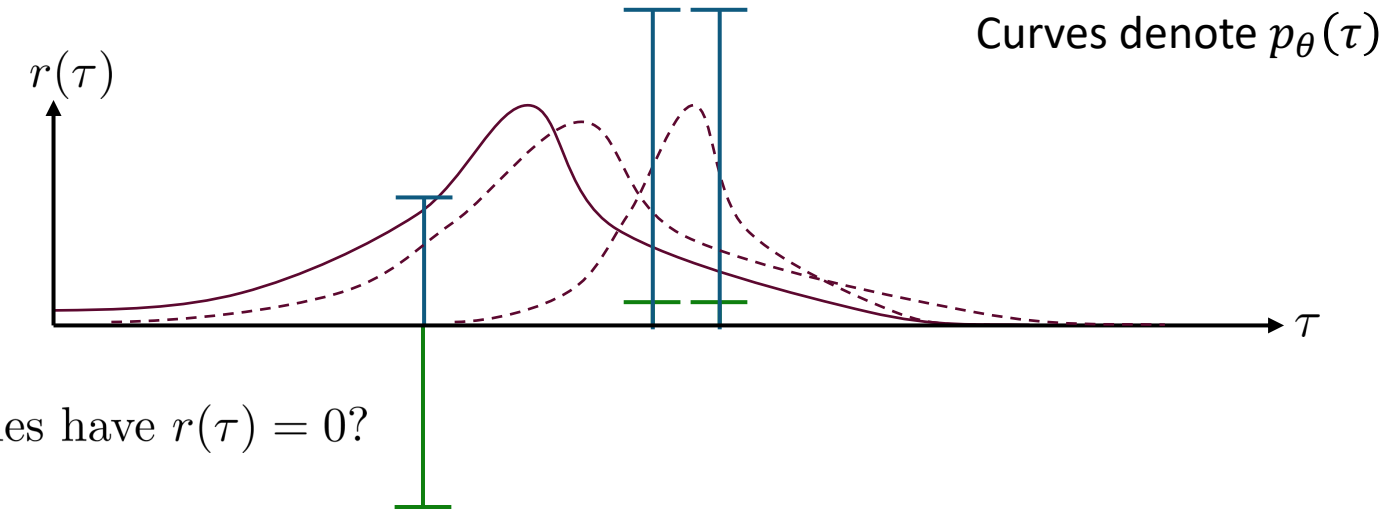
- Remember that the “policy gradient” of expected utility has high variance.
  - Expected utility is estimated by sampling a small number of trajectories from the policy.
  - This isn’t the same as supervised learning!
  - Gradients are often very noisy!
- Consider using much larger batches to reduce the variance
- Tweaking learning rates is very hard
  - Adaptive step size rules like ADAM can be OK-ish
  - We’ll learn about policy gradient-specific learning rate adjustment methods later!
- Popular policy gradient approaches today: PPO, TRPO …
- RL implementation details can be hard to get right. Good to start with **popular repositories: OpenAI stable-baselines, CleanRL etc.**

Sensitivity To Constant Reward Offsets

Consider what should happen if rewards for an MDP were all incremented by a constant value r. Should the optimal policy change? Should a policy gradient change?

No to both, but look at the current policy gradient!

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau) r(\tau)$$



even worse: what if the two "good" samples have $r(\tau) = 0$?

Baselines

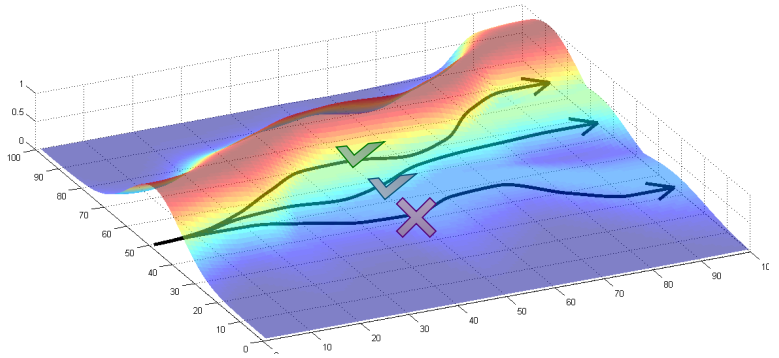
a convenient identity

$$\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) = \nabla_\theta \pi_\theta(\tau)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau)[r(\tau) - b]$$

$$b = \frac{1}{N}\sum_{i=1}^{N} r(\tau)$$

but… are we *allowed* to do that??



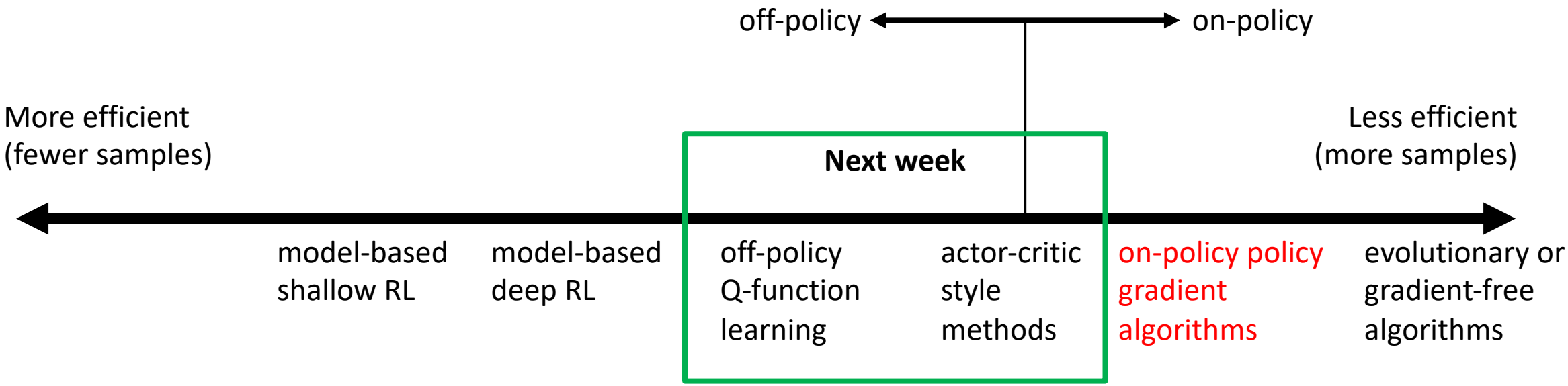
$$E[\nabla_\theta \log \pi_\theta(\tau) b] = \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) b \, d\tau = \int \nabla_\theta \pi_\theta(\tau) b \, d\tau = b\nabla_\theta \int \pi_\theta(\tau) d\tau = b\nabla_\theta 1 = 0$$

subtracting a baseline is *unbiased* in expectation

- Keeps expectation fixed, but reduces the variance of the policy gradients!

average reward is *not* the best baseline, but it's pretty good!

Other RL Algorithms

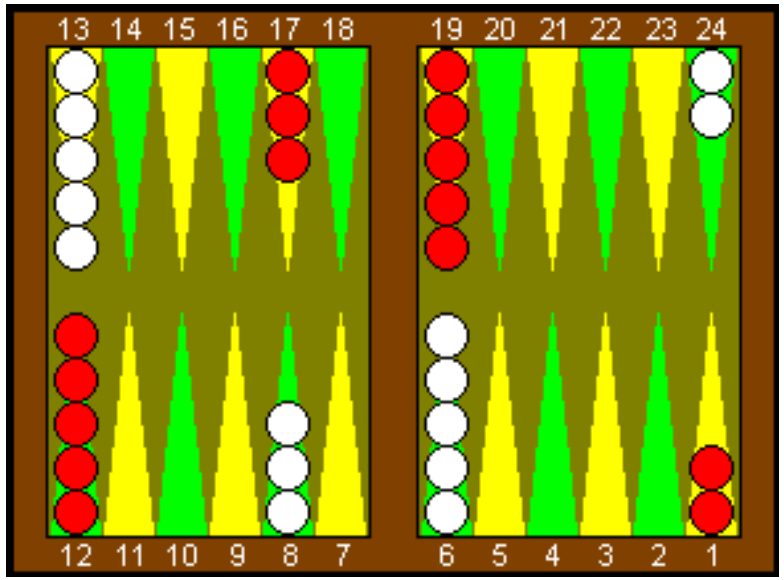


But policy gradients are among the most stable approaches that work most broadly, and take limited wall clock time even though many samples.

Next week: Q learning and actor-critic approaches!

Applications of RL

Test Bed: Video Games and Board Games



TD-Gammon (1992).

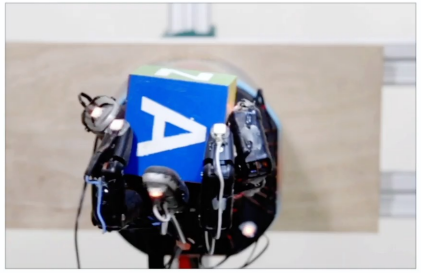
Deep Q Learning (2013)

AlphaGo (2016).

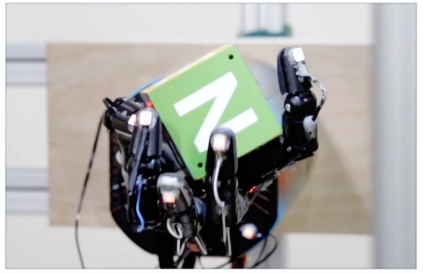
GT Sophy (2023)

Robotics

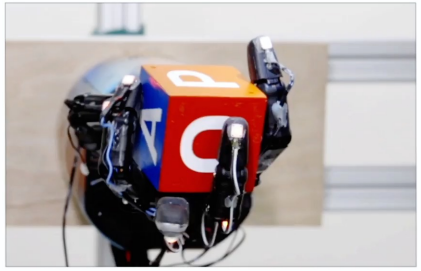
Robotics



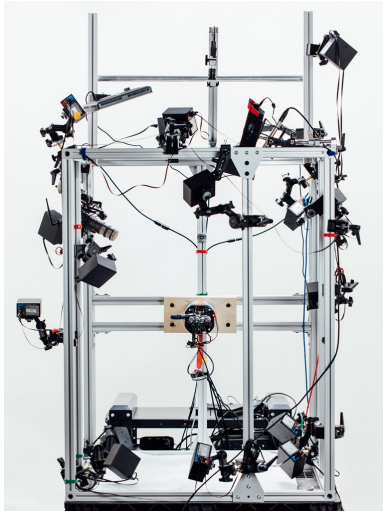
FINGER PIVOTING



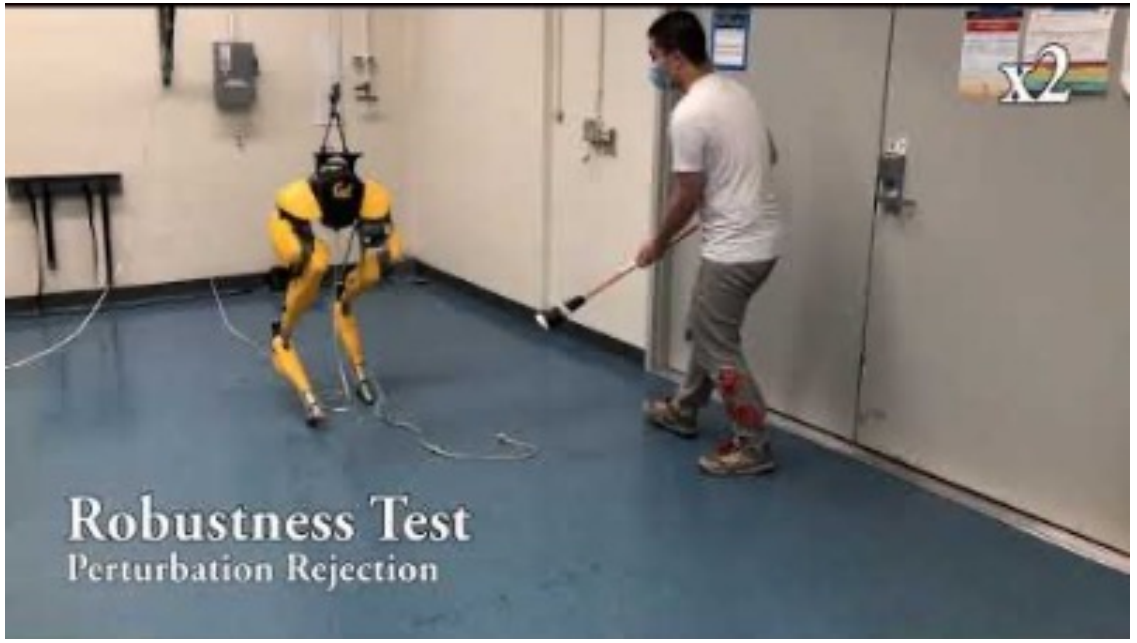
SLIDING



FINGER GAITING



Open AI Dactyl (2018)



Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots, 2022

More RL for Robotics

- Guiness world record in 100 meters by biped robots (Oregon State University)
- Learned quadrupedal locomotion in challenging environments (ETH Zurich)
- Autonomous Navigation of Stratospheric Balloons (Google AI), blog (was real, just Google canceled the whole project.. sadly..)
- Not yet perching (article), but soon? Just for inspiration..
- Video games; car racing in video games, competing with humans

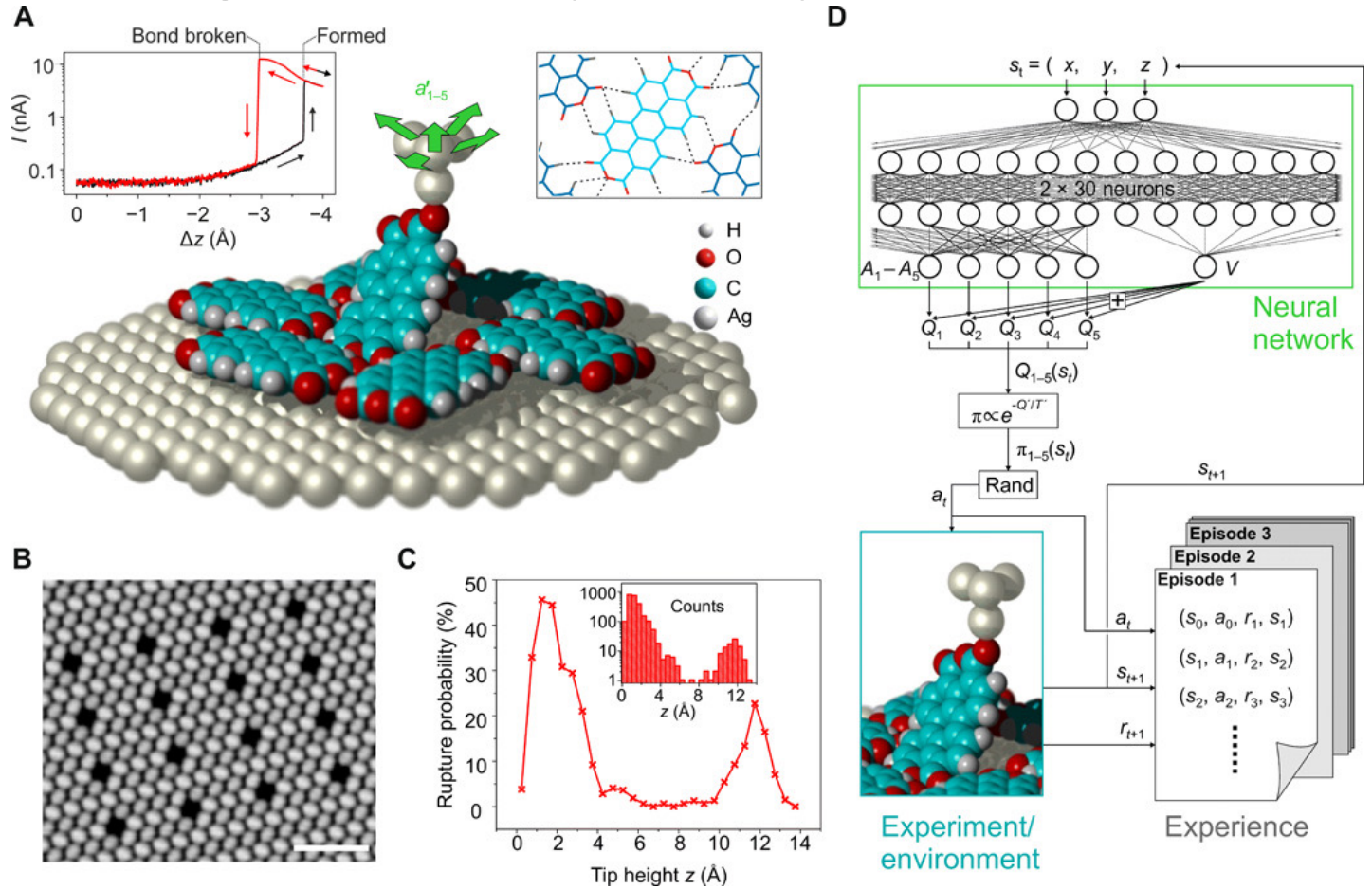
- Vision-based autonomous drone racing (video, UZH RPG)
- Commanding robots using natural language to perform tasks (SayCan project, Google)
- behavioral cloning/imitation learning (not RL) is doing well with transformers in the kitchen (Google)
- Yet it is not enough to learn to drive well
- Quadruped learns to walk in the park in 20 minutes, model-free (UC Berkeley)
    - More of this
- Still, dexterous manipulation is not easy.. (Berkeley, Meta, UW)
- Visual Navigation (Berkeley)
- In the need for resets (Berkeley)

Credit: Csaba Szepasvari

Subtractive Manufacturing

Microscopy

(steering a microscope to separate molecules)



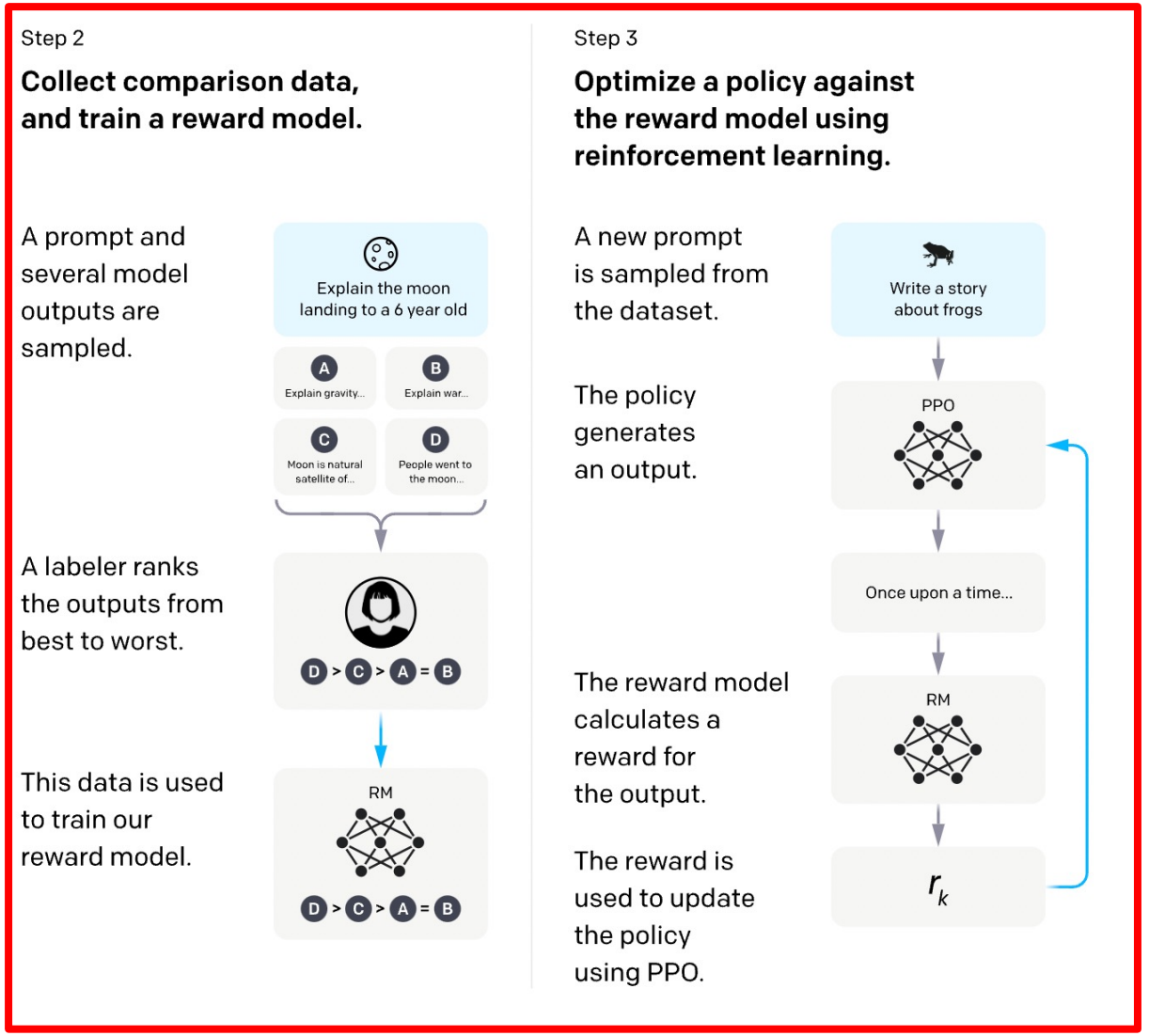
https://www.science.org/doi/10.1126/sciadv.abb6987

Finetuning Conversational Agents

ES

Briefly explain chatGPT using a metaphor that would resonate with an audience of stock-market investors and avoiding AI jargon.

One way to explain chatGPT to stock-market investors is to compare it to a financial advisor who uses past performance data and market trends to make predictions about future investments. Just like a skilled financial advisor, chatGPT uses large amounts of training data to generate intelligent and informed responses to questions, providing valuable insights and suggestions to users.

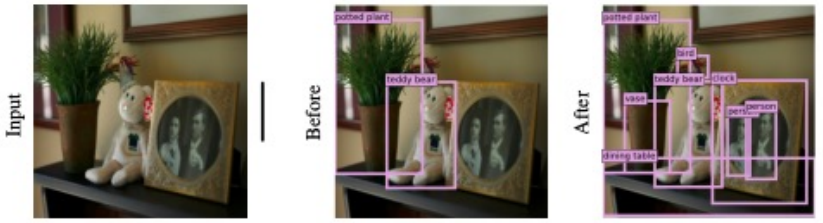


https://openai.com/research/instruction-following

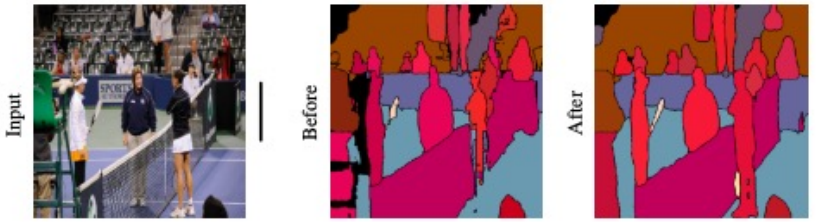
Finetuning Other Pretrained ML Models

- Like ChatGPT, can also use RL to finetune other models to maximize some performance score.

Aside: Why would these models not have been trained directly to maximize the performance scores in the first place?



(a) Optimize mAP: 39 → 54, results in a much high recall and learns box prediction confidences.



(b) Optimize PQ: 43.1 → 46.1, removes many incoherent predictions, especially for small-scale objects.

(c) Optimize "colorfulness" score: 0.41 → 1.79, improves color diversity and saturation.

Figure 1. By tuning a strong, pretrained model with a reward that relates to the task, we can significantly improve the model's alignment with the intended usage.

https://arxiv.org/pdf/2302.08242.pdf

Web Assistants

Web navigation
(e.g. navigating a flight-booking website to make a purchase)

Gur et al 2021, Environment Generation for Zero-Shot Compositional Reinforcement Learning

Real-world applications using RL: Already working

- Applications to algorithms:
  - Video compression on Youtube using nuzero (DeepMind)
  - Faster matrix multiplication (blog, article) (DeepMind)
  - Faster std::sort in the LLVM compiler toolchain, background:, the new part (DeepMind)
  - Chip design applied to Google TPUs (Google AI)
- Industrial automation:
  - Cooling the interior of large commercial buildings (DeepMind), (Vector&Telus, prelim)
  - Amazon “deep” inventory management

Credit: Csaba Szepasvari

Real-world applications using RL: In the works

- Control: Nuclear fusion (DeepMind)
- Education (2021 paper)
- Healthcare (2020 survey)
- Power grids: Reinforcement learning for demand response: A review of algorithms and modeling techniques José R. Vázquez-Canteli, Z. Nagy
- Recommender systems: https://github.com/google-research/recsim
- Automated stock trading: https://github.com/AI4Finance-LLC/FinRL-Library
- And many other "real-world" applications
  - https://arxiv.org/abs/1904.12901
  - https://arxiv.org/abs/2202.11296

RL is not yet "just working", but there is hope. Several important open problems with potential for impact in large numbers of applications!

Credit: Csaba Szepasvari

Initial applications of reinforcement learning span most, if not all, industries.

● Optimizing product development cycles (AI-assisted design) ● Optimizing complex operations ● Informing next best action for each customer

Industry	Sample reinforcement learning applications
Advanced electronics and semiconductors	● Optimize silicon and chip design to increase performance and reduce manufacturing costs ● Optimize fabrication manufacturing process for improved yield and throughput
Agriculture	● Solve scheduling and production allocation challenges to increase yield ● Optimize network and warehouse logistics for reduced waste and costs ● Apply advanced pricing and promotion to improve product margins
Aerospace and defense	● Optimize engineering design processes to reduce time to market for new systems and improve quality
Automotive	● Optimize design processes to shorten development cycle for new cars and features and improve quality ● Deploy advanced predictive maintenance to prevent rare failures and unplanned outages ● Deliver real-time production monitoring and controls to increase manufacturing yield
Financial services	● Apply real-time trading and pricing strategies for greater agility and revenue ● Optimize ATM replenishment and allocation strategies to reduce costs and improve the customer experience ● Deliver advanced personalization capabilities that adapt promotions, offers, and recommendations daily for increased customer satisfaction and sales
Mining	● Optimize design process so teams can explore a greater range of mine designs for improving mine yield ● Use intelligent process controls for managing power generation and bore milling to increase yield and reduce costs ● Apply holistic logistics scheduling to optimize mine-to-shipping operations and reduce costs
Oil and gas	● Enable real-time well monitoring and precision drilling for increased yield ● Optimize tanker routing to reduce costs and ensure on-time delivery ● Enable advanced predictive maintenance to prevent rare equipment failures and unplanned outages
Pharmaceuticals	● Optimize drug discovery, identifying molecules of interest faster to reduce the time and cost of research and bring new therapies to market faster ● Automate chemistry, manufacturing, and controls (CMC) to maximize batch yield and quality ● Optimize biological methods to reach peak production output
Retail	● Optimize routing, logistics network planning, and warehouse operations to reduce costs and keep shelves stocked ● Implement advanced inventory modeling and digitize supply-chain planning to prevent out-of-stocks and waste ● Deliver advanced personalization capabilities that adapt promotions, offers, and recommendations daily for increased customer satisfaction and sales
Telecom	● Optimize network layout to maximize coverage and minimize power consumption ● Manage networks in real time to optimize service quality and reduce downtime ● Apply advanced personalization to increase cross-sell and upsell revenue
Transport and logistics	● Optimize routing, logistics network planning, and warehouse op costs and improve customer satisfaction ● Optimize inbound and outbound delivery networks to minimize s associated costs

McKinsey

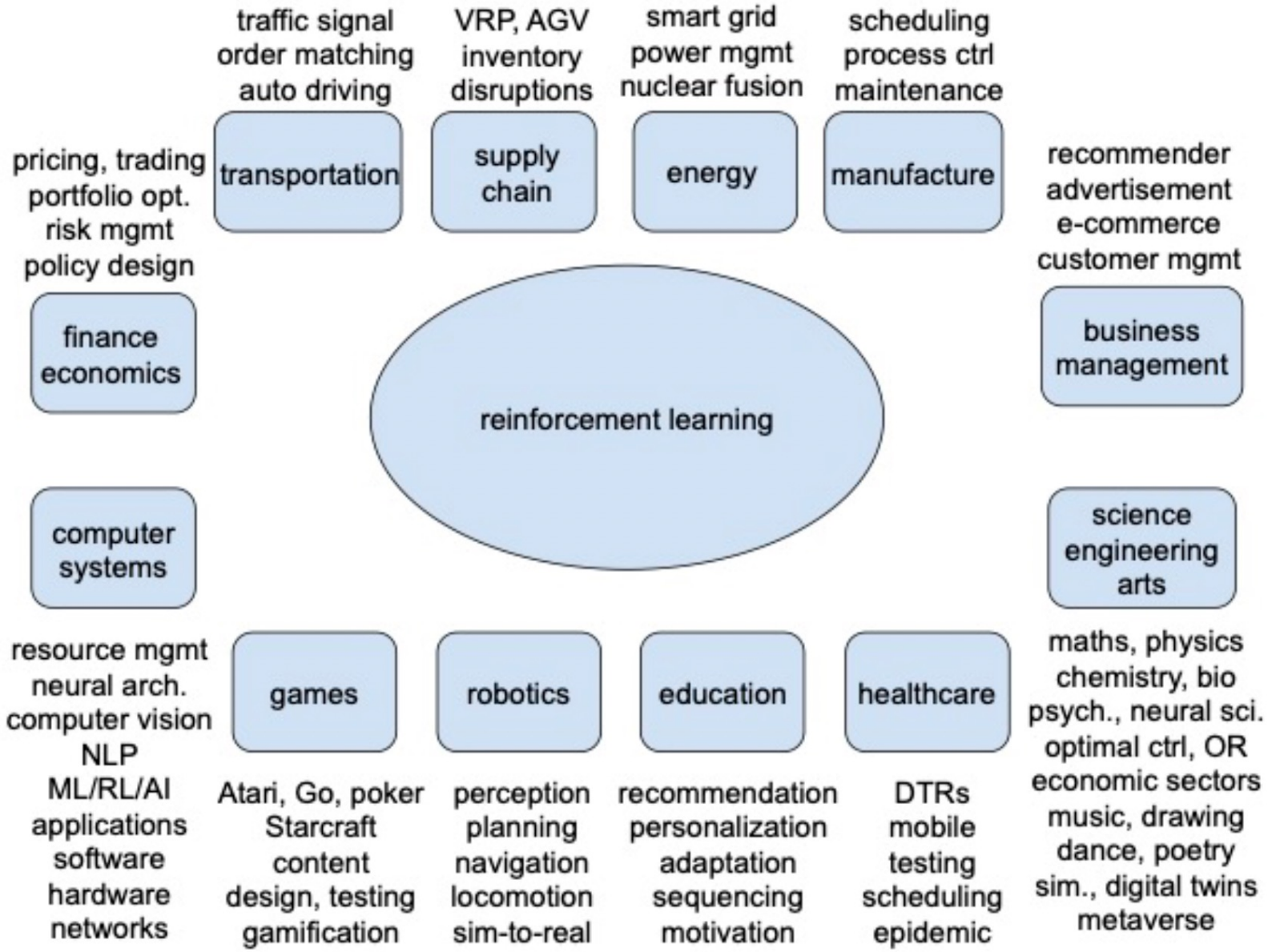
Reinforcement Learning at Work

How top companies are using this breed of AI to solve tough problems.

Company	Application	Sector	Inputs	Actions	Objective
Royal Bank of Canada	Trade execution platform for multiple strategies	Financial services	200-plus market-related data inputs	Sell, buy, hold stocks	To trade as close as possible to VWAP, a common price metric
Netflix	Test schedules for business partner devices	Technology	Historical test and device performance information	Which test to do next	Minimize device failure
Spotify	Recommendation engine	Entertainment	Previous songs liked/disliked/not played	Which songs to put in your playlist	Maximize user listening time
JPMorgan Chase	Financial derivatives risk and pricing calculations	Financial services	Historical market data	Price and sell a financial product	Maximize future cash flows of an investment portfolio
Google	Data center cooling	Technology	Temperature/air pressure	Turn on fan; add water to air unit	Control temperature and reduce energy usage
DiDi	Order dispatching	Ride hailing	Number of idle vehicles, number of orders, location, destination	Match driver to passenger	Minimize pickup time and maximize revenue

Note: Details for use cases can be found in published papers, but we could not verify if they are used in production applications.

HBR



Credit: Yuxi Li