Reinforcement Learning

CS 59300: RL1

August 26, 2025

Joseph Campbell

Department of Computer Science

PURDUE UNIVERSITY®

Today’s lecture

1. A little about me
2. What is reinforcement learning?
3. Course logistics
4. Intro to reinforcement learning

Some content inspired by Katerina Fragkiadaki’s CMU 10-403, Sergey Levine’s Berkeley CS285, and Cathy Wu’s MIT 6.7950 courses

A little about me

BS in CS

2006-2010

Software Engineer

2009-2015

MS in CE + PhD in CS

2013-2021

Postdoc

2021-2024

Assistant Professor

As of Fall 2024

Collaborative **AI** for **M**achines and **P**eople

CAMP Lab

PURDUE UNIVERSITY

PhD Students


Guven Gergerli


Abel Gurung


Muhan Lin


Shahab Rahimirad


Fiona Xie

MS Students

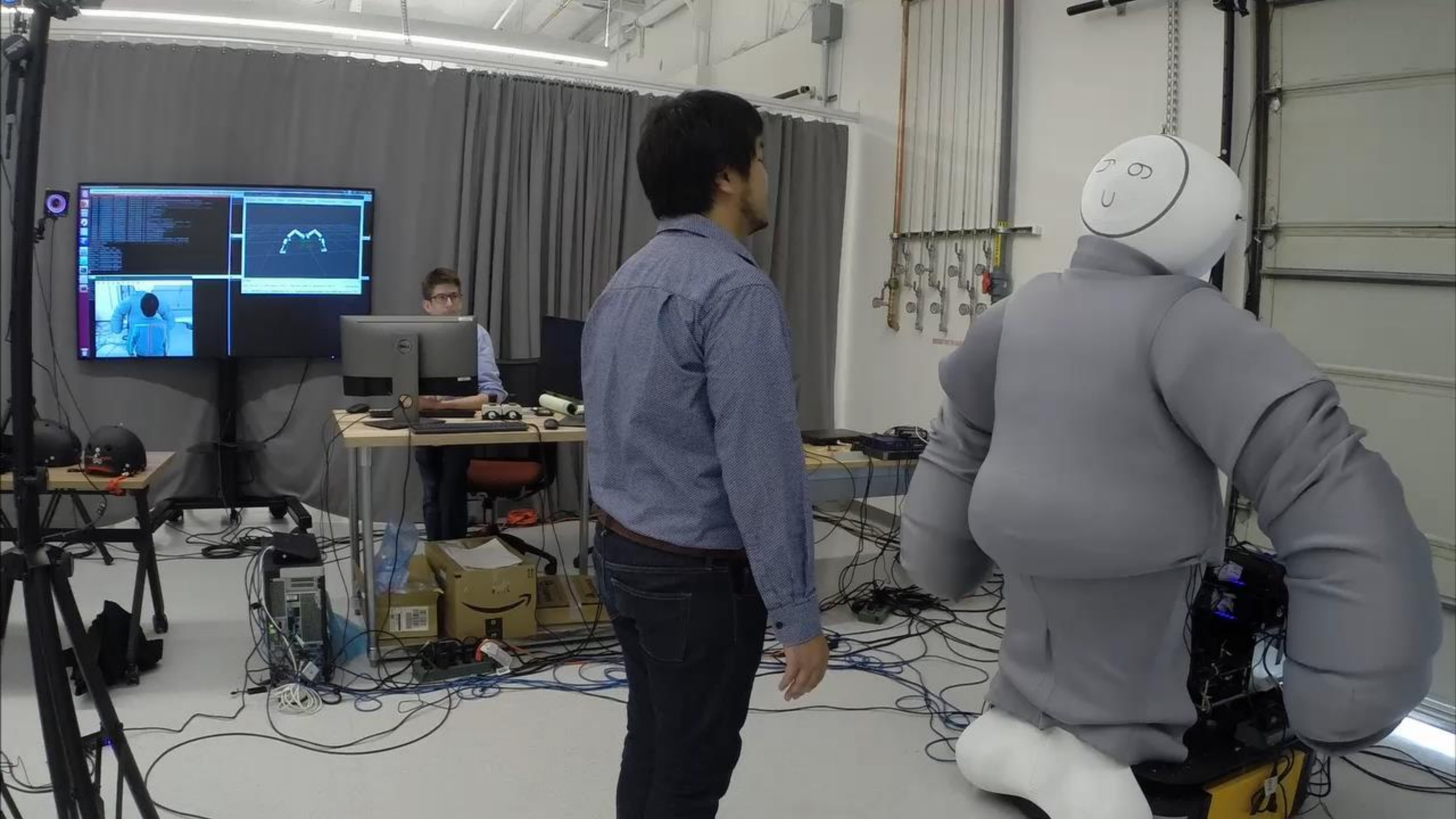

Maheep Brar


Zeyun Deng


Luke Luschwitz


Le Mao

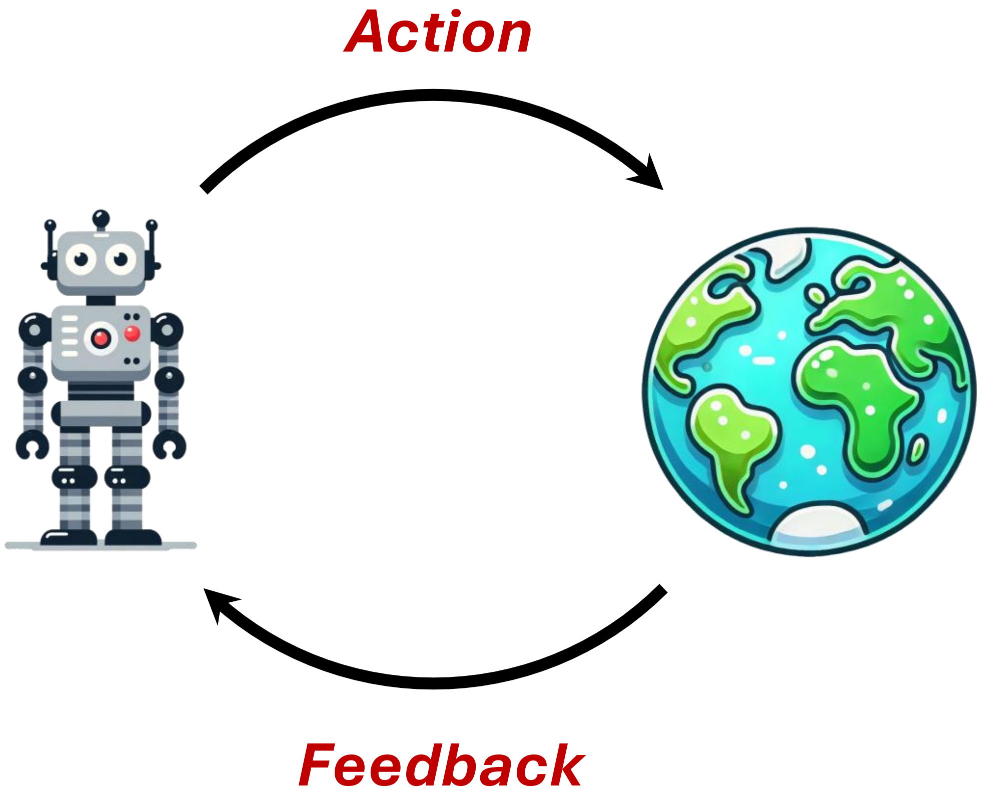

Anusha Sarraf



What is reinforcement learning?


2:11
https://youtu.be/hyNjTrWcn2l?si=jVNLCavRidPba3LO&t=131

Reinforcement learning ≈ trial-and-error!



1. Take some action.

2. Get feedback (a reward).

3. Use feedback to adjust what action we will take next time.

Instrumental conditioning

Law of effect

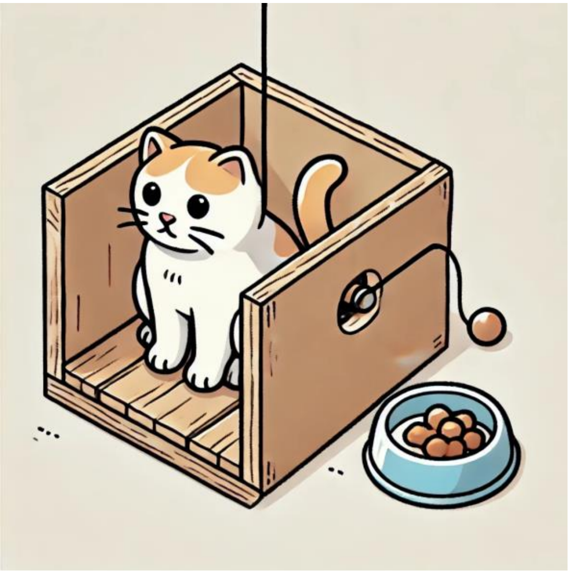
"...responses that produce a satisfying effect in a particular situation become more likely to occur again in that situation, and responses that produce a discomforting effect become less likely to occur again in that situation."

Edward Thorndike
Psychologist @ Columbia

"The cat that is clawing all over the box in her impulsive struggle will probably claw the string or loop or button so as to open the door.

And gradually all the other non-successful impulses will be stamped out and the particular impulse leading to the successful act will be stamped in by the resulting pleasure, until, after many trials, the cat will, when put in the box, immediately claw the button or loop in a definite way."

Edward Thorndike's famous puzzle box.



Terrible illustration courtesy of ChatGPT.

Operant conditioning

Reinforcement schedule

"...any procedure that delivers reinforcement to an organism according to some well-defined rule."

Record effect of schedule on animal's learning rate.

B.F. Skinner
Psychologist @ Harvard

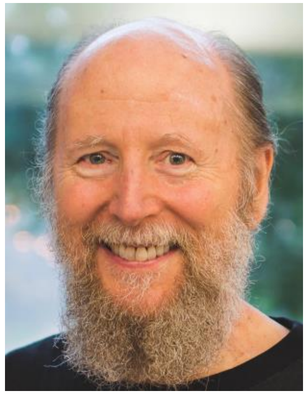
https://youtu.be/yhvaSEJtOV8?si=9ojzWQ9txtAhMkgC

Goal: take actions that maximize total reward

"Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal.

The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them.

In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards."



Richard Sutton
University of Alberta

Andrew Barto
UMass Amherst

Types of feedback: from environment

Types of feedback: from teacher

Types of feedback: from ourselves

Frequency of feedback

Reward shaping

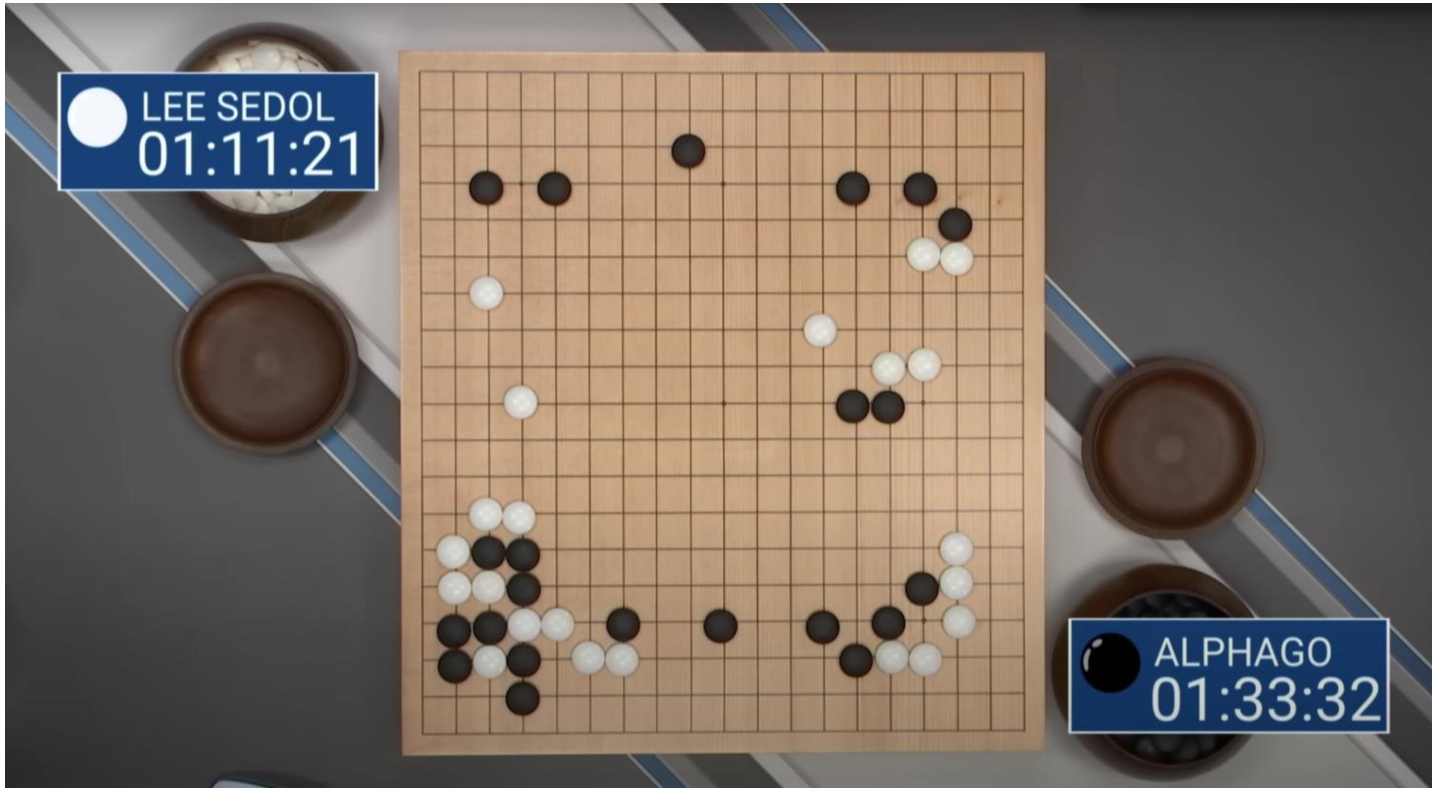
"…decided to reinforce any response that had the slightest resemblance to a swipe—perhaps, at first, merely the behavior of looking at the ball—and then to select responses which more closely approximated the final form.

The result amazed us. In a few minutes, the ball was caroming off the walls of the box as if the pigeon had been a champion squash player."

B.F. Skinner
Psychologist @ Harvard

Why do we care?

We can use this paradigm to build agents that **learn** to **act**!



2002

2016

“No simple yet reasonable evaluation function will ever be found for Go.” - Martin Müller

ELSEVIER

Artificial Intelligence 134 (2002) 145–179

Artificial Intelligence

www.elsevier.com/locate/artint

Computer Go

Martin Müller

Department of Computing Science, University of Alberta, Edmonton, Canada T6G 2E8

Abstract

Computer Go is one of the biggest challenges faced by game programmers. This survey describes the typical components of a Go program, and discusses knowledge representation, search methods and techniques for solving specific subproblems in this domain. Along with a summary of the development of computer Go in recent years, areas for future research are pointed out.

Keywords: Computer Go; Go programs; Game tree search; Knowledge representation

1. Introduction

Of all games of skill, Go is second only to chess in terms of research effort and programming time spent. Yet in playing strength, Go programs lag far behind their counterparts in any other popular board game. While Go programs have advanced considerably in the last 10–15 years, they can still be beaten easily even by human players of only moderate skill. What are the reasons for this state of affairs, and what can we do about it? This survey presents an overview of the specific challenges posed by the domain of computer Go, and the approaches to their solution that have been developed over the years. It also provides a wealth of references for those who want to pursue this fascinating topic deeper.

ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses 'value networks' to evaluate board positions and 'policy networks' to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

All games of perfect information have an optimal value function, $v^*(s)$, which determines the outcome of the game, from every board position or state s, under perfect play by all players. These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where b is the game's breadth (number of legal moves per position) and d is its depth (game length). In large games, such as chess ($b \approx 35, d \approx 80$)[1] and especially Go ($b \approx 250, d \approx 150$)[1], exhaustive search is infeasible[2,3], but the effective search space can be reduced by two general principles. First, the depth of the search may be reduced by position evaluation: truncating the search tree at state s and replacing the subtree below s by an approximate value function $v(s) \approx v^*(s)$ that predicts the outcome from state s. This approach has led to superhuman performance in chess[4], checkers[5] and othello[6], but it was believed to be intractable in Go due to the complexity of the game[7]. Second, the breadth of the search may be reduced by sampling actions from a policy $p(a|s)$ that is a probability distribution over possible moves a in position s. For example, Monte Carlo rollouts[8] search to maximum depth without branching at all, by sampling long sequences of actions for both players from a policy p. Averaging over such rollouts can provide an effective position evaluation, achieving superhuman performance in backgammon[8] and Scrabble[9], and weak amateur level play in Go[10].

Monte Carlo tree search (MCTS)[11,12] uses Monte Carlo rollouts to estimate the value of each state in a search tree. As more simulations are executed, the search tree grows larger and the relevant values become more accurate. The policy used to select actions during search is also improved over time, by selecting children with higher values. Asymptotically, this policy converges to optimal play, and the evaluations converge to the optimal value function[12]. The strongest current Go programs are based on MCTS, enhanced by policies that are trained to predict human expert moves[13]. These policies are used to narrow the search to a beam of high-probability actions, and to sample actions during rollouts. This approach has achieved strong amateur play[13–15]. However, prior work has been limited to shallow policies[13–15] or value functions[16] based on a linear combination of input features.

Recently, deep convolutional neural networks have achieved unprecedented performance in visual domains: for example, image classification[17], face recognition[18], and playing Atari games[19]. They use many layers of neurons, each arranged in overlapping tiles, to construct increasingly abstract, localized representations of an image[20]. We employ a similar architecture for the game of Go. We pass in the board position as a 19×19 image and use convolutional layers to construct a representation of the position. We use these neural networks to reduce the effective depth and breadth of the search tree: evaluating positions using a value network, and sampling actions using a policy network.

We train the neural networks using a pipeline consisting of several stages of machine learning (Fig. 1). We begin by training a supervised learning (SL) policy network p_σ directly from expert human moves. This provides fast, efficient learning updates with immediate feedback and high-quality gradients. Similar to prior work[13,15], we also train a fast policy p_π that can rapidly sample actions during rollouts. Next, we train a reinforcement learning (RL) policy network p_ρ that improves the SL policy network by optimizing the final outcome of games of self-play. This adjusts the policy towards the correct goal of winning games, rather than maximizing predictive accuracy. Finally, we train a value network v_θ that predicts the winner of games played by the RL policy network against itself. Our program AlphaGo efficiently combines the policy and value networks with MCTS.

Supervised learning of policy networks

For the first stage of the training pipeline, we build on prior work on predicting expert moves in the game of Go using supervised learning[13,21–24]. The SL policy network $p_\sigma(a|s)$ alternates between convolutional layers with weights σ, and rectifier nonlinearities. A final softmax layer outputs a probability distribution over all legal moves a. The input s to the policy network is a simple representation of the board state (see Extended Data Table 2). The policy network is trained on randomly

Robotics

D’Ambrosio et al. Achieving Human Level Competitive Robot Table Tennis. 2024.
https://youtu.be/EqQl-JQxToE?si=sRpkHHcF2UPJF5lC

Recommendation engines

What artwork is shown to a user in order to maximize watch-rate?

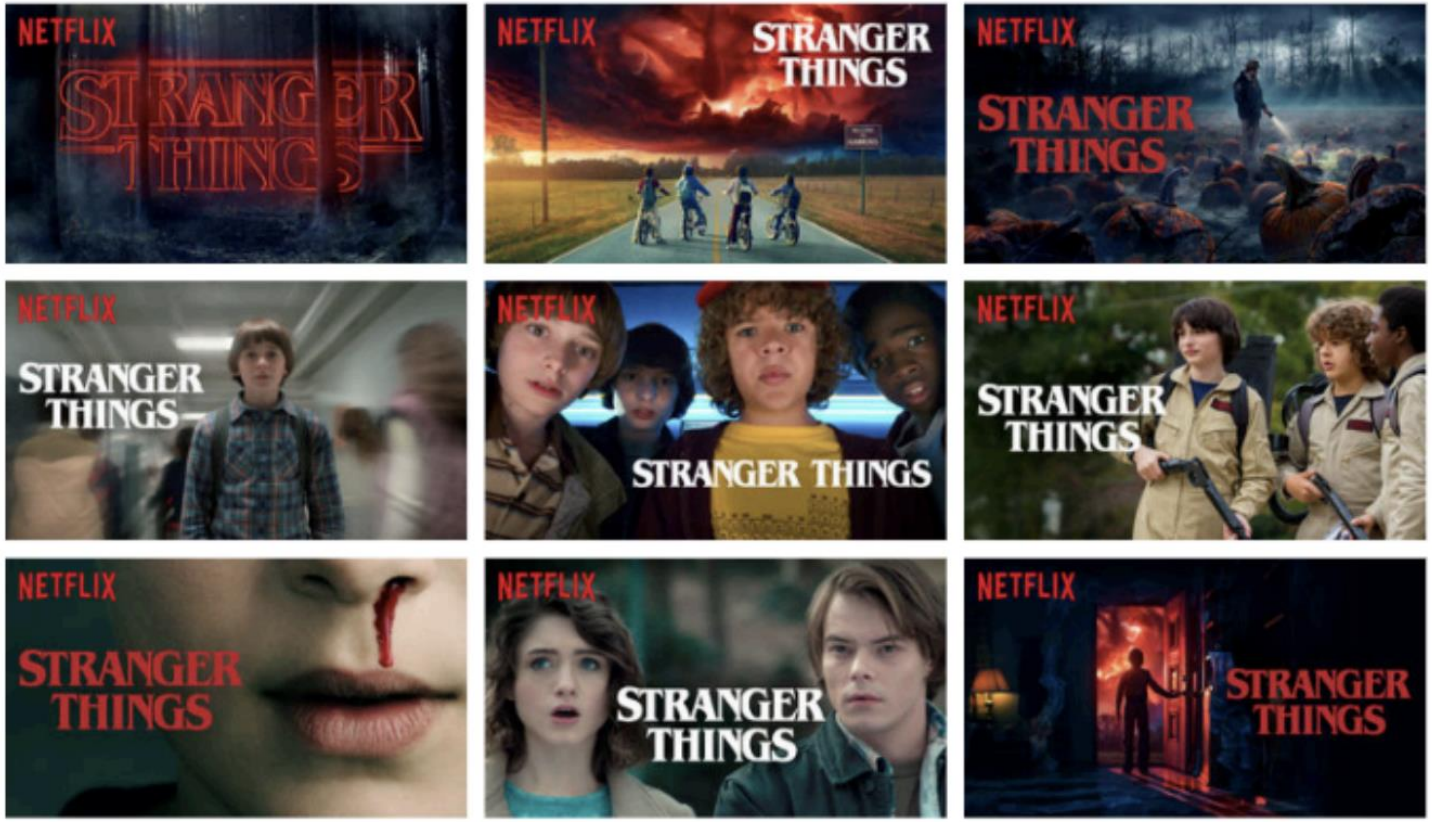
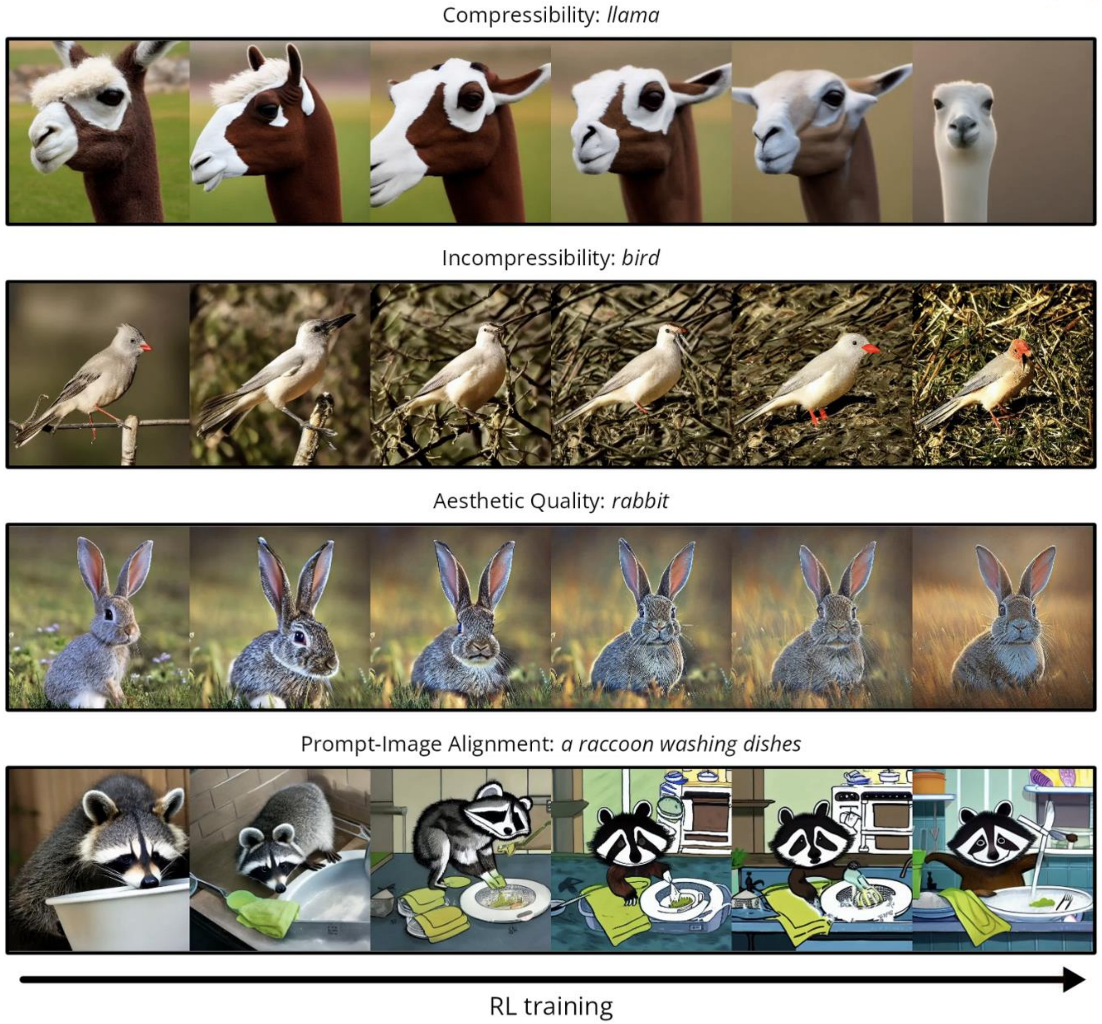


Image generation

How can we generate human-like images?

Black et al. Training Diffusion Models with Reinforcement Learning. 2024.



Large language models

How do we make LLMs generate human-sounding text?

Can you tell me how reinforcement learning from human feedback works?

Reinforcement Learning from Human Feedback (RLHF) is a technique that leverages human input to guide and improve the learning process of a reinforcement learning (RL) agent. The core idea is to use human feedback to shape the reward function or to directly guide the agent's behavior, enabling it to learn more effectively and align its actions with human values or preferences.

Here's a breakdown of how RLHF works:

1. Traditional Reinforcement Learning Basics

- **Agent and Environment:** In a standard RL setup, an agent interacts with an environment to maximize cumulative rewards over time. The agent takes actions, the environment responds with a new state, and the agent receives a reward.
- **Reward Function:** The reward function defines what the agent should strive to achieve. It typically comes from the environment and is designed by the developers. The agent learns a policy, which is a strategy that tells it what actions to take in different states.

2. Human Feedback in RLHF

- **Human-in-the-Loop:** In RLHF, humans are introduced into the loop to provide feedback that

In this course, you will learn…

How to build agents that learn to act!

- How reinforcement learning works
- What types of algorithms exist
- When they should be used
- How we can use reinforcement learning to solve real problems

Course logistics

What, where, when

Will (try to) post all lectures on Brightspace before class and recordings after class

Lecture: TTH 10:30 – 11:45 AM
Room: LWSN 1106

Email: joecamp@purdue.edu
Office hours: F 12:00 – 1:00 PM, DSAI 3047

TA email: ggergerl@purdue.edu
TA office hours: W 11:30 – 12:30 PM, DSAI B063

Please include [593RL] in the subject header

Guven Gergerli

Course structure

Intended to provide a **broad overview** of reinforcement learning

- Focus on important concepts and algorithms
- Build theoretical + practical understanding

Mixture of lectures, homework assignments, and a course project

Later in the semester: a couple of guest lectures

Prerequisites

Introductory-level knowledge of **machine learning** and **linear algebra**
- Do you know what supervised and unsupervised learning is?
- Are you comfortable working with probability distributions?
- Do you understand matrix algebra and other operations (inverse)?

Basic proficiency in **Python**
- *Vast majority* of machine learning libraries are written in Python
- Knowledge of Pytorch will help you greatly

Schedule (Tentative)

Week	Tuesday		Thursday	
1	8/26	Introduction to RL	8/28	Deep Learning Basics and Behavior Cloning
2	9/2	Multi-Armed Bandits & Markov Decision Proc.	9/4	Policy/Value Iteration
3	9/9	Temporal Difference Learning	9/11	Deep Q Learning
4	9/16	Deep Q Learning	9/18	Policy Gradient
5	9/23	Actor-Critic Methods	9/25	Actor-Critic Methods
6	9/30	Model-Based RL	10/2	Model-Based RL
7	10/7	Guest Talk	10/9	RL Framework Tutorial
8	10/14	No Class Fall Break	10/16	Monte Carlo Tree Search
9	10/21	Multi-Agent RL	10/23	Multi-Agent RL
10	10/28	Guest Talk	10/30	Inverse Reinforcement Learning
11	11/4	Offline Reinforcement Learning	11/6	Intelligent Exploration and Curiosity
12	11/11	Intelligent Exploration and Curiosity	11/13	Transfer Learning and Lifelong Learning
13	11/18	Good Presentation Tips	11/20	Large Language Models
14	11/25	Vision Language Models	11/27	No Class Thanksgiving Break
15	12/2	Robotics & Sim-to-Real	12/4	Challenges and Open Problems
16	12/9	Poster Presentations	12/11	Poster Presentations

Grading

Assignments (individual): 40%
- 4x assignments at 10% each

Midterm exam: 15%

Course project (group): 40%
- Midterm update: 10%
- Project report (plus deliverables): 20%
- Poster presentation: 10%

Class participation: 5%

Discussion is highly encouraged!

Ask questions at any time

If you have a question…odds are others do as well

If I ask questions (which I will), please participate

- This is 5% of your grade!

Course project

Goal: gain experience implementing, applying, evaluating, and improving reinforcement learning algorithms

Form groups of 2-4 students

- Tell me your group by **9/5** (end of week 2)

Project proposal must be approved by me

- Proposal due **9/26** (end of week 5)

Course project

Midterm-update due by **10/30** (week 10)

- 2-page research-style paper outlining your project and progress
- IEEE paper format. Recommend using Overleaf

Final project due by **12/11** (week 16)

- $\geq$ 6-page research-style paper outlining your project and findings
- Poster presentation in class on 12/9 and 12/11

Goal is to replicate the lifecycle of a research project

Example project ideas

Implementation-based: implement a deep reinforcement learning algorithm **from scratch** and apply it to a challenging problem, *e.g. robot navigation or manipulation*

- Understand and show me how various hyperparameters and other design choices effect algorithm performance
- If you are interested in robotics or another real-world application, let me know and we can try to find resources to make it work
- Must be a different algorithm/task than in the homeworks!

Example project ideas

Analysis-based: implement multiple algorithms (may use libraries) and apply them to multiple tasks and perform meaningful empirical or theoretical analysis, e.g.

- Try to reproduce a paper's experimental findings
- Compare different variants of curiosity on a benchmark
- Analyze why one algorithm may out-perform another, e.g. when and why does SAC out-perform PPO?

Example project ideas

Improvement-based: identify a weakness with an existing algorithm over a particular benchmark/evaluation metric and introduce a variant of this algorithm which out-performs the original

- This is a classic “research problem” approach
- If you don’t make meaningful improvements, that’s ok! But show me what you tried and why it did/didn’t work

Example project ideas

I will work with you to develop your project ideas!

- If you have an idea but aren't sure if it is viable, come talk to me at office hours or send an email; we can figure something out

If you are interested in research, we can work towards turning your project into a research publication after the semester

Collaboration policy and academic integrity

While discussion and exchanging ideas is encouraged, students and groups must perform their own work separately

- For group projects you may not collaborate outside your group
- For homework assignments you may not collaborate with others

All work must be your own unless otherwise stated

- You may use AI tools (e.g. ChatGPT) as an *assistive tool*
- All uses must be explicitly documented in any submitted work

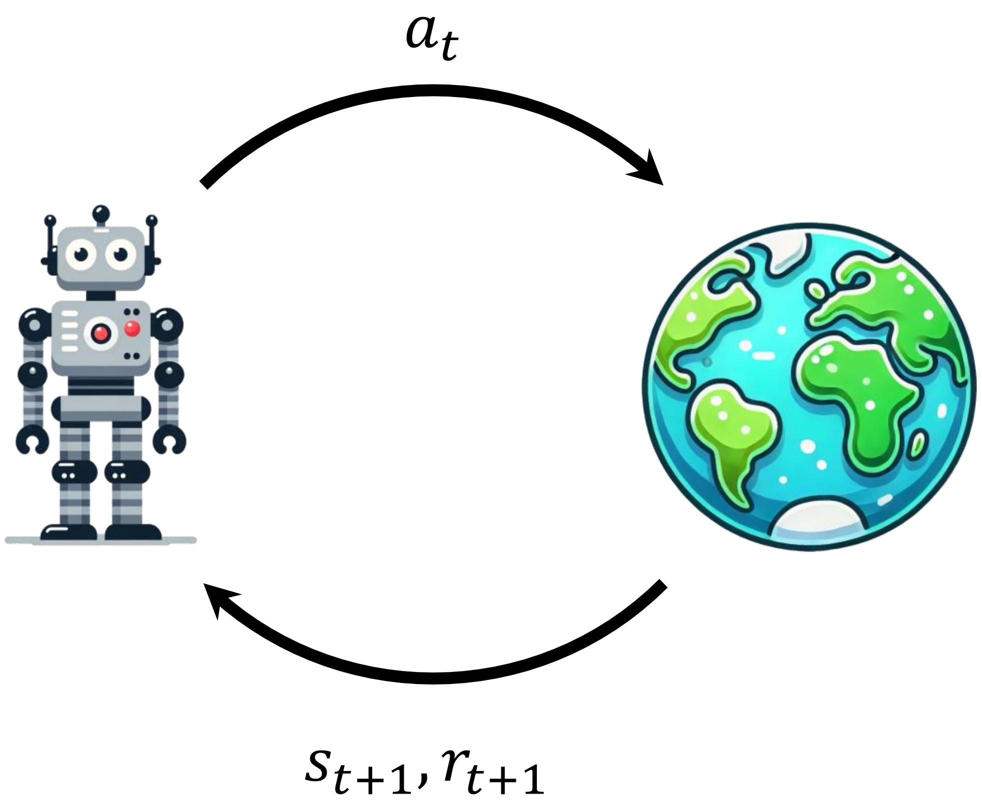
Refer to Purdue's Academic Integrity website

Intro to reinforcement learning

The basics

The agent and environment operate at discrete timesteps $t = 0,1,2, \ldots$

- The agent observes state s_t at time t
- The agent takes action a_t
- The agent gets the resulting reward r_{t+1} and the subsequent state s_{t+1}

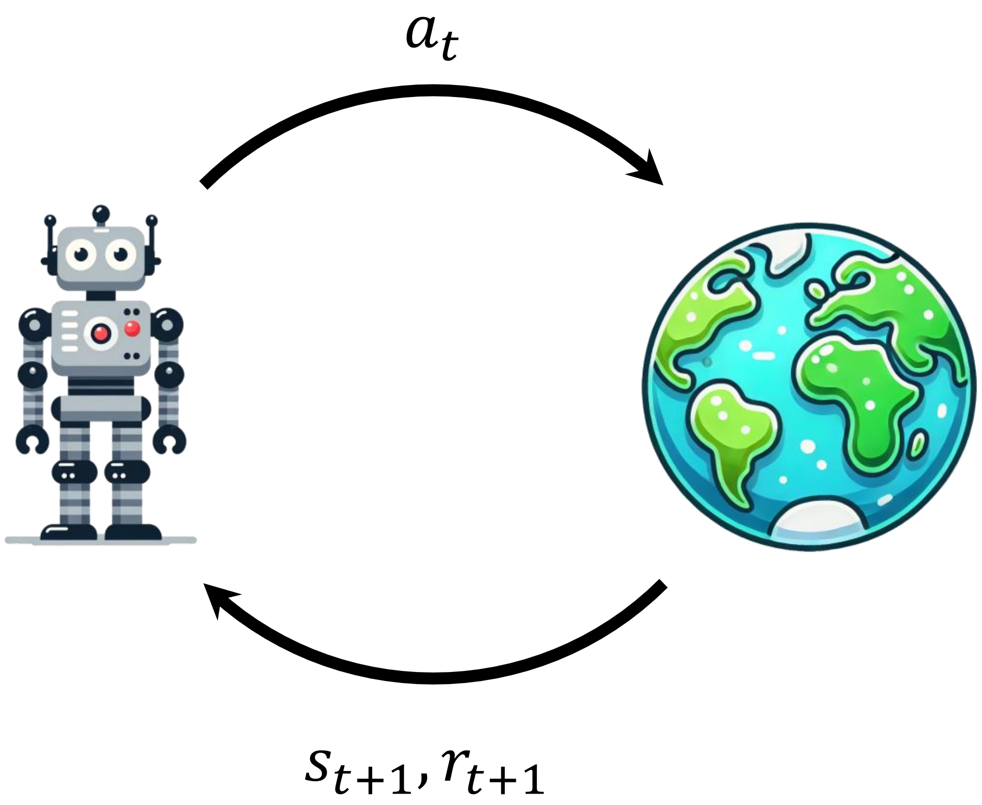


The basics

Action a_t is chosen by sampling actions from a probability distribution

$$a_t \sim \pi(a|s)$$

The probability distribution π is referred to as a policy.



Supervised learning vs reinforcement learning

Supervised/imitation learning

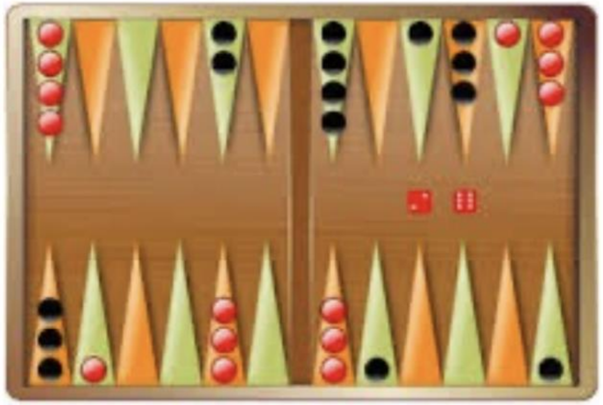
- Agent learns policy that imitates a set of demonstrations

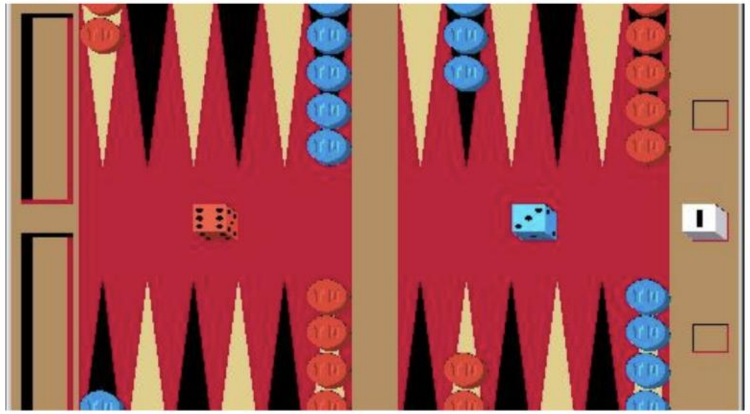
- Learns quickly
- Performance is bounded by the quality of demonstrations
- Agent doesn't know what to do if it makes a mistake

Reinforcement learning

- Agent learns its own policy that maximizes rewards

- Learns slowly
- Can discover novel solutions
- Requires an environment to interact with



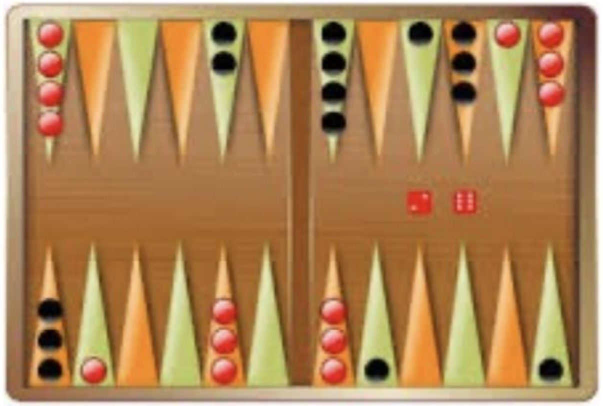


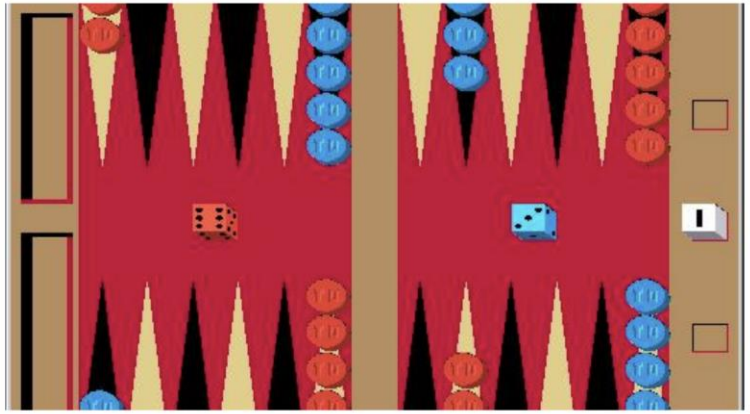
Neurogammon – Supervised

- Developed by Gerald Tesauro in 1989 at IBM Research
- Trained using supervised learning given expert demonstrations
- Intermediate-level play

TD-Gammon – Reinforcement

- Developed by Gerald Tesauro in 1992 at IBM Research
- Trained against itself using self-play beginning with random actions
- Top-level play





Neurogammon – Supervised

TD-Gammon – Reinforcement

- Deve
  in 19
- Train
  learn
  demo

sauro

ing

Why did TD-Gammon play so much better?

Because it discovered novel strategies!

- Intermediate-level play
- Top-level play

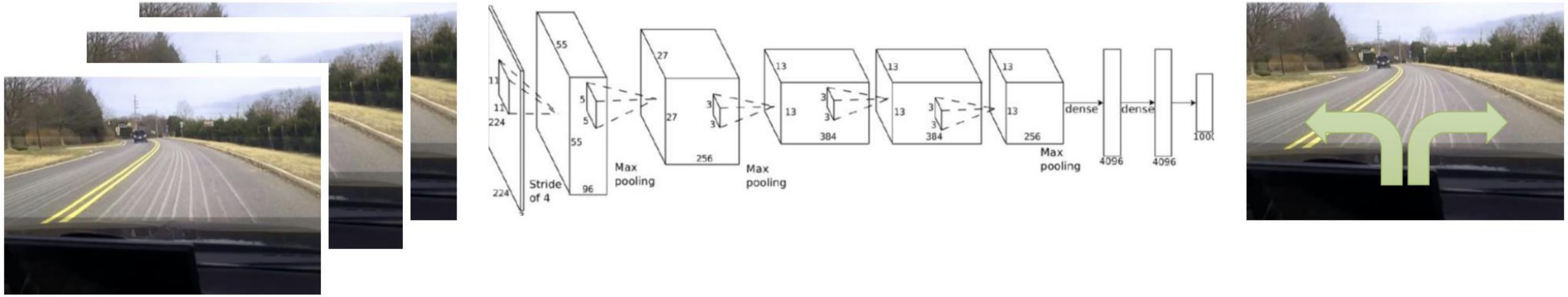
https://youtu.be/WXuK6gekU1Y?si=1ia9W3ve9ulzMZfe&t=2965

49:25

What is deep reinforcement learning?

How can an agent learn over a complex state space, e.g. images?

Policy $\pi(a|s)$ is represented using a deep neural network.



Limitations of reinforcement learning

The agent needs to interact with an environment

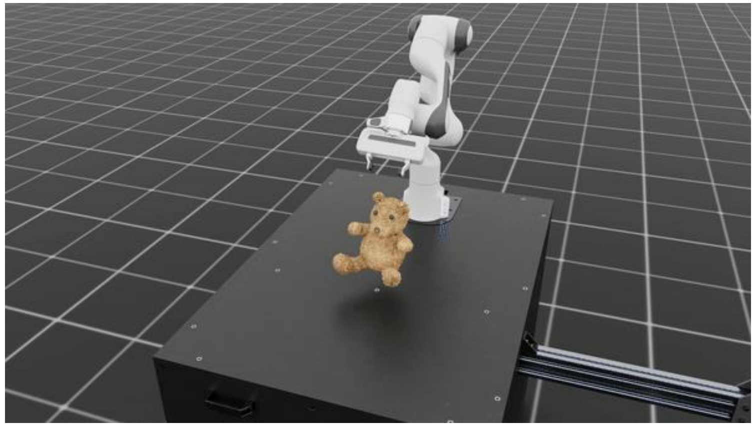
Requires either a simulator or access to the real world

- If simulator:
    - Is it high-enough fidelity to use the policy in the real-world after training?
- If real world:
    - Takes a long time to collect samples + wear and tear (if embodied)
    - May not be safe (can we train autonomous vehicles on city roads?)

Brute force solution: more agents!

Ahn et al. AutoRT: Embodied Foundation Models for Large Scale Orchestration of Robotic Agents. 2024.

Slightly more elegant: fancy simulator!



NVIDIA

Limitations of reinforcement learning

The agent's policy is non-stationary

- Actions depend on agent's policy (bad policy = bad trial-and-error)

What if the agent needs to start from a known "good" state?

- Environment may need to be reset somehow


2:54

Limitations of reinforcement learning

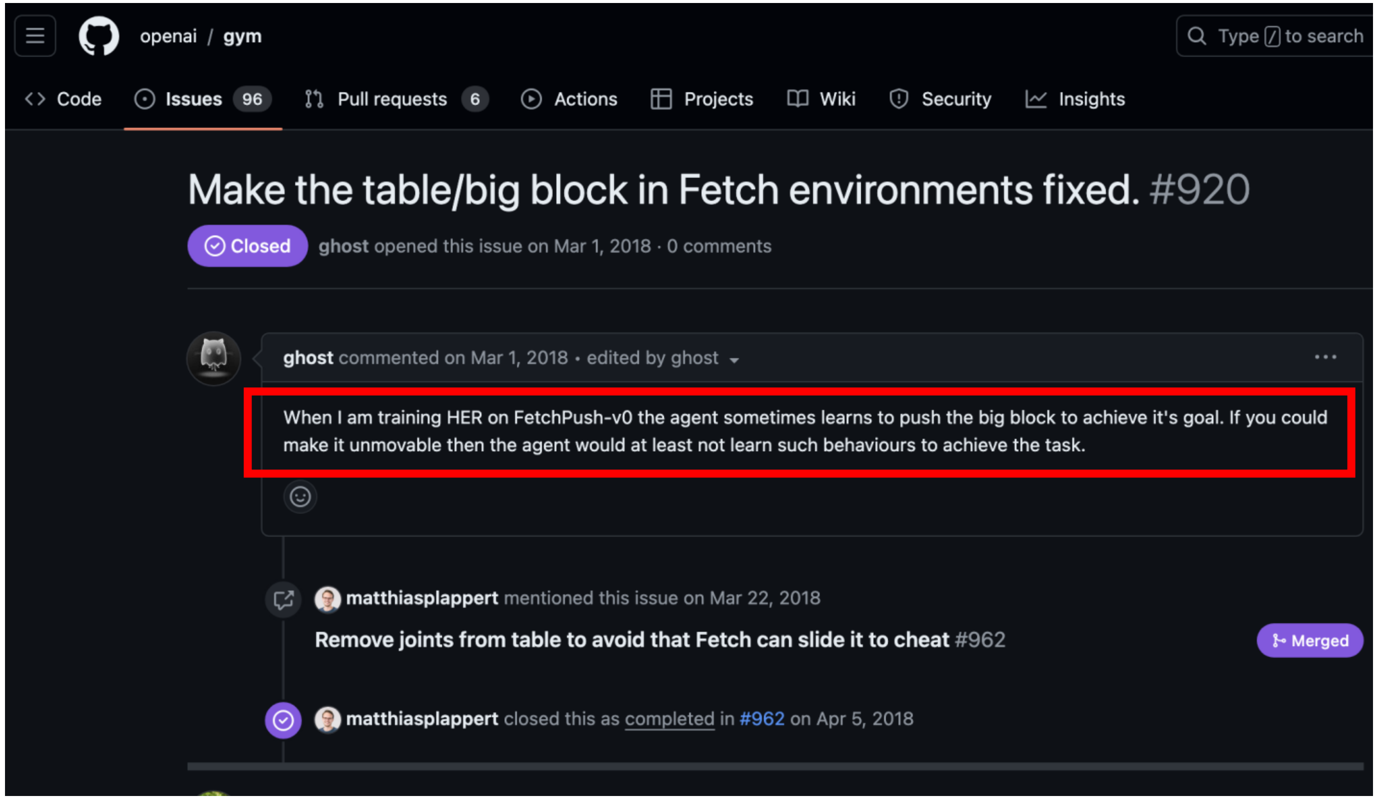
Reward functions may be difficult to specify

- How do you specify social/cultural norms in a reward function?
  - Required for chat bots, social navigation, autonomous driving, …
- How frequently does an agent get a reward?
  - Reward may be difficult to obtain through random exploration
  - Reward may be the result of a long series of actions
  - Hint: remember “reward scheduling” and “reward shaping”?

Reward hacking

Rewards are hard for people to design!

Agents may find “loopholes” to exploit reward process



Skalse et al. Defining and Characterizing Reward Hacking. 2022.

https://openai.com/index/faulty-reward-functions/
https://youtu.be/tlOIHko8ySg?si=bMZ5lpjTqFaoo44j

Take-aways

- In reinforcement learning an agent discovers its own actions which maximize its received reward

- Differs from supervised learning which imitates demonstrations

- Reinforcement learning has many challenging problems