BOSTON UNIVERSITY

Deep Learning for Data Science DS 542

Lecture 25
Reinforcement Learning

Slides originally by Thomas Gardos.

Images from Understanding Deep Learning unless otherwise cited.

Last Lecture of This Course

Previously covered (among others)

- Supervised learning
- Neural Networks
- Convolutional Networks
- Transformers
- Generative Models

Today

- Reinforcement learning

Reinforcement Learning (RL)

An intelligent agent uses reinforcement learning to maximize a sequence of rewards arising from actions over time.

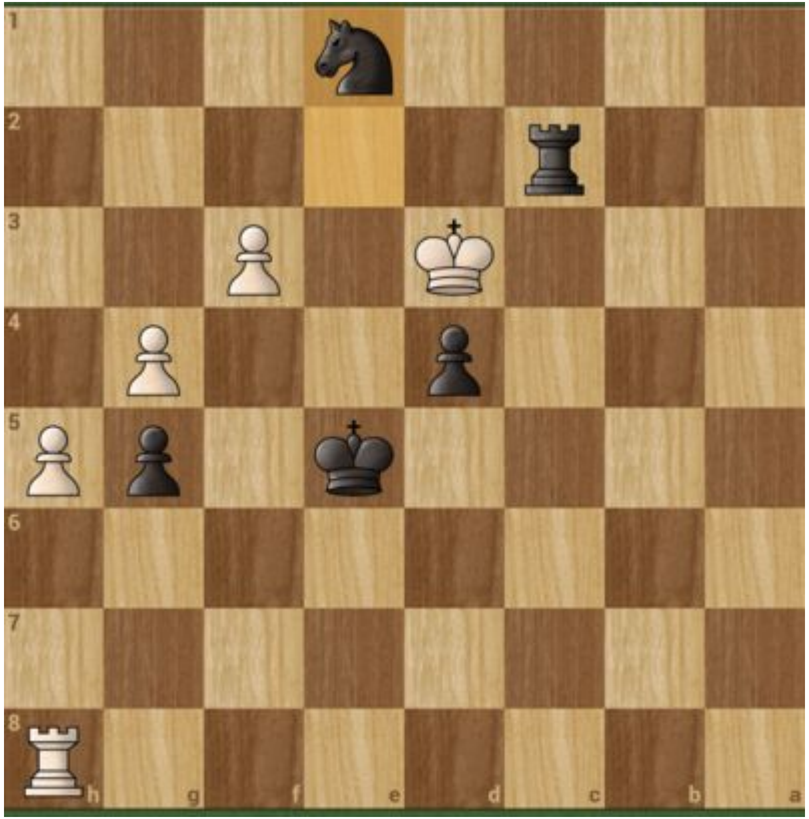
Which of these moves were good or bad?

PGN

1. e4 c5 2. Nc3 Nc6 3. Bc4 e6 4. Nf3 Nf6 5. O-O d5 6. exd5 exd5 7. Bb5 Bd7 8. Re1+ Be7 9. Bxc6 Bxc6 10. Ne5 Qc7 11. d4 O-O 12. Nxc6 bxc6 13. Bg5 h6 14. Bxf6 Bxf6 15. dxc5 Rad8 16. Qh5 Qa5 17. a3 Bxc3 18. bxc3 Qxc3 19. Qe5 Qxc5 20. c3 Qd6 21. Qxd6 Rxd6 22. Re7 a6 23. Rd1 Rb8 24. h4 Rb3 25. c4 Rxa3 26. c5 Re6 27. Rxe6 fxe6 28. Rb1 Kf7 29. Rb6 Ke7 30. Rxc6 Rc3 31. Rxa6 Rxc5 32. Ra7+ Kf6 33. g4 d4 34. Kf1 e5 35. Ke2 g5 36. h5 e4 37. f3 e3 38. Ra6+ Ke5 39. Rxh6 Rc2+ 40. Kd3 e2 41. Rh8 e1=N#

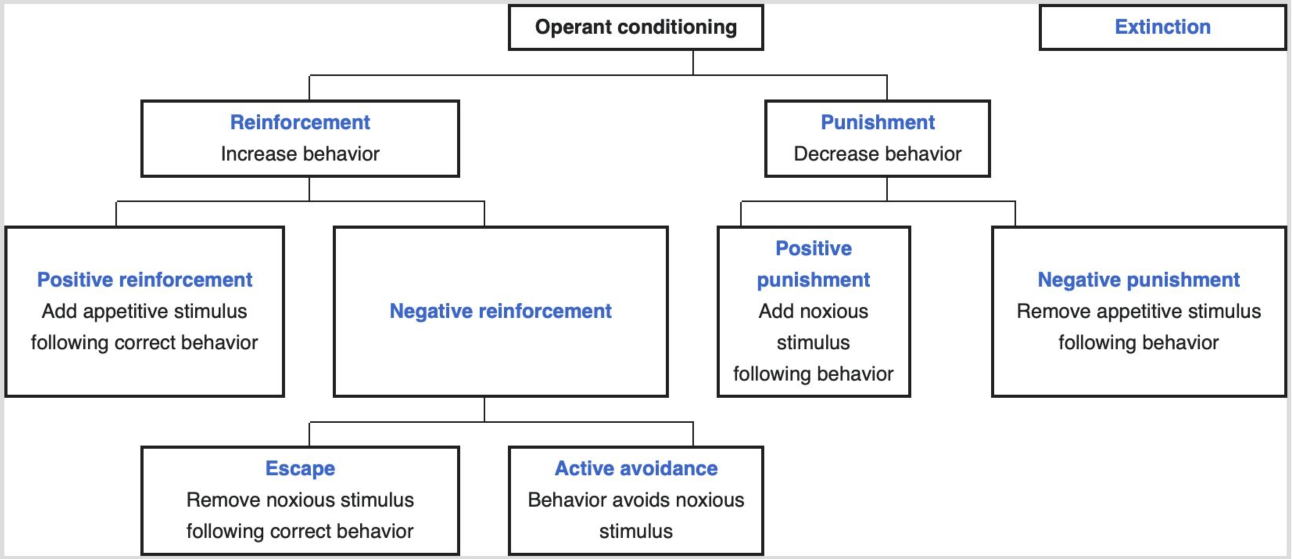
Only defined reward.

Source: https://www.reddit.com/r/chess/comments/25y7o5/cool_underpromotion_checkmate_in_a_game_i_just/



Re: name origin

Curiously, this area is named after the early solving tactics, not the problem definition…



Operant conditioning

Extinction

Reinforcement
Increase behavior

Punishment
Decrease behavior

Positive reinforcement
Add appetitive stimulus following correct behavior

Negative reinforcement

Positive punishment
Add noxious stimulus following behavior

Negative punishment
Remove appetitive stimulus following behavior

Escape
Remove noxious stimulus following correct behavior

Active avoidance
Behavior avoids noxious stimulus

Image source: https://en.wikipedia.org/wiki/Operant_conditioning

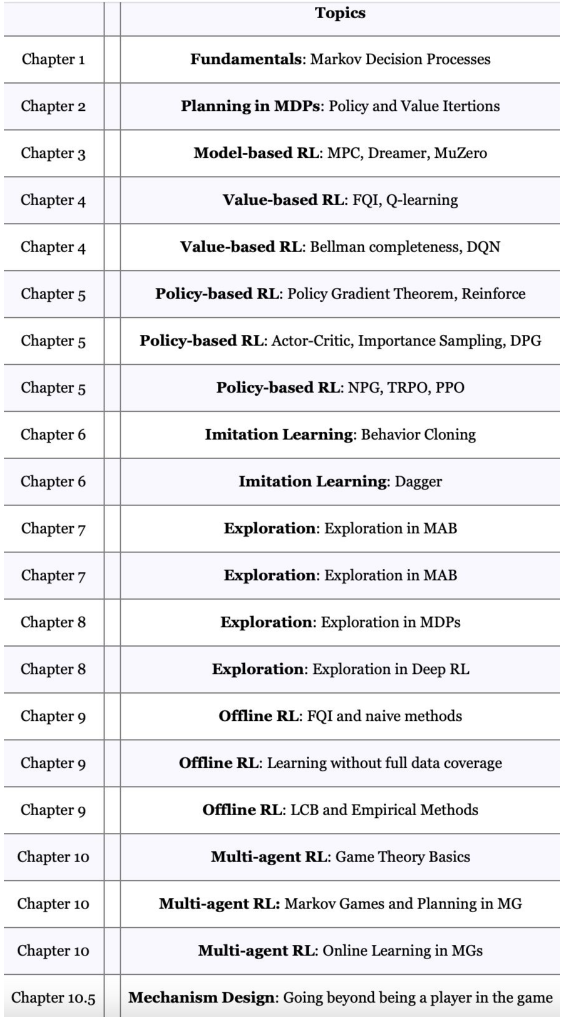
Why is Reinforcement Learning Hard?

- Rewards due to an action may be delayed arbitrarily long.
- Distributions of rewards may change due to agents changing their behavior.
- Probabilistic scenarios make systematic coverage more difficult.
- Opponents (e.g. in 2 player games) actively try to minimize agent rewards.

DS 543 Introduction to Reinforcement Learning

- We actually have a whole course about reinforcement learning.
- Today will focus on the learning-specific challenges of reinforcement learning.
- And connect various techniques used in this course to the RL state of the art.

Will give a brief overview, but this will be very light compared to the real course.



		Topics
Chapter 1		**Fundamentals**: Markov Decision Processes
Chapter 2		**Planning in MDPs**: Policy and Value Itertions
Chapter 3		**Model-based RL**: MPC, Dreamer, MuZero
Chapter 4		**Value-based RL**: FQI, Q-learning
Chapter 4		**Value-based RL**: Bellman completeness, DQN
Chapter 5		**Policy-based RL**: Policy Gradient Theorem, Reinforce
Chapter 5		**Policy-based RL**: Actor-Critic, Importance Sampling, DPG
Chapter 5		**Policy-based RL**: NPG, TRPO, PPO
Chapter 6		**Imitation Learning**: Behavior Cloning
Chapter 6		**Imitation Learning**: Dagger
Chapter 7		**Exploration**: Exploration in MAB
Chapter 7		**Exploration**: Exploration in MAB
Chapter 8		**Exploration**: Exploration in MDPs
Chapter 8		**Exploration**: Exploration in Deep RL
Chapter 9		**Offline RL**: FQI and naive methods
Chapter 9		**Offline RL**: Learning without full data coverage
Chapter 9		**Offline RL**: LCB and Empirical Methods
Chapter 10		**Multi-agent RL**: Game Theory Basics
Chapter 10		**Multi-agent RL:** Markov Games and Planning in MG
Chapter 10		**Multi-agent RL**: Online Learning in MGs
Chapter 10.5		**Mechanism Design**: Going beyond being a player in the game

Two Targets for Today

- AlphaGo + successors
  - State of the art in board game playing
- Reinforcement Learning with Human Feedback
  - Final tuning stage applied to ChatGPT

Example: Tic-Tac-Toe

- What move to win?

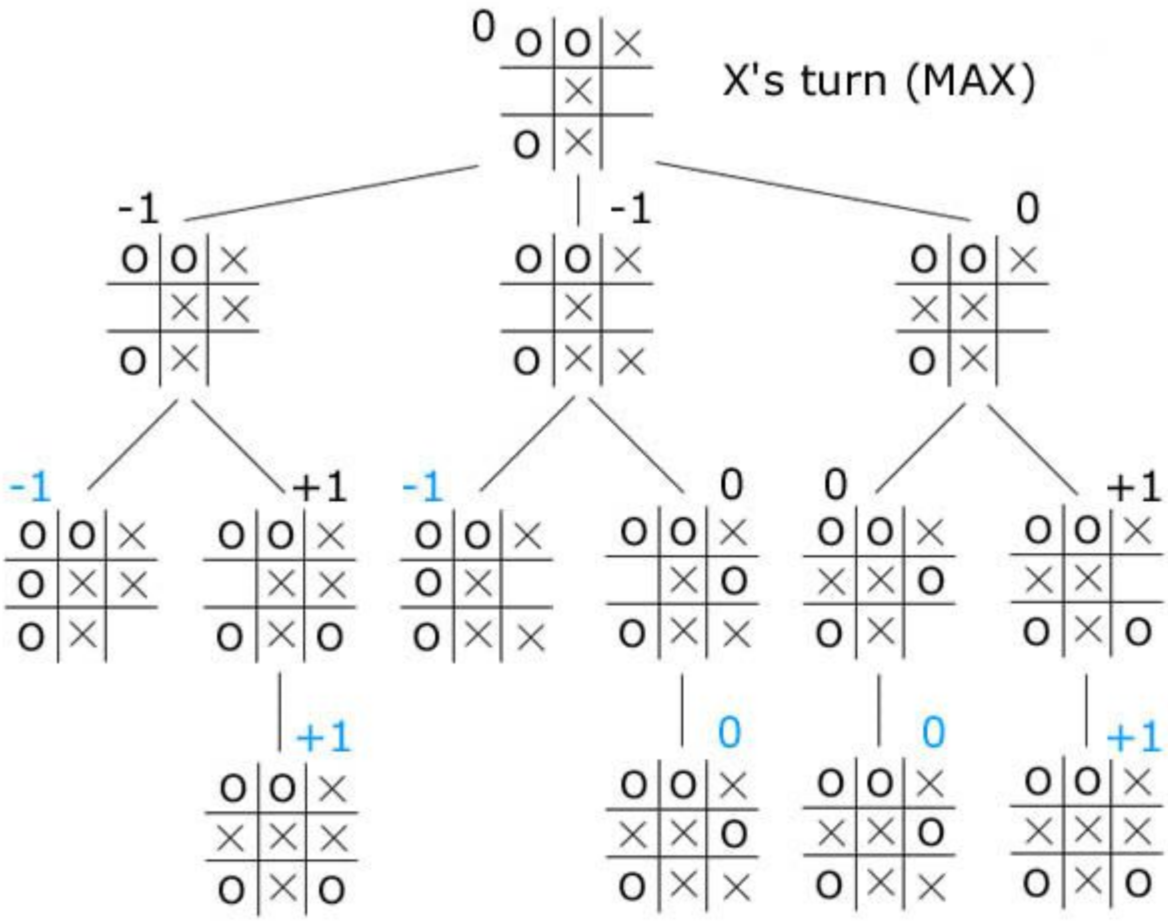
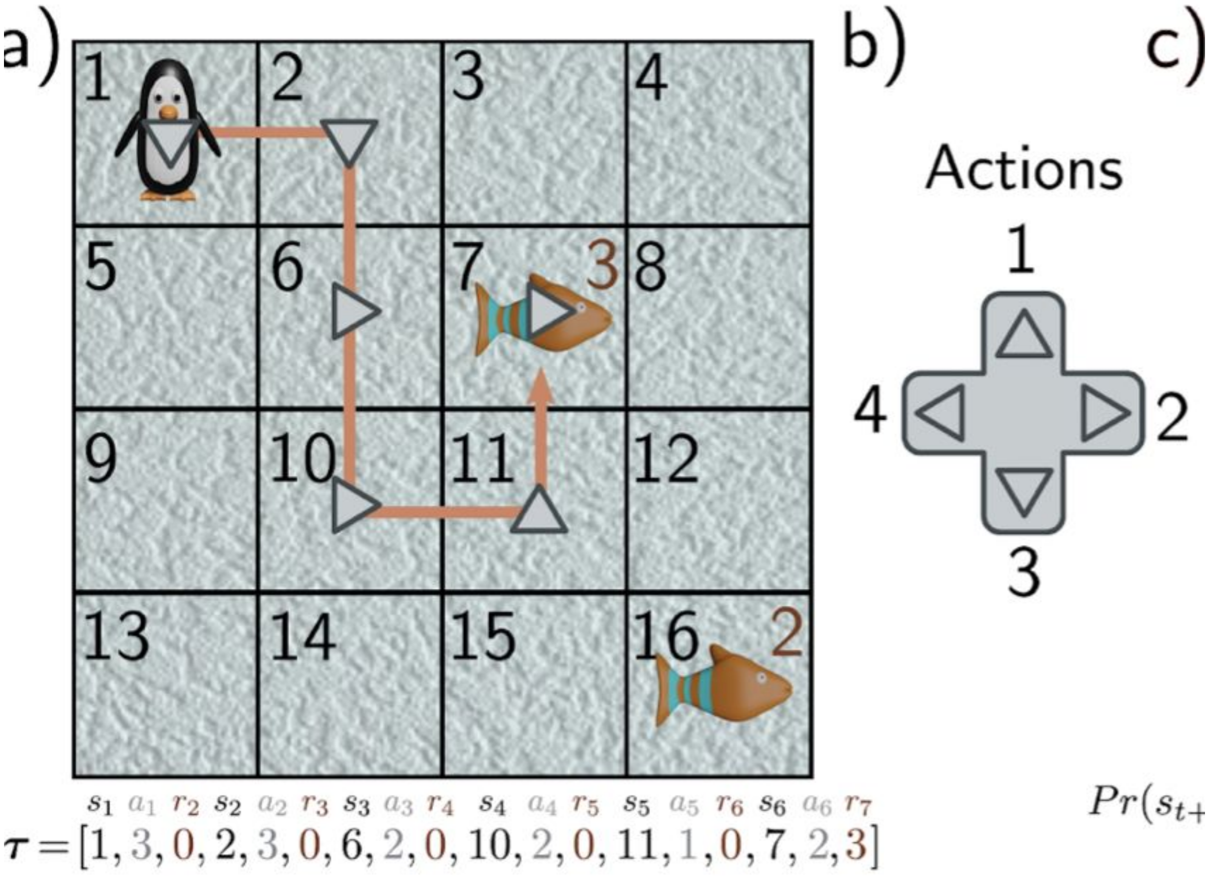


Image source: https://github.com/LewisMatos/MiniMaxTicTacToe

Example: Penguin Game

- How should the penguin move to get the fish?
- Slippery ice makes moves imprecise.



$$\begin{array}{cccccccccccccccccc} s_1 & a_1 & r_2 & s_2 & a_2 & r_3 & s_3 & a_3 & r_4 & s_4 & a_4 & r_5 & s_5 & a_5 & r_6 & s_6 & a_6 & r_7 \end{array}$$

$$\boldsymbol{\tau} = [1, 3, 0, 2, 3, 0, 6, 2, 0, 10, 2, 0, 11, 1, 0, 7, 2, 3]$$

Example: Data Center Cooling

“Despite the impressive advances in reinforcement learning (RL) algorithms, their deployment to real-world physical systems is often complicated by unexpected events and the potential for expensive failures. In this paper we describe an application of RL “in the wild” to the task of regulating temperatures and airflow inside a large-scale data center (DC). Adopting a data-driven model-based approach, we demonstrate that an RL agent is able to effectively and safely regulate conditions inside a server floor in just a few hours, while improving operational efficiency relative to existing controllers.”

“Data Center Cooling using Model-predictive Control” (2018)

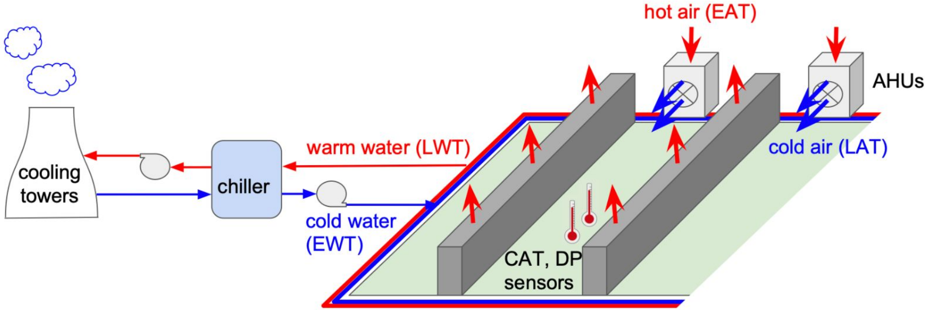
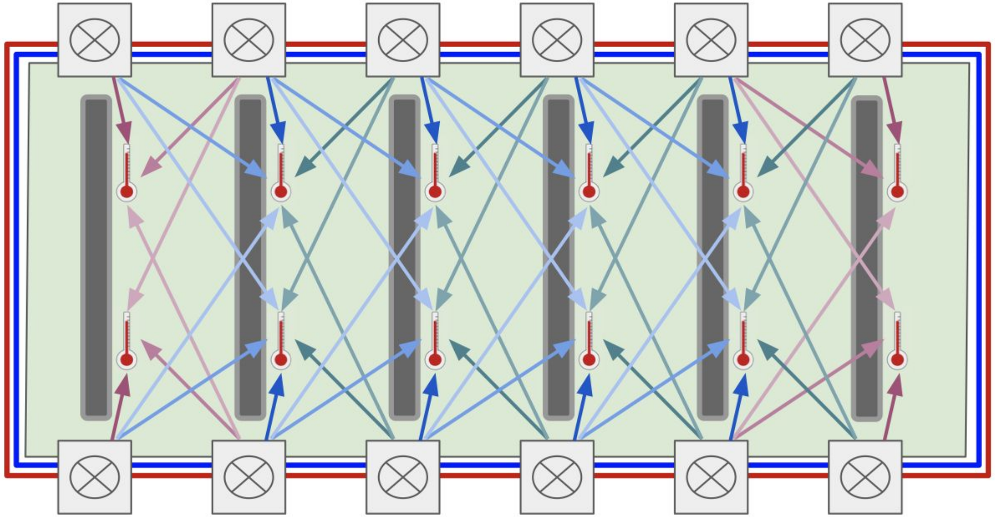


Figure 1: Data center cooling loop. AHUs on the server floor regulate the air temperature through air-water heat exchange. Warmed water is cooled in the chiller and evaporative cooling towers.



Example: Gran Turismo Sophie

- I saw a talk about this last week.
- Racing fast is the easy part.
  - Well-behaved driving is much more difficult.
  - Do not ride the walls.
  - Stop crashing to knock other cars off the course.

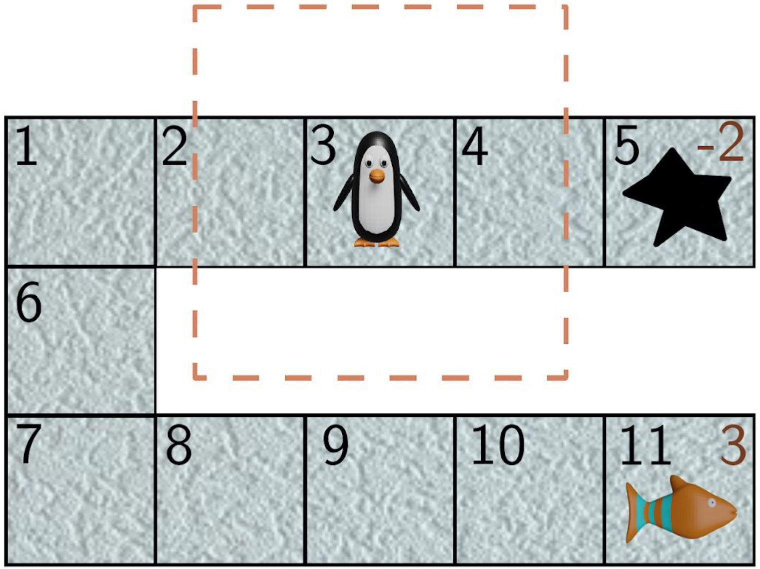
https://www.gran-turismo.com/jp/gran-turismo-sophy/

The international journal of science / 10 February 2022
nature
DRIVING FORCE
AI algorithm outcompetes human champions in *Gran Turismo* racing game
RACE TOGETHER 2021
Sony AI

What is Reinforcement Learning?

- Reinforcement learning covers learning problems where actions are chosen and rewards are received over time.
- The goal of reinforcement learning is to maximize the sum of those rewards.

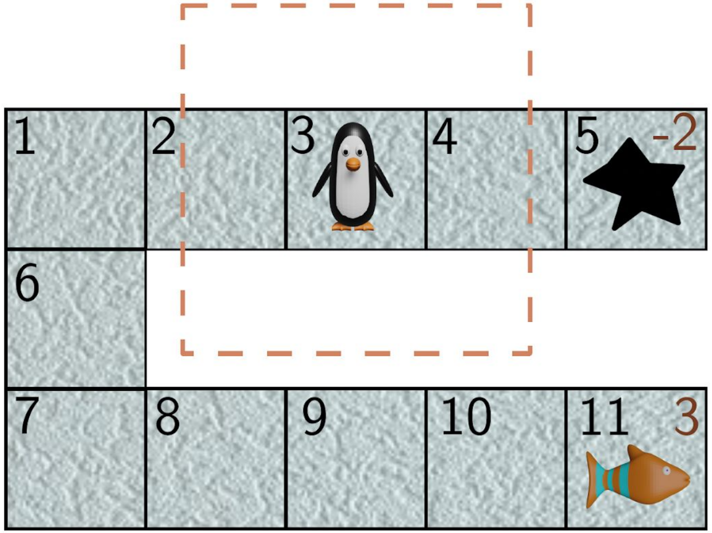
Figure 19.4 Partially observable Markov decision process (POMDP). In a POMDP, the agent does not have access to the entire state. Here, the penguin is in state three and can only see the region in the dashed box. This is indistinguishable from what it would see in state nine. In the first case, moving right leads to the hole in the ice (with -2 reward) and, in the latter, to the fish (with +3 reward).



Horizon Effect

- Positive rewards may be delayed by many steps.
  - Horizon problem - brute force search of a finite number of steps may not see the reward.
  - Some problems may even put negative rewards on the path to the real reward.
    - Often “toy” problems, but analogous to investments with long term payoff.

Figure 19.4 Partially observable Markov decision process (POMDP). In a POMDP, the agent does not have access to the entire state. Here, the penguin is in state three and can only see the region in the dashed box. This is indistinguishable from what it would see in state nine. In the first case, moving right leads to the hole in the ice (with -2 reward) and, in the latter, to the fish (with +3 reward).



Partial Observability

- Real world problems often have unknown state information.
  - Current state may only be partially observed.
  - Problem rules may not be known.
  - Simultaneous actions are unknown.

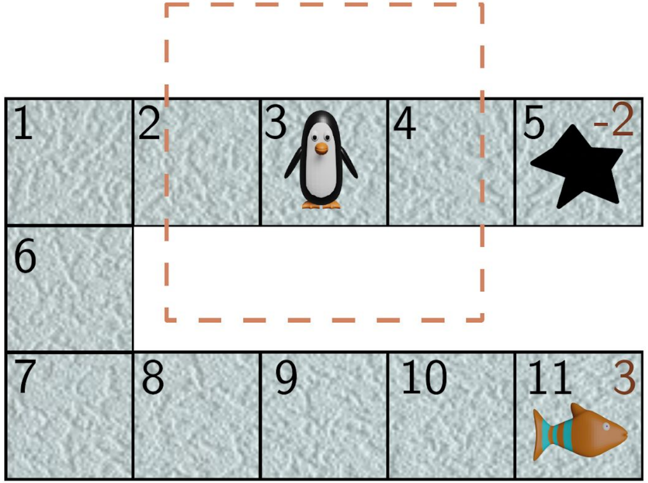


Figure 19.4 Partially observable Markov decision process (POMDP). In a POMDP, the agent does not have access to the entire state. Here, the penguin is in state three and can only see the region in the dashed box. This is indistinguishable from what it would see in state nine. In the first case, moving right leads to the hole in the ice (with -2 reward) and, in the latter, to the fish (with +3 reward).

Unknown Dynamics

- In many real-world problems, the true rules are unknown.
  - We may have approximate versions of the rules.
  - If we can safely attempt the problem, we can gather data to infer the rules.
  - Simulators based on approximate rules will let us trade compute power for real world costs.
- Problems with known dynamics tend to be much easier.
  - Board games in particular.

Side Note: Solving Problems

- Reinforcement learning is more concerned with acting effectively than perfect solutions.
  - Perfect solutions require actual knowledge of the real dynamics.
  - Errors modeling the dynamics tend to blow up.
  - Solving tactics tend to be pretty different and more methodical.
  - But, given enough resources, reinforcement learning methods can get perfect solutions too.

Building up Complexity

Easy mode

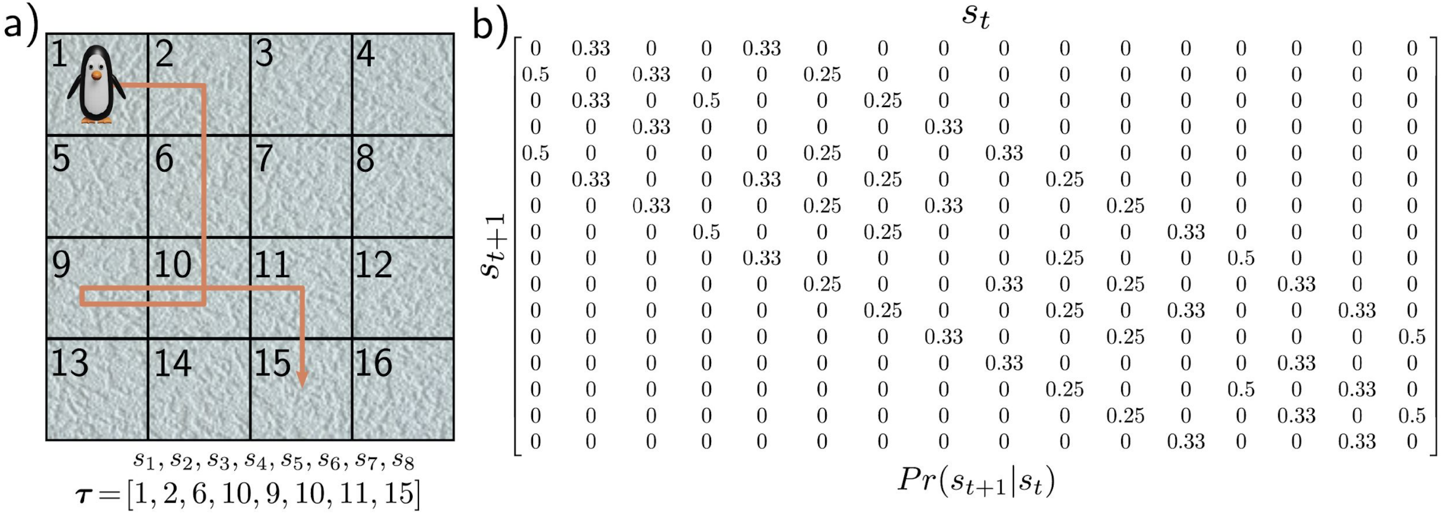
- Markov processes
  - State → probabilistic transition
  - Just a Markov chain.

Focus today

- Markov decision processes (MDPs)
  - State + action → probabilistic transition
  - Puzzles and full information games are deterministic special cases.
- Partially observable Markov decision processes (POMDPs)
  - Like MDPs, but only partial state known.

Harder RL here

Markov Processes


a)
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
$s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8$
$\boldsymbol{\tau} = [1, 2, 6, 10, 9, 10, 11, 15]$
b)
s_t
s_{t+1}
$Pr(s_{t+1}|s_t)$

$$\begin{bmatrix}
0 & 0.33 & 0 & 0 & 0.33 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.5 & 0 & 0.33 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.33 & 0 & 0.5 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.33 & 0 & 0 & 0 & 0 & 0.33 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.5 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.33 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.33 & 0 & 0 & 0.33 & 0 & 0.25 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.33 & 0 & 0 & 0.25 & 0 & 0.33 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 & 0.33 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.33 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.5 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.33 & 0 & 0.25 & 0 & 0 & 0.33 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.25 & 0 & 0.33 & 0 & 0 & 0.33 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.33 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 & 0.5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.33 & 0 & 0 & 0 & 0 & 0.33 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.5 & 0 & 0.33 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.33 & 0 & 0.5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.33 & 0 & 0 & 0.33 & 0
\end{bmatrix}$$

Figure 19.1 Markov process. A Markov process consists of a set of states and transition probabilities $Pr(s_{t+1}|s_t)$ that define the probability of moving to state s_{t+1} given the current state is s_t. a) The penguin can visit 16 different positions (states) on the ice. b) The ice is slippery, so at each time, it has an equal probability of moving to any adjacent state. For example, in position 6, it has a 25% chance of moving to states 2, 5, 7, and 10. A trajectory $\tau = [s_1, s_2, s_3, \ldots]$ from this process consists of a sequence of states.

Markov Processes

- A finite Markov process with n states is easy to learn.
  - Run a lot of simulations and track stats…
- Learn transition probabilities
  - n x n transition probabilities.
- Or learn expected total reward from a particular state
  - n values to learn.

Just stats, no deep learning necessary.

- These processes can run infinitely long, so usually adding a discount factor to rewards.
  - Compare finite discounted rewards instead of everything infinite…

Discounted Rewards

γ is the discounting factor.

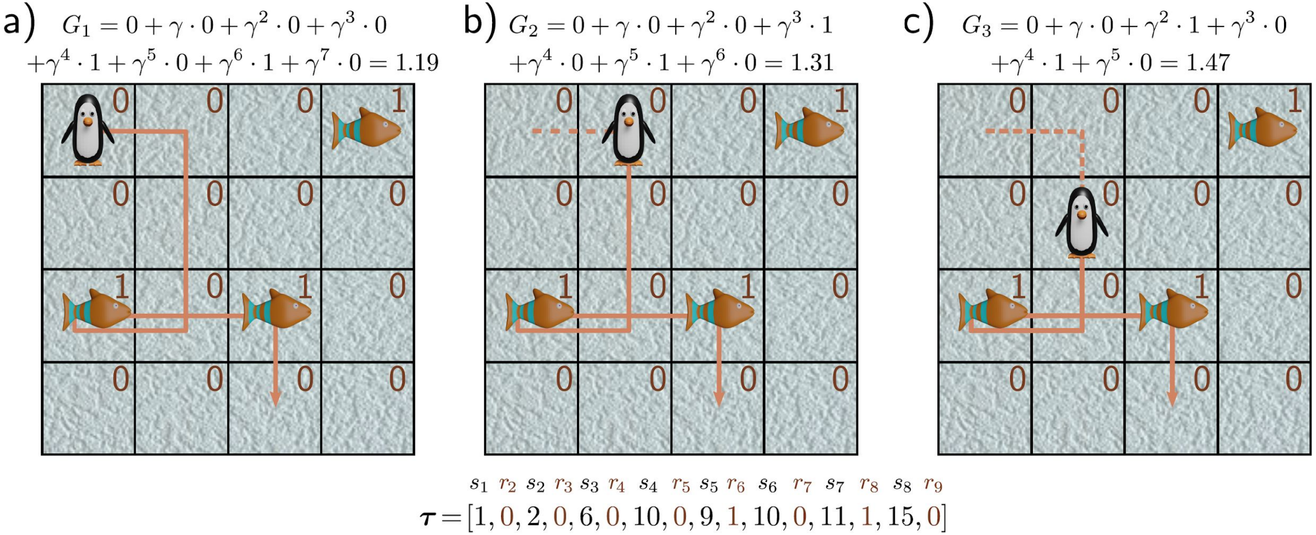

a) $G_1 = 0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \gamma^3 \cdot 0 + \gamma^4 \cdot 1 + \gamma^5 \cdot 0 + \gamma^6 \cdot 1 + \gamma^7 \cdot 0 = 1.19$
b) $G_2 = 0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \gamma^3 \cdot 1 + \gamma^4 \cdot 0 + \gamma^5 \cdot 1 + \gamma^6 \cdot 0 = 1.31$
c) $G_3 = 0 + \gamma \cdot 0 + \gamma^2 \cdot 1 + \gamma^3 \cdot 0 + \gamma^4 \cdot 1 + \gamma^5 \cdot 0 = 1.47$
0 0 0 1
0 0 0 0
1 0 1 0
0 0 0 0
s_1 r_2 s_2 r_3 s_3 r_4 s_4 r_5 s_5 r_6 s_6 r_7 s_7 r_8 s_8 r_9
$\boldsymbol{\tau} = [1, 0, 2, 0, 6, 0, 10, 0, 9, 1, 10, 0, 11, 1, 15, 0]$

Figure 19.2 Markov reward process. This associates a distribution $Pr(r_{t+1}|s_t)$ of rewards r_{t+1} with each state s_t. a) Here, the rewards are deterministic; the penguin will receive a reward of +1 if it lands on a fish and 0 otherwise. The trajectory τ now consists of a sequence $s_1, r_2, s_2, r_3, s_3, r_4 \ldots$ of alternating states and rewards, terminating after eight steps. The return G_t of the sequence is the sum of discounted future rewards, here with discount factor $\gamma = 0.9$. b-c) As the penguin proceeds along the trajectory and gets closer to reaching the rewards, the return increases.

Markov Processes

- Do not model how we want our agents to act.
- More a model of other things happening to our agent.
  - Agent does nothing, or is just swept along.
  - Why do these things keep happening to me?

Markov Decision Processes

- Add agent actions to the model.
    - Not necessarily deterministic.
    - This model has the action choice work 50% of the time.

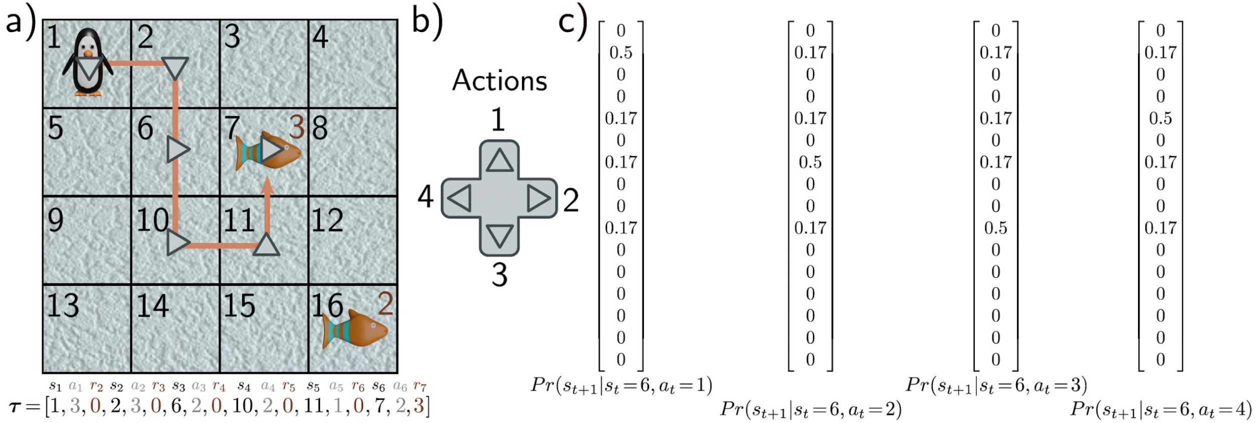

a)
1
2
3
4
5
6
7
3
8
9
10
11
12
13
14
15
16
2
s_1 a_1 r_2 s_2 a_2 r_3 s_3 a_3 r_4 s_4 a_4 r_5 s_5 a_5 r_6 s_6 a_6 r_7
$\boldsymbol{\tau} = [1, 3, 0, 2, 3, 0, 6, 2, 0, 10, 2, 0, 11, 1, 0, 7, 2, 3]$
b)
Actions
1
4
2
3
c)
$[0, 0.5, 0, 0, 0.17, 0, 0.17, 0, 0, 0.17, 0, 0, 0, 0, 0, 0]^T$
$Pr(s_{t+1}|s_t=6, a_t=1)$
$[0, 0.17, 0, 0, 0.17, 0, 0.5, 0, 0, 0.17, 0, 0, 0, 0, 0, 0]^T$
$Pr(s_{t+1}|s_t=6, a_t=2)$
$[0, 0.17, 0, 0, 0.17, 0, 0.17, 0, 0, 0.5, 0, 0, 0, 0, 0, 0]^T$
$Pr(s_{t+1}|s_t=6, a_t=3)$
$[0, 0.17, 0, 0, 0.5, 0, 0.17, 0, 0, 0.17, 0, 0, 0, 0, 0, 0]^T$
$Pr(s_{t+1}|s_t=6, a_t=4)$

Figure 19.3 Markov decision process. a) The agent (penguin) can perform one of a set of actions in each state. The action influences both the probability of moving to the successor state and the probability of receiving rewards. b) Here, the four actions correspond to moving up, right, down, and left. c) For any state (here, state 6), the action changes the probability of moving to the next state. The penguin moves in the intended direction with 50% probability, but the ice is slippery, so it may slide to one of the other adjacent positions with equal probability. Accordingly, in panel (a), the action taken (gray arrows) doesn't always line up with the trajectory (orange line). Here, the action does not affect the reward, so $Pr(r_{t+1}|s_t, a_t) = Pr(r_{t+1}|s_t)$. The trajectory $\boldsymbol{\tau}$ from an MDP consists of a sequence $s_1, a_1, r_2, s_2, a_2, r_3, s_3, a_3, r_4 \ldots$ of alternating states s_t, actions a_t, and rewards, r_{t+1}. Note that here the penguin receives the reward when it *leaves* a state with a fish (i.e., the reward is received for passing through the fish square, regardless of whether the penguin arrived there intentionally or not).

Markov Decision Processes

- Add actions to Markov processes
- Rewards and next state depend on (state, action) instead of just state.
- Two new potential functions to model
  - (state, action) → expected reward (given current or optimal action choices)
  - (state, action) → distribution of rewards and next states

Why is this qualification needed?

model-based reinforcement learning

Policies

A policy is a function taking in a state and returning an action to take.

- Move selection in board games
- May be probabilistic or deterministic.
- Usually represented with variable π, maybe with subscripts to distinguish…

Mechanically, policy output is a list of probabilities of each action…

- Just like classification outputs.

Given a policy π, we can quantify

- Value function:
  - The total expected rewards of state s playing with policy π.
- Action function:
  - The total expected rewards of state s picking action a and then playing with policy π.

Why Probabilistic Policies?

- To facilitate learning.
    - Can run gradient descent with probabilistic policies.
- Some problems require probabilistic policies.
    - Easy example: Rock / Paper / Scissors
      https://hackaday.com/2015/10/06/robot-cheats-at-rock-paper-scissors/

Why Not Probabilistic Policies?

- Some problems have clear best answers.
  - Pick any of them deterministically.
  - Full information and no simultaneous moves.
    - Puzzles
    - Two player games with alternating turns.
- With these problems, probabilities usually represent uncertainty.
  - Not sure about the right action.
  - Or rarely, choice between actions with non-zero probabilities does not matter.

Is the Perfect Policy Probabilistic or Deterministic?

Probabilistic

- Rock-paper-scissors
- Poker
- Car racing

- Navigation (congestion aware)

Deterministic

- Chess
- Go
- Data center cooling?

- Navigation (single car)

Bellman Equation

$$v[s_t] = \sum_{a_t} \pi[a_t \,|\, s_t] \; q[s_t, a_t]$$

$$q[s_t, a_t] = r[s_t, a_t] + \gamma \cdot \sum_{s_{t+1}} Pr(s_{t+1} \,|\, s_t, a_t) \; v[s_{t+1}]$$

This version omits optimization of π.

Q-Learning

- Just saw the q-function in the Bellman equation as value of a (state, action) choice.
- More properly used when the policy π is optimal.
    - The q-learning process actually optimizes the policy to be optimal.

“Learning from Delayed Rewards” (1989)

Comparison of Value vs Policy vs Q Functions

- Policy picks actions.
- Value function estimates rewards from state given policy.
- Q function estimates rewards from state and action given policy.

Side Note: Tabular Reinforcement Learning

For small problems, such as Tic-Tac-Toe or the Penguin game, there are few enough states that the value and q functions can be feasibly stored in a table.

- If the tabular representation is feasible, then learning tends to be simplified.
  - If dynamics are known, usually solving instead of learning.
  - Iterative methods still converge with enough samples.
- This tabular representation is essentially the infinite resource ideal.
  - May see table-oriented notation if you read more about this.
  - Reinforcement learning uses function approximation when this ideal is not practical.
  - This is where deep learning comes in.

Looking at the RL Process

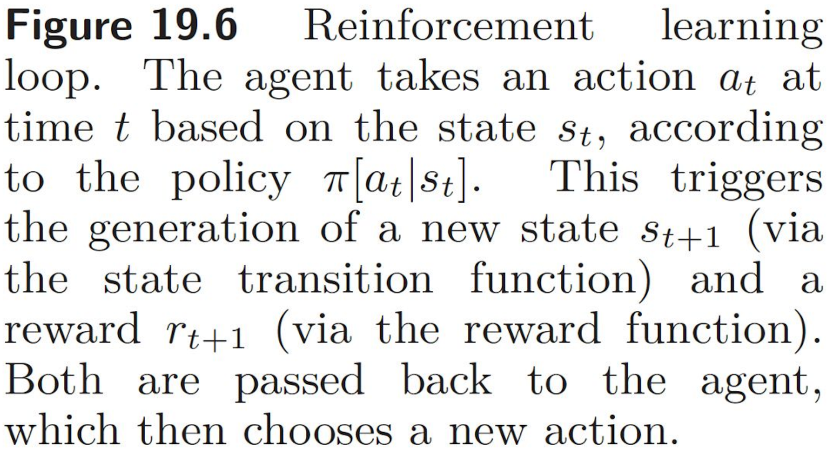
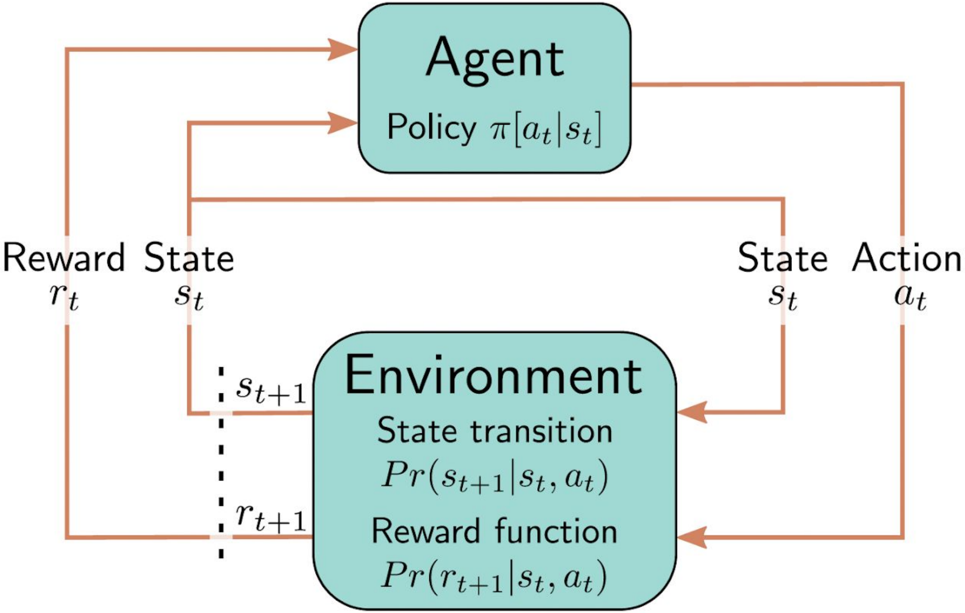


Figure 19.6 Reinforcement learning loop. The agent takes an action a_t at time t based on the state s_t, according to the policy $\pi[a_t|s_t]$. This triggers the generation of a new state s_{t+1} (via the state transition function) and a reward r_{t+1} (via the reward function). Both are passed back to the agent, which then chooses a new action.



Generic RL Approach

- Learn approximate behavior over many instances.
  - Simulations preferred if possible.
  - Optimize based on learned (modeled) best behavior
- Very different from classical AI approach
  - Minimax trees
  - Alpha beta search
  - Classical AI approach usually assumes problem can be simulated faithfully.
- Hybrid approaches are possible
  - Monte Carlo Tree Search (will cover today)
  - State of the art game play works this way
  - AlphaGo / AlphaZero / Leela Chess Zero
  - StockFish (recent versions)

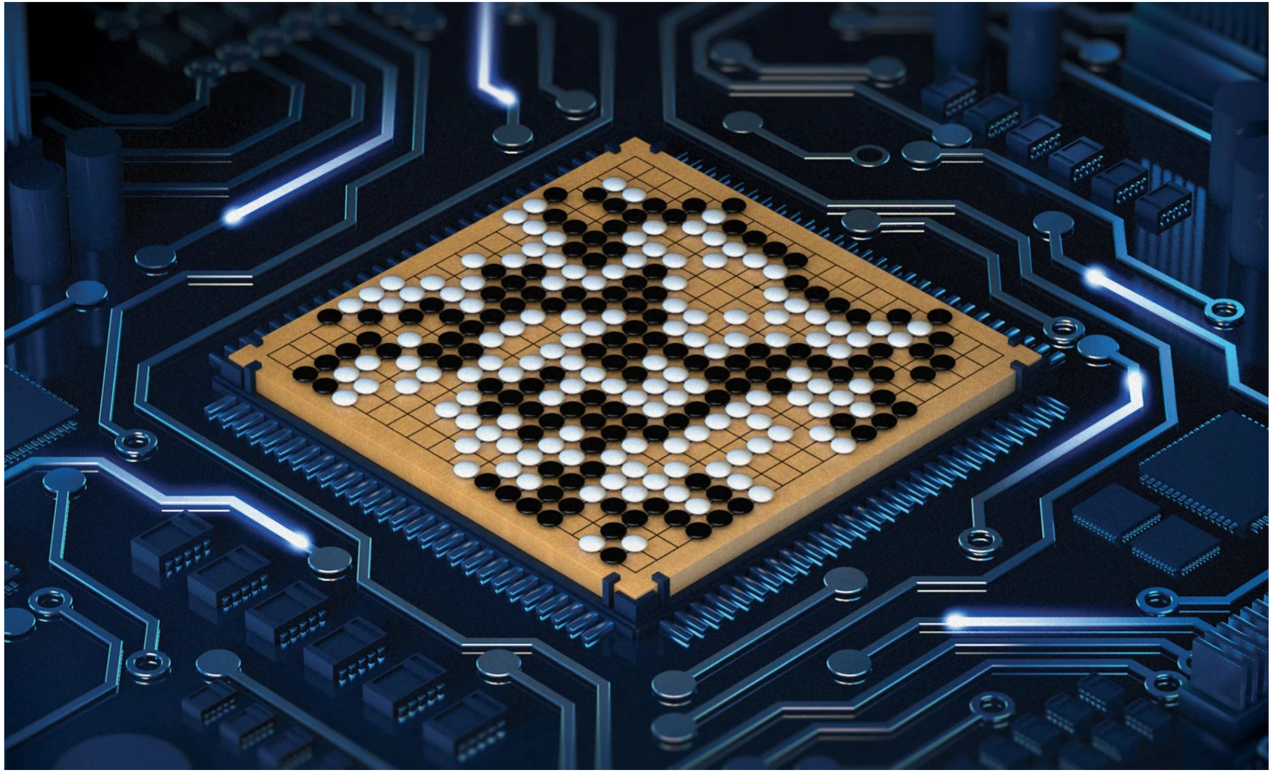
How I learned AI

Current Chess State of the Art

AlphaGo

- Game of Go
  - Ancient game
  - Played on 19x19 board
  - Generally considered one of the hardest games that we play
  - Was notoriously difficult for computers to play above amateur level.
- Needs intelligence, cannot brute force like chess?
- Many hard subproblems

https://deepmind.google/research/breakthroughs/alphago/



Supervised learning of policy networks (imitation learning)

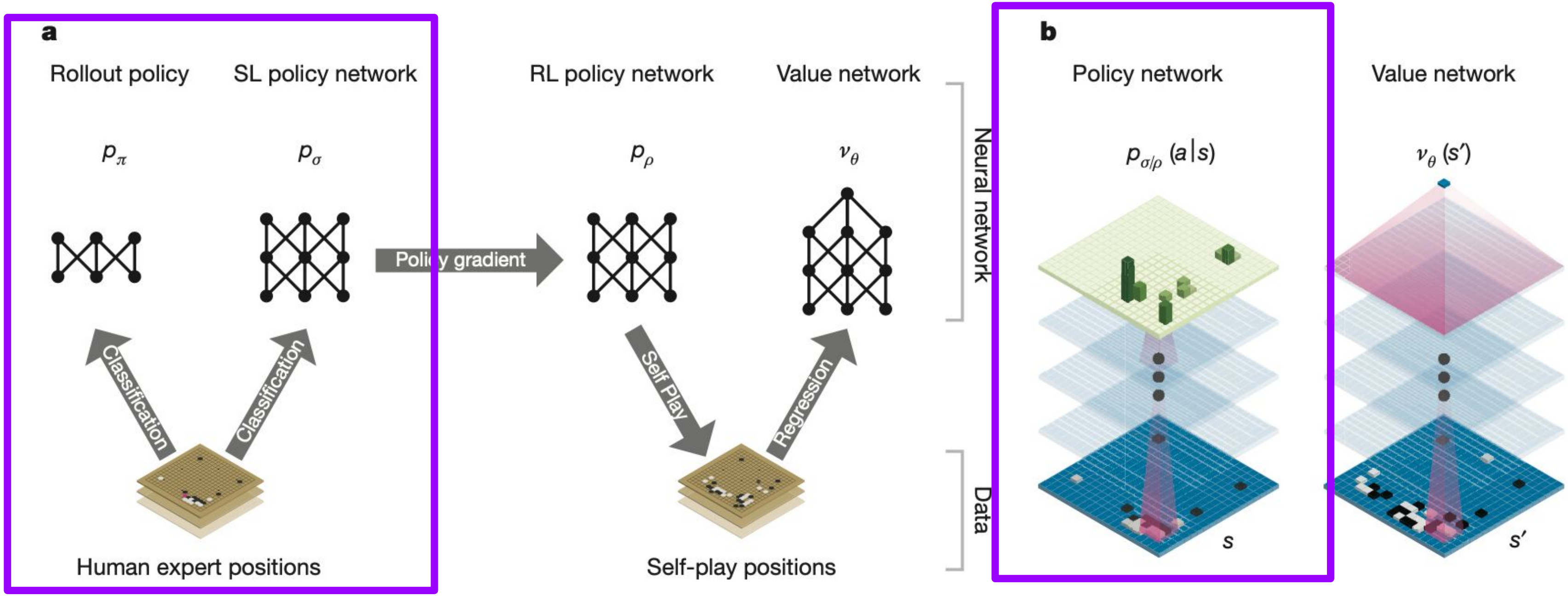
Phase 1:

- Collect a database of expert-level go games.
- Train a policy network to pick the same moves as the experts.
- This had been done before, but only with linear policies or shallow neural networks…

“Mastering the game of Go with deep neural networks and tree search” (2016)

Supervised learning of policy networks (imitation learning)

TLDR: a deep convolutional network treating move selection as classification.



“Mastering the game of Go with deep neural networks and tree search” (2016)

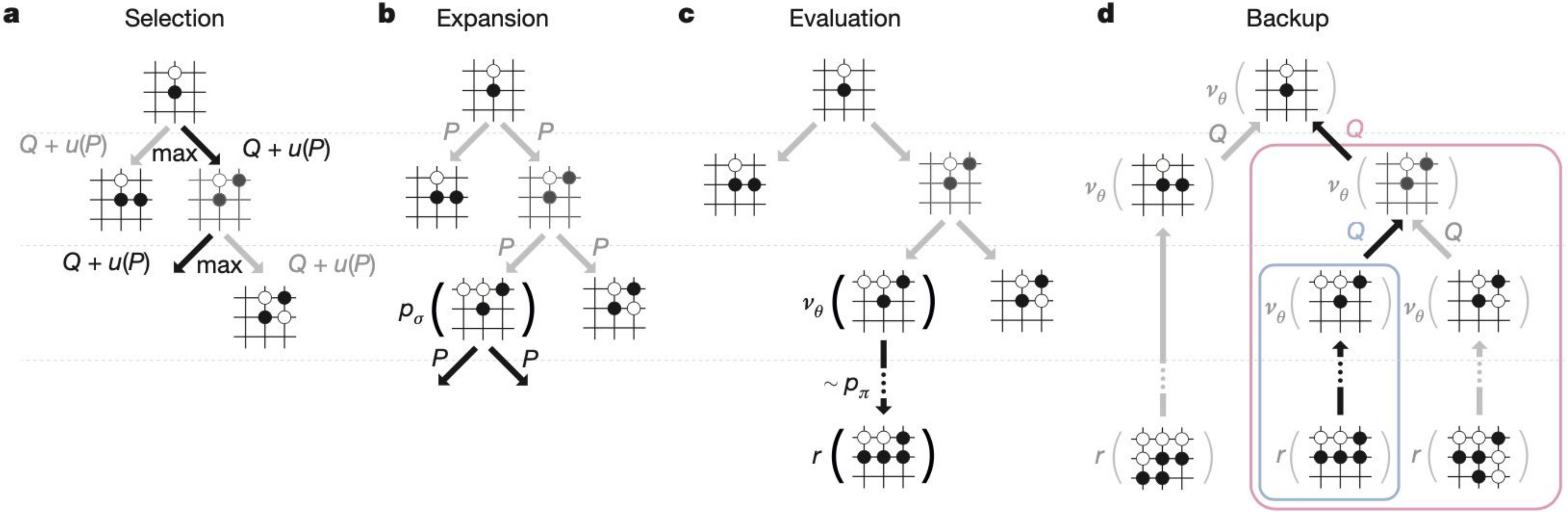
Supervised learning of policy networks (imitation learning)

- Downloaded 30 million positions from KGS Go Server.
- 13 layer policy network
- Predicted expert moves 55-57% accurately (depending on features allowed)
  - Previous best ~44% accurate
- Position evaluation in 3ms
  - Also trained a simplified network running in 2μs but only 24% accurate

"Mastering the game of Go with deep neural networks and tree search" (2016)

Monte Carlo Tree Search (MCTS)

TLDR: play a lot of games (in your head) remembering which moves worked out.



"Mastering the game of Go with deep neural networks and tree search" (2016)

Reinforcement learning of policy networks

- Copy supervised policy from imitation learning to initialize new rollout policy.
- Run Monte Carlo Tree Search to play games between rollout policy and previous snapshot.
    - Previous snapshot is chosen randomly to overfitting on particular snapshot's weaknesses.
    - No rewards until the end of the game.
    - Final reward is ±1 depending on result of the game.
    - After each game, update the current rollout policy based on game result.
        - Use policy to calculate move probabilities of the game played.
        - Calculate gradients of picking those moves.
        - One step of gradient descent with direction based on final reward.

"Mastering the game of Go with deep neural networks and tree search" (2016)

Policy Improvement with Monte Carlo Tree Search

- If MCTS using a policy shows the policy made mistakes, update the policy.
  - If policy did not pick winning move, but MCTS found a win, update the policy.
  - If policy picked an avoidable losing move and MCTS avoided the loss, update the policy.
- Using MCTS forces policy to align with best rewards.
  - This can work even starting from a random policy.
- Even if one side always wins with the current policy, the losing side will explore all their alternative moves looking for a way out.

Reinforcement learning of value networks

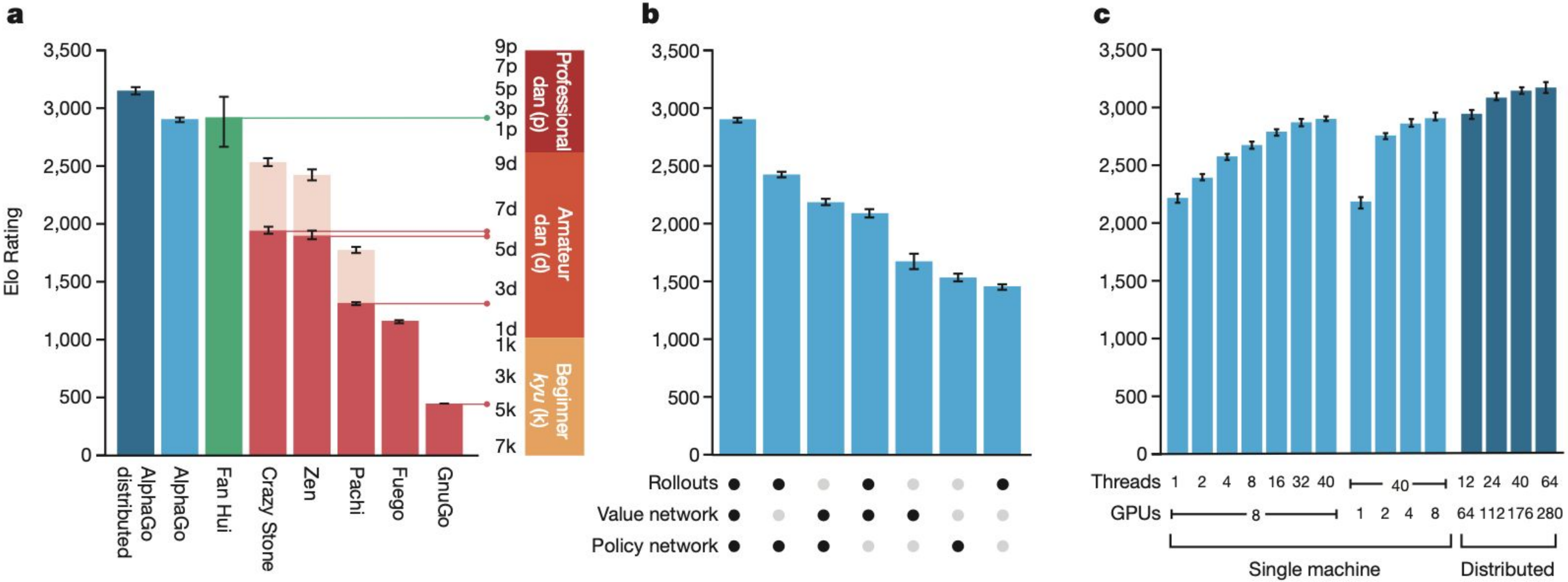
- Policies help pick moves, but do not say whether you will win or not.
- Also would like a value function (network) to directly estimate value of a position or move.
    - If you have a value function, you can also pick moves by applying it to all possible moves.
    - This is faster than simulating whole games.
- Treat this as another supervised learning problem.
    - Play games using the current policy until the end./
    - Pick one move during the game and train value network with game result as target.
    - Only pick one move per game to avoid overfitting - many similar positions in a game.

"Mastering the game of Go with deep neural networks and tree search" (2016)

Playing with AlphaGo

- Build an MCTS game tree as follows.
  - Repeatedly simulate game rollouts from the root expanding the tree one node at a time.
  - MCTS leaf evaluations are weighted average of value network and rollout result.
  - Repeated rollouts through a node update that node’s stats.
  - Move selection mixes picking for diversity early on and getting more data for winning moves.
- Interesting notes
  - The supervised policy gave better MCTS trees, apparently due to diversity.
  - The rollout policy gave a better value function since it was tuned for better one move choices.
  - Ablation of other choices on next slide.

Evaluation of AlphaGo



“Mastering the game of Go with deep neural networks and tree search” (2016)

Evaluation of AlphaGo

- AlphaGo vs Fan Hui (2015)
  - European Go champion
  - Won 5-0
- AlphaGo vs Lee Sedol (2016):
  - Won 4-1 (technically 3-0 but they continued)
  - AlphaGo play was shocking to spectators.
  - Lee won the 4th game by staying up working out an anti-computer strategy with other experts.

"Mastering the game of Go with deep neural networks and tree search" (2016)

Later Developments

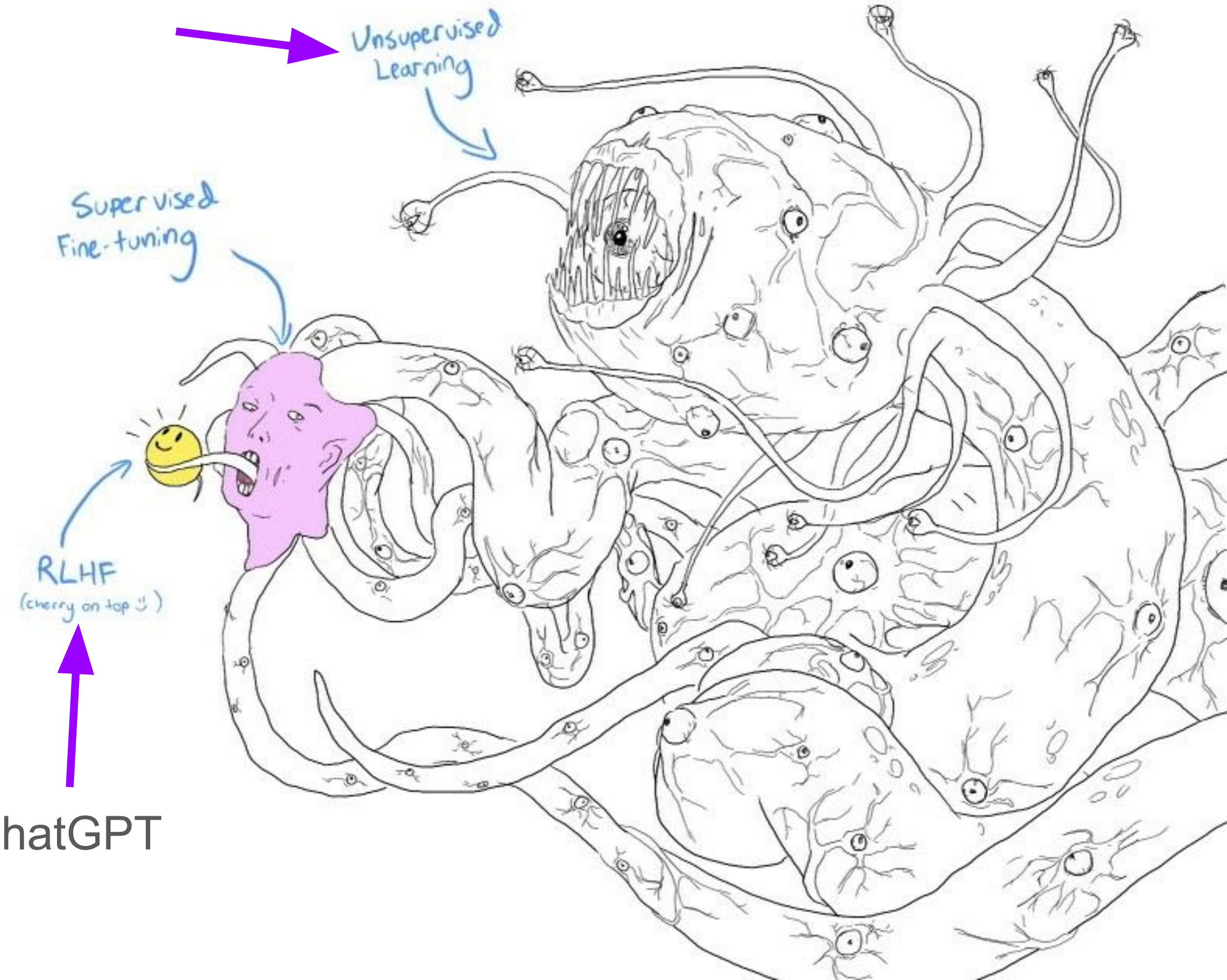
- AlphaGo Zero:
  - Same design, but skip imitation learning from expert games.
- AlphaZero:
  - Simplify design removing go-specific features (particularly augmentations)
  - Apply common design to chess and shogi too.
  - Claimed to crush all the state of the art programs.
- MuZero:
  - Further simplifications, do not even wire up game rules!!!

Later Developments

- Leela Chess Zero (lc0)
  - Open source re-implementation of AlphaZero
  - https://lczero.org/
- StockFish
  - Relatively old open source chess program that was long time state of the art.
  - Based on really efficient search and a good evaluation function.
  - Integrated “Efficiently Updatable Neural Networks” (NNUE) to replace previous bespoke evaluation function.
  - Gained >80 ELO “overnight”
  - https://stockfishchess.org/blog/2020/introducing-nnue-evaluation/

Large Language Model Tuning

- Large language models existed for a while before ChatGPT.
- GPT 3.0 was available to the public via API.
- Why did ChatGPT make a bigger splash?

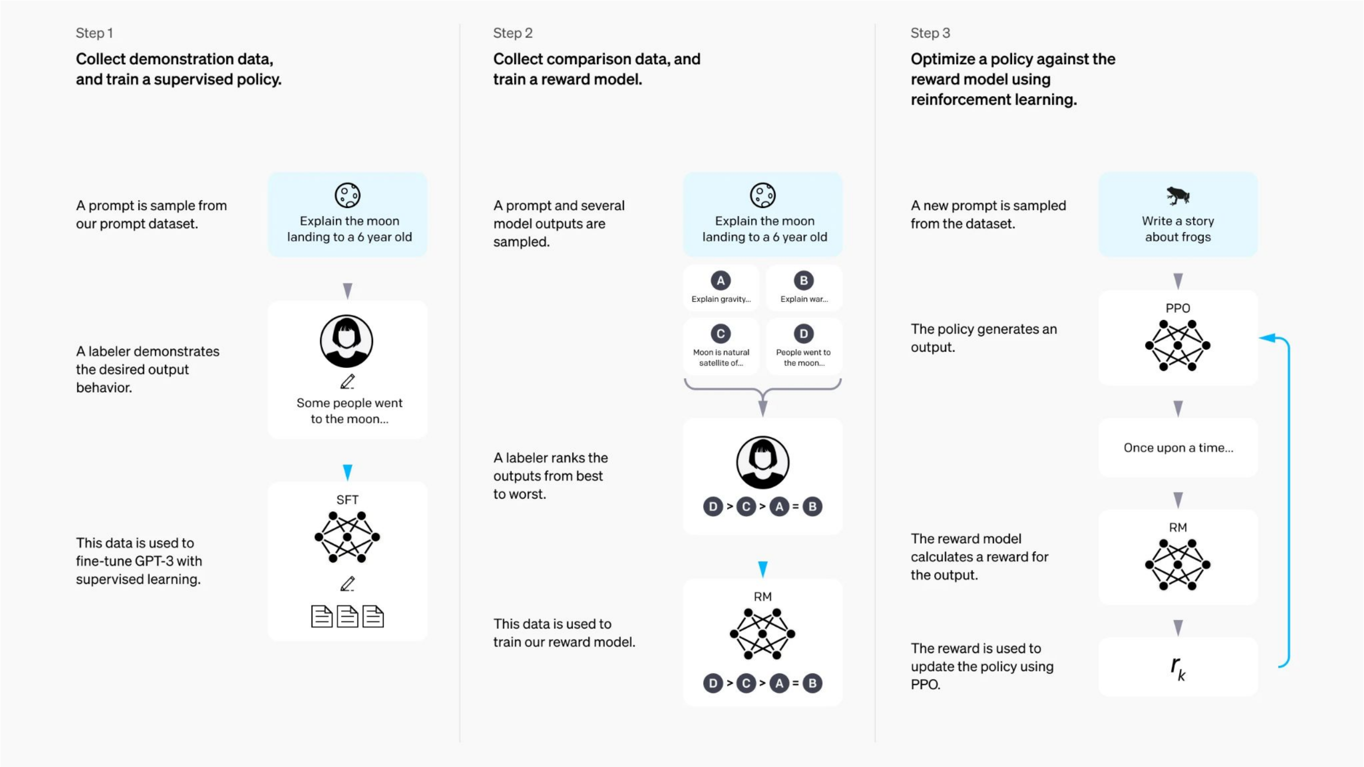


GPT 3.0/3.5

ChatGPT

Image source: https://x.com/anthrupad/status/1622349563922362368

Training an LLM to Take Instructions



Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sample from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.

Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Step 3

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

Explain the moon landing to a 6 year old
Some people went to the moon...
SFT
Explain the moon landing to a 6 year old
A
Explain gravity...
B
Explain war...
C
Moon is natural satellite of...
D
People went to the moon...
D > C > A = B
RM
D > C > A = B
Write a story about frogs
PPO
Once upon a time...
RM
r_k

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

Pre-Training a Large Language Model

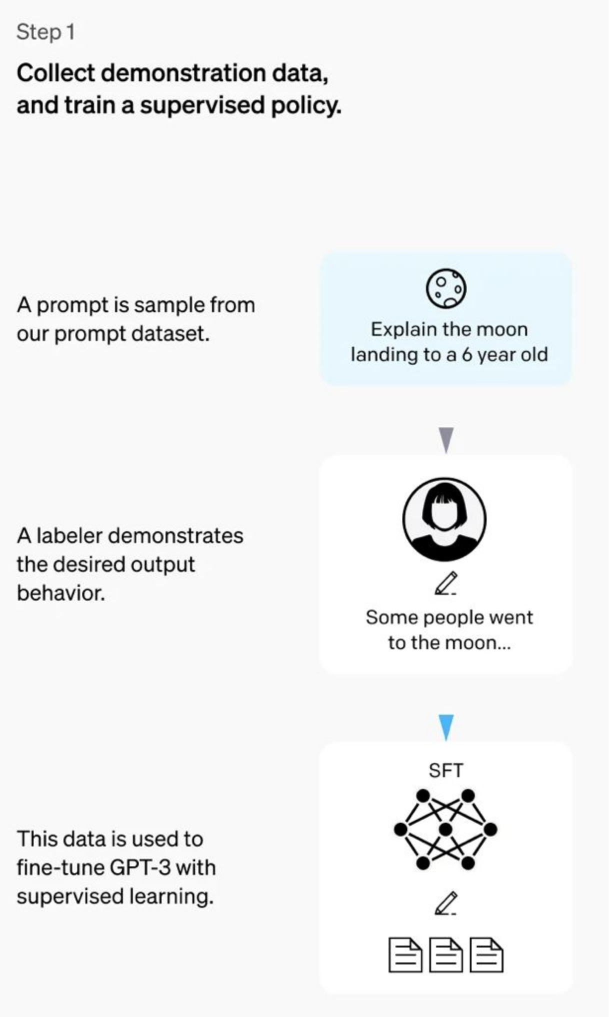
- Download all the text on the internet.
  - Train your favorite decoder-only model on that regardless of good or bad.
  - But feel to upweight good stuff if you can tell.
  - Use Wikipedia twice, skip 4chan.
- Nowadays, most LLM providers are training on a significant fraction of publicly accessible text on the Internet.
  - Some papers show they are getting more selective about what is included in the training data.
  - Anecdotally, also using synthetic data but seeing mixed results. (Also mode collapse risk.)
- The messy problem
  - There is a lot of toxic and made up content on the Internet.

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

Supervising Your Language Model

- Collect many examples of good responses.
- Directly tune language model weights to increase probabilities of good responses.
- This process tends to make the model responses much more polite and helpful.

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



Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sample from our prompt dataset.

Explain the moon landing to a 6 year old

A labeler demonstrates the desired output behavior.

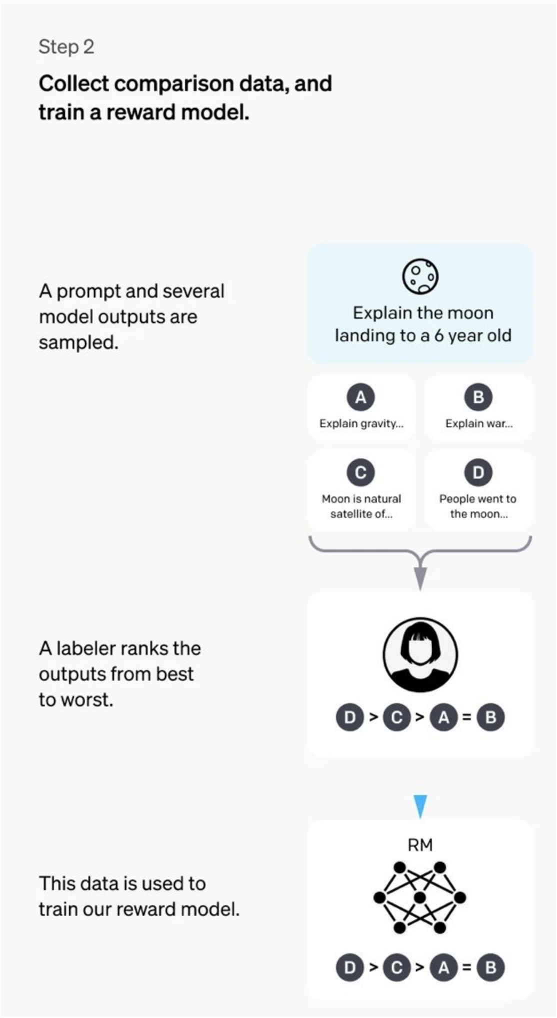
Some people went to the moon...

This data is used to fine-tune GPT-3 with supervised learning.

Language Models - Collecting Human Preferences

- Build another dataset of model outputs for the same prompts.
- Collect labeler rankings of those model outputs.
- Build a reward model to rank responses.
  - The reward model built off the pre-trained model.
  - Key idea: the reward model generalizes the labeler preferences.

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



Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

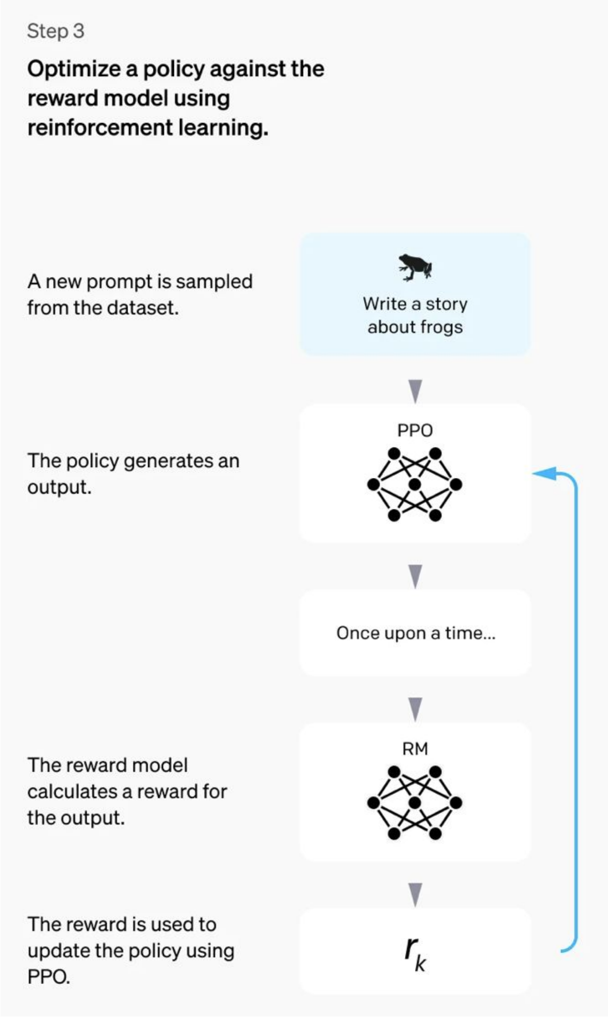
This data is used to train our reward model.

Explain the moon landing to a 6 year old
A
Explain gravity...
B
Explain war...
C
Moon is natural satellite of...
D
People went to the moon...
D > C > A = B
RM
D > C > A = B

Reinforcement Learning from Human Preferences

- Build a new policy to choose actions (tokens) and optimize it to maximize the previous reward model.
  - This policy is also initialized with the pre-trained model.
  - Did not share since hard to balance small amount of labeler data vs lots of reward samples.

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



Step 3

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

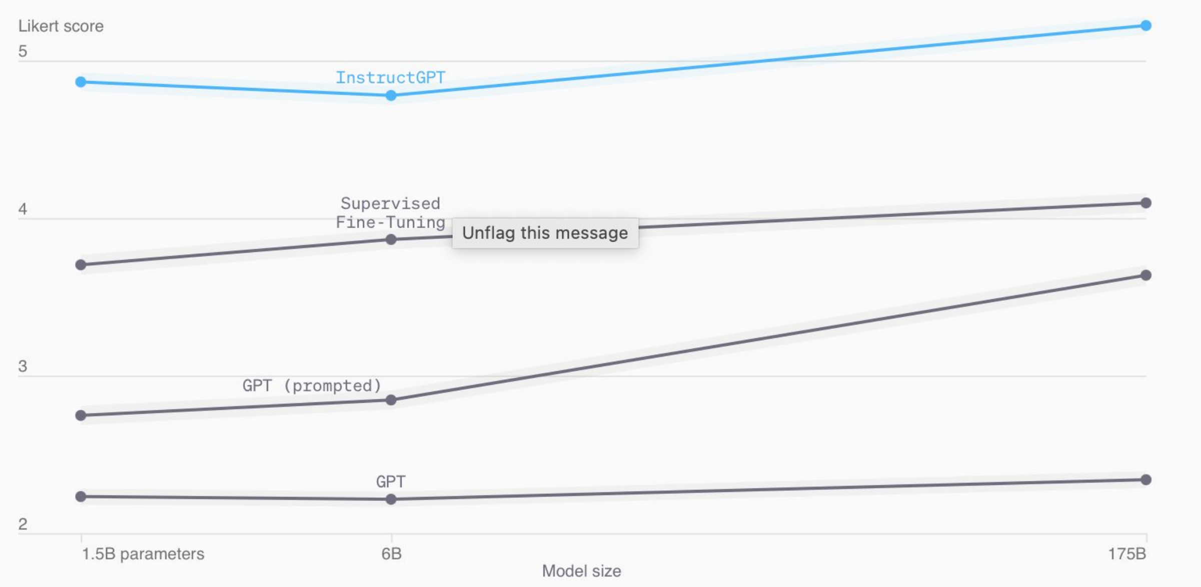
The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

LLM Evaluations

“Our labelers prefer outputs from our 1.3B InstructGPT model over outputs from a 175B GPT-3 model, despite having more than 100x fewer parameters. At the same time, we show that we don’t have to compromise on GPT-3’s capabilities, as measured by our model’s performance on academic NLP evaluations.”

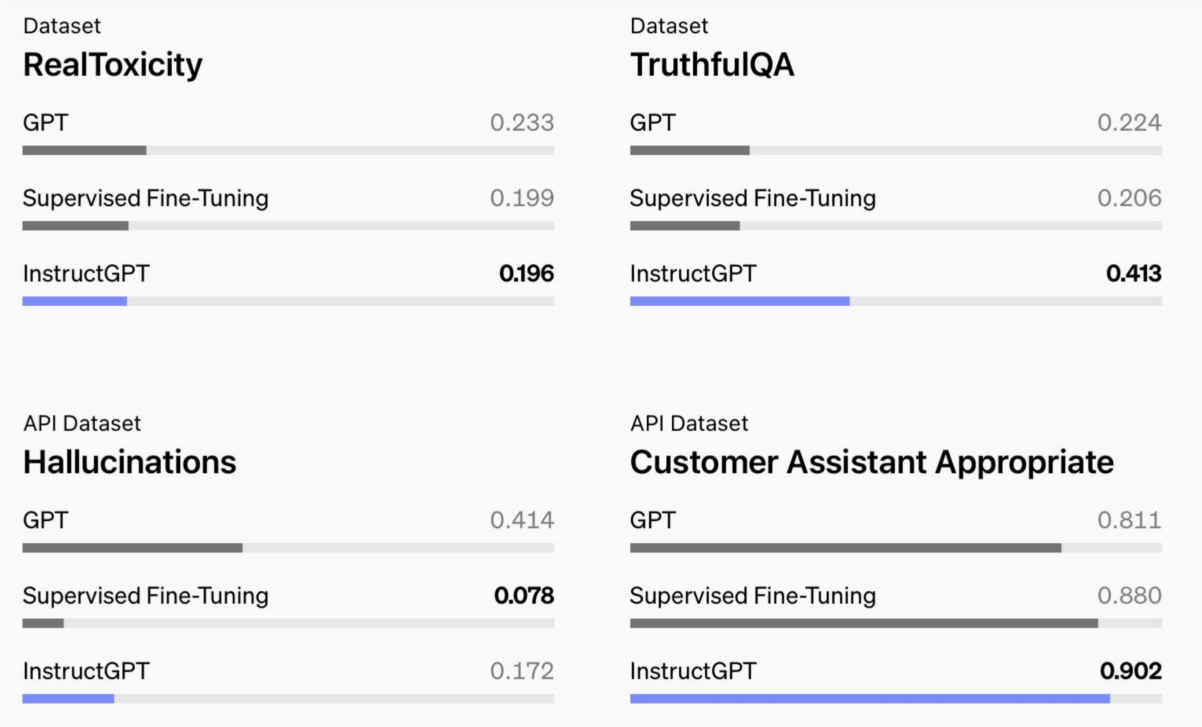
https://openai.com/index/instruction-following/



Quality ratings of model outputs on a 1–7 scale (y-axis), for various model sizes (x-axis), on prompts submitted to InstructGPT models on our API. InstructGPT outputs are given much higher scores by our labelers than outputs from GPT-3 with a few-shot prompt and without, as well as models fine-tuned with supervised learning. We find similar results for prompts submitted to GPT-3 models on the API.

LLM Evaluations

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



Neat Things That Were Skipped Today

- https://openai.com/index/solving-rubiks-cube/
- “Playing Atari with Deep Reinforcement Learning”
- “Simple random search of static linear policies is competitive for reinforcement learning” by Mania et al (2018)
- https://ai.sony/publications/A-Super-human-Vision-based-Reinforcement-Learning-Agent-for-Autonomous-Racing-in-Gran-Turismo/

Project Presentation Schedule

Wednesday 12/4

- Gukai Chen
- Houssain Ababou
- Yuchen Huang
- Zachary Meurer
- Akshara Ramprasad
- Chuqiao Feng, Assylnur Lesken, Jingyuan Liu
- Apoorva Gupta
- Kaya Daylor

Monday 12/9

- Hemangi Suthar
- Yangu Chen
- Fengyuan Shen, Jinhu Sun
- Qiuyi Feng
- Aashrey Jain
- Daniel Foley
- Matthew Maslow, Jonathan Neimann
- Can Erozer, Ozgur Sen

Feedback?