CSE 574 Planning and Learning Methods in AI

Ransalu Senanayake

Week 4

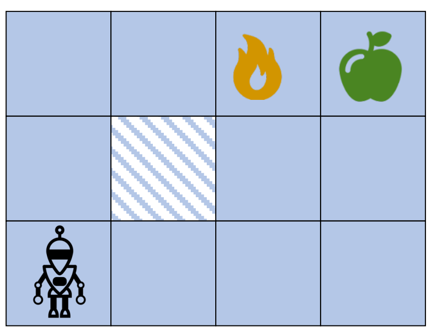
https://www.ponggame.org/

Supervised Learning vs. Reinforcement Learning

- Datasets (use static datasets vs. collect data trail & error)
- Data distribution (iid vs. action dependent states)
- Labels
- Objective (maximum likelihood vs maximum expected reward)

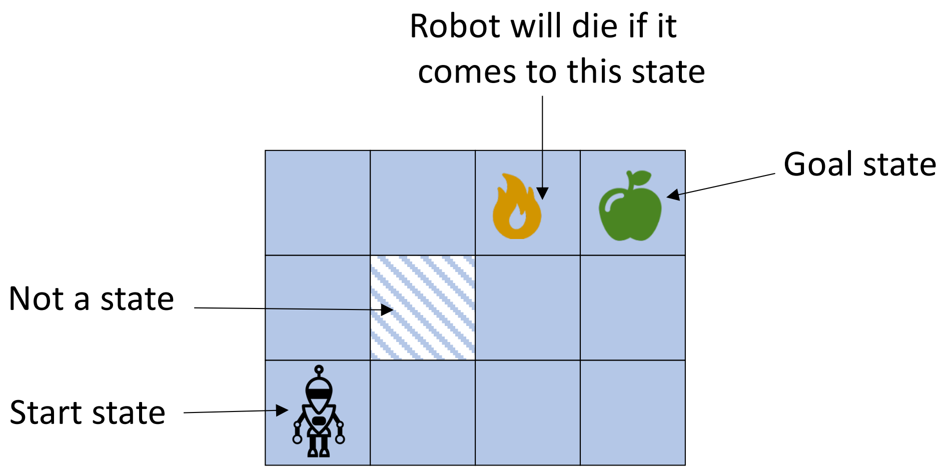
Markov Decision Process

$$\mathcal{M}: \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$$



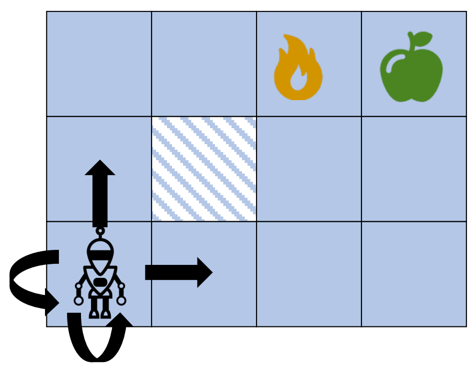
Markov Decision Process: State Space $(s \in \mathcal{S})$

- There are 11 states. Here, a state is simply a position in the world.



Markov Decision Process: Action Space ($a \in \mathcal{A}$)

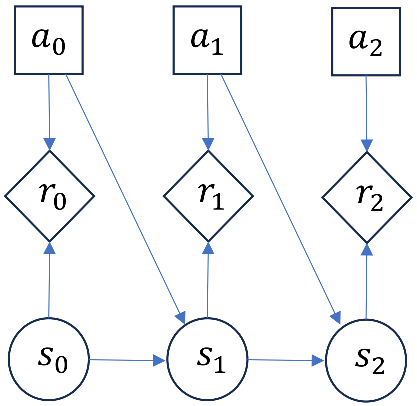
- The robot can take 4 actions (move up, down, left, or right). It can move to only 4 neighboring cells.
- Discrete time (step 1, step 2, step 3, …)
  - Markov property: Next state depends only on the current state and the action you take at the current state

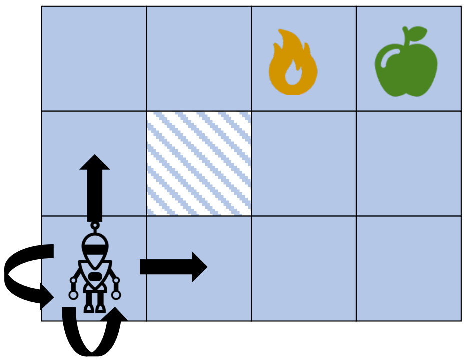


Andrey Markov

Markov Decision Process: Action Space $(a \in \mathcal{A})$

	Fully observable	Partially observable
Without action	Markov Chain	HMM
With action	MDP	POMDP

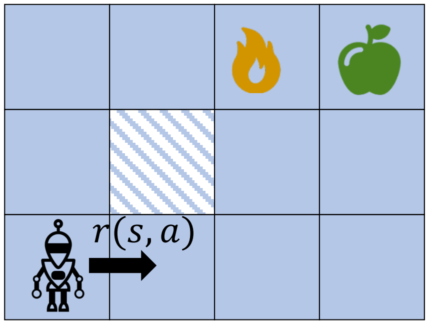




Markov Decision Process: Rewards (r)

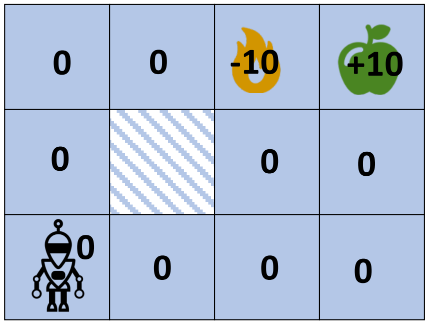
- A reward is received after transitioning from previous state to new state by taking a particular action $r(s_{t+1}, s_t, a_t)$ = r(next state, current state, current action). We can simply consider $r(s_t, a_t)$.
- Rewards can be negative or positive. Rewards can be engineered or learned.

$$r: S \times \mathcal{A} \rightarrow \mathbb{R}$$



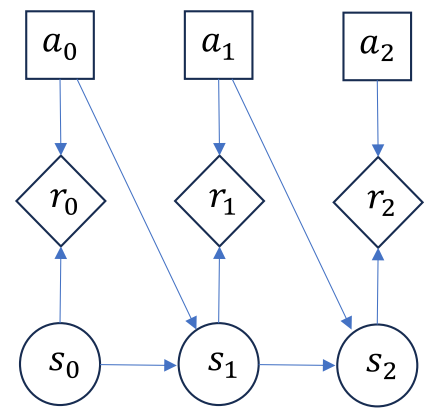
Markov Decision Process: Rewards

- The objective is to maximize the cumulative reward. To this end, we engineer rewards as, E.g.,
  1. If we go to the goal from any cell by taking whatever the action, r=+10
  2. If we go to the dead zone from any cell by taking whatever the action, r=-10
  3. No reward for any other action in any state

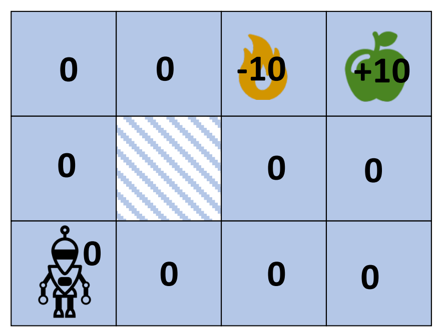


Markov Decision Process: Transition Operator

- Or the transition dynamics or the environment
- Transition dynamics and/or reward function constitutes a *model*



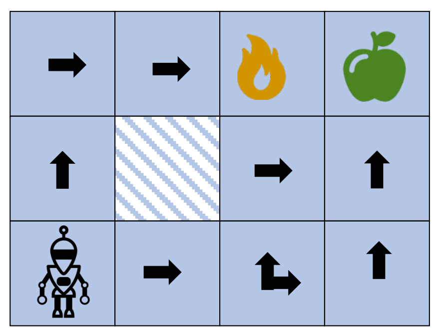
$$p(s_{t+1}|s_t, a_t)$$

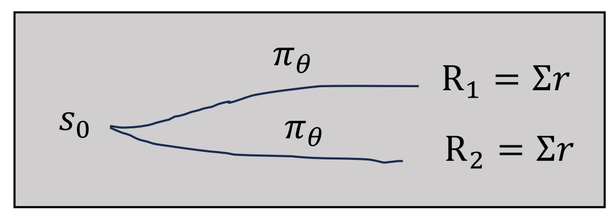


Reinforcement Learning: Policy

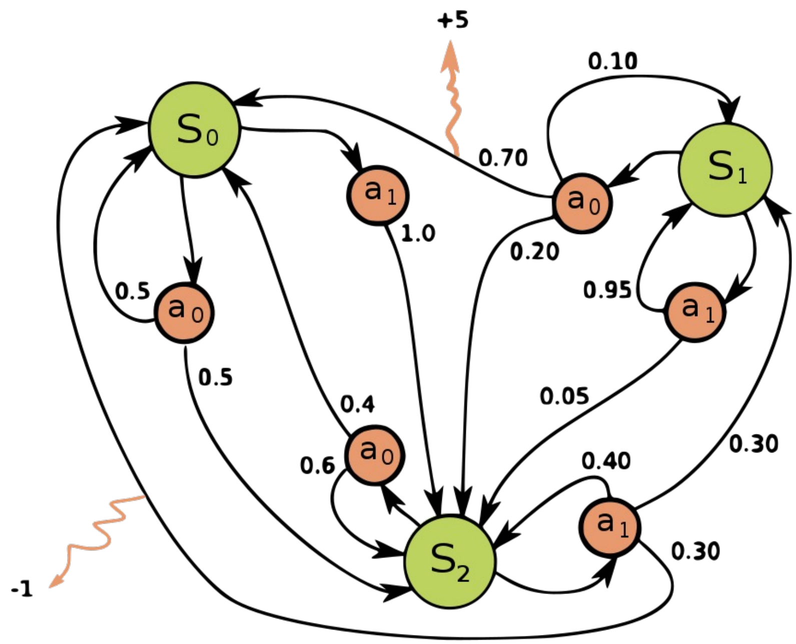
- Our objective is to find a policy ($\pi: S \to \mathcal{A}$) that maximizes the cumulative sum of rewards.
- If we have a policy, then we know what actions to take at any state.
- A deterministic policy maps states to actions (if we take action a at s, we'll end up in s_{t+1})
- A stochastic policy map states to distributions of actions (if we take action a_t at s_t, we'll end up in various s_{t+1}, each with a different probability)

$$\max_\theta \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \right]$$





Reinforcement Learning: Policy

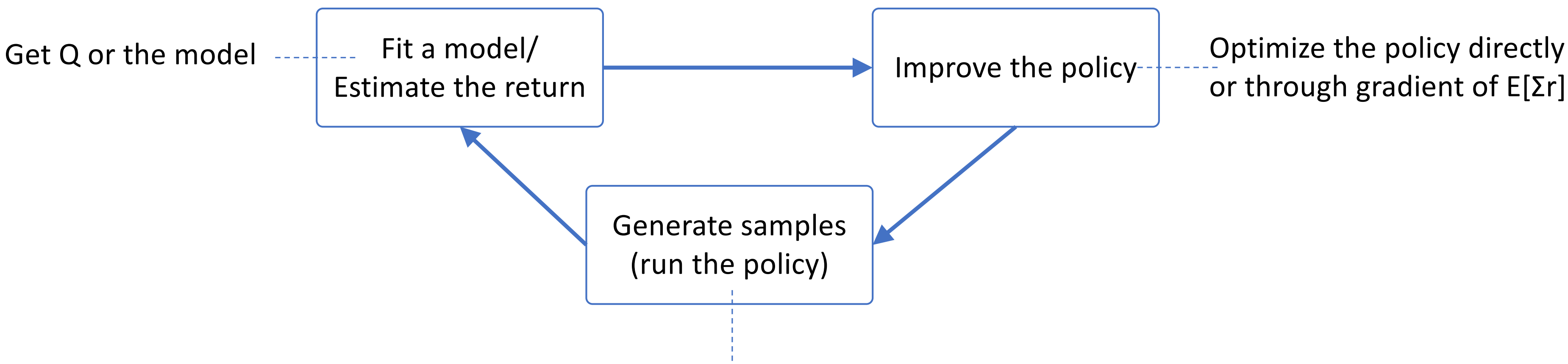


$$\max_\theta \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\sum_{t'=t}^{T} r(s_{t'},a_{t'})\right]$$

and find the policy

$$\pi_\theta(\tau) = \pi_\theta(s_1, a_1, \ldots, s_T, a_T) = p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)$$

The RL loop



On-policy: Use our current policy for sampling. Hence, if we update the policy at iteration t, we have to resample. Hence, sample inefficient. E.g., SARSA

Off-policy: Use samples by other means (e.g., previously collected samples, randomly/greedily collected samples, etc.). Hence, sample efficient. We might even not need the policy at all. E.g., Q-learning

Some resources adapted from Chelsea Finn's CS224R Reinforcement Learning

Q-function vs. value function

- Q-function: Expected return of taking an action at a given state

$$Q^{\pi}(s_t, a_t) = \sum_{t'=t}^{T} \mathbb{E}_{\pi}[r(s_{t'}, a_{t'}) | s_t, a_t]$$

- Value-function: Expected return of an action

$$V^{\pi}(s_t) = \mathbb{E}_{a_t \sim \pi(a_t|s_t)}[Q^{\pi}(s_t, a_t)]$$

$$= \sum_{t'=t}^{T} \mathbb{E}_{\pi}[r(s_{t'}, a_{t'}) | s_t]$$

Model-based vs. model-free RL

A **model** represents how the environment behaves i.e., $p(s_{t+1}|s_t, a_t)$ and/or $r(s_t, a_t)$ E.g., rules of a game, physics, human interactions

Model-based: Learn the model then plan to find the policy. Planning algorithms such as MCTS or dynamic programming can be used for the second stage. If we learn a good model, then planning is sample efficient. But it's difficult to learn a good model for complex environments.
E.g., trajectory optimization (LQR), MPC, PILCO (or NN based), MCTS, Cross Entropy Method, Dyna

Model-free: Instead of learning an explicit model, it interacts with the environment to collect state-action pair data to learn a policy. Sample inefficient because it has to interact a lot. This is especially true for high-dimensional state/action spaces. E.g.,

- value-based (estimate the value) e.g., Q-learning/DQN, DDQN, Dueling DQN, SARSA
- policy gradient methods (learn a parameterized policy) e.g., REINFORCE , TRPO, PPO
- Actor-Critic methods (combines value based with policy-based by having an actor network and a policy network e.g., A2C, TRPO
- Entropy-regularized methods (for better exploration) e.g., SAC, TRPO
- For handling sparse rewards E.g., HER

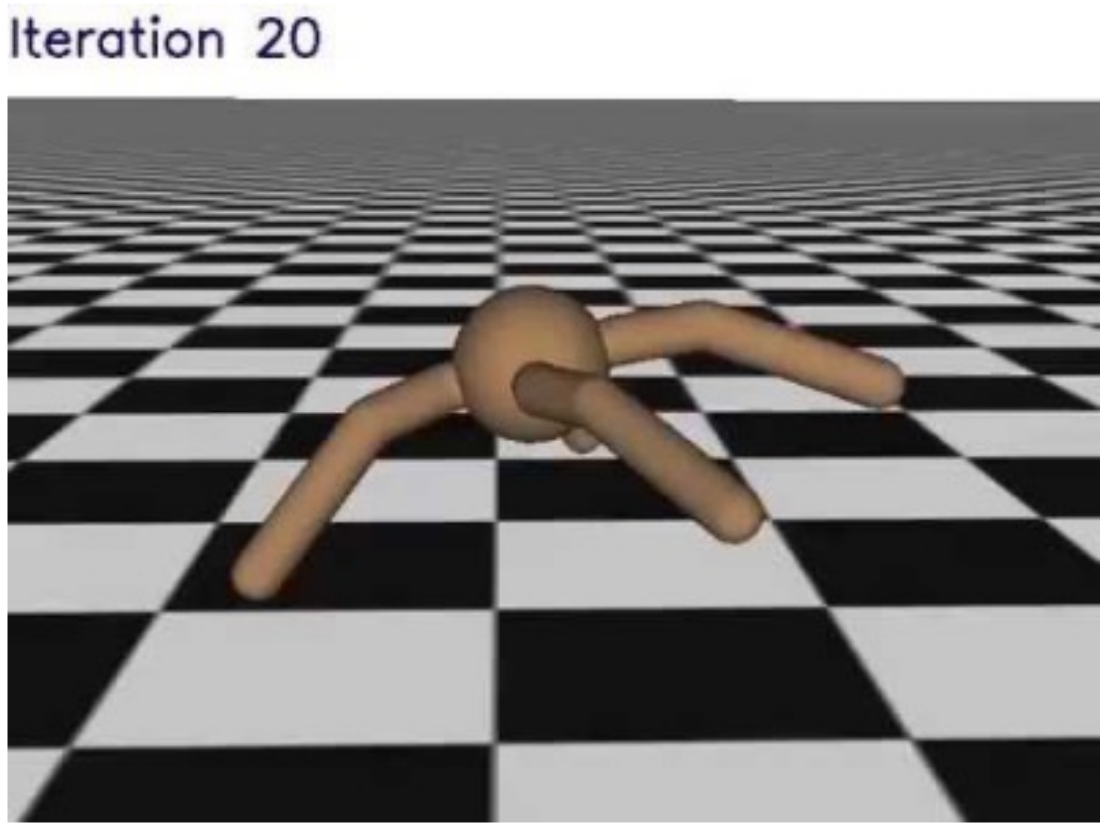
Policy Iteration for Pong

https://www.ponggame.org/

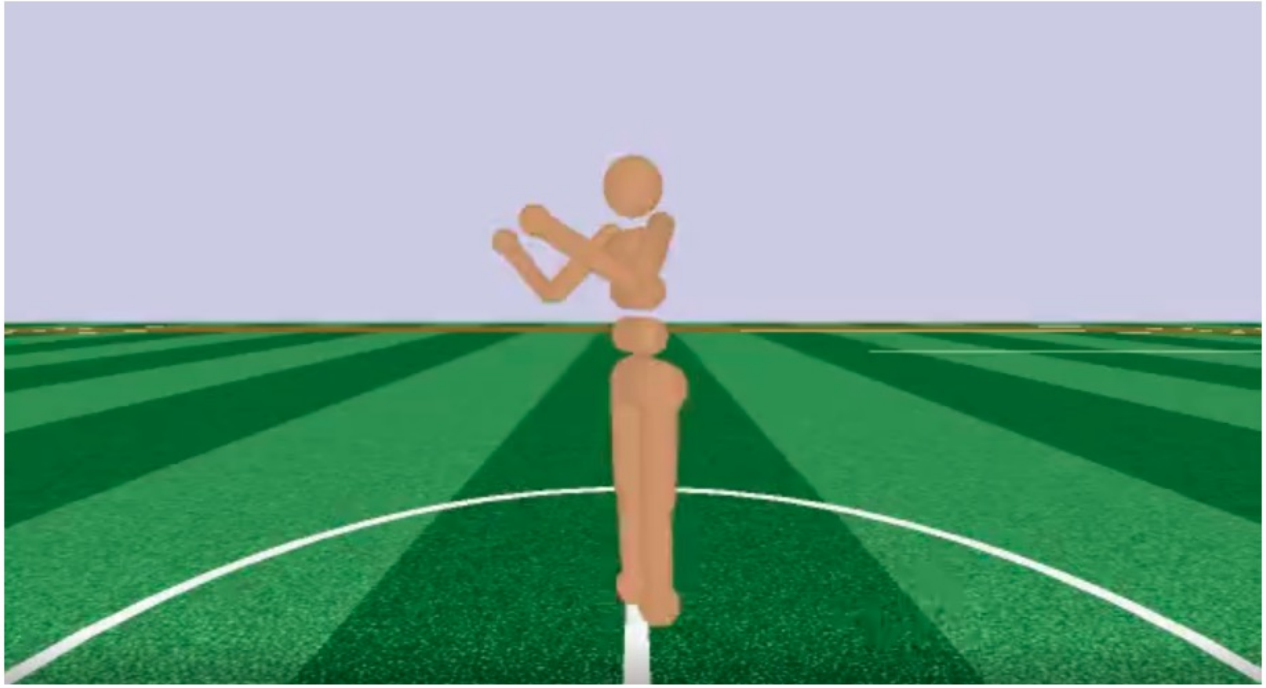
DQN


```python
import gym

from stable_baselines.common.vec_env import DummyVecEnv
from stable_baselines.deepq.policies import MlpPolicy
from stable_baselines import DQN

env = gym.make('CartPole-v1')

model = DQN(MlpPolicy, env, verbose=1)
model.learn(total_timesteps=25000)
model.save("deepq_cartpole")

del model # remove to demonstrate saving and loading

model = DQN.load("deepq_cartpole")

obs = env.reset()
while True:
    action, _states = model.predict(obs)
    obs, rewards, dones, info = env.step(action)
    env.render()
```

TRPO

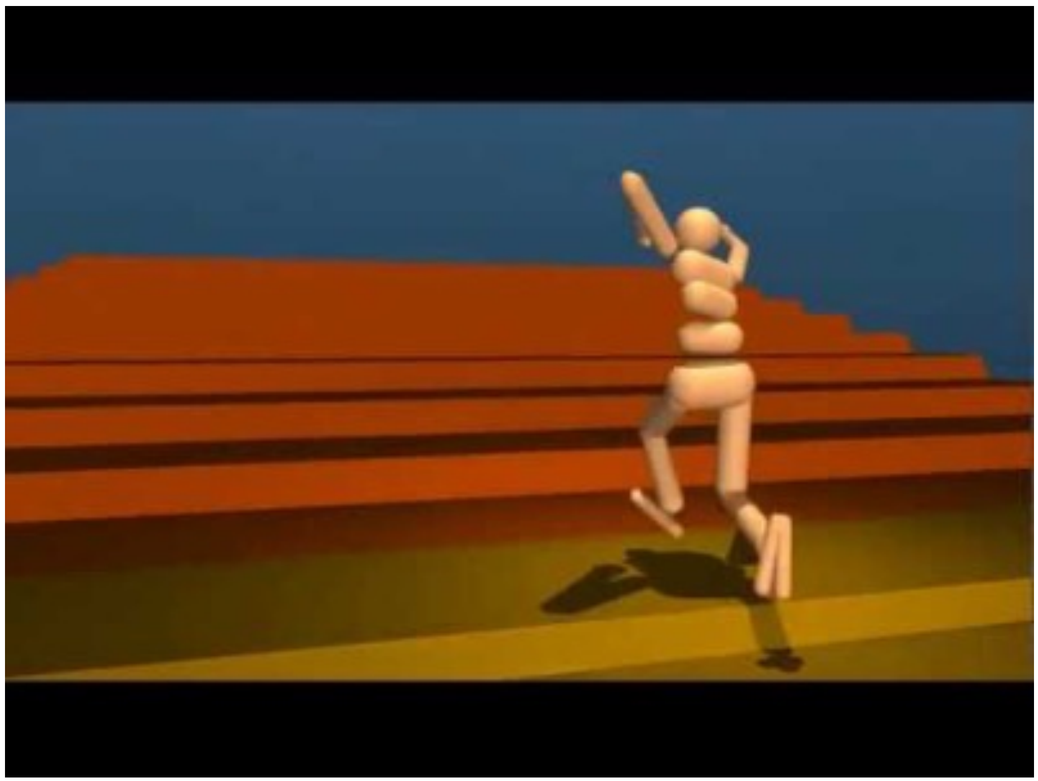


PPO



https://openai.com/blog/openai-baselines-ppo/

DeepMind Navigating Obstacles

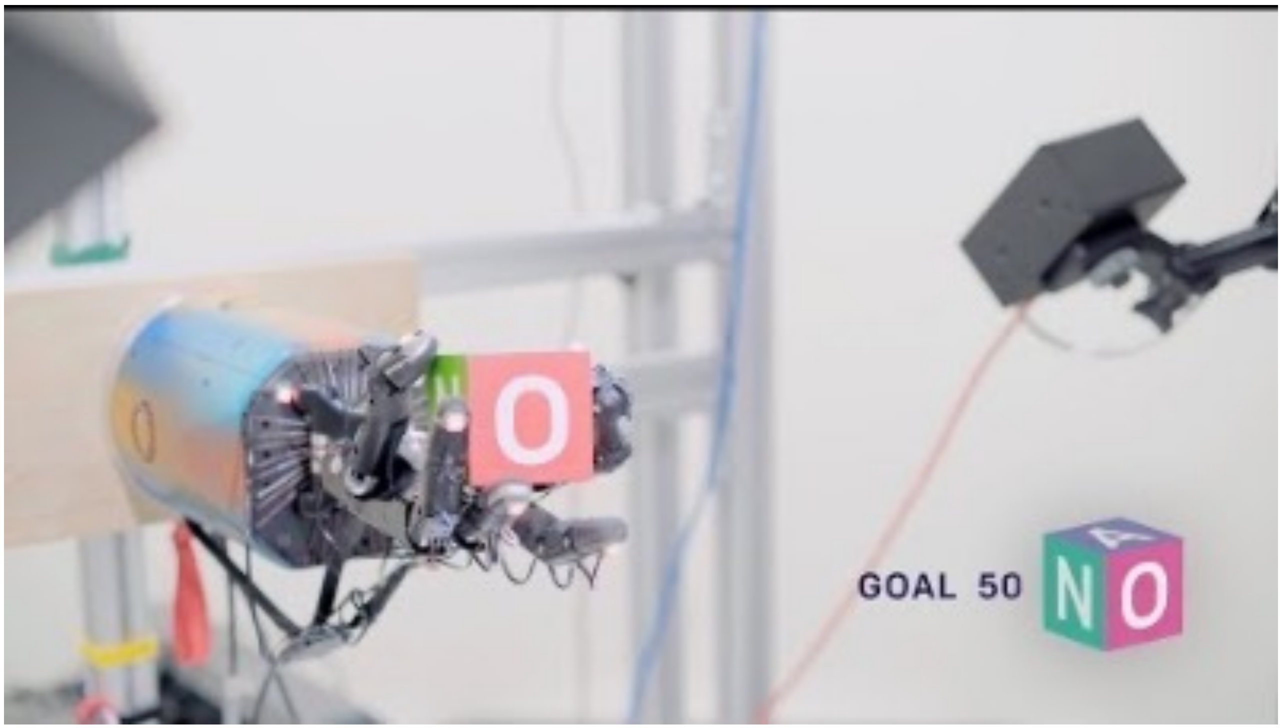


DeepMind AlphaGo Computer Player

- AlphaZero, AlphaGo Zero (2017), AlphaGo Master, AlphaGo Lee, AlphaGo Fan
- Uses neural networks and Monte Carlo Tree Search (MCTS)

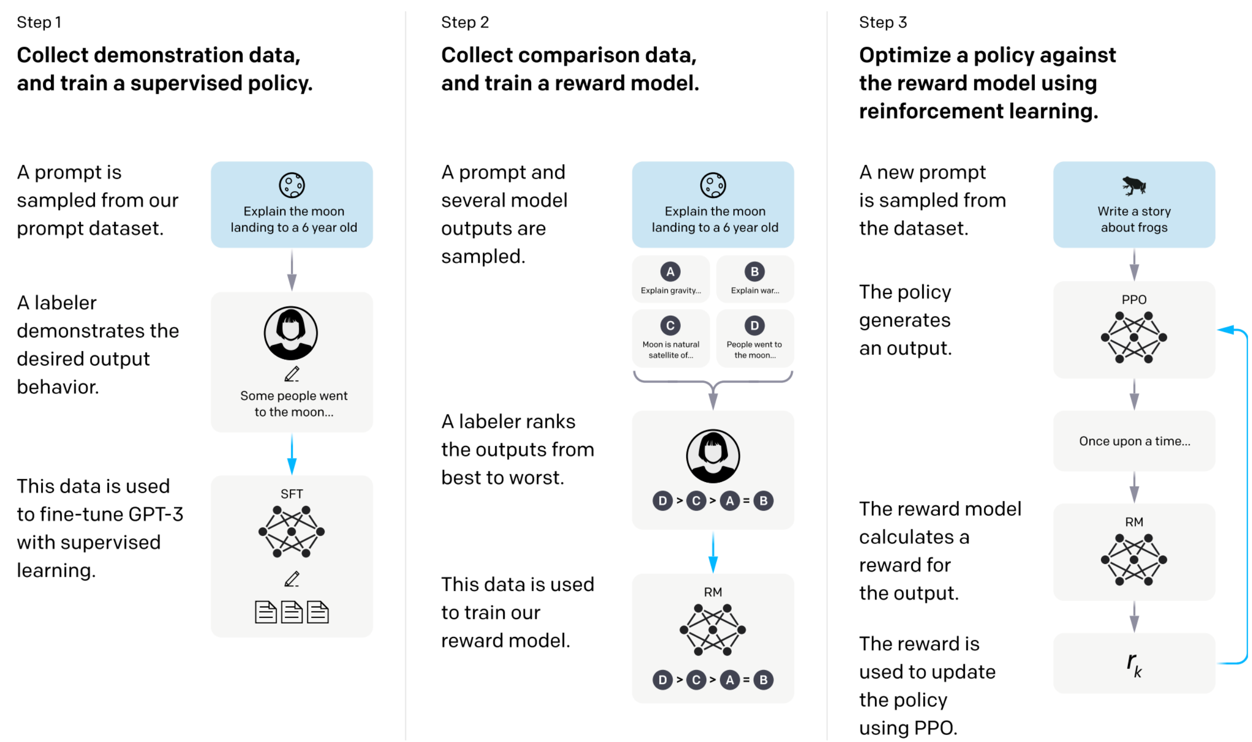
Playing DOTA2 OpenAI Five (2018)

OpenAI Dexterous Manipulation



Wayve.ai Learning to Drive in a Day

OpenAI ChatGPT



InstructGPT

Some resources adapted from Chelsea Finn's CS 224R Reinforcement Learning