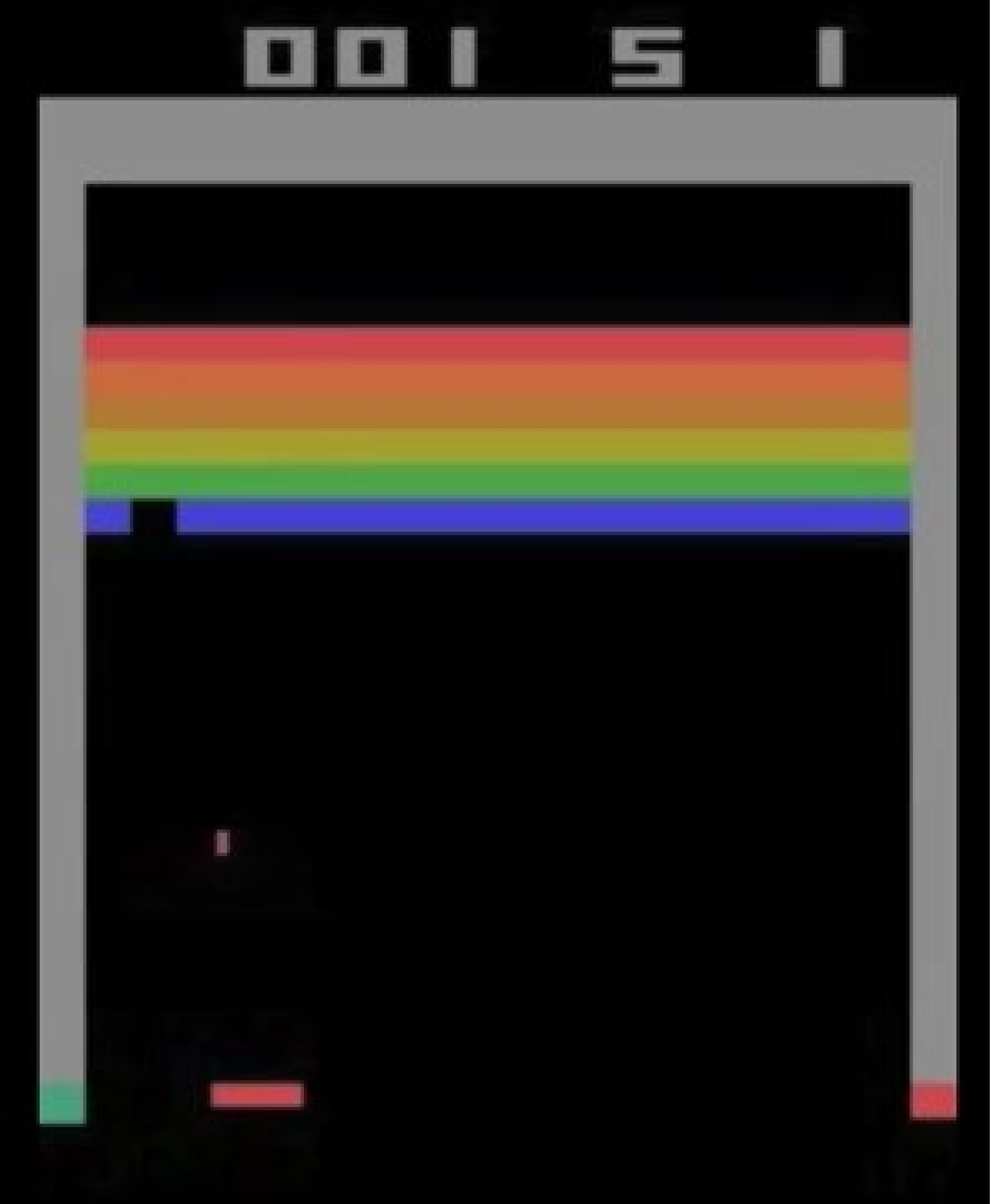
Reinforcement Learning

Applied Machine Learning
Derek Hoiem



So far, we've learned how to train models to predict some value.

How could we train an AI agent to play Breakout?

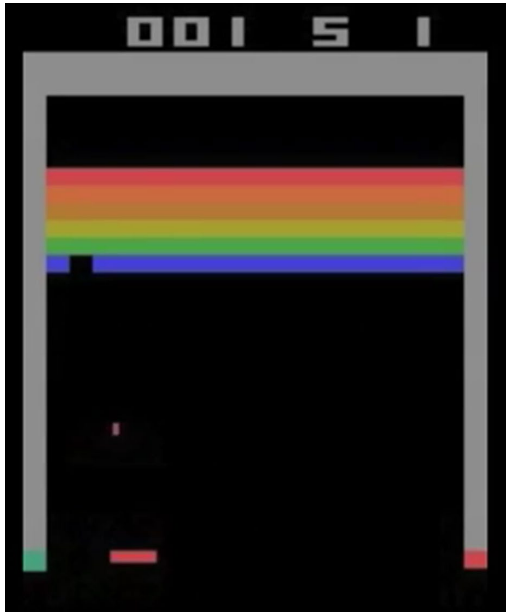
This class: Reinforcement Learning

- Deep Q-Learning with Atari Games
- Proximal Policy Optimization for Hide and Seek
- Other approaches to RL, and RLHF

Problem statement for RL

Learn a **policy** that, given **state,** outputs the **action** with maximum **expected reward**

- State: everything the agent knows about the environment, e.g. images of current and past screens
- Action: things the agent can do, e.g. move left, stay, move right
- Policy: function that outputs an action given a state
- Reward: the thing that's being maximized, e.g. the score
- Expected Reward: maximize the sum of the current reward and *discounted future rewards*, with the expectation over possible sequences

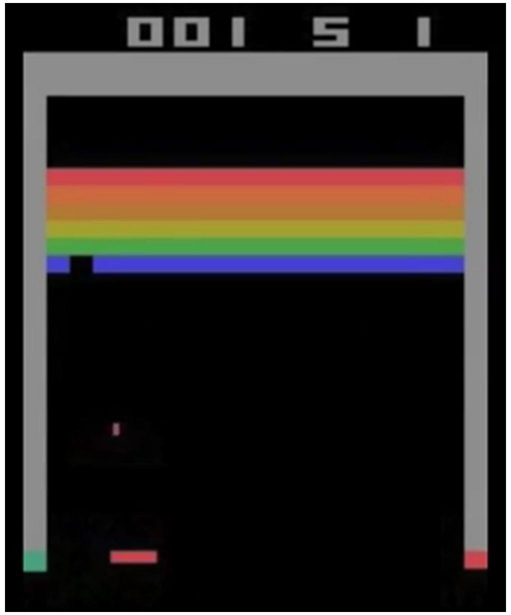


Q-Learning

Learn $Q(s_t, a_t) \rightarrow v$: given a state s_t and action a_t, what is the expected total future reward value v?

If I move left/stay/right, what will be my time-discounted score?

Once learned, the optimal policy is to choose the action a_t that maximizes $Q(s_t, a_t)$



Discounted Rewards and Bellman Equation

The value function is the expected total rewards received from following the optimal policy

$$Q(s_t, a_t) = \mathbb{E}[R(s_t, a_t) + \sum_{i=1}^{\infty} \gamma^i \max_{a_{t+i}} R(s_{t+i}, a_{t+i})]$$

E.g., suppose

- reward is points added to score
- $\gamma = \frac{1}{2}$
- 10 points are earned at t and again at $t + 3$, and at no other time.

What is the total discounted reward obtained at t?

$$10 + 10 * \frac{1}{2}^3 = 10.125$$

Discounted Rewards and Bellman Equation

The value function is the expected total rewards received from following the optimal policy

$$Q(s_t, a_t) = \mathbb{E}[R(s_t, a_t) + \sum_{i=1}^{\infty} \gamma^i \max_{a_{t+i}} R(s_{t+i}, a_{t+i})]$$

The *Bellman Equation* models this recursively, i.e. the expected total rewards are the immediate rewards plus the expected rewards at the next time step

$$Q(s_t, a_t) = \mathbb{E}[R(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})]$$

How do we learn this Q function?

- Play lots of games, and use Bellman's equation to update the parameters of the Q-function

$$L_i\left(\theta_i\right) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[\left(y_i - Q\left(s,a;\theta_i\right)\right)^2\right]$$

$$\nabla_{\theta_i} L_i\left(\theta_i\right) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\underbrace{\left(r+\gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\right)} \underbrace{\nabla_{\theta_i} Q(s,a;\theta_i)}\right]$$

How much we want to increase or decrease Q to match our "oracle" estimate from the next time step, i.e. difference between:

1. total reward under previous parameters (current reward, plus prediction at t+1)
2. predicted reward under current parameters

How much changing each parameter will change Q

[Minh et al. 2013, Deep Mind]

What are some drawbacks to playing one game at a time and learning from each step?

- May not be efficient
  - Need to play a lot of games before you start making progress, and could get stuck in some long games
  - Consecutive states are very similar
- May be unstable
  - Due to correlated consecutive states, may overfit to particular cases or swing back and forth with each episode

[Minh et al. 2013, Deep Mind]

Deep Q-Learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

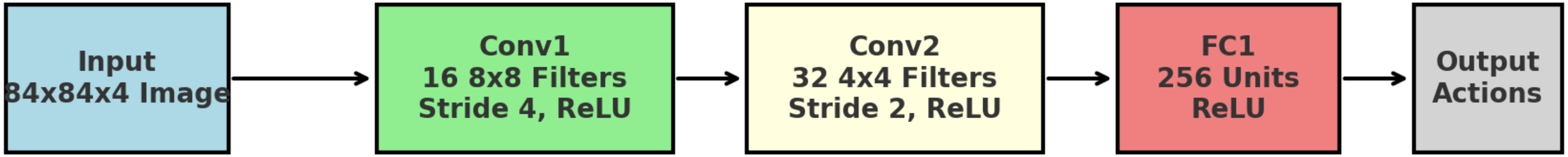
Generate next action in episode and store results

Train on random sample of stored state/action/reward experiences

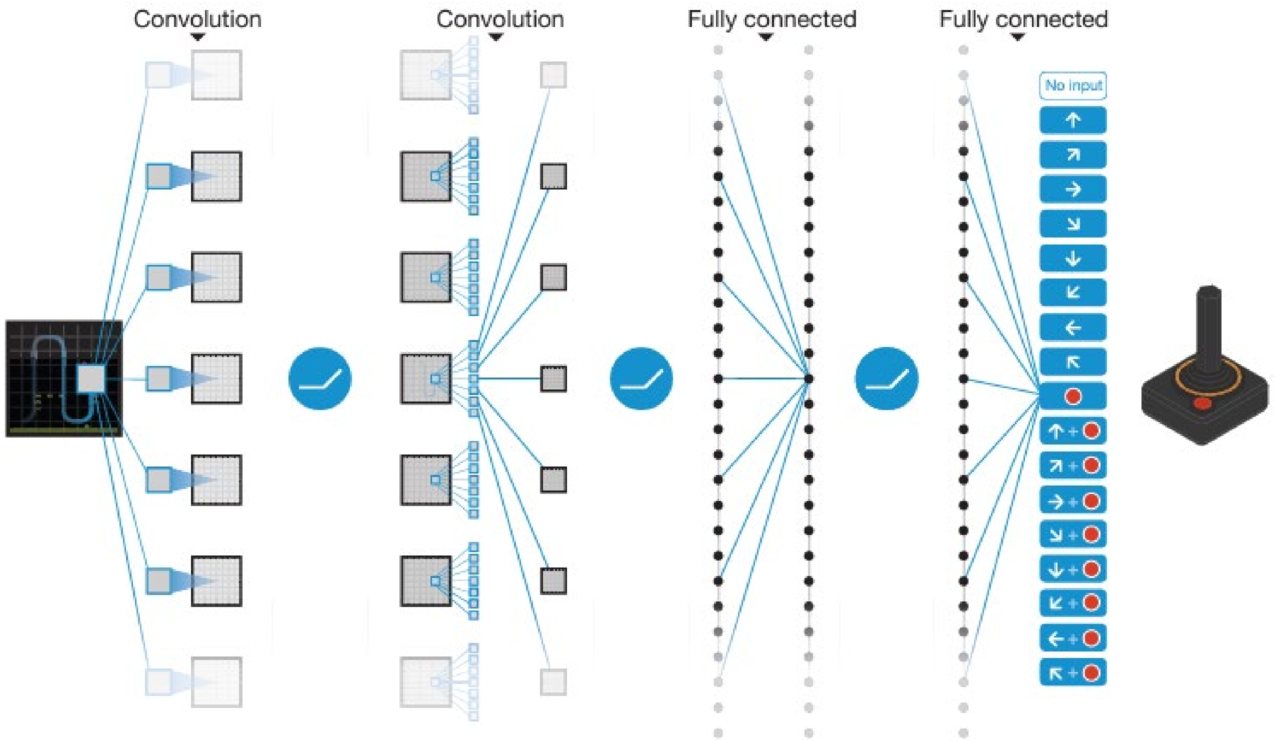
[Minh et al. 2013, Deep Mind]

Modeling the Q function with a neural network

Four grayscale cropped frames



Outputs the *value* for each action



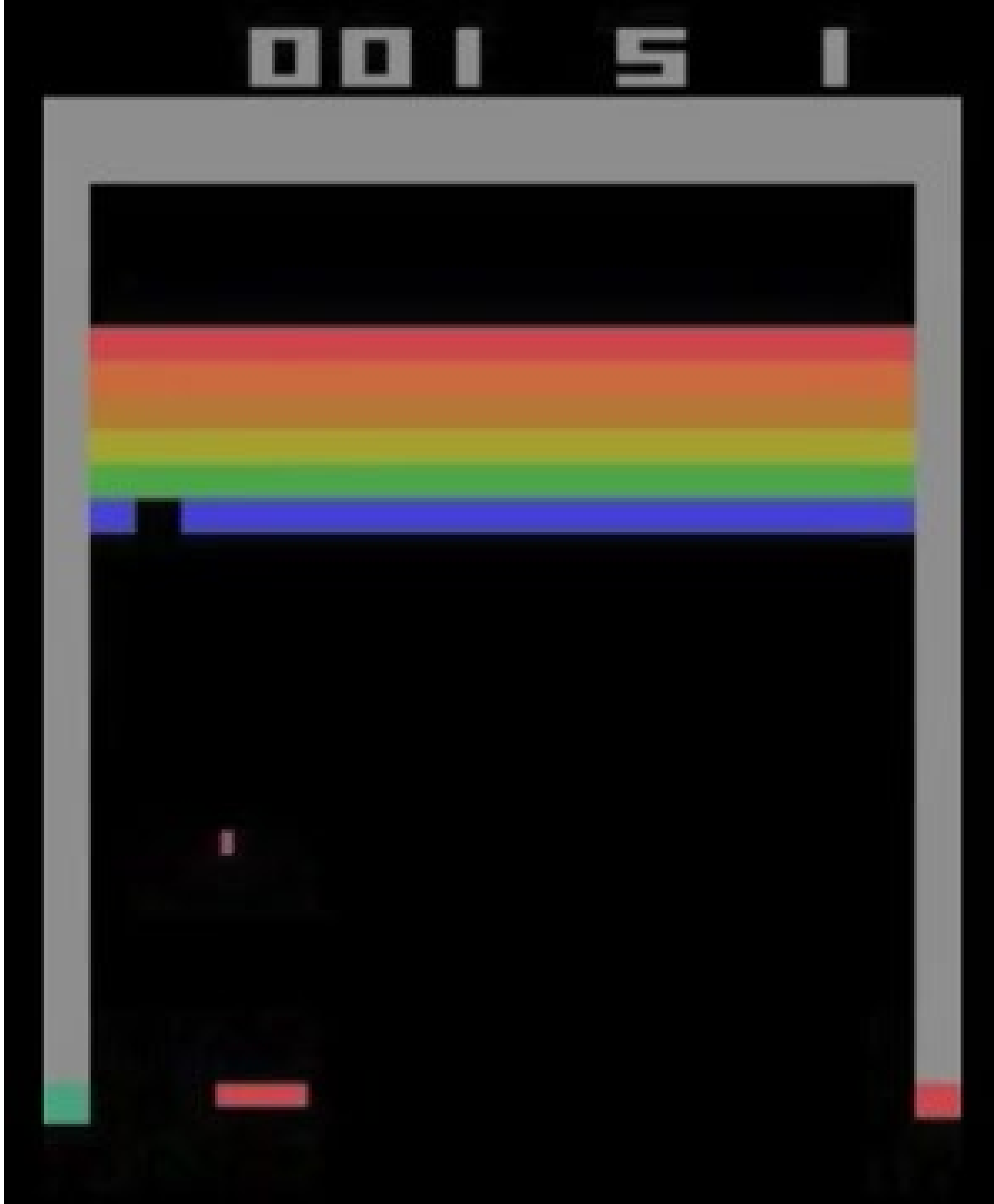
[Minh et al. 2013, Deep Mind]
[Minh et al. 2015, Nature]

- What should the reward for a given timestep be?

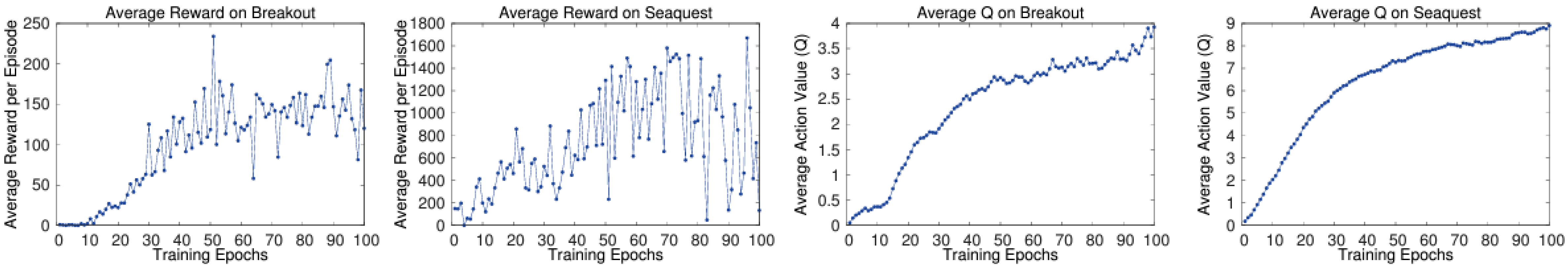
- What is a reasonable discount γ factor? (assume that one time step is one animation frame or 1/15 second of normal play)

Reward is 1, 0, or -1 for score increased, unchanged, or decreased

$\gamma = 0.99$



Reward improves overall, and Q-value improves stably

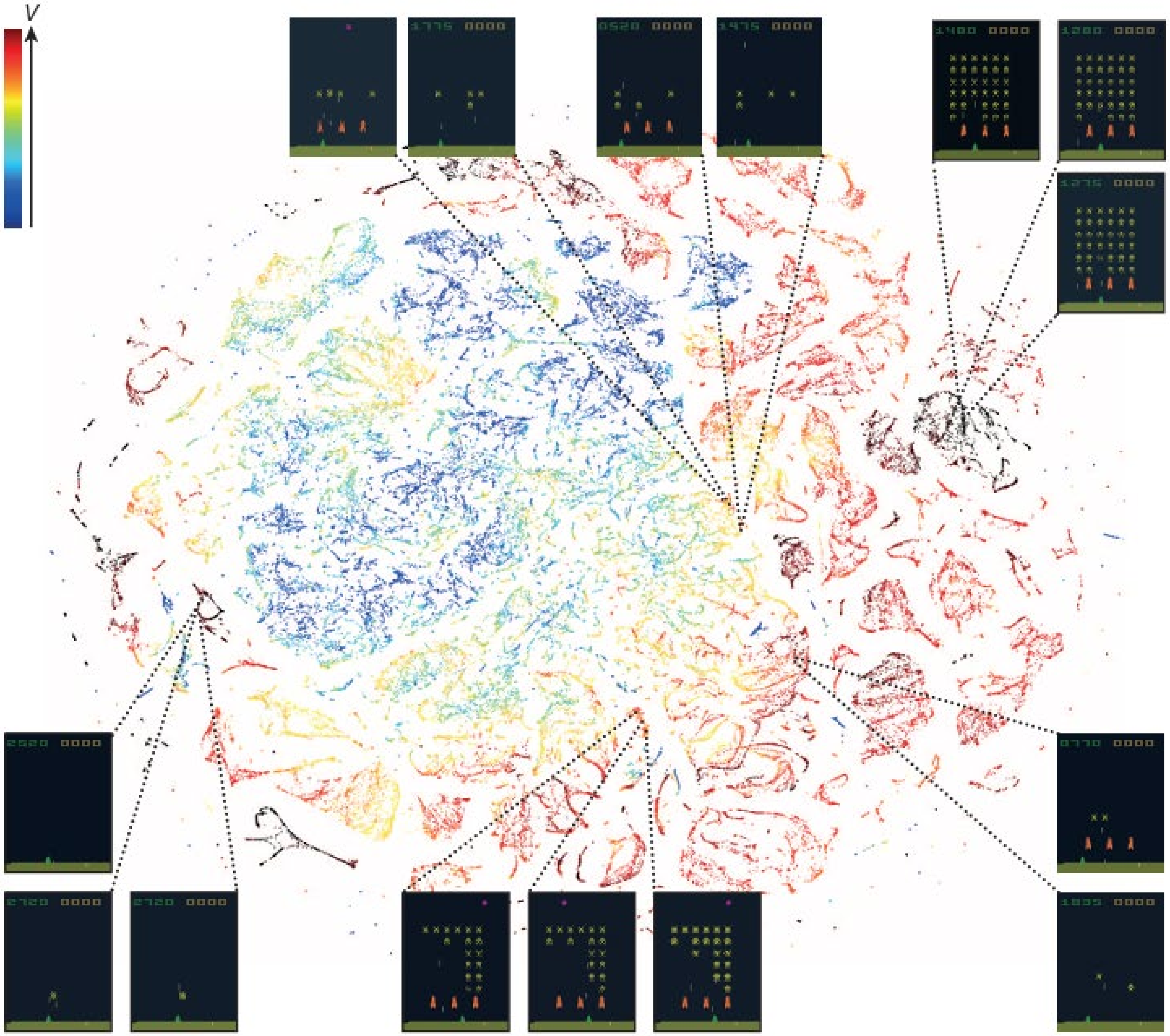


A: Enemy appears (scoring opportunity)

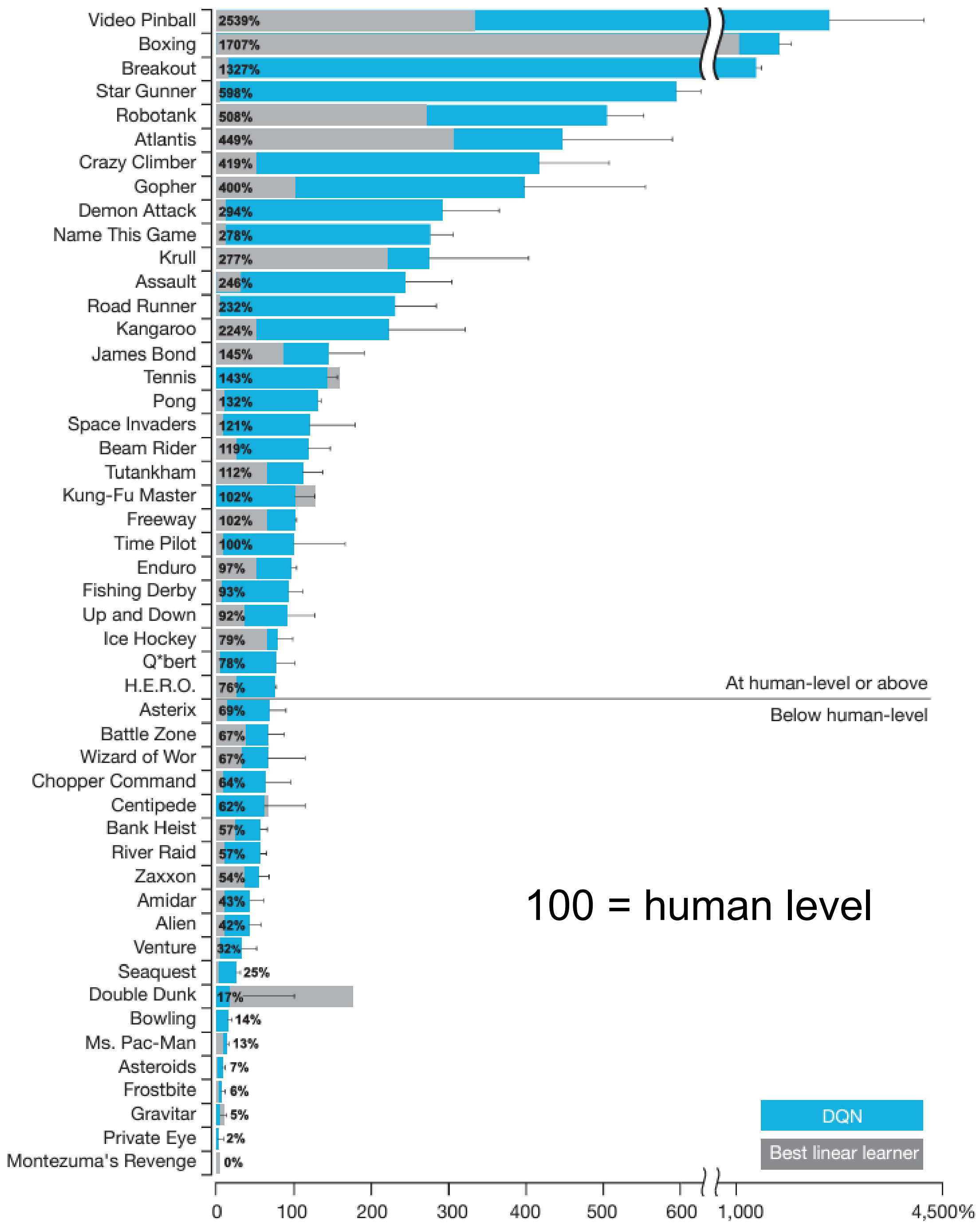
B: Enemy about to be hit by torpedo (soon to score)

C: Scored! Back to previous level

T-SNE of hidden state for Space Invaders



[Minh et al. 2015, Nature]



100 = human level

[Minh et al. 2015, Nature]

Simple code with nice explanation

https://medium.com/@shruti.dhumne/deep-q-network-dqn-90e1a8799871

Q1-Q4

https://tinyurl.com/441-fa24-L25

State encodes distances to objects and positions of other visible agents, boxes, and ramps

Action is directly predicted from state

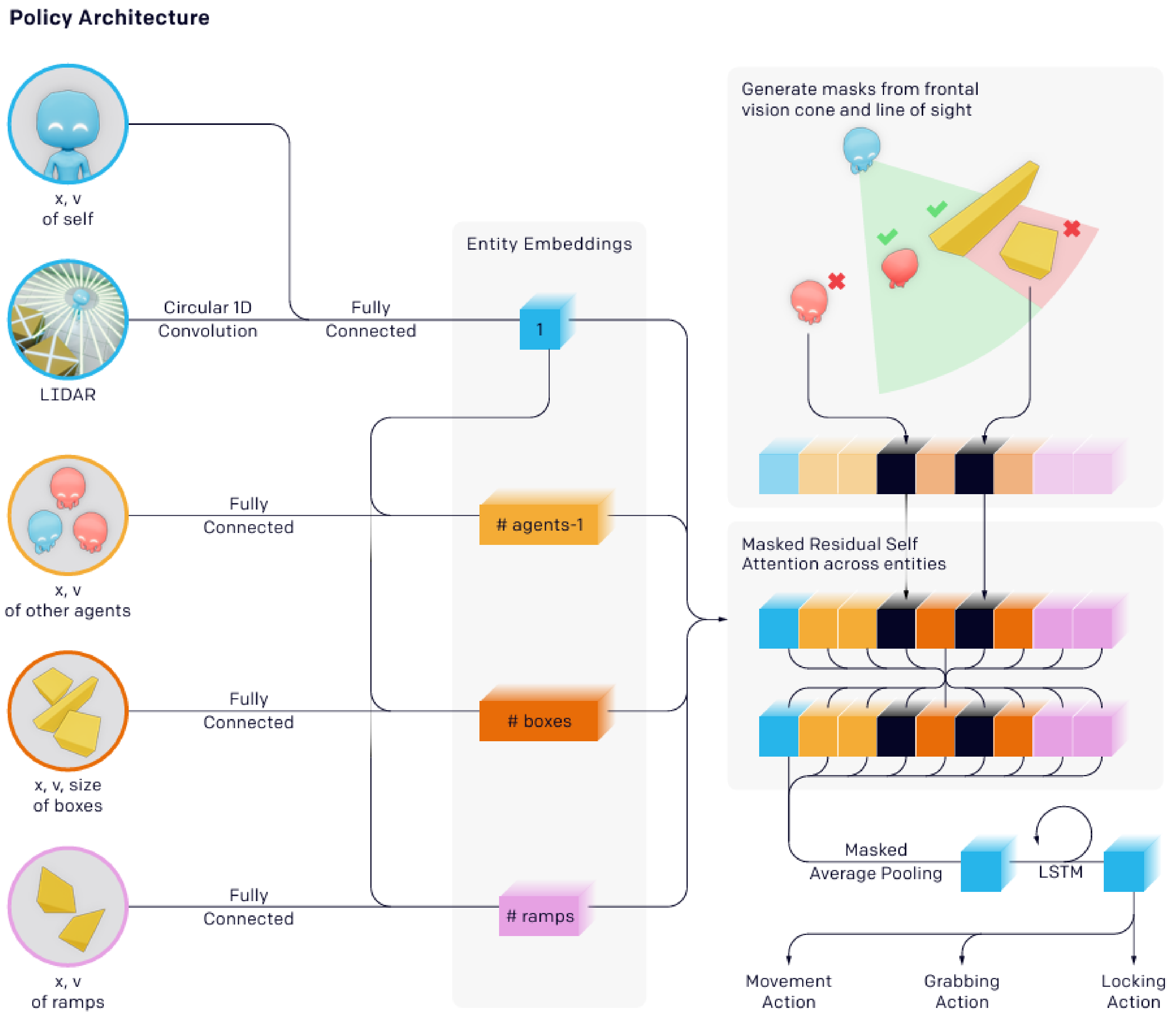
(reward value of a state-action pair is not modeled)

Teams keep challenging each other to develop new strategies

Q: What do you think is the reward?

A: Hider = 0 if seen, 1 if not

Seeker = 1 if sees, 0 if not



Policy Optimization

Typically, optimize a value function $V_\theta(s)$ and a policy function $\pi_\theta(a_t|s_t)$

$$\hat{g} = \hat{\mathbb{E}}_t\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\hat{A}_t\right]$$

Solve for parameters that maximize "Advantage"

$$A_t = r_t + V(S_{t+1}) - V(S)$$

Advantage is reward plus improvement in expected future reward

[Shulman et al. 2017]

Trust Region Policy Optimization (TRPO)

Improve likelihood of increasing advantage without diverging too much from previous policy

$$\underset{\theta}{\text{maximize}}\,\hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t - \beta\,\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]\right]$$

[Shulman et al. 2017]

Proximal Policy Optimization (PPO)

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right]$$

Clipped advantage objective

Value prediction loss
$(V_\theta(s_t) - V_t^{\text{targ}})^2$

Entropy bonus (don't be too confident)

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

[Shulman et al. 2017]

Proximal Policy Optimization (PPO)

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right]$$

Clipped advantage objective

Value prediction loss

Entropy bonus (don't be too confident)

Algorithm 1 PPO, Actor-Critic Style

---

for iteration$=1, 2, \ldots$ **do**
    for actor$=1, 2, \ldots, N$ **do**
        Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
        Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    end for
    Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
    $\theta_{\text{old}} \leftarrow \theta$
end for

[Shulman et al. 2017]

Comparison on Atari Games

PPO beats updated version of Deep Q-Learning (A2C) in almost all cases

	A2C	ACER	PPO	Tie
(1) avg. episode reward over all of training	1	18	**30**	0
(2) avg. episode reward over last 100 episodes	1	**28**	19	1

PPO learns faster than ACER (a different policy optimization algorithm) but tends not to do better in the end

Another use of PPO

https://www.youtube.com/watch?v=L_4BPjLBF4E

Comparing DQN and PPO

- PPO can handle continuous action space, while DQN can't easily (as it outputs value for each action)

- On-policy: PPO generates trajectories using the same policy that it updates (on-policy), while DQN's experience replay updates based on a previous policies and sometimes tries uniformly random actions (off-policy)

- PPO can often explore better, as it chooses next action stochastically, rather than DQN's ϵ-greedy sampling (usually best, else uniformly random action)

Four RL Approaches

- Behavioral cloning: Predict which action an expert would take, given the current state
- Policy learning: Learn a value function (predicting future reward) from state, and a policy function that predicts which action maximizes value
- Q-learning: Learn which state-action pairs have the highest value

"Model Free"

- World Models: Learn value function and to predict the next state given an action and current state; then choose action that maximizes value of next state

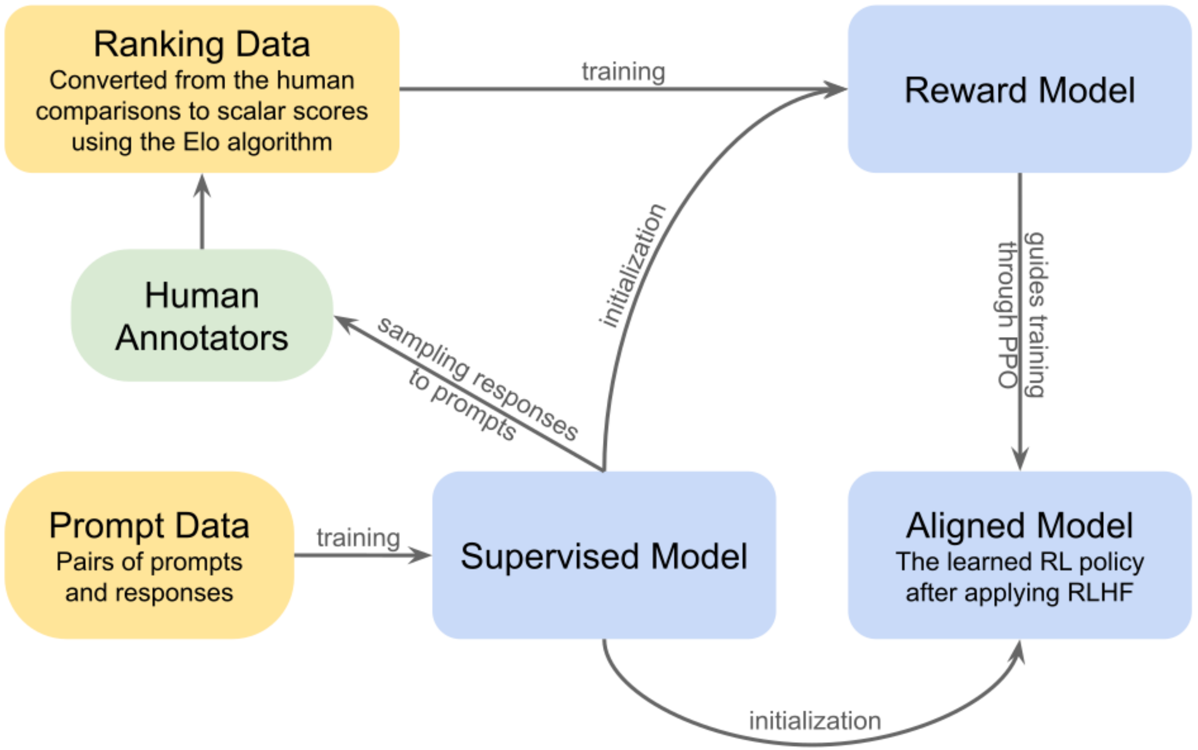
"Model-based"

Q5-Q8

https://tinyurl.com/441-fa24-L25

RLHF: Reinforcement Learning with Human Feedback

ChatGPT improves on GPT by using RLHF, using PPO to generate outputs that users prefer



https://commons.wikimedia.org/wiki/File:RLHF_diagram.svg

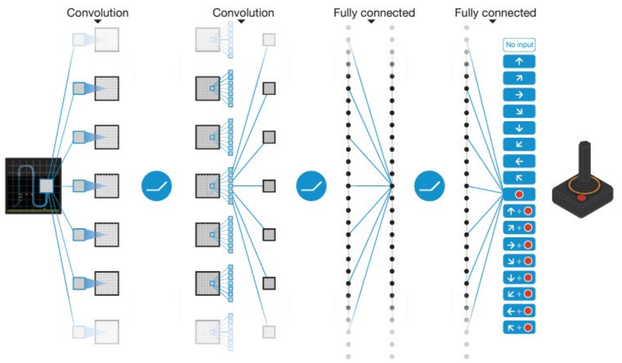
Things to remember

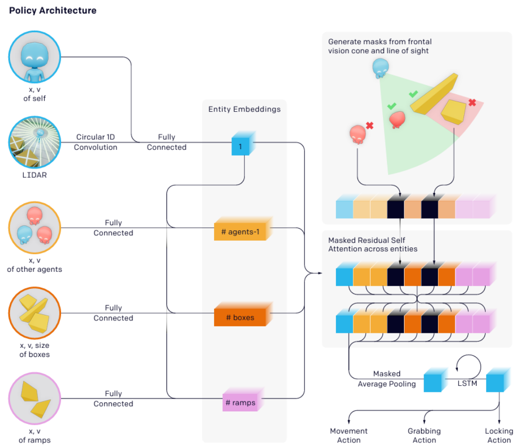
- Reinforcement learning applies when a sequence of actions is needed to complete a goal

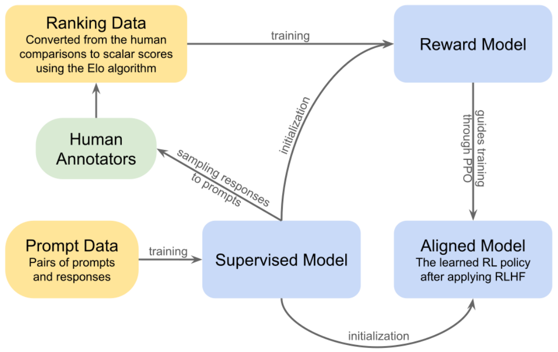
- Q-Learning predicts the long-term rewards that will result from a given state and action

- PPO predicts which action will result in the highest value state

- To better align with user preference, a common solution is to train a model (self-supervised and/or on datasets) and then tune it using RLHF to create more preferred results







Almost done!

- Thursday – Semester Wrap-up and Review
- Exam 3 – Thursday to Tuesday (Dec 5-10)
- Final Project – due Dec 15