Reinforcement Learning

Applied Machine Learning
Joshua Levine

Today’s Lecture

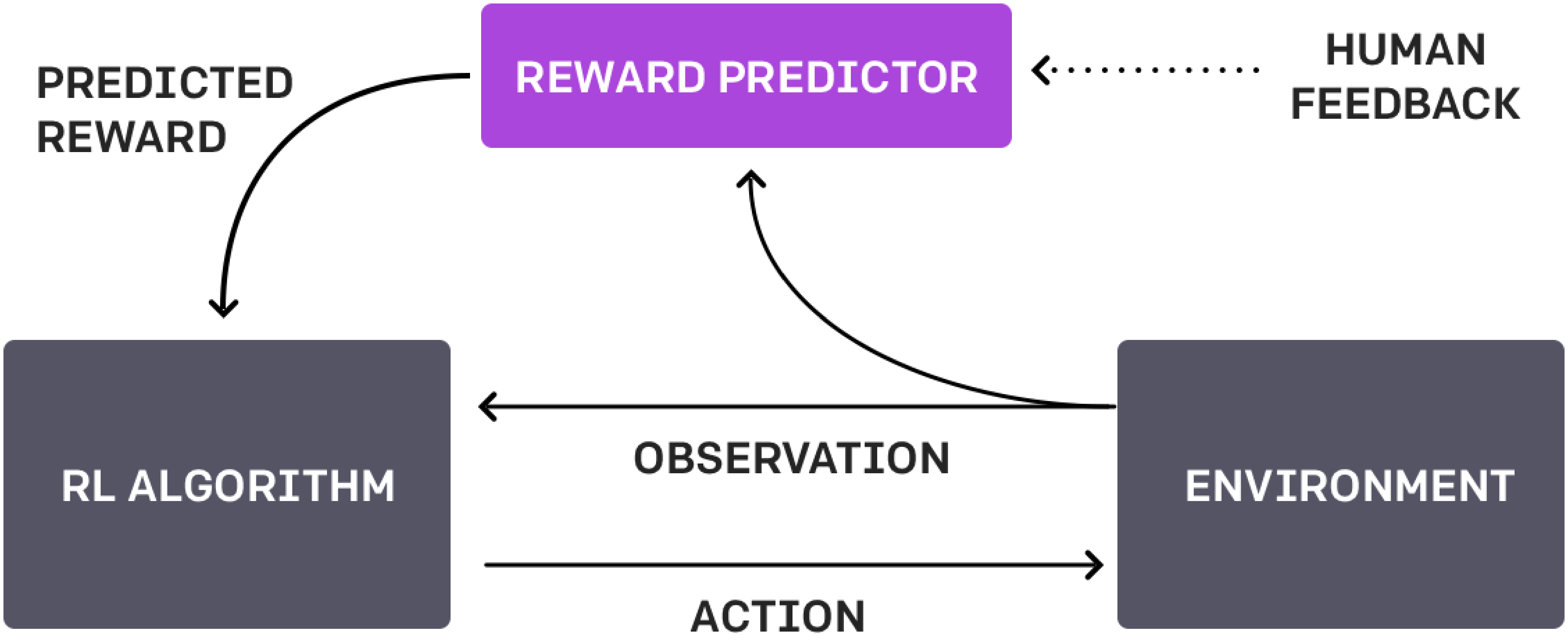
- Markov Decision Process
- Overview of Reinforcement Learning
- Q-Learning
- Example Applications

Reinforcement Learning

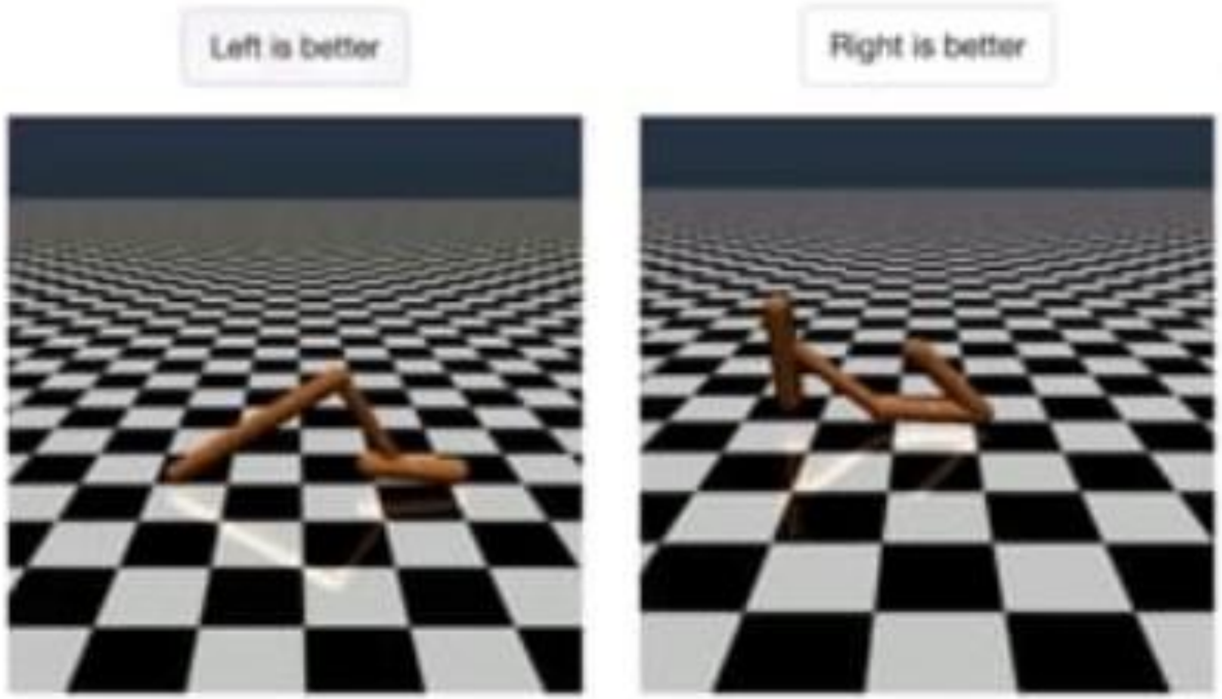
Main idea: an agent learns to take actions by simulating episodes and collecting rewards.

Learning To Park

Reinforcement Learning from Human Feedback (RLHF)

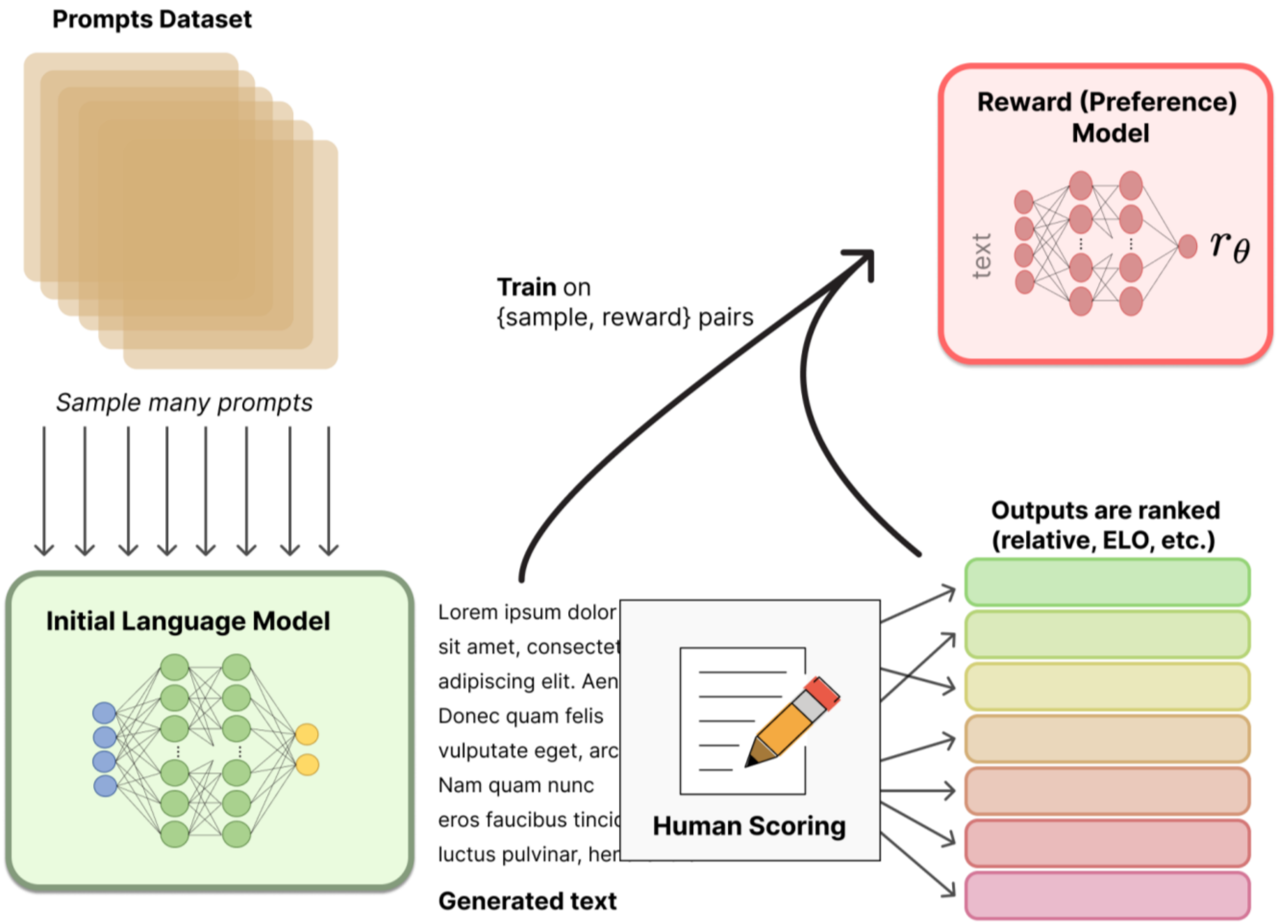


Reinforcement Learning from Human Feedback (RLHF)



RLHF for Large Language Models

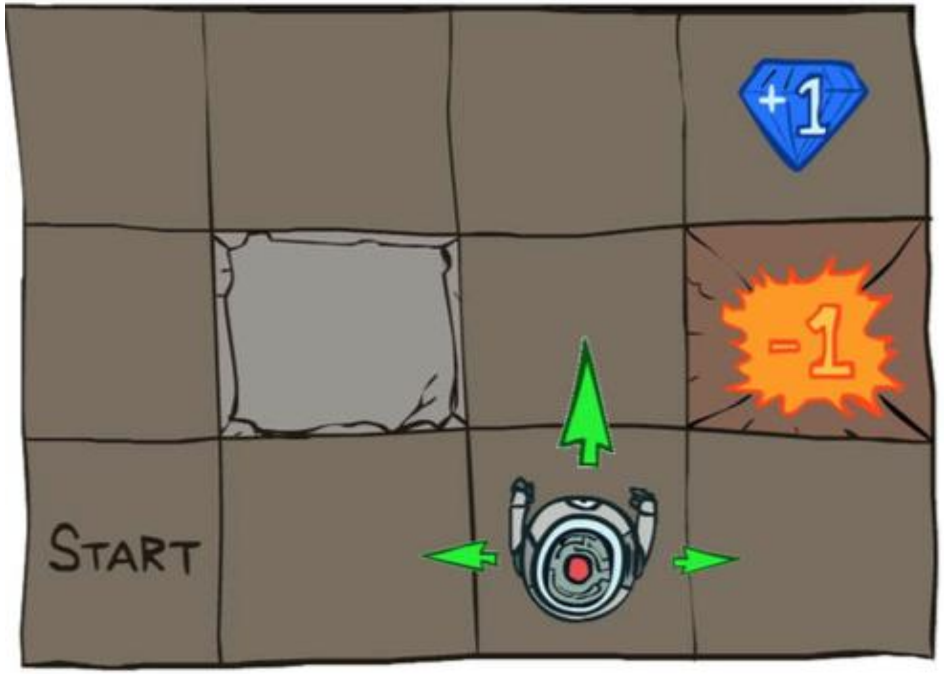
LLMs can learn from human feedback.


Prompts Dataset
Sample many prompts
Initial Language Model
Lorem ipsum dolor sit amet, consectet adipiscing elit. Aen Donec quam felis vulputate eget, arc Nam quam nunc eros faucibus tincid luctus pulvinar, her
Generated text
Human Scoring
Outputs are ranked (relative, ELO, etc.)
Train on {sample, reward} pairs
Reward (Preference) Model
text
r_θ

Markov Decision Process

MDP is defined by:

- Set of **states** $s \in S$
- Set of **actions** $a \in A$
- **Transition function** $T(s, a, s')$
  - Probability that a in s leads to s': $P(s'|s, a)$
- **Reward function** $R(s, a, s')$
- A **start state**
- Possibly one or more **terminal states**
- Possibly a **discount factor** γ



Policies

- The **policy** determines the actions that an agent will take
- Policies can be deterministic or stochastic
- The goal of an agent is to learn an optimal policy π^*
- In Deep RL we define the policy with learned parameters θ

$$a_t = \pi_\theta(s_t)$$

Discounted Rewards

- Solving an MDP: maximize cumulative reward
- Convergence: we use γ to give less weight to samples that are further in the future. This causes the utility to converge
- Discounted utility:

$$U([s_0, a_0, s_1, a_1, s_2, \ldots]) = R(s_0, a_0, s_1) + \gamma R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3) + \cdots$$

Compute Rewards

$$U([s_0, a_0, s_1, a_1, \dots]) = R(s_0, a_0, s_1) + \gamma R(s_1, a_1, s_2) + \cdots$$

- States: $S = \{a, b, c, d, e\}$
- Actions: $A = \{Left, Right, Exit\}$, and $Exit$ is only valid in states a and e
- Discount factor: $\gamma = 0.1$

10				**1**
a	b	c	d	e

Policy:

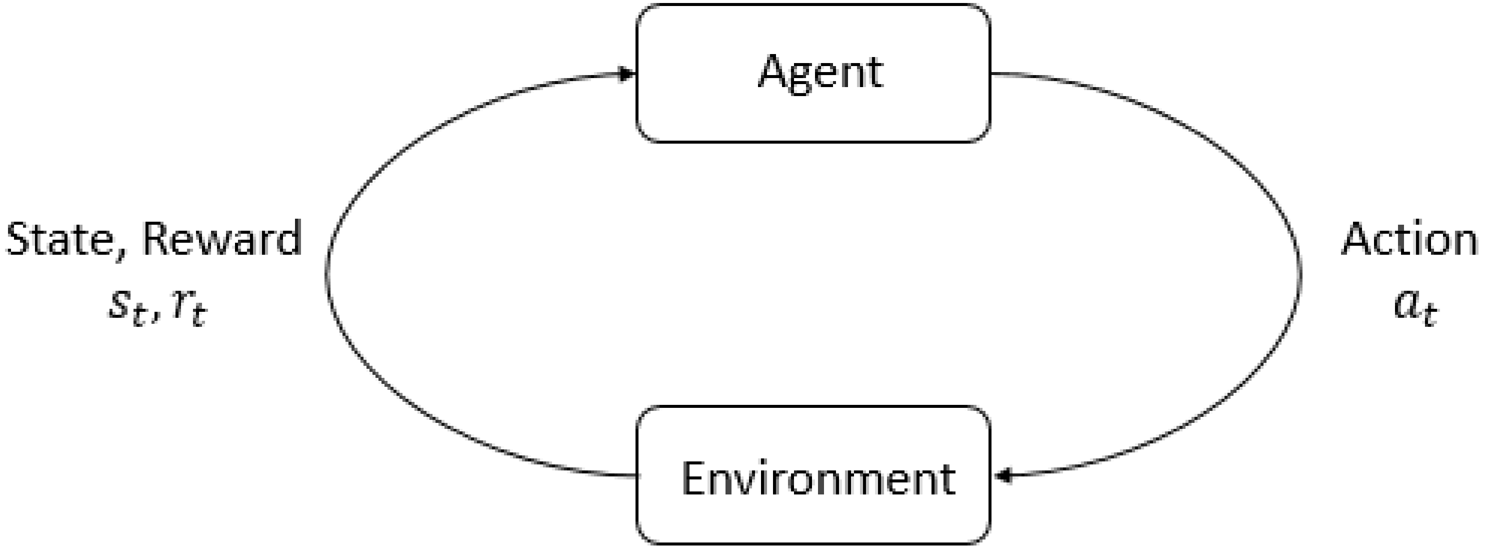
Exit	←	←	→	**Exit**
a	b	c	d	e

Start State	Reward
a	10
b	1
c	0.1
d	0.1
e	1

What if we don’t know *T* and *R*?

- Usually we don’t know the transition probabilities or the reward function
- The agent needs to learn a policy from the unknown probabilities and reward function

Reinforcement Learning



- The **environment** is the world that the **agent** acts in
- The agent receives a **reward** to represent how good or bad the current state is
- RL: the agent learns to maximize cumulative reward

Value Functions

- **On-Policy Value Function**, $V^{\pi}(s)$: expected return starting in state s acting according to policy π
- **Q-state Value Function**, $Q^{\pi}(s, a)$: expected return if we start in state s, take a, then act according to π
- $V^{*}(s)$ and $Q^{*}(s, a)$ are the optimal functions, used with the optimal policy, π^{*}

Bellman Equations

- Main Idea: the value of the starting point is the reward for being there plus the value of the next state

$$V^*(s) = max_a \sum_{s'} T(s,a,s')\boxed{[R(s,a,s') + \gamma V^*(s')]}$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

$$a_t^* = \arg\max_a Q^*(s_t, a)$$

Model-Free vs Model-Based RL

- Model-Based: the agent either has access to or learns a model of the environment
  - Often used for games

- **Model-Free**: the agent neither learns nor has access to a model of the environment
  - Policy Optimization
  - Q-Learning

Policy Optimization

- Optimize π_θ either directly or using gradient ascent on an objective function, which depends on the cumulative reward
- The optimization is generally done **on-policy**
  - The policy can only be updated by data from the policy we want to update

Q-Learning

- Learn an approximator $Q_\theta(s, a)$, using the Bellman equation for an objective function
- Optimization is performed **off-policy**, the Q-values can be updated using data from any time during training
- Recall that once we learn the Q-values, our policy becomes:

$$\pi(s) = \arg\max_a Q_\theta\ (s, a)$$

Q-Learning

- Collect samples to update $Q(s, a)$:

$$\text{sample} = R(s, a, s') + \gamma \max_{a'} Q(s', a')$$

- Incorporate samples into an exponential moving average with learning rate α:

$$Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha \cdot \text{sample}$$

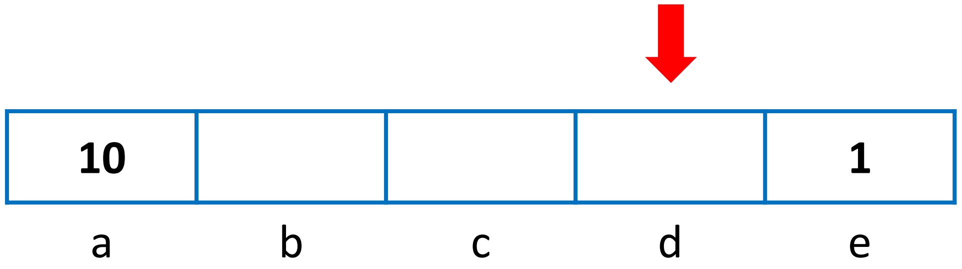
Q-Learning

- Collect samples to update $Q(s,a)$:

$$\text{sample} = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

- Incorporate samples into an exponential moving average:

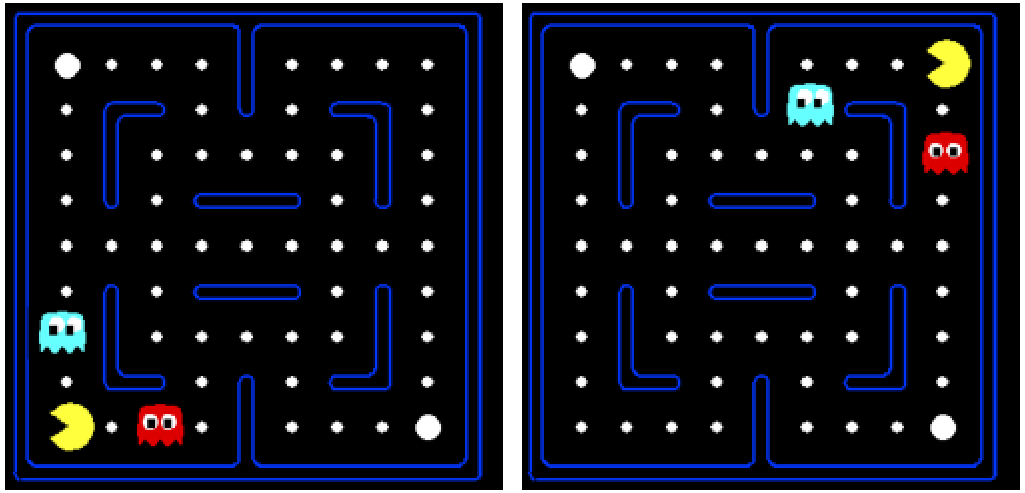
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \cdot \text{sample}$$



$\gamma = 0.1$
$\alpha = 1$

State	Left	Right	Exit
a	0	0	10
b	1	0	-
c	0.1	0	-
d	0.01	0	-
e	0	0	0

Approximate Q-Learning



- A Q-value table would be too big
- Represent states with **feature vectors**
- Feature vector might include:
    - Distance to nearest ghost
    - Distance to nearest food pellet
    - Number of ghosts
    - Is pacman trapped?
- Value of Q-states becomes a linear value function
- We can do the same for state values

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \cdots + w_n f_n(s) = \vec{w} \cdot \vec{f}(s,a)$$

$\vec{f}(s,a)$ is the feature vector for Q-state (s,a)
$\vec{w}$ is a weight vector

Approximate Q-Learning

- Collect samples to update $Q(s,a)$:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

- Define Difference:

$$\text{difference} = sample - Q(s,a)$$

- Update with learning rate α:

$$w_i \leftarrow w_i + \alpha \cdot \text{difference} \cdot f_i(s,a)$$

* Note: Exact Q-learning can be expressed as $Q(s,a) \leftarrow Q(s,a) + \alpha * \text{difference}$

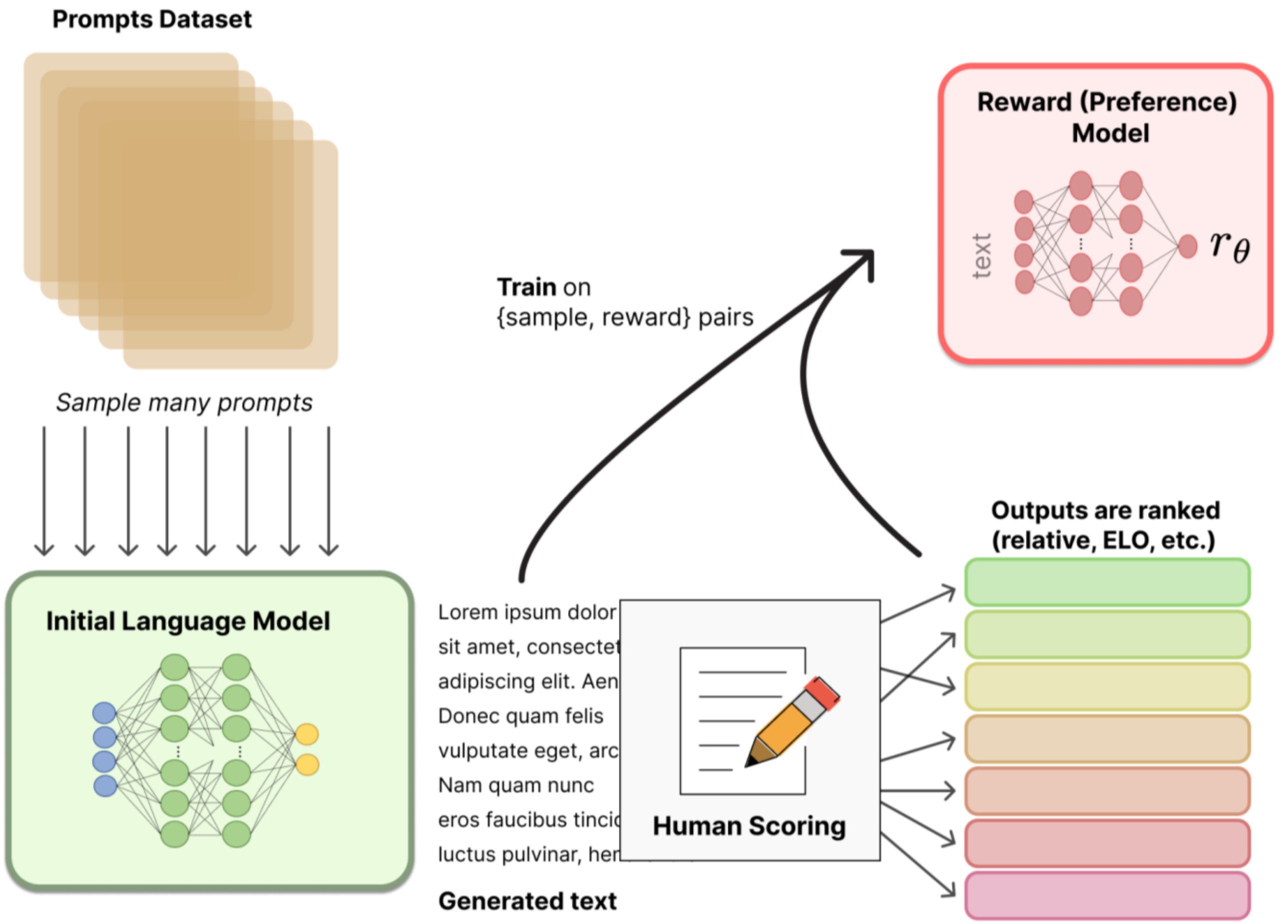
Trade-offs

- Policy optimization optimizes the thing you want
  - More stable and reliable; less efficient

- Q-learning can reuse data better because it uses off-policy optimization
  - More efficient; less stable

RLHF for Large Language Models

LLMs can learn from human feedback.



Prompts Dataset
Sample many prompts
Initial Language Model
Lorem ipsum dolor sit amet, consectet adipiscing elit. Aen Donec quam felis vulputate eget, arc Nam quam nunc eros faucibus tincid luctus pulvinar, her
Generated text
Human Scoring
Outputs are ranked (relative, ELO, etc.)
Train on {sample, reward} pairs
Reward (Preference) Model
text
r_θ

Training

- **Exploration**: explore new states to update approximators
- **Exploitation**: rely on approximators for actions
- Encourage exploration in the objective function or by randomly choosing some actions

Stretch Break

Think about:

How does reinforcement learning differ from other types of machine learning, such as supervised and unsupervised learning?

What are some advantages and disadvantages of reinforcement learning compared to these other types?

After the break: Applications of RL

Applications

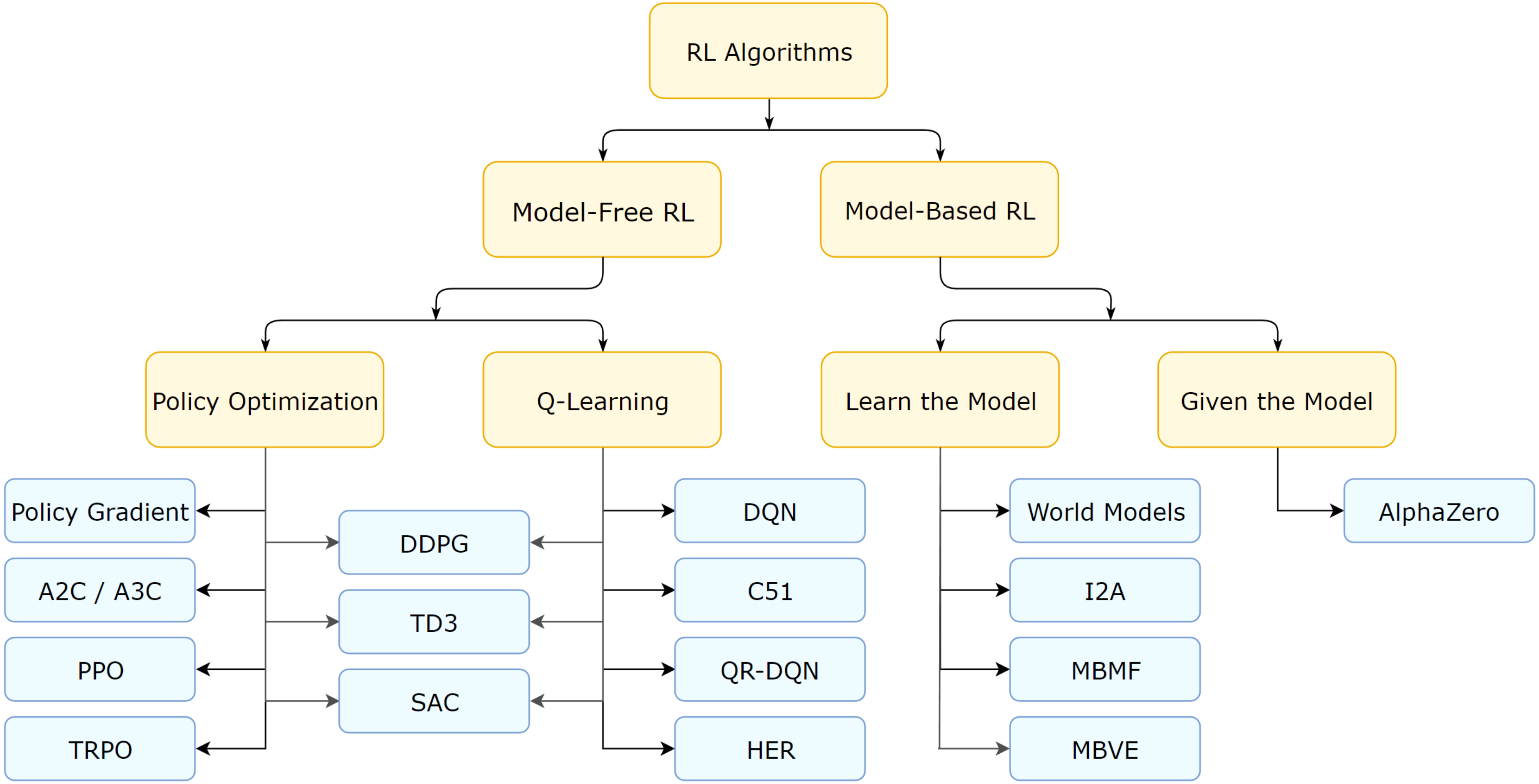
Deep RL: Steps to train an agent

1. Choose or design an algorithm
2. If you are doing deep RL, construct π_θ which is a deep neural network that can be optimized
3. Define a reward function
4. Start training and tune your reward function as necessary

Hide & Seek

Baker et al., 2020 Emergent Tool Use From Multi-Agent Autocurricula

Popular Algorithms



Taxonomy of RL Algorithms [Open AI]

Aircraft Controller

- Soft Actor-Critic
  - Hybrid between policy optimization and Q-learning
  - Actor learns policy while critic learns values
  - Entropy regularization
- Balancing convergence time and sparse rewards
- Creating a realistic environment
- Developing cooperative controllers

Simulated Cockpit [Viper Wing]

Additional Challenges

- **Multi-Agent Reinforcement Learning**: multiple agents in an environment learn policies and can either compete or cooperate
- **Sim-to-Real Gap**: policies in simulation don't usually transfer well to the real world
- **Tuning LMs:** using PPO variants for LLM tuning
- **Imitation Learning:** learning from examples

Source

Things to remember

- Markov Decision Process:
  - States, actions, transition probabilities, reward
  - Discounted utility
  - Policy determines actions
- Reinforcement Learning
  - Agents live in and get rewards from the environment
  - On-policy Value Function and Q-State Value Function
  - Bellman Equations
  - Model-Based vs. Model-free RL
    - Policy Optimization
    - Q-Learning
    - Approximate Q-Learning

References

- Open AI Spinning Up Introduction to RL
- UC Berkeley CS188
- UIUC CS440