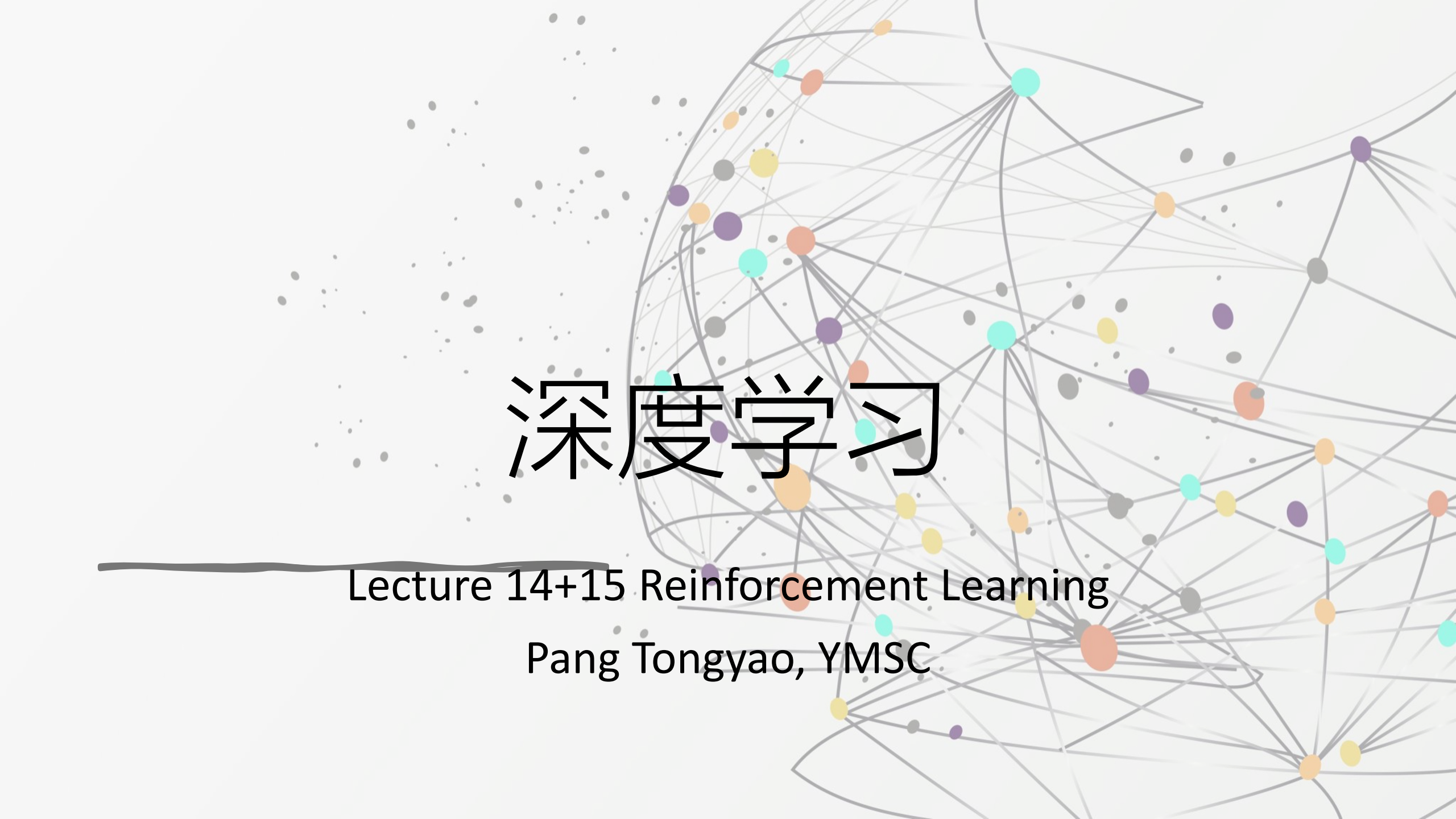


深度学习

Lecture 14+15 Reinforcement Learning

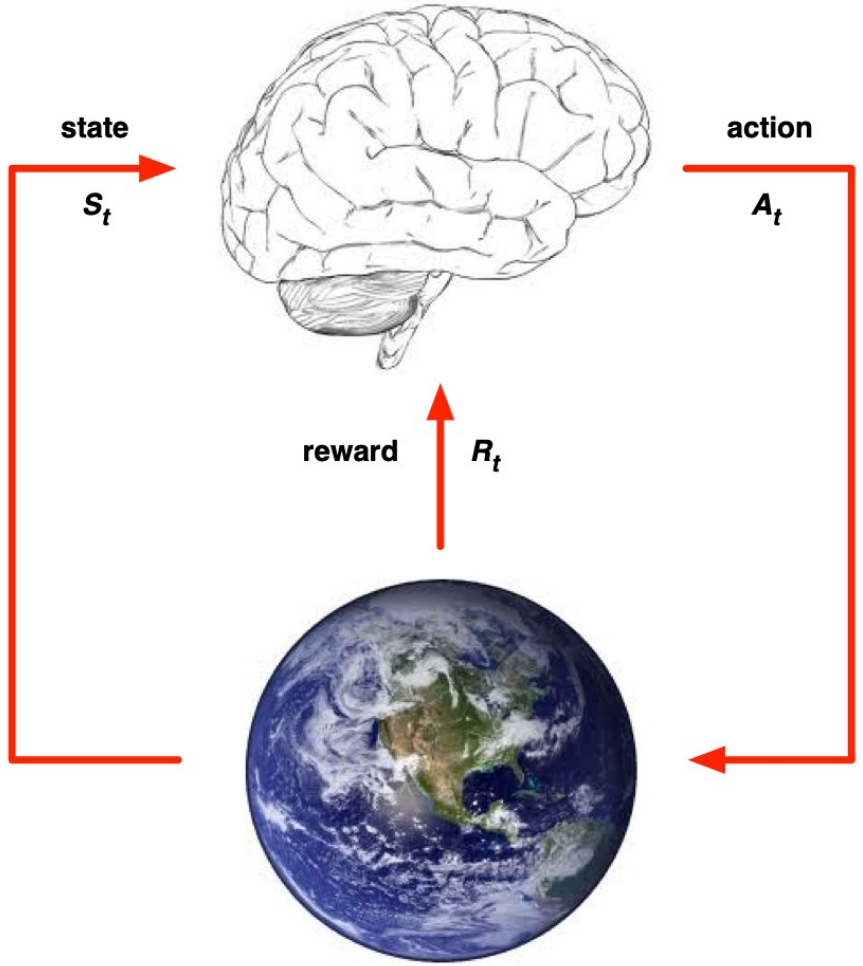
Pang Tongyao, YMSC

Reinforcement learning

- Reinforcement learning (RL) lies somewhere in between supervised Learning and unsupervised Learning.
- There is no supervisor, only a reward signal
  - Feedback is delayed, not instantaneous
  - Time really matters (sequential, non i.i.d data)
  - Agent's actions affect the subsequent data it receives
- RL is a sequential decision problem which is usually of high dimension (large state/action spaces) and highly nonconvex.

Reinforcement Learning

- Environment:
  - receives action A_t
  - transfers to a new state S_{t+1}
  - provides reward R_t
- Agent:
  - receives reward R_t
  - observes state S_{t+1}
  - selects and executes action A_{t+1}



Reinforcement Learning

Major component

- Policy: agent's behaviour function; a map from state to action

Deterministic policy: $\pi(s) = \mathrm{a}$; Stochastic policy: $\pi(a|s) = \mathbb{P}(a|s)$

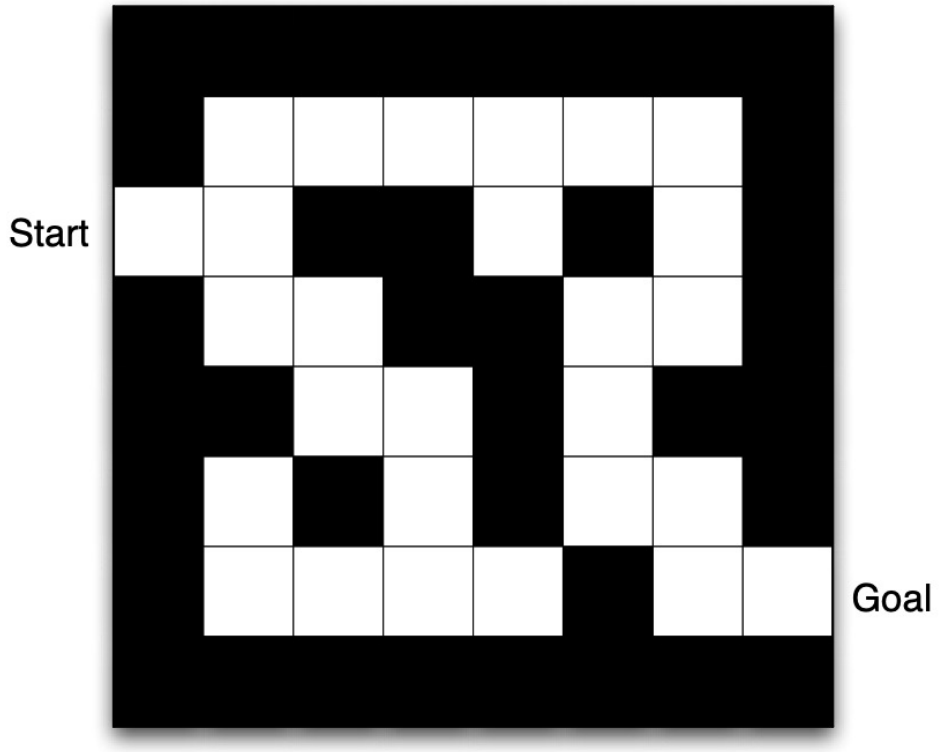
- Value function: how good is each state and/or action; a prediction of future reward:

$$V_\pi(s_t) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s]$$

- Model: predicts what the environment will do next

transition: $\mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$; Reward: $\mathbb{E}(R_{t+1} | S_t = s, A_t = a)$

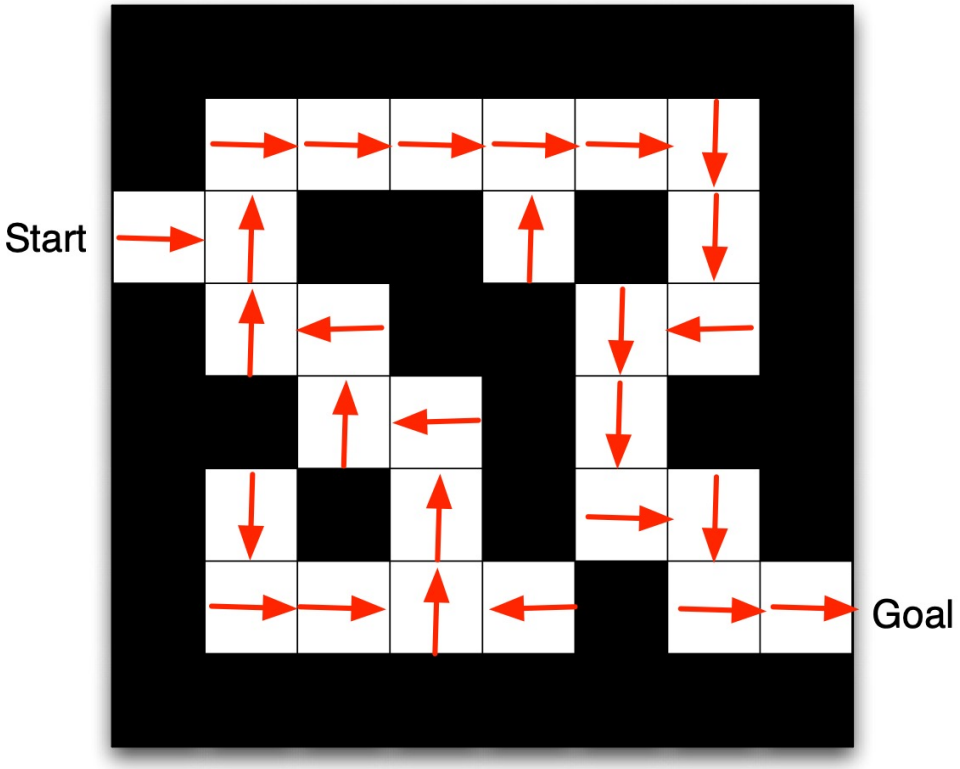
Maze Example

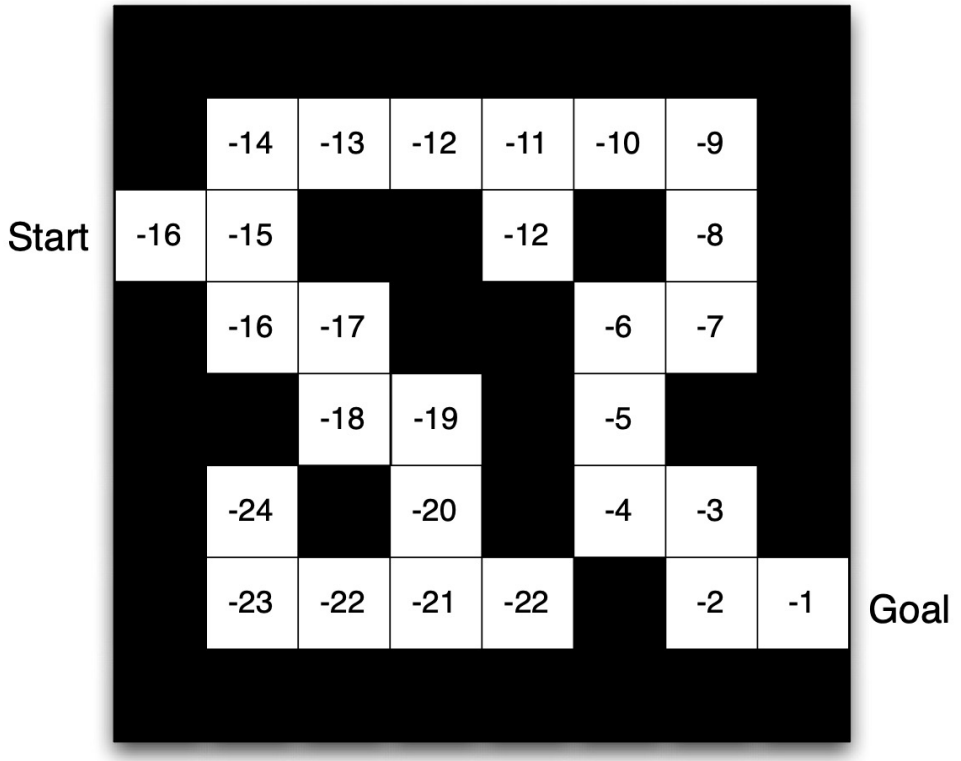


- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

Source: https://www.davidsilver.uk/wp-content/uploads/2020/03/intro_RL.pdf

Maze Example

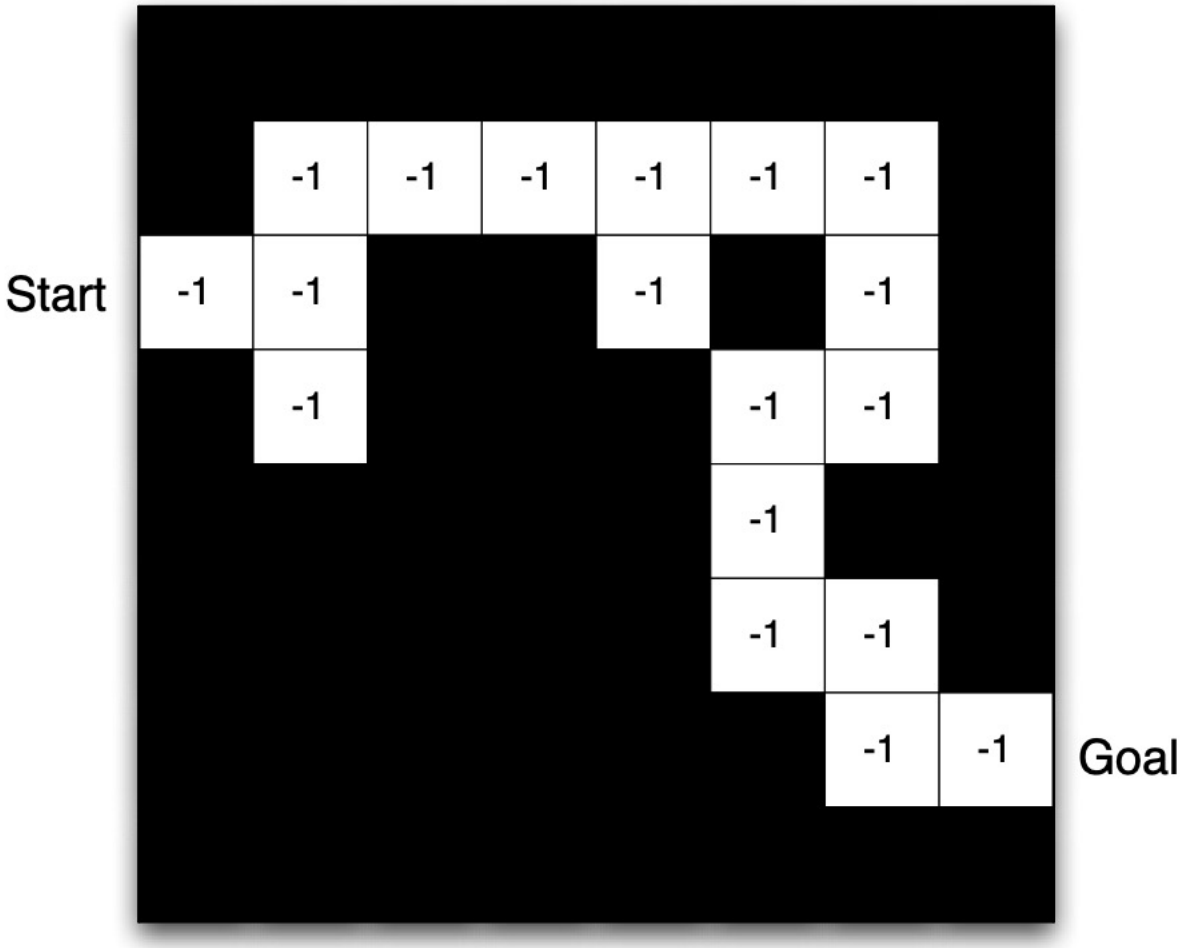




- Arrows represent policy $\pi(s)$ for each state s
- Numbers represent value $v_\pi(s)$ of each state s

Source: https://www.davidsilver.uk/wp-content/uploads/2020/03/intro_RL.pdf

Maze Example



- Grid layout represents transition model
- Numbers represent immediate reward at each state

Source: https://www.davidsilver.uk/wp-content/uploads/2020/03/intro_RL.pdf

Categorizing RL Algorithms

- Model Free
  - Policy and/or Value Function
  - No Model
- Model Based
  - Policy and/or Value Function
  - Model

- Value Based
  - No Policy (Implicit)
  - Value Function
- Policy Based
  - Policy
  - No Value Function
- Actor Critic
  - Policy
  - Value Function

Why So Many RL Algorithms

- Different tradeoffs:
  - Sample efficiency
  - Stability & ease of use
  - exploitation and exploration
- Different Assumptions:
  - Stochastic or deterministic policy
  - Continuous or discrete states and/or actions
- Different Settings: which one is easier, policy or model?

Model-Based Algorithms

Markov Decision Process

- Markov Property

$$\mathbb{P}(S_{t+1} = s' | S_t = s, S_{t-1} = s_{t-1}, \dots, S_0 = s_0) = \mathbb{P}(S_{t+1} = s' | S_t = s)$$

- Time Homogeneous Markov Chain
  - The transition probability is independent of time, i.e.

$$\mathbb{P}(S_{t+1} = s' | S_t = s) = P_{ss'} = p(s'|s)$$

  - The matrix $P = \{P_{ss'}\}$ is called the **transition (probability) matrix**
- Markov decision processes (MDP) is a generalization of Markov processes, with actions A_t and rewards R_t.

Markov Decision Process

- A Markov decision process (MDP) is the evolution of S_t, R_t according to

$$p(s', r|s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a]$$

(Model)

- Policy: Agent controls the system through the policy

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$$

Deterministic policies: $\pi(a|s) = \mathbb{I}_{a=a_0(s)}$

- A MDP is finite if the state space and the action space are finite.

Choosing a Policy

- The goal of choosing a policy is to maximize the long-term rewards:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

  Here, γ is the discount rate.

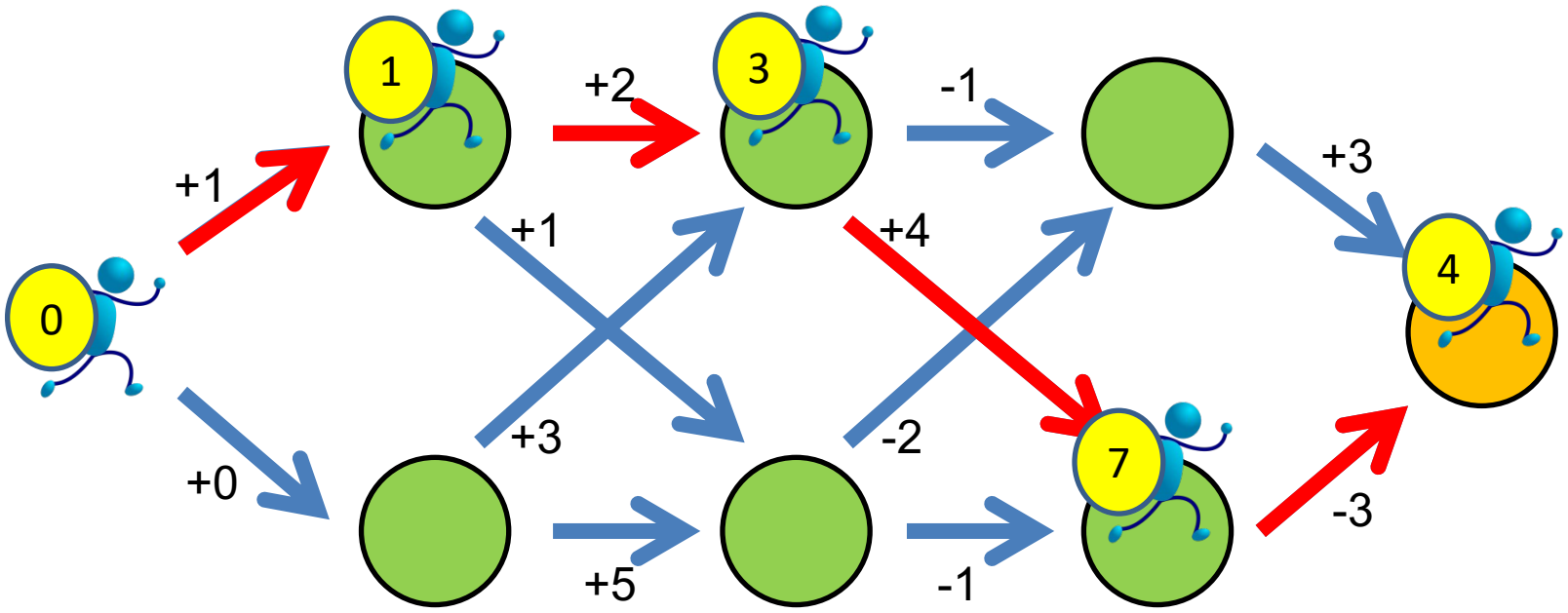
- The goal of RL is the maximize, by choosing a good policy π, the expected return (the time-homogeneous case):

$$\mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \, | S_t = s \right]$$

Curse of Dimensionality

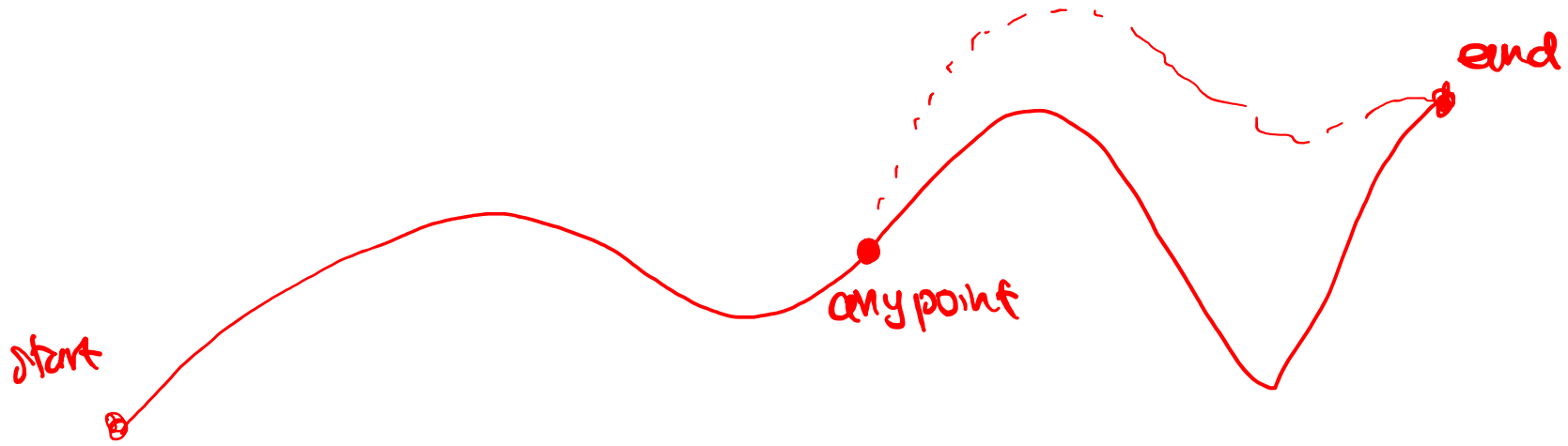
- How long does it take to check all possibilities?

$$N^T! \quad N: number\ of\ states, T: number\ of\ steps$$



Dynamic Programming

- On an optimal path (following the optimal policy), if we start at any state in that path, the rest of path must again be optimal.



Bellman Equation

- As motivated earlier, we define the state value function

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi\left[\sum_k \gamma^k R_{t+k+1} | S_t = s\right]$$

and the action value (Q-value) function

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi[\textstyle\sum_k \gamma^k R_{t+k+1} | S_t = s, A_t = a]$$

- Our goal: derive a recursion for $v_\pi(s)$ and $q_\pi(s,a)$
- These are known as Bellman equations

Bellman Equation for Value Function

- Using the definitions, we can show the following relationships:

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a)$$

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r(s,a,s') + \gamma v_\pi(s')]$$

- Combining, we get

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r(s,a,s') + \gamma v_\pi(s')]$$

- This is known as the Bellman equation for the value function.

Bellman Equation for Value Function

- For finite MDPs, the Bellman equation can be written as

$$v_\pi = \gamma P(\pi) v_\pi + b(\pi),$$

$$b(\pi) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) r(s, a, s') \,, [P(\pi)]_{ss'} = \sum_a \pi(a|s) p(s', r|s, a)$$

- This is a linear equation, and we can show that there exists a unique solution for v_π.
- In fact, it is just

$$v_\pi = \big(I - \gamma P(\pi)\big)^{-1} b(\pi)$$

whose existence and uniqueness follow from the invertibility of $I - \gamma P(\pi)$, which in turn follows from $\|P\|_\infty = 1$.

Bellman Equation for Action-Value Function

- Using similar methods, one can show that the action value function satisfies a similar recursion

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a')\right]$$

Comparing Policies

- We can compare policies via their values
  - Given π, π', we say $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)$ for all s
  - This is a **partial order**
- Examples
  - $\mathcal{S} = \{s_1, s_2, s_3\}$, $v_\pi = (2, 4, 6)$, $v_{\pi'} = (1, 3, 6)$. Then $v_\pi \geq v_{\pi'}$
  - $\mathcal{S} = \{s_1, s_2, s_3\}$, $v_\pi = (2, 4, 6)$, $v_{\pi'} = (3, 3, 6)$. Then neither $v_\pi \geq v_{\pi'}$ nor $v_{\pi'} \geq v_\pi$ holds

Optimal Policy

- We define an optimal policy π_* to be any policy satisfying

$$\pi_* \geq \pi \ \ \forall \text{ other policies } \pi$$

- In other words, $\mathrm{V}_{\pi^*}(s) \geq V_\pi(s)$ for all s and all π
  - Does such a π_* exist?
  - Is it unique?

Policy Improvement

- For any two policies π, π', if

$$\sum_a \pi'(a|s) q_{\pi'}(s,a) \geq \sum_a \pi(a|s) q_\pi(s,a) \;\; \forall\, s$$

  Then we must have

$$v_{\pi'}(s) \geq v_\pi(s) \;\; \forall\, s$$

- In addition, if the first inequality is strict for some s, then the second equality is strict for at least one s.

Policy Improvement

- **[Policy Improvement Theorem]** For any two (deterministic) policies π, π', if

$$q_\pi\big(s, \pi'(s)\big) \geq v_\pi(s) \;\; \forall\, s$$

  Then we must have

$$v_{\pi'}(s) \geq v_\pi(s) \;\; \forall\, s$$

- In addition, if the first inequality is strict for some s, then the second equality is strict for at least one s.
- Hence:
  - Given any π, we can derive $\pi'(s) = \arg\max_a q_\pi(s, a)$
  - $\pi' \geq \pi$ with equality if and only if π is optimal

Proof of Policy Improvement Theorem [Finite case]:

$$q_\pi\big(s,\pi'(s)\big) = \sum_{s',r} p(s',r|s,\pi'(s))\left[r + \gamma \sum_{a\prime} \pi(a'|s')q_\pi(s',a')\right]$$

$$\triangleq [b(\pi') + \gamma P(\pi')V^\pi](\mathrm{s}) \geq V^\pi(s)$$

which gives

$$\big(I - \gamma P(\pi')\big)V^\pi \leq b(\pi')$$

By Bellman's Equation:

$$\big(I - \gamma P(\pi')\big)V^{\pi'} = b(\pi')$$

Since $\mathrm{I} - \gamma P(\pi')$ is positive definite, we have

$$V^\pi \leq V^{\pi'}$$

If no improvement is available, then for any π',

$$[R(\pi') + \gamma P(\pi')V^\pi](\mathrm{s}) \leq V^\pi(s)$$

Similarly, we can get that

$$V^\pi \geq V^{\pi'}, \forall\, \pi'.$$

Thus, π is optimal.

Bellman Optimality Equation

- Corresponding to an optimal policy

$$\pi_*(s) \in \arg\max_a q_*(s,a)$$

we obtain the following recursion

$$v_*(s) = \max_{a \in \mathcal{A}} \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

$$q_*(s,a) = \sum_{s',r} p\,(s',r|s,a)\left[r + \gamma \max_{a' \in \mathcal{A}} q_*(s',a')\right]$$

- These are known as the Bellman optimality equations

Value Iteration

- Write the Bellman's optimality equation as

$$v_* = F(v_*)$$

where

$$F: \mathbb{R}^n \to \mathbb{R}^n$$

is called the Bellman optimality operator

$$F(v)(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v(s')]$$

- Recall that in matrix notations, we can write

$$F(v)(s) = \max_\pi [\gamma P(\pi)v + b(\pi)]$$

Fixed Point Iteration

- Since the optimal value function satisfies

$$v_* = F(v_*)$$

  i.e. it is a fixed point of F, we can solve it via fixed point iteration:

$$v_{k+1} = F(v_k) \quad k = 0,1,2, \ldots$$

- If this converges to v_∞ as $k \to \infty$, then
  - v_∞ is a fixed point of F, i.e. $v_\infty = F(v_\infty)$
  - If the optimal value function is unique, $v_\infty = v_*$
- To retrieve the optimal policy

$$\pi_*(a|s) = \begin{cases} 1 & \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \\ 0 & \text{otherwise.} \end{cases}$$

Contraction Mapping Theorem

- Let V be a complete normed space and $F: V \to V$ be a contraction, i.e.

$$\|F(v) - F(u)\| \leq \gamma \|v - u\|, \qquad \forall v, u \in V, \qquad 0 \leq \gamma < 1$$

Then, there exists a unique $v_* \in V$ such that $v_* = F(v_*)$. Moreover,

$$v_* = \lim_{k \to \infty} v_k$$

for any sequence $\{v_k : k = 0,1, \dots\}$ such that $v_{k+1} = F(v_k)$ and $v_0 \in V$ is arbitrary.

- The Bellman optimality operator is contractive.

Value Iteration Algorithm

Algorithm 10: Value Iteration Algorithm

Model Parameters: MDP transition probability $p(s', r|s, a)$, discount rate $\gamma < 1$

Input: Policy π, Stopping criterion

Initialize: $v \in \mathbb{R}^n$

while *stopping criterion not reached* **do**

  $v \leftarrow F(v) = \max_\pi \{P(\pi)v + b(\pi)\}$

end

return v

By the contraction mapping theorem, v_k converges to the unique optimal value function.

Policy Iteration Algorithm

- There are two ingredients in Policy Iteration (PI)

  Policy Evaluation:

$$v_{\pi_k} = \gamma P(\pi_k) v_{\pi_k} + b(\pi_k) \text{ (Bellman Equation)}$$

  Policy Improvement:

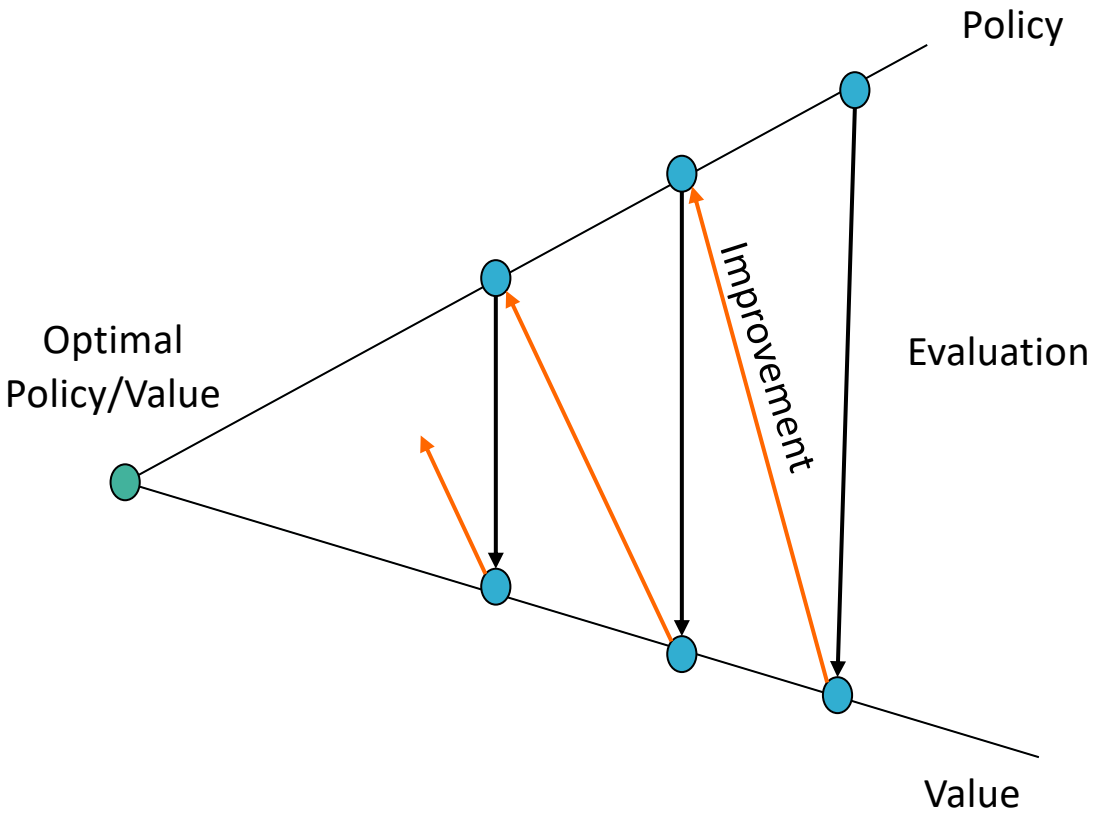
$$\pi_{k+1}(a|s) = \begin{cases} 1 & \arg\max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_{\pi_k}(s')\right] \\ 0 & \text{otherwise.} \end{cases}$$

- Policy Improvement Theorem guarantees that the value function gets strict improvement.

Policy Iteration Algorithms

- Policy Iteration Algorithms
  1. Initialize π_0
  2. Repeat Until Convergence:
     - Policy Evaluation
     - Policy Improvement

- Because the policy improves strictly unless it's already optimal, **the process must terminate** after a finite number of steps when there are only finitely many states and actions.



Value Iteration vs. Policy Iteration

Aspect	Value Iteration	Policy Iteration
Core Idea	Iteratively improve value estimates	Iteratively evaluate and improve a policy
Main Steps	1. Bellman Update2. Greedy Policy Selection	1. Policy Evaluation2. Policy Improvement
Value Updates	Bellman optimality update: 1-step lookahead per iteration	Full policy evaluation: solve or iterate until convergence
Policy Evaluation Target	No specific policy; approximates V^* directly	V^{π_k} — the value of current policy
Does Value Satisfy Bellman Equation for Some Policy?	❌ Not until convergence; intermediate $V_k \neq V^\pi$	✅ Yes, always corresponds to V^{π_k}
Per-Iteration Cost	Low (one-step updates per state)	High (policy evaluation is expensive)
Number of Iterations	More (each one is cheap)	Fewer (each one is more expensive)
Convergence Type	Asymptotic (converges to V^*)	Finite (in exact tabular case)
Suitability	- Large MDPs- Online or approximate settings- Where fast, lightweight updates are needed	- Small/medium MDPs with exact evaluation- Useful when policy evaluation can be done efficiently

Model-Free Algorithms

Model-free

Why Model-free

- Computing/estimating $p(s', r|s, a)$ is prohibitively expensive
- Reward and state evolutions are from some black-box system

In the model-free setting, we are given an environment simulator

$$(s', r) = \text{EnvironmentSimulator}(s, a) \sim p(\cdot, \cdot\,|s, a)$$

Monte-Carlo Policy Evaluation

- Recall that given π, in the model-based case we evaluate it via

$$v_\pi = \big(I - \gamma P(\pi)\big)^{-1} b(\pi)$$

  where the computation of P, b depends on $p(s', r|s, a)$.

- In the model free case, we turn to the original definition of value function

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_k \gamma^k R_{k+1} \,|S_0 = s \right]$$

Monte-Carlo Policy Evaluation

- We approximate the expectation

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_k \gamma^k R_{k+1} \,|S_0 = s \right]$$

by Monte-Carlo!

- Using the black-box simulator, draw **episodes** of states and rewards

$$S_t^{(n)}, R_t^{(n)}, \qquad t \geq 0, n = 1,2, \dots, N$$

where $S_0^{(n)} = s$.

- Estimate $v_\pi(s)$ via Monte-Carlo

$$v_\pi(s) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_k \gamma^k R_{k+1}^{(n)}$$

Monte-Carlo Policy Improvement

- Estimate action value function via Monte-Carlo

$$q_\pi(s,a) \approx \frac{1}{N}\sum_{n=1}^{N}\left[\sum_k \gamma^k R_{k+1}^{(n)}\right], \qquad S_0^{(n)} = s, A_0^{(n)} = a$$

Improve policy via

$$\pi'(s) = \arg\max_a q_\pi(s,a)$$

Algorithm 12: Model-free Policy Iteration Algorithm

Input: State-reward simulator, Sample size N, Stopping criterion

Initialize: Initial policy π, Initial action value function $q(s, a)$

while *stopping criterion not reached* **do**

  Policy evaluation:

  for s *in* $\mathcal{S}$ **do**

    for a *in* $\mathcal{A}(s)$ **do**

      Sample episodes according to π:

      $S_0^{(n)} = s, S_1^{(n)}, S_2^{(n)}, \ldots$ and $R_1^{(n)}, R_2^{(n)}, \ldots$ for $n = 1, 2, \ldots, N$, Set action-value function:

      $q(s, a) \leftarrow \frac{1}{N} \sum_{n=1}^{N} \left[\sum_{k=0}^{\infty} \gamma^k R_{k+1}^{(n)} | S_0^{(n)} = s, A_0^{(n)} = a \right]$.

    end

  end

  Policy improvement:

  for s *in* $\mathcal{S}$ **do**

    $\pi(s) \leftarrow \arg\max_a q(a, s)$

  end

end

return $\pi \approx \pi_*, q \approx q_*$

This Primitive Monte Carlo (MC) Learning is inefficient:

- **Delayed Updates**: Must wait for episode to finish before updating.
- **High Variance**: Variance increases with episode length→ unstable learning.
- **Sample Inefficiency**: Each visit to a state is used once per episode for update.
- **Poor Credit Assignment:** All actions in the episode share the same return, even if one was crucial (or terrible).

Temporal Differencing Methods

- Recall the $Bellman\ Equation$:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

- Fixed point iteration

$$v(s) \leftarrow (1 - \alpha)v(s) + \alpha \left(\mathbb{E}_{a \sim \pi(a \mid s)} \mathbb{E}_{s', r \sim p(s', r|s, a)} [r + \gamma v(s')] \right)$$

- Temporal Differencing (TD) methods approximate the expectation using single sample! (like stochastic gradient descent)

Temporal Differencing Methods

- Given policy π, suppose we are at state s
  - Sample action $a \sim \pi(\cdot\,|s)$
  - Sample s', r using environment simulator
  - Update

$$v(s) \leftarrow (1-\alpha)v(s) + \alpha\big(r + \gamma v(s')\big)$$

- Suppose such iteration converges, then take expectation over environment, $v \rightarrow v_\pi$
- TD evaluation (vs. MC evaluation):
  - more efficient:exploits MDP structure
  - low variance: one step sampling; but biased
  - bootstrapping: updating value estimates after every time step

Why is it called temporal differencing?

- We can rewrite

$$v(s) \leftarrow v(s) + \alpha[r + \gamma v(s') - v(s)]$$

- $r + \gamma v(s')$ is the updated value function prediction given the new sample s', r drawn from the environment
- Compares the temporal difference

$$r + \gamma v(s') \text{ v.s. } v(s)$$

  - If v "over-values" s, then increase it
  - If v "under-values" s, then decrease it

TD(0) Algorithm for Policy Evaluation

Algorithm 13: TD(0) Algorithm for Policy Evaluation

Input: State-reward simulator, initial state sampler, Stopping criterion, Policy π
Initialize: Initial value v_0, Episode length T
$v \leftarrow v_0$;
while *stopping criterion not reached* **do**
    $s \leftarrow$ InitialStateSampler();
    for $t = 0$ *to* $T - 1$ **do**
        $a \rightarrow a \sim \pi(\cdot|s)$;
        $s', r \leftarrow$ StateRewardSimulator(s, a);
        $v(s) \leftarrow (1 - \alpha)v(s) + \alpha[r + \gamma v(s')]$;
    end
end
return v

TD Policy Evaluation of Action Values

- Bellman Equation for Action Value Functions

$$q_\pi(s,a) = \mathbb{E}_{s',r\sim p(s',r|s,a)}\mathbb{E}_{a\prime\sim\pi(a\prime\,|\,s\prime)}[r + \gamma q_\pi(s',a')]$$

- Fixed point iteration

$$q_\pi(s,a) \leftarrow (1-\alpha)q_\pi(s,a) + \alpha\left(\mathbb{E}_{s',r\sim p(s',r|s,a)}\mathbb{E}_{a\prime\sim\pi(a\prime\,|\,s\prime)}[r + \gamma q_\pi(s',a')]\right)$$

- SARSA: Given policy π, current state-action (s,a)
  - Sample a tuple (s',a',r) from the same policy π and the environment simulator
  - Update:

$$q_\pi(s,a) \leftarrow (1-\alpha)q_\pi(s,a) + \alpha\big(r + \gamma q_\pi(s',a')\big)$$

  - SARSA stands for: $state \rightarrow action \rightarrow reward \rightarrow state \rightarrow action$
  - It is an **on-policy** Temporal Difference (TD) method for learning action-value functions

SARSA

Algorithm 1 SARSA with ε-greedy policy

Require: Learning rate $\alpha \in (0, 1)$, discount factor $\gamma \in [0, 1)$, exploration rate ε

Initialize $q(s, a)$ arbitrarily for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
for each episode **do**
  Initialize state s
  Choose action a from s using ε-greedy policy:

$$a \leftarrow \begin{cases} \arg\max_{a'} q(s, a'), & \text{with probability } 1 - \varepsilon \\ \text{random action from } \mathcal{A}(s), & \text{with probability } \varepsilon \end{cases}$$

  repeat
    Take action a, observe reward r and next state s'
    Choose action a' from s' using ε-greedy policy:

$$a' \leftarrow \begin{cases} \arg\max_{a''} q(s', a''), & \text{with probability } 1 - \varepsilon \\ \text{random action from } \mathcal{A}(s'), & \text{with probability } \varepsilon \end{cases}$$

    Update: $q(s, a) \leftarrow q(s, a) + \alpha \left[r + \gamma q(s', a') - q(s, a) \right]$
    $s \leftarrow s'$
    $a \leftarrow a'$
  until s is terminal
end for

The **ε-greedy policy** is a simple but effective method for balancing **exploration** and **exploitation**

Q-Learning Algorithm

- **Q-learning** is a foundational **off-policy algorithm** that learns the **optimal action-value function** $q_*(s, a)$, regardless of the policy being followed.

$$q_*(s, a) = \mathbb{E}_{s', r \sim p(\cdot, \cdot | s, a)} \left[r + \gamma \max_{a'} q_*(s', a') \right]$$

(Bellman Optimality Equation)

- Similarly, we replace the expectation by samples for updates

$$q(s, a) \leftarrow (1 - \alpha) q(s, a) + \alpha \left[r + \gamma \max_{a'} q(s', a') \right]$$

- Q-learning is a model free implementation of Q-value iteration.

Q-Learning Algorithm

Algorithm 14: Q-learning Algorithm

Input: State-reward simulator, initial state sampler, Stopping criterion, Policy π

Initialize: Initial value q_0, Episode length T

$q \leftarrow q_0$;

while *stopping criterion not reached* **do**

  $s \leftarrow$ InitialStateSampler();

  for $t = 0$ *to* $T - 1$ **do**

    $a \rightarrow a \sim \pi(\cdot|s)$;

    $s', r \leftarrow$ StateRewardSimulator(s, a);

    $q(s, a) \leftarrow (1 - \alpha)q(s, a) + \alpha[r + \gamma \max_{a'} q(s', a')]$;

  end

end

return q

SARSA vs. Q-Learning

- SARSA uses the **action it actually takes at the next state**, sampled from the same ε-greedy policy.
- **On-policy**: the update is based on what the policy **actually did.**

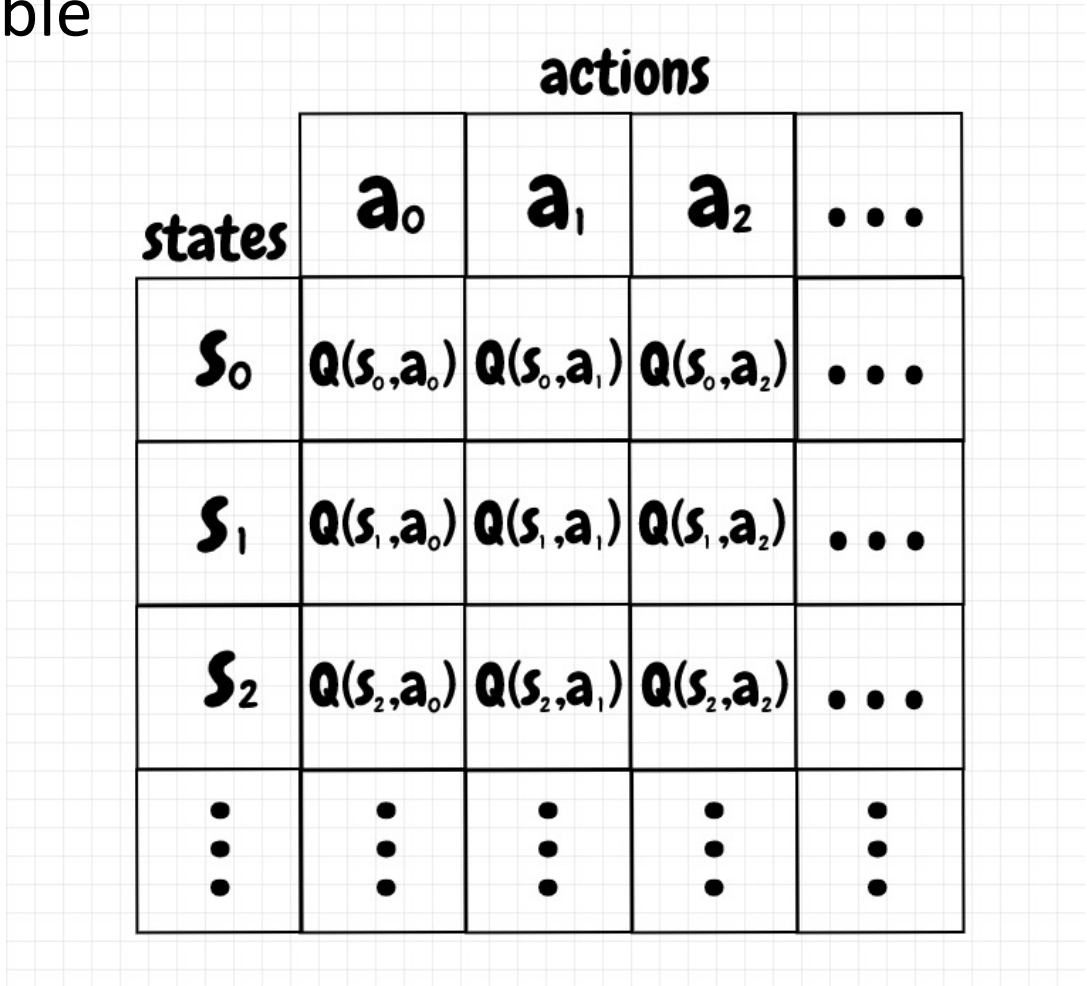
- Q-learning uses the **greedy action** for the next state, regardless of what was actually done.
- **Off-policy**: the update is based on a **different policy** than the one used to act.

- **On-policy** methods must **use the policy they're improving** to collect experience**. (More stable)**
- **Off-policy** methods can **use different data** (e.g., old policies or exploratory policies) and still learn about the optimal policy. **(More flexible)**

Value Function Approximation

Q-function Representation

- Q-table



Problems with large MDPs

- Enormously many states/actions or continuous state/action space
- Inefficient to learn every state/action value individually

Q-function Representation

- State $s = (x_1, x_2, \ldots, x_n)^T$
- Linear Approximation:

$$Q(s, a) = \sum_i w_{ai} x_i$$

- Non-linear (e.g. neural networks):

$$Q(s, a) = g(x; w)$$

Gradient Q-learning

- Training goal: learn $Q_w(s,a)$ to approximate the optimal value $Q_*(s,a)$

$Q_*(s,a) = \mathbb{E}_{s',r\sim p(\cdot,\cdot|s,a)}\left[r + \gamma \max_{a'} Q_*(s',a')\right]$ (Bellman Optimality Equation)

- Model free TD evaluation: use $r + \gamma \max_{a'} Q_{\bar{w}}(s',a')$ as the training target. ($\bar{w}$ is the old parameter)
- Loss function: $\delta = Q_w(s,a) - \left(r + \gamma \max_{a'} Q_{\bar{w}}(s',a')\right)$

$$L = \frac{1}{|B|}\sum_{s,a,s',r\in B} l(\delta), \qquad l(\delta): \ell_2\ loss, Huber\ loss\ \ldots$$

- Update parameters via gradient descent: e.g. for $\ell_2\ loss$:

$$w \leftarrow w - \alpha\delta\frac{\partial Q_w}{\partial w}$$

Gradient Q-learning

Experience Replay:

- Store agent's experiences (s, a, s', r) in a replay buffer.
- During training, randomly sample batches of experiences from the replay buffer to update the network parameters.

Advantages:

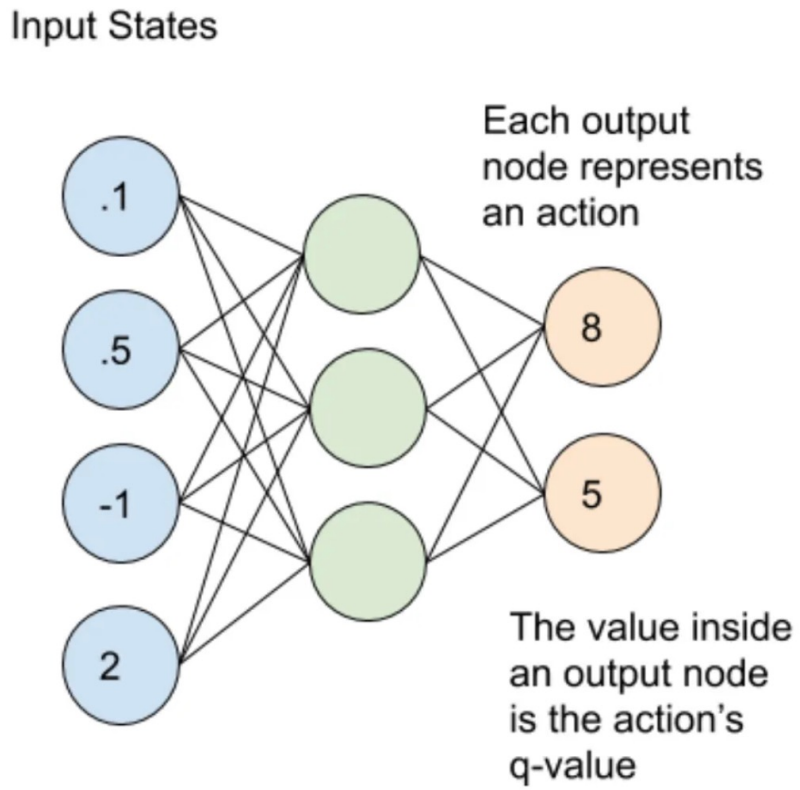
- Reduces the correlation between consecutive updates, leading to more stable training.
- Enables more efficient use of past experiences, improving learning performance.

Gradient Q-learning

Target Network：

- has the same architecture as the Q-network.
- its parameters are periodically updated by copying the parameters from the online Q-network.
- decoupling the target network from the online Q-network’s updates help prevent overestimation bias and improve the stability of the learning process.

Deep Q-Learning



Loop

Select action a and execute it

Receive immediate reward r

Observe new state s'

Add (s, a, s', r) to experience buffer

Sample mini-batch of experiences from buffer

For each experience $(\hat{s}, \hat{a}, \hat{s}', \hat{r})$ in mini-batch

Gradient: $\frac{\partial Err}{\partial \boldsymbol{w}} = \left[Q_{\boldsymbol{w}}(\hat{s}, \hat{a}) - \hat{r} - \gamma \max_{\hat{a}'} \mathrm{Q}_{\bar{\boldsymbol{w}}}(\hat{s}', \hat{a}') \right] \frac{\partial Q_{\boldsymbol{w}}(\hat{s}, \hat{a})}{\partial \boldsymbol{w}}$

Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \frac{\partial Err}{\partial \boldsymbol{w}}$

Update state: $s \leftarrow s'$

Every c steps, update target: $\bar{\boldsymbol{w}} \leftarrow \boldsymbol{w}$

Volodymyr et al. "Human-level control through deep reinforcement learning", 2015

Policy-Based RL

- Reparametrize and learn policy instead of the value function
- Advantages:
  - Better convergence properties
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies
- Disadvantages:
  - Typically converge to a local rather than global optimum
  - Evaluating a policy is typically inefficient and high variance

Policy Gradient

Represent Policy function $\pi_\theta(a|s)$:

- Finitely many discrete actions

  Softmax: $\pi_\theta(a|s) = \frac{\exp(h(s,a;\theta))}{\sum_{a'} \exp(h(s,a';\theta))}$

  where $h(s,a;\theta)$ might be

  linear in θ: $h(s,a;\theta) = \sum_i \theta_i f_i(s,a)$

  or **non-linear** in θ: $h(s,a;\theta) = neuralNet(s,a;\theta)$

- Continuous actions:

  Gaussian: $\pi_\theta(a|s) = N(a|\mu(s;\theta), \Sigma(s;\theta))$

Policy Gradient

- Consider policy $\pi_\theta(a|s)$
- Data: state-action-reward triples

$$\{(s_1, a_1, r_1), (s_2, a_2, r_2), \dots\}$$

- Maximize discounted sum of rewards

$$J(\theta) = \mathbb{E}_{\tau \sim p^{\pi_\theta}(\tau)}(G_0) = \mathbb{E}_{s_0 \sim p_0} V^{\pi_\theta}(s_0) = \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^{\pi_\theta}(s)} V^{\pi_\theta}(s)$$

where $d^{\pi_\theta}(s)$ is the stationary distribution of Markov chain for π_θ

- Gradient Ascent

$$\theta_{t+1} \leftarrow \theta_t + \alpha_t \nabla_\theta J(\theta)$$

The (unnormalized) γ-discounted state visitation distribution

$$d^\pi(s) = \sum_{t=0}^{\infty} \gamma^t \Pr(s_t = s \mid \pi).$$

$$G_0 = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$$

Policy Gradient

- The Policy Gradient is

$$\nabla J(\theta) = \frac{1}{1-\gamma}\,\mathbb{E}_{s\sim d^{\pi_\theta}, a\sim\pi_\theta(a|s)}\left[\nabla \log \pi_\theta(a|s)\;\; Q^{\pi_\theta}(s,a)\right]$$

- $\nabla \log \pi_\theta(a|s)$ is the score function

$$\nabla J(\theta) = \nabla \mathbb{E}_{\tau\sim p^{\pi_\theta}(\tau)}\,(G_0) = \mathbb{E}(\nabla p^{\pi_\theta}(\tau) G_0) = \mathbb{E}_{\tau\sim p^{\pi_\theta}(\tau)}\,(\nabla \log p^{\pi_\theta}(\tau)\, G_0) = \mathbb{E}_\tau\left(\sum_{t\geq 0}\gamma^t \nabla \log \pi_\theta\,(a_t|s_t) G_t\right)$$

$$= \mathbb{E}_\tau\left(\sum_{t\geq 0}\gamma^t \nabla \log \pi_\theta\,(a_t|s_t)\, Q^{\pi_\theta}(s_t,a_t)\right) = \frac{1}{1-\gamma}\,\mathbb{E}_{s\sim d^{\pi_\theta}, a\sim\pi_\theta(a|s)}\left[\nabla \log \pi_\theta(a|s)\;\; Q^{\pi_\theta}(s,a)\right]$$

$$\left(\nabla_\theta p^{\pi_\theta}(\tau) = p^{\pi_\theta}(\tau)\frac{\nabla p^{\pi_\theta}(\tau)}{p^{\pi_\theta}(\tau)} = p^{\pi_\theta}(\tau)\nabla \log p^{\pi_\theta}(\tau)\right)$$

Refer to Lilian Wong's blog (https://lilianweng.github.io/posts/2018-04-08-policy-gradient/) for another proof.

Policy Gradient: REINFORCE

- Monte-Carlo Gradient Update:

$$\theta_{n+1} \leftarrow \theta_n + \alpha_n v_t \nabla \log \pi_\theta(a_n|s_n)$$

- Using return v_t as an unbiased sample of $Q^{\pi_\theta}(s, a)$

function REINFORCE
Initialise θ arbitrarily
for each episode $\{s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
for $t = 1$ to $T - 1$ **do**
$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
end for
end for
return θ
end function

$$v_t = \sum_{n=0}^{T-t} \gamma^n r_{t+n}$$

Actor-critic algorithms

- Monte-Carlo policy gradient still has high variance
- We use a critic to estimate the action-value function
- Actor-critic algorithms maintain two sets of parameters
  - Critic Updates action-value function parameters w: SARSA, Q-learning…
  - Actor Updates policy parameters θ, in direction suggested by critic

$$\nabla J(\theta) = \mathbb{E}_{\pi_\theta}[Q_w(\mathrm{s}, \mathrm{a}) \nabla \log \pi_\theta(a|s)]$$

Actor-critic algorithms

function QAC

Initialise s, θ

Sample $a \sim \pi_\theta$

for each step **do**

Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$.

Sample action $a' \sim \pi_\theta(s', a')$

$\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$

$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$

$w \leftarrow w + \beta \delta \phi(s, a)$

$a \leftarrow a', s \leftarrow s'$

end for

end function

Time Difference

More General: $w \leftarrow w + \beta \delta \frac{\partial Q_w}{\partial w}$

Linear Q-function:
$Q(s, a) = \phi(s, a)^T w$

Policy Gradient With a Baseline

- Subtract a Baseline function $B(s)$ from the policy gradient

$$\mathbb{E}_{\pi_\theta}[\nabla \log \pi_\theta(a|s)\,(Q^{\pi_\theta}(s,a) - B(s))]$$

- This can reduce variance, without changing expectation

$$\mathbb{E}_{\pi_\theta}[\nabla \log \pi_\theta(a|s)\,B(s)] = \mathbb{E}_{s\sim d^{\pi_\theta}} \sum_{\mathrm{a}} \nabla\, \pi_\theta(a|s) B(s) = \mathbb{E}_{s\sim d^{\pi_\theta}} B(s) \nabla \sum_{a} \pi_\theta(a|s) = 0$$

$\sum_a \pi_\theta(a|s) = 1$

- Baseline often chosen to be the value funciton $V^{\pi_\theta}(s)$
- Advantage function:

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$$

- Policy Gradient Update:

$$\mathbb{E}_{\pi_\theta}[\nabla \log \pi_\theta(a|s)\,A^{\pi_\theta}(s,a)]$$

Advantage Actor Critic (A2C)

Advantage Actor Critic (A2C)

A2C()

Initialize π_θ to anything

Loop forever (for each episode)

Initialize s_0 and set $n \leftarrow 0$

Loop while s is not terminal (for each time step n)

Select a_n

Execute a_n, observe s_{n+1}, r_n

$\delta \leftarrow r_n + \gamma \max_{a_{n+1}} Q_w(s_{n+1}, a_{n+1}) - Q_w(s_n, a_n)$

$A(s_n, a_n) \leftarrow r_n + \gamma \max_{a_{n+1}} Q_w(s_{n+1}, a_{n+1})$

$- \sum_a \pi_\theta(a|s_n) Q_w(s_n, a)$

Update Q: $w \leftarrow w + \alpha_w \gamma^n \delta \nabla_w Q_w(s_n, a_n)$

Update π: $\theta \leftarrow \theta + \alpha_\theta \gamma^n A(s_n, a_n) \nabla \log \pi_\theta(a_n|s_n)$

$n \leftarrow n + 1$

Value function

γ: discount ratio

TD A2C

- For the true value function, the TD (Time Difference) error

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

is an unbiased estimate of

$$\begin{aligned}\mathbb{E}_{\pi_\theta}\left[\delta^{\pi_\theta}|s,a\right] &= \mathbb{E}_{\pi_\theta}\left[r + \gamma V^{\pi_\theta}(s')|s,a\right] - V^{\pi_\theta}(s) \\ &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \\ &= A^{\pi_\theta}(s,a)\end{aligned}$$

- So the policy gradient can be calculated as $\mathbb{E}_{\pi_\theta}[\nabla \log \pi_\theta(a|s)\, \delta^{\pi_\theta}]$
- In practice, we use $V_w(s) \approx V^{\pi_\theta}(s')$

Policy Gradient

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ v_t\right] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)\right] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^w(s,a)\right] && \text{Advantage Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta\right] && \text{TD Actor-Critic}
\end{aligned}
$$

Trust Region Policy Optimization (TRPO)

- Let $\eta(\pi)$ denote the expected discounted reward of π ($J(\theta) = \eta(\pi_\theta)$)

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right], \; s_0 \sim \rho_0(s_0), \; a_t \sim \pi(a_t|s_t), \; s_{t+1} \sim P(s_{t+1}|s_t, a_t).$$

- The expected return difference of two policies

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0, a_0, \cdots \sim \tilde{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)\right]$$

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{d^{\tilde{\pi}}}(\tilde{\pi}(a|s) A_\pi(s, a))$$

"Trust Region Policy Optimization", John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, Pieter Abbeel, 2015

Trust Region Policy Optimization (TRPO)

- Local approximation to η

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{d^\pi}(\tilde{\pi}(a|s)A_\pi(s,a))$$

- A sufficiently small step that improves $L_\pi(\tilde{\pi})$ will also improve $\eta(\tilde{\pi})$, but does not give us any guidance on how big of a step to take

$$D_{\mathrm{TV}}^{\max}(\pi, \tilde{\pi}) = \max_s D_{TV}(\pi(\cdot|s) \parallel \tilde{\pi}(\cdot|s)). \qquad (7)$$

Theorem 1. *Let* $\alpha = D_{\mathrm{TV}}^{\max}(\pi_{\mathrm{old}}, \pi_{\mathrm{new}})$. *Then the following bound holds:*

$$\eta(\pi_{\mathrm{new}}) \geq L_{\pi_{\mathrm{old}}}(\pi_{\mathrm{new}}) - \frac{4\epsilon\gamma}{(1-\gamma)^2}\alpha^2$$

$$\textit{where } \epsilon = \max_{s,a} |A_\pi(s,a)| \qquad (8)$$

The theorem implies that a policy update that improves the right-hand side is guaranteed to improve the true performance.

Trust Region Policy Optimization (TRPO)

- TRPO proposes to solve

$$\underset{\theta}{\text{maximize}} \left[L_{\theta_{\text{old}}}(\theta) - C D_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta) \right] \qquad \text{where } C = \frac{4\epsilon\gamma}{(1-\gamma)^2}.$$

  - When $\theta = \theta_{old}$, the objective is the just $\eta(\theta)$

- More robustly, solve

$$\underset{\theta}{\text{maximize}}\, L_{\theta_{\text{old}}}(\theta)$$
$$\text{subject to } D_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta) \le \delta.$$

$$\underset{\theta}{\text{maximize}}\, L_{\theta_{\text{old}}}(\theta)$$
$$\text{subject to } \overline{D}_{\text{KL}}^{\rho_{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) \le \delta.$$

$$\overline{D}_{\text{KL}}^{\rho}(\theta_1, \theta_2) := \mathbb{E}_{s\sim\rho}\left[D_{\text{KL}}(\pi_{\theta_1}(\cdot|s) \,\|\, \pi_{\theta_2}(\cdot|s)) \right].$$

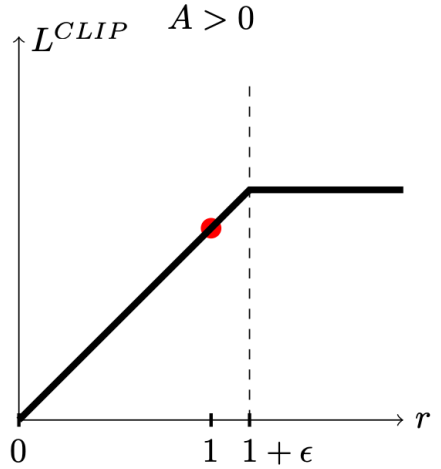
Proximal Policy Optimization (PPO)

- TRPO loss can be rewritten as

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{d^\pi}\left(\tilde{\pi}(a|s)A_\pi(s,a)\right) - \beta \mathrm{KL}^{\max}(\pi|\tilde{\pi})$$

$$= \eta(\pi) + \mathbb{E}_{d^\pi,\pi}\left(\frac{\tilde{\pi}(a|s)}{\pi(a|s)}A_\pi(s,a)\right) - \beta \mathrm{KL}^{\max}(\pi|\tilde{\pi})$$

- A major disadvantage of TRPO is that it's computationally expensive. PPO is proposed to simplify it:



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

"Proximal Policy Optimization Algorithms", John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

Proximal Policy Optimization (PPO)

- Adaptive KL Penalty Coefficient
  - Using several epochs of minibatch SGD, optimize the KL-penalized objective

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

  - Compute $d = \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$
    - If $d < d_{\text{targ}}/1.5$, $\beta \leftarrow \beta/2$
    - If $d > d_{\text{targ}} \times 1.5$, $\beta \leftarrow \beta \times 2$

algorithm	avg. normalized score
No clipping or penalty	-0.39
Clipping, $\epsilon = 0.1$	0.76
Clipping, $\epsilon = 0.2$	**0.82**
Clipping, $\epsilon = 0.3$	0.70
Adaptive KL $d_{\text{targ}} = 0.003$	0.68
Adaptive KL $d_{\text{targ}} = 0.01$	0.74
Adaptive KL $d_{\text{targ}} = 0.03$	0.71
Fixed KL, $\beta = 0.3$	0.62
Fixed KL, $\beta = 1.$	0.71
Fixed KL, $\beta = 3.$	0.72
Fixed KL, $\beta = 10.$	0.69

Reinforcement Learning from Human Feedback (RLHF)

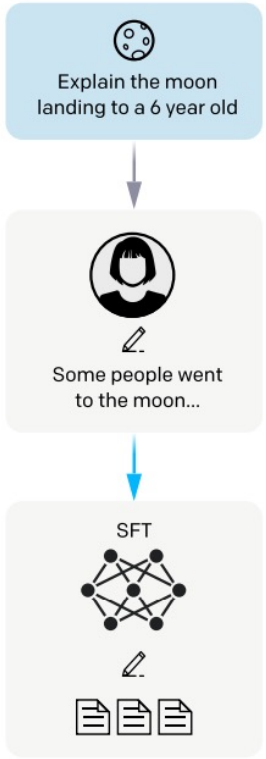
Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

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



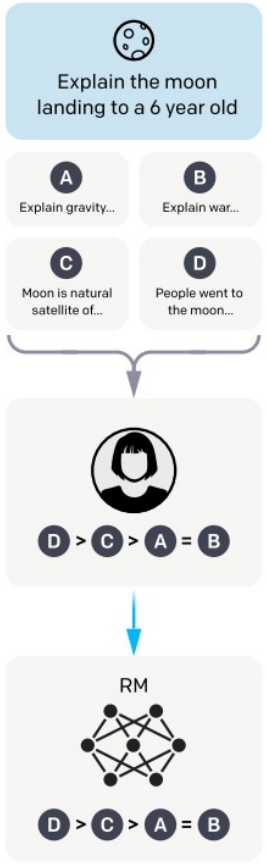
Step 2

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

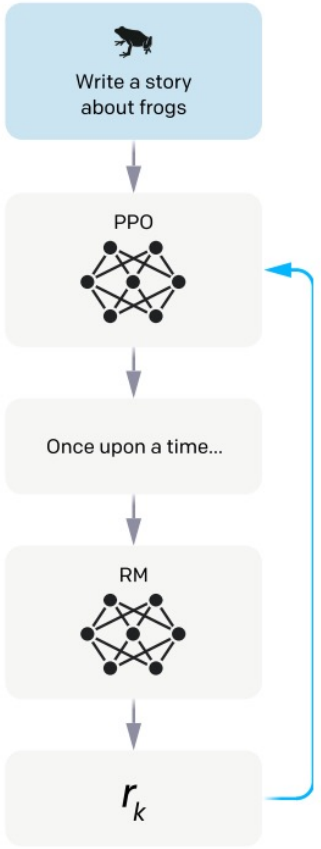
Optimize a policy against the reward model using reinforcement learning.

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.



InstructGPT:Training language models to follow instructions with human feedback,2022

RLHF

- The loss function for the reward model is (a binary classification)

$$\text{loss}\,(\theta) = -\frac{1}{\binom{K}{2}} E_{(x,y_w,y_l)\sim D}\left[\log\left(\sigma\left(r_\theta\left(x,y_w\right) - r_\theta\left(x,y_l\right)\right)\right)\right]$$

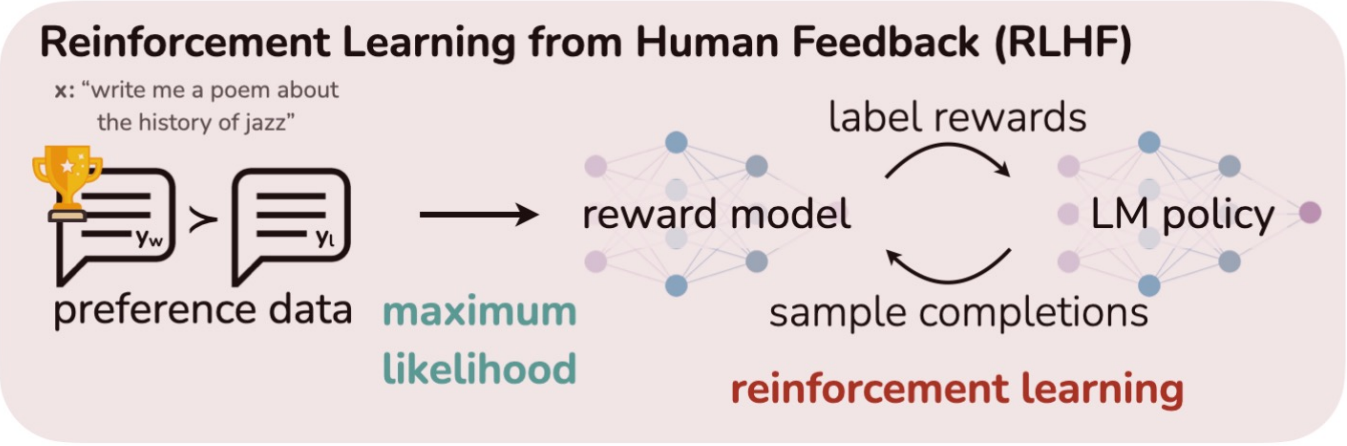
  - σ is the logistic function
  - All comparisons from each prompt are trainedas a single batch element.
  - y_w is the preferred completion out of the pair of y_w and y_l

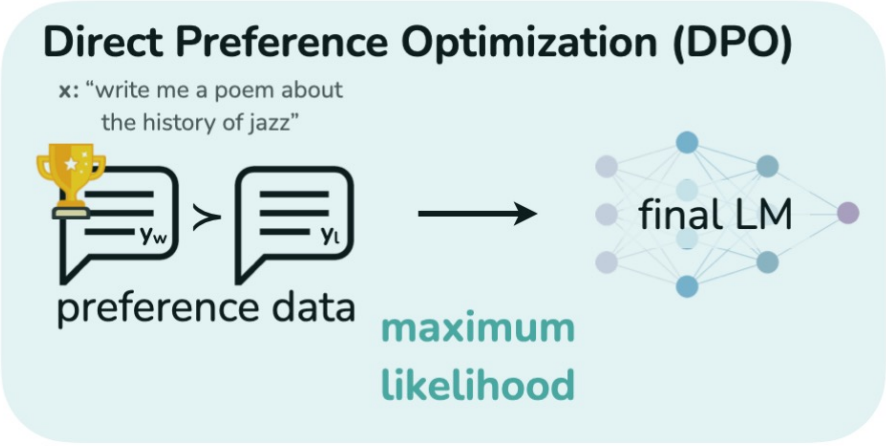
- PPO-ptx: mixing the pretraining gradients into the PPO gradients

$$\text{objective}\,(\phi) = E_{(x,y)\sim D_{\pi_\phi^{\text{RL}}}}\left[r_\theta(x,y) - \beta\log\left(\pi_\phi^{\text{RL}}(y \mid x)/\pi^{\text{SFT}}(y \mid x)\right)\right] +$$
$$\gamma E_{x\sim D_{\text{pretrain}}}\left[\log(\pi_\phi^{\text{RL}}(x))\right]$$

Direct Preference Optimization(DPO)

- RLHF fits a reward model to human preferences, and then use RL to find a policy that maximizes the learned reward.
- DPO directly optimizes for the policy best satisfying the preferences.





Direct Preference Optimization: Your Language Model is Secretly a Reward Mode,2023

Direct Preference Optimization(DPO)

- RLHF

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)}\left[r_\phi(x,y)\right] - \beta\mathbb{D}_{\mathrm{KL}}\left[\pi_\theta(y \mid x) \,||\, \pi_{\mathrm{ref}}(y \mid x)\right],$$

- The optimal solution is

$$\pi_r(y \mid x) = \frac{1}{Z(x)}\pi_{\mathrm{ref}}(y \mid x)\exp\left(\frac{1}{\beta}r(x,y)\right), \quad \Longrightarrow \quad r(x,y) = \beta\log\frac{\pi_r(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)} + \beta\log Z(x).$$

- Bradley-Terry (BT) human preference model

$$p^*(y_1 \succ y_2 \mid x) = \frac{\exp\left(r^*(x,y_1)\right)}{\exp\left(r^*(x,y_1)\right) + \exp\left(r^*(x,y_2)\right)}.$$

- The optimal RLHF policy $\pi*$ under the Bradley-Terry model satisfies the preference model:

$$p^*(y_1 \succ y_2 \mid x) = \frac{1}{1+\exp\left(\beta\log\frac{\pi^*(y_2|x)}{\pi_{\mathrm{ref}}(y_2|x)} - \beta\log\frac{\pi^*(y_1|x)}{\pi_{\mathrm{ref}}(y_1|x)}\right)}$$

Direct Preference Optimization(DPO)

- DPO objective (Maximum likelihood of BT model)becomes

$$\mathcal{L}_{\mathrm{DPO}}(\pi_\theta; \pi_{\mathrm{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\mathrm{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\mathrm{ref}}(y_l \mid x)}\right)\right].$$

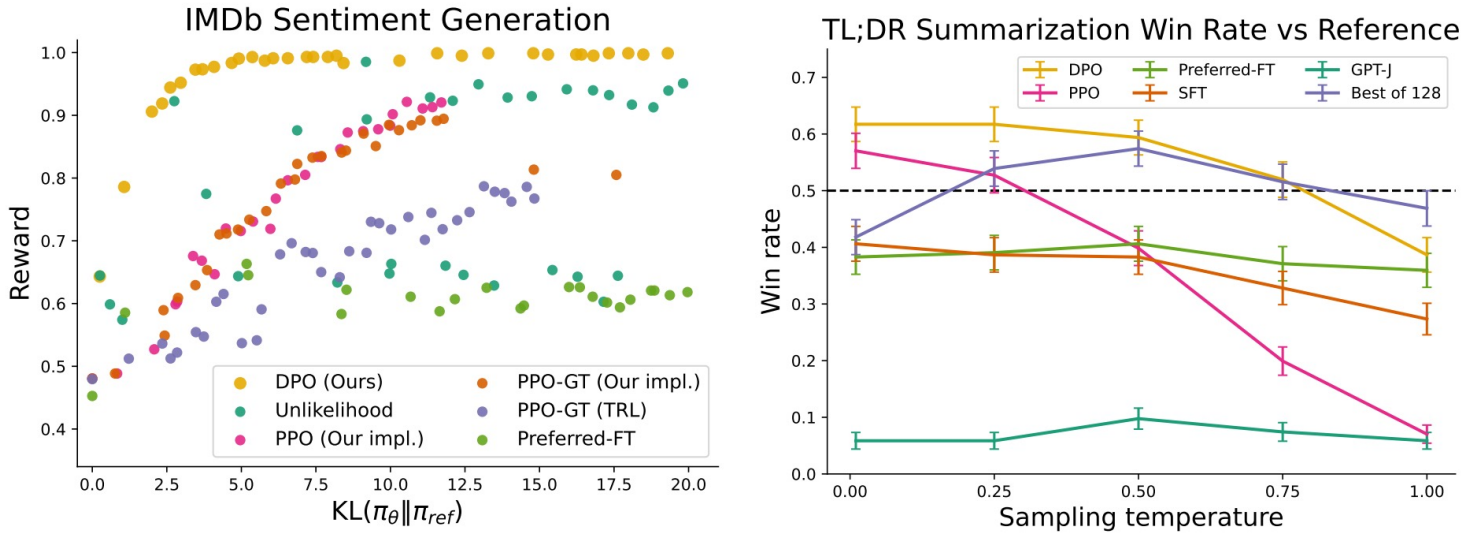


Figure 2: **Left.** The frontier of expected reward vs KL to the reference policy. DPO provides the highest expected reward for all KL values, demonstrating the quality of the optimization. **Right.** TL;DR summarization win rates vs. human-written summaries, using GPT-4 as evaluator. DPO exceeds PPO's best-case performance on summarization, while being more robust to changes in the sampling temperature.

Group Relative Policy Optimization (GRPO)

- Like DPO, GRPO eliminates the need for a separate critic model. Instead it uses the **group average** of multiple outputs as its baseline.
- Workflow
  - For each prompt p, sample a group of N responses
  - Compute rewards
  - Calculate advantage

$$A_i = \frac{R_\phi(r_i) - \text{mean}(\mathcal{G})}{\text{std}(\mathcal{G})}$$

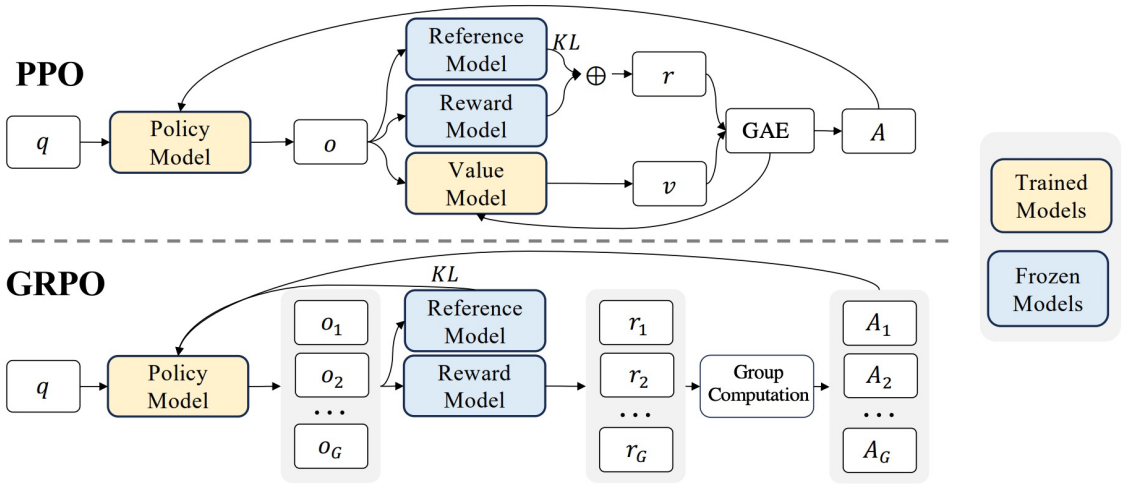


Figure 4 | Demonstration of PPO and our GRPO. GRPO foregoes the value model, instead estimating the baseline from group scores, significantly reducing training resources.

DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models,2024