Short Seminar about Reinforcement Learning

Dorian Gailhard

doriangailhard.github.io

June 25, 2026

Context: Why Reinforcement Learning?

In supervised learning, the model learns from labels:

$$(x, y) \sim \mathcal{D}, \qquad f_\theta(x) \approx y.$$

Context: Why Reinforcement Learning?

In supervised learning, the model learns from labels:

$$(x, y) \sim \mathcal{D}, \qquad f_\theta(x) \approx y.$$

In reinforcement learning, an **agent** learns from consequences.

Context: Why Reinforcement Learning?

In supervised learning, the model learns from labels:

$$(x, y) \sim \mathcal{D}, \qquad f_\theta(x) \approx y.$$

In reinforcement learning, an **agent** learns from consequences.

The agent interacts with an **environment**:

$$s_t \xrightarrow{a_t} s_{t+1}, \qquad r_t \in \mathbb{R}$$

Context: Why Reinforcement Learning?

In supervised learning, the model learns from labels:

$$(x, y) \sim \mathcal{D}, \qquad f_\theta(x) \approx y.$$

In reinforcement learning, an **agent** learns from consequences.

The agent interacts with an **environment**:

$$s_t \xrightarrow{a_t} s_{t+1}, \qquad r_t \in \mathbb{R}$$

The goal is not to predict labels, but to choose actions that maximize future reward.

Context: Why Reinforcement Learning?

In supervised learning, the model learns from labels:

$$(x, y) \sim \mathcal{D}, \qquad f_\theta(x) \approx y.$$

In reinforcement learning, an **agent** learns from consequences.

The agent interacts with an **environment**:

$$s_t \xrightarrow{a_t} s_{t+1}, \qquad r_t \in \mathbb{R}$$

The goal is not to predict labels, but to choose actions that maximize future reward.

Remember: RL is about learning to make decisions.

Where is RL Used?

Typical applications:

- **Games:** Go, Chess, Shogi, Atari, StarCraft

Where is RL Used?

Typical applications:

- **Games:** Go, Chess, Shogi, Atari, StarCraft
- **Robotics:** locomotion, manipulation, control

Where is RL Used?

Typical applications:

- **Games:** Go, Chess, Shogi, Atari, StarCraft
- **Robotics:** locomotion, manipulation, control
- **Recommendation systems:** sequential user interaction

Where is RL Used?

Typical applications:

- **Games:** Go, Chess, Shogi, Atari, StarCraft
- **Robotics:** locomotion, manipulation, control
- **Recommendation systems:** sequential user interaction
- **Resource allocation:** scheduling, logistics, datacenters

Where is RL Used?

Typical applications:

- **Games:** Go, Chess, Shogi, Atari, StarCraft
- **Robotics:** locomotion, manipulation, control
- **Recommendation systems:** sequential user interaction
- **Resource allocation:** scheduling, logistics, datacenters
- **Language models:** RLHF, reasoning, alignment

Where is RL Used?

Typical applications:

- **Games:** Go, Chess, Shogi, Atari, StarCraft
- **Robotics:** locomotion, manipulation, control
- **Recommendation systems:** sequential user interaction
- **Resource allocation:** scheduling, logistics, datacenters
- **Language models:** RLHF, reasoning, alignment

Common structure: decisions influence future observations.

Formalization: Markov Decision Process

An RL problem is often formalized as a **Markov Decision Process**:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma).$$

Formalization: Markov Decision Process

An RL problem is often formalized as a **Markov Decision Process**:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma).$$

- $\mathcal{S}$: set of states

Formalization: Markov Decision Process

An RL problem is often formalized as a **Markov Decision Process**:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma).$$

- $\mathcal{S}$: set of states
- $\mathcal{A}$: set of actions

Formalization: Markov Decision Process

An RL problem is often formalized as a **Markov Decision Process**:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma).$$

- $\mathcal{S}$: set of states
- $\mathcal{A}$: set of actions
- $P(s' \mid s, a)$: transition dynamics

Formalization: Markov Decision Process

An RL problem is often formalized as a **Markov Decision Process**:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma).$$

- $\mathcal{S}$: set of states
- $\mathcal{A}$: set of actions
- $P(s' \mid s, a)$: transition dynamics
- $r(s, a)$: reward function

Formalization: Markov Decision Process

An RL problem is often formalized as a **Markov Decision Process**:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma).$$

- $\mathcal{S}$: set of states
- $\mathcal{A}$: set of actions
- $P(s' \mid s, a)$: transition dynamics
- $r(s, a)$: reward function
- $\gamma \in [0, 1]$: discount factor

Formalization: Markov Decision Process

An RL problem is often formalized as a **Markov Decision Process**:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma).$$

- $\mathcal{S}$: set of states
- $\mathcal{A}$: set of actions
- $P(s' \mid s, a)$: transition dynamics
- $r(s, a)$: reward function
- $\gamma \in [0, 1]$: discount factor

A policy $\pi(a \mid s)$ chooses actions from states.

Objective: Maximize Return

Given a policy $\pi_\theta(a \mid s)$, the agent generates trajectories:

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots).$$

Objective: Maximize Return

Given a policy $\pi_\theta(a \mid s)$, the agent generates trajectories:

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots).$$

The objective is to maximize the expected discounted return:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right].$$

Objective: Maximize Return

Given a policy $\pi_\theta(a \mid s)$, the agent generates trajectories:

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots).$$

The objective is to maximize the expected discounted return:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right].$$

Problem: the agent does not know which actions lead to high long-term reward.

Objective: Maximize Return

Given a policy $\pi_\theta(a \mid s)$, the agent generates trajectories:

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots).$$

The objective is to maximize the expected discounted return:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right].$$

Problem: the agent does not know which actions lead to high long-term reward.

Unlike supervised learning, the data distribution is not fixed: it depends on the policy itself. The agent must actively **explore** the state space to discover good behavior.

Two Major Paradigms in Modern RL

Modern systems often combine several ideas, but two important paradigms are:

Search / Planning	**Policy Optimization**
Monte Carlo Tree Search	Policy gradients
AlphaGo, AlphaZero, MuZero	PPO, RLHF, GRPO
Uses simulations of the future	Directly updates the policy
Strong when planning is possible	Scales well with neural networks

Two Major Paradigms in Modern RL

Modern systems often combine several ideas, but two important paradigms are:

Search / Planning	**Policy Optimization**
Monte Carlo Tree Search	Policy gradients
AlphaGo, AlphaZero, MuZero	PPO, RLHF, GRPO
Uses simulations of the future	Directly updates the policy
Strong when planning is possible	Scales well with neural networks

Other important directions exist: Q-learning / DQN, SAC, offline RL, model-based RL, evolutionary methods, etc.

Two Major Paradigms in Modern RL

Modern systems often combine several ideas, but two important paradigms are:

Search / Planning	**Policy Optimization**
Monte Carlo Tree Search	Policy gradients
AlphaGo, AlphaZero, MuZero	PPO, RLHF, GRPO
Uses simulations of the future	Directly updates the policy
Strong when planning is possible	Scales well with neural networks

Other important directions exist: Q-learning / DQN, SAC, offline RL, model-based RL, evolutionary methods, etc.

In this seminar: **MCTS-family** and **PPO-family** methods.

Monte Carlo Tree Search

Idea: use simulated futures to decide what to do now.

MCTS builds a partial search tree by repeating four steps:

Monte Carlo Tree Search

Idea: use simulated futures to decide what to do now.

MCTS builds a partial search tree by repeating four steps:

1. **Selection:** follow promising actions in the tree

Monte Carlo Tree Search

Idea: use simulated futures to decide what to do now.

MCTS builds a partial search tree by repeating four steps:

1. **Selection:** follow promising actions in the tree
2. **Expansion:** add a new node to the tree

Monte Carlo Tree Search

Idea: use simulated futures to decide what to do now.

MCTS builds a partial search tree by repeating four steps:

1. **Selection:** follow promising actions in the tree
2. **Expansion:** add a new node to the tree
3. **Evaluation / simulation:** estimate the value of the new state

Monte Carlo Tree Search

Idea: use simulated futures to decide what to do now.

MCTS builds a partial search tree by repeating four steps:

1. **Selection:** follow promising actions in the tree
2. **Expansion:** add a new node to the tree
3. **Evaluation / simulation:** estimate the value of the new state
4. **Backup:** propagate the estimate back to previous nodes

Monte Carlo Tree Search

Idea: use simulated futures to decide what to do now.

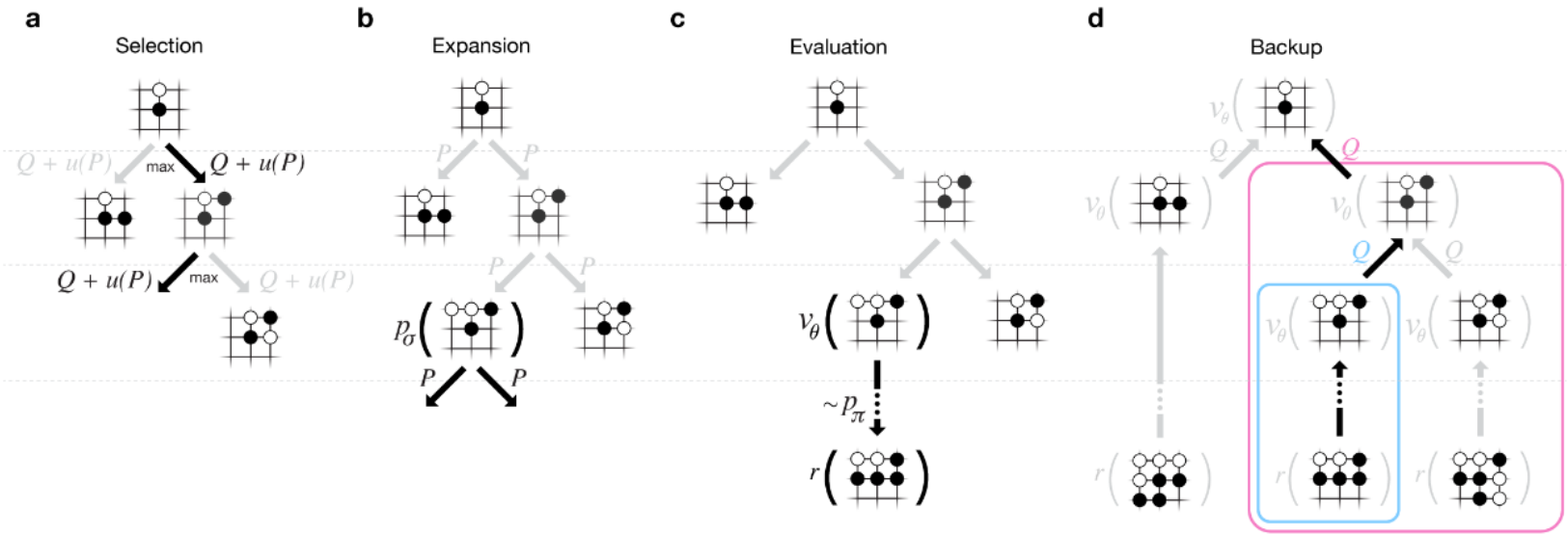
MCTS builds a partial search tree by repeating four steps:

1. **Selection:** follow promising actions in the tree
2. **Expansion:** add a new node to the tree
3. **Evaluation / simulation:** estimate the value of the new state
4. **Backup:** propagate the estimate back to previous nodes

After many simulations, choose the action with the best search statistics.

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016



Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

AlphaGo combines **MCTS** with deep neural networks.

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

AlphaGo combines **MCTS** with deep neural networks.

Two main learned models:

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

AlphaGo combines **MCTS** with deep neural networks.

Two main learned models:

- **Policy network** $p_\sigma(a \mid s)$: proposes promising moves

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

AlphaGo combines **MCTS** with deep neural networks.

Two main learned models:

- **Policy network** $p_\sigma(a \mid s)$: proposes promising moves
- **Value network** $v_\theta(s)$: estimates the probability of winning

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

AlphaGo combines **MCTS** with deep neural networks.

Two main learned models:

- **Policy network** $p_\sigma(a \mid s)$: proposes promising moves
- **Value network** $v_\theta(s)$: estimates the probability of winning

Role in MCTS:

- policy network makes search more focused
- value network avoids expensive random rollouts

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

AlphaGo combines **MCTS** with deep neural networks.

Two main learned models:

- **Policy network** $p_\sigma(a \mid s)$: proposes promising moves
- **Value network** $v_\theta(s)$: estimates the probability of winning

Role in MCTS:

- policy network makes search more focused
- value network avoids expensive random rollouts

More prosaically:

- you do not try moves completely at random: the policy network proposes promising actions
- you do not need to simulate the whole game until the end: the value network estimates whether a position is good or bad

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

During search, actions are selected using the Predictor + Upper Confidence bound applied to Trees (PUCT) formula:

$$a = \arg\max_a \left[Q(s,a) + c_{\text{puct}}\, P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \right].$$

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

During search, actions are selected using the Predictor + Upper Confidence bound applied to Trees (PUCT) formula:

$$a = \arg\max_a \left[Q(s,a) + c_{\text{puct}}\, P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \right].$$

- $Q(s,a)$: value estimated from previous simulations

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

During search, actions are selected using the Predictor + Upper Confidence bound applied to Trees (PUCT) formula:

$$a = \arg\max_a \left[Q(s,a) + c_{\text{puct}}\, P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \right].$$

- $Q(s,a)$: value estimated from previous simulations
- $P(s,a)$: prior probability from the policy network

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

During search, actions are selected using the Predictor + Upper Confidence bound applied to Trees (PUCT) formula:

$$a = \arg\max_a \left[Q(s,a) + c_{\text{puct}}\, P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \right].$$

- $Q(s,a)$: value estimated from previous simulations
- $P(s,a)$: prior probability from the policy network
- $N(s,a)$: number of times action a was visited

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

During search, actions are selected using the Predictor + Upper Confidence bound applied to Trees (PUCT) formula:

$$a = \arg\max_a \left[Q(s,a) + c_{\text{puct}}\, P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \right].$$

- $Q(s,a)$: value estimated from previous simulations
- $P(s,a)$: prior probability from the policy network
- $N(s,a)$: number of times action a was visited
- c_{puct}: exploration strength

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

During search, actions are selected using the Predictor + Upper Confidence bound applied to Trees (PUCT) formula:

$$a = \arg\max_a \left[Q(s,a) + c_{\text{puct}}\, P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \right].$$

- $Q(s,a)$: value estimated from previous simulations
- $P(s,a)$: prior probability from the policy network
- $N(s,a)$: number of times action a was visited
- c_{puct}: exploration strength

Intuition: try promising actions, but also explore promising under-visited actions. Once explored enough, select actions based on the value estimation.

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

At each state s_t, MCTS performs many simulations and builds visit counts: $N(s_t, a)$.

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

At each state s_t, MCTS performs many simulations and builds visit counts: $N(s_t, a)$.

The search policy is then defined from these visit counts:

$$\pi_{\text{MCTS}}(a \mid s_t) = \frac{N(s_t, a)^{1/\tau}}{\sum_b N(s_t, b)^{1/\tau}},$$

where τ controls exploration.

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

At each state s_t, MCTS performs many simulations and builds visit counts: $N(s_t, a)$.

The search policy is then defined from these visit counts:

$$\pi_{\text{MCTS}}(a \mid s_t) = \frac{N(s_t, a)^{1/\tau}}{\sum_b N(s_t, b)^{1/\tau}},$$

where τ controls exploration.

Once the search is complete, select an action from the MCTS policy:

$$a_t \sim \pi_{\text{MCTS}}(\cdot \mid s_t).$$

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

At each state s_t, MCTS performs many simulations and builds visit counts: $N(s_t, a)$.

The search policy is then defined from these visit counts:

$$\pi_{\text{MCTS}}(a \mid s_t) = \frac{N(s_t, a)^{1/\tau}}{\sum_b N(s_t, b)^{1/\tau}},$$

where τ controls exploration.

Once the search is complete, select an action from the MCTS policy:

$$a_t \sim \pi_{\text{MCTS}}(\cdot \mid s_t).$$

Repeat this process until the end of the game, and generate many self-play games to build a training dataset.

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

After self-play, each position gives a training example:

$$(s, \pi_{\text{MCTS}}, z),$$

where:

- s: game state
- π_{MCTS}: improved search policy
- $z \in \{-1, +1\}$: final game outcome

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

After self-play, each position gives a training example:

$$(s, \pi_{\text{MCTS}}, z),$$

where:

- s: game state
- π_{MCTS}: improved search policy
- $z \in \{-1, +1\}$: final game outcome

The network is trained with:

$$\mathcal{L}(\theta) = \big(z - v_\theta(s)\big)^2 - \pi_{\text{MCTS}}^\top \log p_\theta(\cdot \mid s) + \lambda \|\theta\|^2.$$

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

After self-play, each position gives a training example:

$$(s, \pi_{\text{MCTS}}, z),$$

where:

- s: game state
- π_{MCTS}: improved search policy
- $z \in \{-1, +1\}$: final game outcome

The network is trained with:

$$\mathcal{L}(\theta) = \big(z - v_\theta(s)\big)^2 - \pi_{\text{MCTS}}^\top \log p_\theta(\cdot \mid s) + \lambda \|\theta\|^2.$$

- value loss: predict who eventually wins
- policy loss: imitate the stronger MCTS policy
- regularization: avoid overfitting

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

AlphaGo was not trained from scratch.

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

AlphaGo was not trained from scratch.

1. **Supervised learning from expert games**
   Train a policy network to imitate human expert moves.

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

AlphaGo was not trained from scratch.

1. **Supervised learning from expert games**
   Train a policy network to imitate human expert moves.
2. **Self-play with MCTS**
   Use MCTS guided by the current networks to generate stronger games.

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

AlphaGo was not trained from scratch.

1. **Supervised learning from expert games**
   Train a policy network to imitate human expert moves.
2. **Self-play with MCTS**
   Use MCTS guided by the current networks to generate stronger games.
3. **Policy improvement**
   Train the policy network to imitate the improved search policy produced by MCTS.

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

AlphaGo was not trained from scratch.

1. **Supervised learning from expert games**
   Train a policy network to imitate human expert moves.
2. **Self-play with MCTS**
   Use MCTS guided by the current networks to generate stronger games.
3. **Policy improvement**
   Train the policy network to imitate the improved search policy produced by MCTS.
4. **Value network training**
   Predict game outcomes from self-play positions.

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

AlphaGo was not trained from scratch.

1. **Supervised learning from expert games**
   Train a policy network to imitate human expert moves.
2. **Self-play with MCTS**
   Use MCTS guided by the current networks to generate stronger games.
3. **Policy improvement**
   Train the policy network to imitate the improved search policy produced by MCTS.
4. **Value network training**
   Predict game outcomes from self-play positions.
5. **Inference with MCTS**
   Combine search and neural networks again at test time.

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

Why does this work?

- Pure search is too expensive: Go has a huge branching factor.

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

Why does this work?

- Pure search is too expensive: Go has a huge branching factor.
- Pure neural prediction is too imprecise: one mistake can lose the game.

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

Why does this work?

- Pure search is too expensive: Go has a huge branching factor.
- Pure neural prediction is too imprecise: one mistake can lose the game.
- AlphaGo combines both:
  - neural networks provide intuition
  - search provides lookahead and correction

Mastering the game of Go with deep neural networks and tree search

David Silver et al., *Nature*, 2016

Why does this work?

- Pure search is too expensive: Go has a huge branching factor.
- Pure neural prediction is too imprecise: one mistake can lose the game.
- AlphaGo combines both:
  - neural networks provide intuition
  - search provides lookahead and correction

Remember: performance can improve by spending more computation at inference time, not only by training larger models.

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, 2017

AlphaZero generalizes AlphaGo-style learning.

AlphaGo	**AlphaZero**
Starts from expert games	Starts from random play
Uses human imitation	No expert imitation
Designed for Go	Same algorithm for Go, Chess, Shogi
Policy/value + MCTS	Policy/value + MCTS

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, 2017

AlphaZero generalizes AlphaGo-style learning.

AlphaGo	**AlphaZero**
Starts from expert games	Starts from random play
Uses human imitation	No expert imitation
Designed for Go	Same algorithm for Go, Chess, Shogi
Policy/value + MCTS	Policy/value + MCTS

Key idea: expert knowledge can be replaced by self-play at scale.

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, 2017

Training loop:

1. Use the current network to guide MCTS.

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, 2017

Training loop:

1. Use the current network to guide MCTS.
2. Use MCTS to choose stronger actions during self-play.

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, 2017

Training loop:

1. Use the current network to guide MCTS.
2. Use MCTS to choose stronger actions during self-play.
3. Store states, improved search policies, and final outcomes.

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, 2017

Training loop:

1. Use the current network to guide MCTS.
2. Use MCTS to choose stronger actions during self-play.
3. Store states, improved search policies, and final outcomes.
4. Train the network to predict:
   - the MCTS action distribution
   - the final game outcome

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, 2017

Training loop:

1. Use the current network to guide MCTS.
2. Use MCTS to choose stronger actions during self-play.
3. Store states, improved search policies, and final outcomes.
4. Train the network to predict:
   - the MCTS action distribution
   - the final game outcome

At test time: the network and MCTS are again combined to choose actions.

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, 2017

AlphaZero was trained at very large scale.

	Chess	**Shogi**	**Go**
Mini-batches	700k	700k	700k
Training time	9h	12h	34h
Training games	44M	24M	21M
MCTS simulations	800	800	800
Thinking time / move	40ms	80ms	200ms

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, 2017

AlphaZero was trained at very large scale.

	Chess	**Shogi**	**Go**
Mini-batches	700k	700k	700k
Training time	9h	12h	34h
Training games	44M	24M	21M
MCTS simulations	800	800	800
Thinking time / move	40ms	80ms	200ms

Inference setup: each MCTS executed on 4 TPUs

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, 2017

AlphaZero was trained at very large scale.

	Chess	**Shogi**	**Go**
Mini-batches	700k	700k	700k
Training time	9h	12h	34h
Training games	44M	24M	21M
MCTS simulations	800	800	800
Thinking time / move	40ms	80ms	200ms

Inference setup: each MCTS executed on 4 TPUs

Important perspective:

- massive self-play
- neural network training
- substantial test-time computation

Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model

Julian Schrittwieser et al., *Nature*, 2020

AlphaZero assumes we know the environment dynamics.

Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model

Julian Schrittwieser et al., *Nature*, 2020

AlphaZero assumes we know the environment dynamics.

For board games, the rules are known:

$$(s_t, a_t) \mapsto s_{t+1}.$$

Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model

Julian Schrittwieser et al., *Nature*, 2020

AlphaZero assumes we know the environment dynamics.

For board games, the rules are known:

$$(s_t, a_t) \mapsto s_{t+1}.$$

But in many environments, the dynamics are unknown or too complex.

Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model

Julian Schrittwieser et al., *Nature*, 2020

AlphaZero assumes we know the environment dynamics.

For board games, the rules are known:

$$(s_t, a_t) \mapsto s_{t+1}.$$

But in many environments, the dynamics are unknown or too complex.

MuZero idea: learn a model that is only useful for planning.

Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model

Julian Schrittwieser et al., *Nature*, 2020

AlphaZero assumes we know the environment dynamics.

For board games, the rules are known:

$$(s_t, a_t) \mapsto s_{t+1}.$$

But in many environments, the dynamics are unknown or too complex.

MuZero idea: learn a model that is only useful for planning.

It does not try to reconstruct the full observation; it predicts rewards, values, and policies in a latent space.

Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model

Julian Schrittwieser et al., *Nature*, 2020

MuZero learns a compact model for planning.

- **Representation model** h_θ:

$$s^0 = h_\theta(o_1, \ldots, o_t)$$

Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model

Julian Schrittwieser et al., *Nature*, 2020

MuZero learns a compact model for planning.

- **Representation model** h_θ:

$$s^0 = h_\theta(o_1, \ldots, o_t)$$

- **Dynamics model** g_θ:

$$r^k, s^k = g_\theta(s^{k-1}, a^k)$$

Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model

Julian Schrittwieser et al., *Nature*, 2020

MuZero learns a compact model for planning.

- **Representation model** h_θ:

$$s^0 = h_\theta(o_1, \ldots, o_t)$$

- **Dynamics model** g_θ:

$$r^k, s^k = g_\theta(s^{k-1}, a^k)$$

- **Prediction model** f_θ:

$$p^k, v^k = f_\theta(s^k)$$

Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model

Julian Schrittwieser et al., *Nature*, 2020

MuZero learns a compact model for planning.

- **Representation model** h_θ:

$$s^0 = h_\theta(o_1, \ldots, o_t)$$

- **Dynamics model** g_θ:

$$r^k, s^k = g_\theta(s^{k-1}, a^k)$$

- **Prediction model** f_θ:

$$p^k, v^k = f_\theta(s^k)$$

MCTS is then performed inside this learned latent model.

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

Deep RL often needs a very large number of environment interactions.

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

Deep RL often needs a very large number of environment interactions.

- This is acceptable in cheap simulators.

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

Deep RL often needs a very large number of environment interactions.

- This is acceptable in cheap simulators.
- It is problematic in robotics, medicine, science, or real-world control.

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

Deep RL often needs a very large number of environment interactions.

- This is acceptable in cheap simulators.
- It is problematic in robotics, medicine, science, or real-world control.
- We want agents that learn from limited data.

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

Deep RL often needs a very large number of environment interactions.

- This is acceptable in cheap simulators.
- It is problematic in robotics, medicine, science, or real-world control.
- We want agents that learn from limited data.

EfficientZero: a sample-efficient MuZero-style algorithm for visual RL.

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

EfficientZero keeps the MuZero philosophy:

learn a latent world model + plan with MCTS.

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

EfficientZero keeps the MuZero philosophy:

learn a latent world model + plan with MCTS.

But it improves training when data is scarce.

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

EfficientZero keeps the MuZero philosophy:

learn a latent world model + plan with MCTS.

But it improves training when data is scarce.

- Learn more useful latent dynamics

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

EfficientZero keeps the MuZero philosophy:

learn a latent world model + plan with MCTS.

But it improves training when data is scarce.

- Learn more useful latent dynamics
- Improve value consistency across imagined trajectories

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

EfficientZero keeps the MuZero philosophy:

learn a latent world model + plan with MCTS.

But it improves training when data is scarce.

- Learn more useful latent dynamics
- Improve value consistency across imagined trajectories
- Use self-supervised objectives to stabilize representation learning

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

EfficientZero keeps the MuZero philosophy:

learn a latent world model + plan with MCTS.

But it improves training when data is scarce.

- Learn more useful latent dynamics
- Improve value consistency across imagined trajectories
- Use self-supervised objectives to stabilize representation learning
- Reuse limited experience more efficiently

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

EfficientZero keeps the MuZero philosophy:

$$\text{learn a latent world model} \quad + \quad \text{plan with MCTS.}$$

But it improves training when data is scarce.

- Learn more useful latent dynamics
- Improve value consistency across imagined trajectories
- Use self-supervised objectives to stabilize representation learning
- Reuse limited experience more efficiently

Intuition: if real interaction is expensive, learn a model and train as much as possible inside it.

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

Where sample-efficient model-based RL could matter:

- **Atari 100k:** learning from limited game interaction

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

Where sample-efficient model-based RL could matter:

- **Atari 100k:** learning from limited game interaction
- **Robotics:** physical trials are slow and expensive

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

Where sample-efficient model-based RL could matter:

- **Atari 100k:** learning from limited game interaction
- **Robotics:** physical trials are slow and expensive
- **Scientific control:** experiments can be costly

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

Where sample-efficient model-based RL could matter:

- **Atari 100k:** learning from limited game interaction
- **Robotics:** physical trials are slow and expensive
- **Scientific control:** experiments can be costly
- **Healthcare:** real-world trial-and-error is unsafe

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

Where sample-efficient model-based RL could matter:

- **Atari 100k:** learning from limited game interaction
- **Robotics:** physical trials are slow and expensive
- **Scientific control:** experiments can be costly
- **Healthcare:** real-world trial-and-error is unsafe
- **Industrial systems:** downtime and failures are expensive

Mastering Atari Games with Limited Data

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, Yang Gao, 2021

Where sample-efficient model-based RL could matter:

- **Atari 100k:** learning from limited game interaction
- **Robotics:** physical trials are slow and expensive
- **Scientific control:** experiments can be costly
- **Healthcare:** real-world trial-and-error is unsafe
- **Industrial systems:** downtime and failures are expensive

General direction: combine planning, representation learning, and data reuse.

Policy Optimization

Now we switch from search-based RL to **direct policy optimization**.

Policy Optimization

Now we switch from search-based RL to **direct policy optimization**.

A policy $\pi_\theta(a \mid s)$ assigns probabilities to actions.

Policy Optimization

Now we switch from search-based RL to **direct policy optimization**.

A policy $\pi_\theta(a \mid s)$ assigns probabilities to actions.

The basic policy-gradient idea:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[A_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right].$$

Policy Optimization

Now we switch from search-based RL to **direct policy optimization**.

A policy $\pi_\theta(a \mid s)$ assigns probabilities to actions.

The basic policy-gradient idea:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[A_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right].$$

The **advantage** compares an action to the average value of the state:

$$A_t = Q(s_t, a_t) - V(s_t).$$

Policy Optimization

Now we switch from search-based RL to **direct policy optimization**.

A policy $\pi_\theta(a \mid s)$ assigns probabilities to actions.

The basic policy-gradient idea:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[A_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right].$$

The **advantage** compares an action to the average value of the state:

$$A_t = Q(s_t, a_t) - V(s_t).$$

- increase probability of actions with positive advantage
- decrease probability of actions with negative advantage

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

PPO is a popular policy-gradient method.

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

PPO is a popular policy-gradient method.

Problem: policy-gradient updates can be unstable.

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

PPO is a popular policy-gradient method.

Problem: policy-gradient updates can be unstable.

If the policy changes too much after one update, performance can collapse.

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

PPO is a popular policy-gradient method.

Problem: policy-gradient updates can be unstable.

If the policy changes too much after one update, performance can collapse.

PPO idea: improve the policy, but prevent it from moving too far from the policy that generated the data.

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

PPO is a popular policy-gradient method.

Problem: policy-gradient updates can be unstable.

If the policy changes too much after one update, performance can collapse.

PPO idea: improve the policy, but prevent it from moving too far from the policy that generated the data.

This makes PPO simple, robust, and widely used.

PPO is usually implemented as an **actor-critic** method.

- **Actor** $\pi_\theta(a \mid s)$:
  - chooses actions
  - is used at inference time

PPO is usually implemented as an **actor-critic** method.

- **Actor** $\pi_\theta(a \mid s)$:
  - chooses actions
  - is used at inference time
- **Critic** $V_\phi(s)$:
  - estimates expected return
  - helps compute advantages
  - reduces gradient variance

PPO is usually implemented as an **actor-critic** method.

- **Actor** $\pi_\theta(a \mid s)$:
  - chooses actions
  - is used at inference time
- **Critic** $V_\phi(s)$:
  - estimates expected return
  - helps compute advantages
  - reduces gradient variance

Analogy: the actor acts, the critic judges how good the situation is.

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

Define the probability ratio:

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

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

Define the probability ratio:

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

PPO optimizes the objective:

$$\mathcal{L}^{\text{PPO}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right] + \beta \, \mathcal{H}(\pi_\theta(\cdot \mid s_t)) \, .$$

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

Define the probability ratio:

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

PPO optimizes the objective:

$$\mathcal{L}^{\text{PPO}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right] + \beta \, \mathcal{H}(\pi_\theta(\cdot \mid s_t)) \, .$$

- clipping prevents overly large policy updates
- the entropy bonus encourages exploration

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

Define the probability ratio:

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

PPO optimizes the objective:

$$\mathcal{L}^{\text{PPO}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right] + \beta \, \mathcal{H}(\pi_\theta(\cdot \mid s_t)) \, .$$

- clipping prevents overly large policy updates
- the entropy bonus encourages exploration

Intuition: improve the policy conservatively while maintaining exploration.

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

Training loop:

1. Run the current policy in the environment.

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

Training loop:

1. Run the current policy in the environment.
2. Collect trajectories (s_t, a_t, r_t).

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

Training loop:

1. Run the current policy in the environment.
2. Collect trajectories (s_t, a_t, r_t).
3. Estimate returns and advantages.

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

Training loop:

1. Run the current policy in the environment.
2. Collect trajectories (s_t, a_t, r_t).
3. Estimate returns and advantages.
4. Update the actor with the clipped PPO objective.

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

Training loop:

1. Run the current policy in the environment.
2. Collect trajectories (s_t, a_t, r_t).
3. Estimate returns and advantages.
4. Update the actor with the clipped PPO objective.
5. Update the critic to predict returns.

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

Training loop:

1. Run the current policy in the environment.
2. Collect trajectories (s_t, a_t, r_t).
3. Estimate returns and advantages.
4. Update the actor with the clipped PPO objective.
5. Update the critic to predict returns.
6. Repeat.

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017

Training loop:

1. Run the current policy in the environment.
2. Collect trajectories (s_t, a_t, r_t).
3. Estimate returns and advantages.
4. Update the actor with the clipped PPO objective.
5. Update the critic to predict returns.
6. Repeat.

At inference time: usually only the actor is needed.

Training language models to follow instructions with human feedback

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al., 2022

Start from a pretrained language model:

$$\pi_\theta(y \mid x),$$

where:

- x: prompt
- y: generated answer

Training language models to follow instructions with human feedback

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al., 2022

Start from a pretrained language model:

$$\pi_\theta(y \mid x),$$

where:

- x: prompt
- y: generated answer

For a given prompt, humans compare multiple answers:

$$y_w \succ y_l$$

(preferred answer vs rejected answer).

Training language models to follow instructions with human feedback

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al., 2022

Start from a pretrained language model:

$$\pi_\theta(y \mid x),$$

where:

- x: prompt
- y: generated answer

For a given prompt, humans compare multiple answers:

$$y_w \succ y_l$$

(preferred answer vs rejected answer).

Train a reward model:

$$r_\phi(x, y) \in \mathbb{R}$$

that assigns higher scores to preferred outputs.

Training language models to follow instructions with human feedback

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al., 2022

The reward model is trained with:

$$\mathcal{L}_{\text{RM}} = -\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l)) \,.$$

Training language models to follow instructions with human feedback

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al., 2022

The reward model is trained with:

$$\mathcal{L}_{\text{RM}} = -\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l)) \,.$$

This encourages:

$$r_\phi(x, y_w) > r_\phi(x, y_l),$$

meaning preferred answers receive larger rewards.

Training language models to follow instructions with human feedback

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al., 2022

The reward model is trained with:

$$\mathcal{L}_{\text{RM}} = -\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l)) .$$

This encourages:

$$r_\phi(x, y_w) > r_\phi(x, y_l),$$

meaning preferred answers receive larger rewards.

Then PPO optimizes the language model against the learned reward:

$$\max_\theta \mathbb{E}_{y \sim \pi_\theta} [r_\phi(x, y)] .$$

Training language models to follow instructions with human feedback

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al., 2022

The reward model is trained with:

$$\mathcal{L}_{\text{RM}} = -\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l)) .$$

This encourages:

$$r_\phi(x, y_w) > r_\phi(x, y_l),$$

meaning preferred answers receive larger rewards.

Then PPO optimizes the language model against the learned reward:

$$\max_\theta \mathbb{E}_{y \sim \pi_\theta} [r_\phi(x, y)] .$$

Important distinction: the reward is learned from human preferences, not directly provided by the environment.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, Daya Guo, 2024

GRPO is a PPO-style method designed for LLM reasoning.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, Daya Guo, 2024

GRPO is a PPO-style method designed for LLM reasoning.

For a prompt x, sample a group of outputs: $y_1, \ldots, y_G \sim \pi_\theta(\cdot \mid x)$.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, Daya Guo, 2024

GRPO is a PPO-style method designed for LLM reasoning.

For a prompt x, sample a group of outputs: $y_1, \ldots, y_G \sim \pi_\theta(\cdot \mid x)$.

Compute rewards for each output: $r_i = r(x, y_i)$.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, Daya Guo, 2024

GRPO is a PPO-style method designed for LLM reasoning.

For a prompt x, sample a group of outputs: $y_1, \ldots, y_G \sim \pi_\theta(\cdot \mid x)$.

Compute rewards for each output: $r_i = r(x, y_i)$.

Instead of learning a critic, GRPO normalizes rewards inside the group:

$$A_i = \frac{r_i - \text{mean}(r)}{\text{std}(r)}.$$

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, Daya Guo, 2024

GRPO is a PPO-style method designed for LLM reasoning.

For a prompt x, sample a group of outputs: $y_1, \ldots, y_G \sim \pi_\theta(\cdot \mid x)$.

Compute rewards for each output: $r_i = r(x, y_i)$.

Instead of learning a critic, GRPO normalizes rewards inside the group:

$$A_i = \frac{r_i - \text{mean}(r)}{\text{std}(r)}.$$

These normalized rewards are directly used as advantages.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, Daya Guo, 2024

GRPO then optimizes a PPO-style objective:

$$\nabla_\theta J(\theta) \approx \mathbb{E}\left[A_i \nabla_\theta \log \pi_\theta(y_i \mid x)\right].$$

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, Daya Guo, 2024

GRPO then optimizes a PPO-style objective:

$$\nabla_\theta J(\theta) \approx \mathbb{E}\left[A_i \nabla_\theta \log \pi_\theta(y_i \mid x)\right].$$

Main difference from PPO: there is no learned critic:

$$V_\phi(s).$$

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, Daya Guo, 2024

GRPO then optimizes a PPO-style objective:

$$\nabla_\theta J(\theta) \approx \mathbb{E}\left[A_i \nabla_\theta \log \pi_\theta(y_i \mid x)\right].$$

Main difference from PPO: there is no learned critic:

$$V_\phi(s).$$

Advantages are computed from relative rewards inside the sampled group instead.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, Daya Guo, 2024

GRPO then optimizes a PPO-style objective:

$$\nabla_\theta J(\theta) \approx \mathbb{E}\left[A_i \nabla_\theta \log \pi_\theta(y_i \mid x)\right].$$

Main difference from PPO: there is no learned critic:

$$V_\phi(s).$$

Advantages are computed from relative rewards inside the sampled group instead.

Why useful?

- avoids training a large value model
- reduces memory/computation cost
- relative ranking is often easier than estimating absolute value

Challenges in Reinforcement Learning

RL is powerful, but difficult.

- **Sample inefficiency:** many interactions may be needed

Challenges in Reinforcement Learning

RL is powerful, but difficult.

- **Sample inefficiency:** many interactions may be needed
- **Exploration:** the agent must discover useful behavior

Challenges in Reinforcement Learning

RL is powerful, but difficult.

- **Sample inefficiency:** many interactions may be needed
- **Exploration:** the agent must discover useful behavior
- **Instability:** training data changes as the policy changes

Challenges in Reinforcement Learning

RL is powerful, but difficult.

- **Sample inefficiency:** many interactions may be needed
- **Exploration:** the agent must discover useful behavior
- **Instability:** training data changes as the policy changes
- **Reward hacking:** optimizing the reward may not optimize what we want

Challenges in Reinforcement Learning

RL is powerful, but difficult.

- **Sample inefficiency:** many interactions may be needed
- **Exploration:** the agent must discover useful behavior
- **Instability:** training data changes as the policy changes
- **Reward hacking:** optimizing the reward may not optimize what we want
- **Long horizons:** delayed consequences are hard to assign credit to

Challenges in Reinforcement Learning

RL is powerful, but difficult.

- **Sample inefficiency:** many interactions may be needed
- **Exploration:** the agent must discover useful behavior
- **Instability:** training data changes as the policy changes
- **Reward hacking:** optimizing the reward may not optimize what we want
- **Long horizons:** delayed consequences are hard to assign credit to
- **Sim-to-real gap:** policies trained in simulation may fail in reality

Current Directions

Some active directions:

- **World models:** learn dynamics and plan in latent space

Current Directions

Some active directions:

- **World models:** learn dynamics and plan in latent space
- **Offline RL:** learn from fixed datasets without new interaction

Current Directions

Some active directions:

- **World models:** learn dynamics and plan in latent space
- **Offline RL:** learn from fixed datasets without new interaction
- **RL for LLMs:** preference optimization and reasoning

Current Directions

Some active directions:

- **World models:** learn dynamics and plan in latent space
- **Offline RL:** learn from fixed datasets without new interaction
- **RL for LLMs:** preference optimization and reasoning
- **Test-time search:** spend more computation when solving a task

Current Directions

Some active directions:

- **World models:** learn dynamics and plan in latent space
- **Offline RL:** learn from fixed datasets without new interaction
- **RL for LLMs:** preference optimization and reasoning
- **Test-time search:** spend more computation when solving a task
- **Policy distillation:** compress expensive policies or search procedures into cheaper models

Conclusion

Modern RL is not a single algorithm.

Conclusion

Modern RL is not a single algorithm.

Main message: different RL methods make different trade-offs:

- search vs direct policy optimization
- sample efficiency vs scalability
- planning vs fast inference
- learned world models vs direct interaction

Conclusion

Modern RL is not a single algorithm.

Main message: different RL methods make different trade-offs:

- search vs direct policy optimization
- sample efficiency vs scalability
- planning vs fast inference
- learned world models vs direct interaction

There is currently no single general-purpose RL algorithm that dominates all settings.

Conclusion

Modern RL is not a single algorithm.

Main message: different RL methods make different trade-offs:

- search vs direct policy optimization
- sample efficiency vs scalability
- planning vs fast inference
- learned world models vs direct interaction

There is currently no single general-purpose RL algorithm that dominates all settings.

Modern systems increasingly combine:

$$\text{learning} + \text{search} + \text{world models} + \text{test-time computation}.$$