Machine Learning

Lecture 18: Reinforcement Learning

Fall 2023

Instructor: Xiaodong Gu

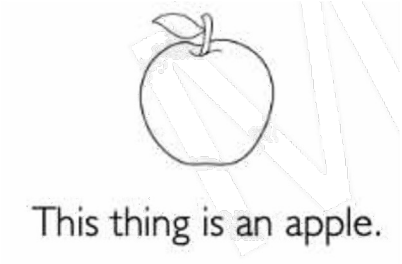
What we have learned so far?

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn to map
$x \rightarrow y$

Examples: Classification, regression, object detection, semantic segmentation, etc.

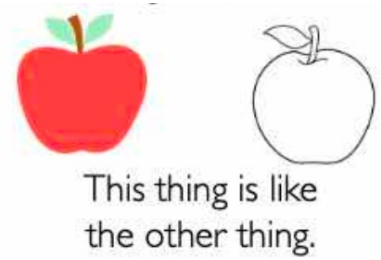


Unsupervised Learning

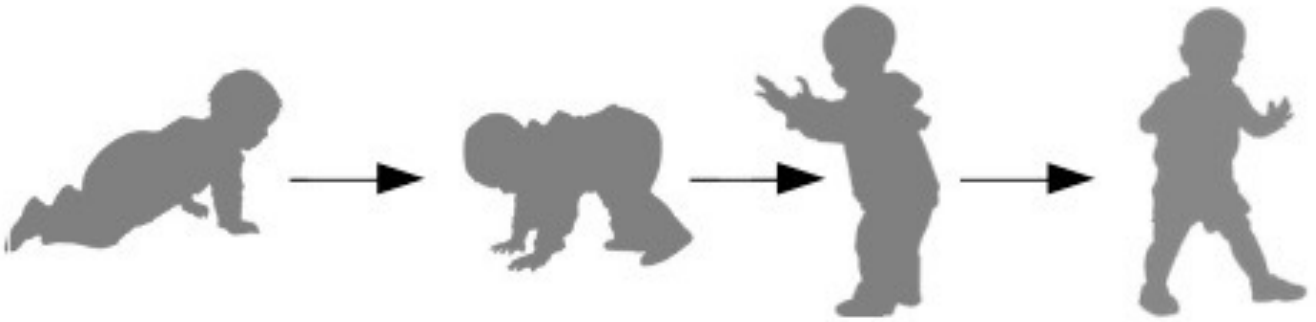
Data: x
x is data, no label

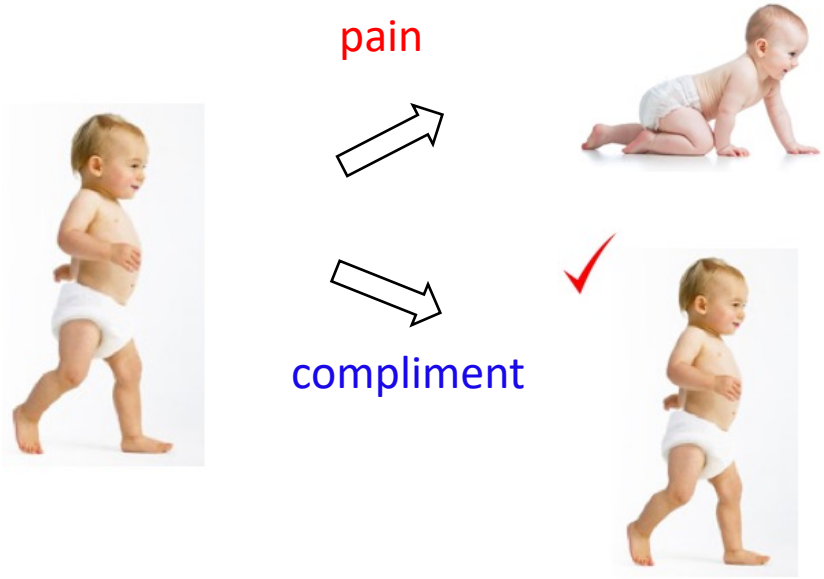
Goal: Learn underline structure.

Examples: Clustering, dimensionality reduction, probability estimation, etc.



Learning in Dynamic Environments





Example: Games, Robots, the World

The Third Class of Machine Learning

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn to map
$x \rightarrow y$

Examples:
Classification, regression, etc.

Unsupervised Learning

Data: x
x is data, no label

Goal: Learn underline structure.

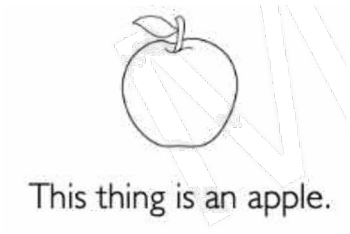
Examples:
Clustering, dim reduction, etc.

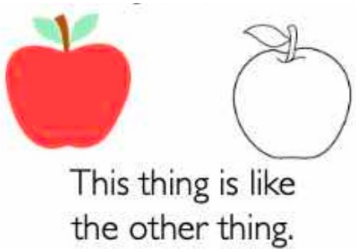
Reinforcement Learning

Data: *state-action* pairs + rewards

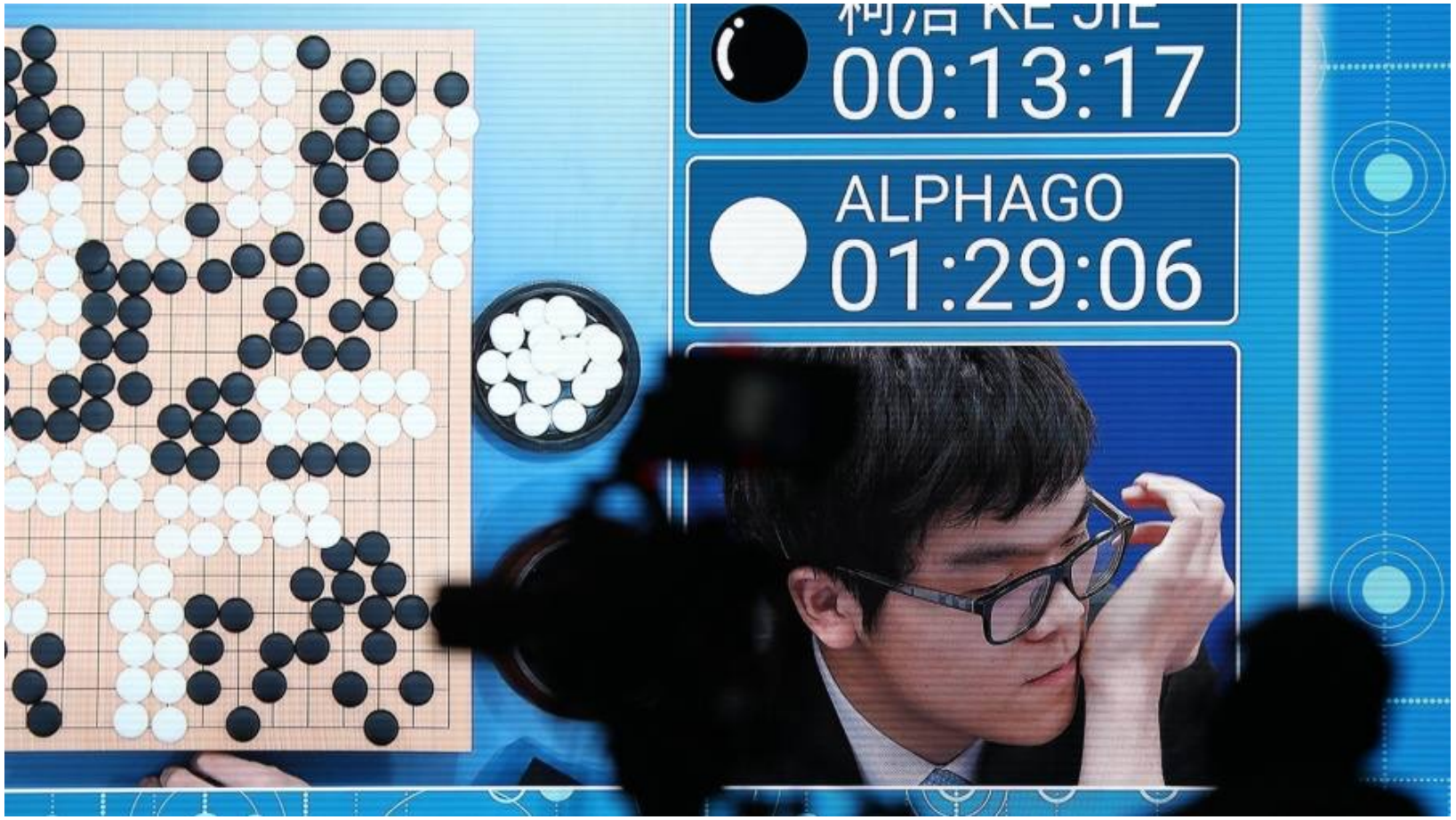
Goal: Learn from interacting with environment.

Examples: Robot, Game, etc.



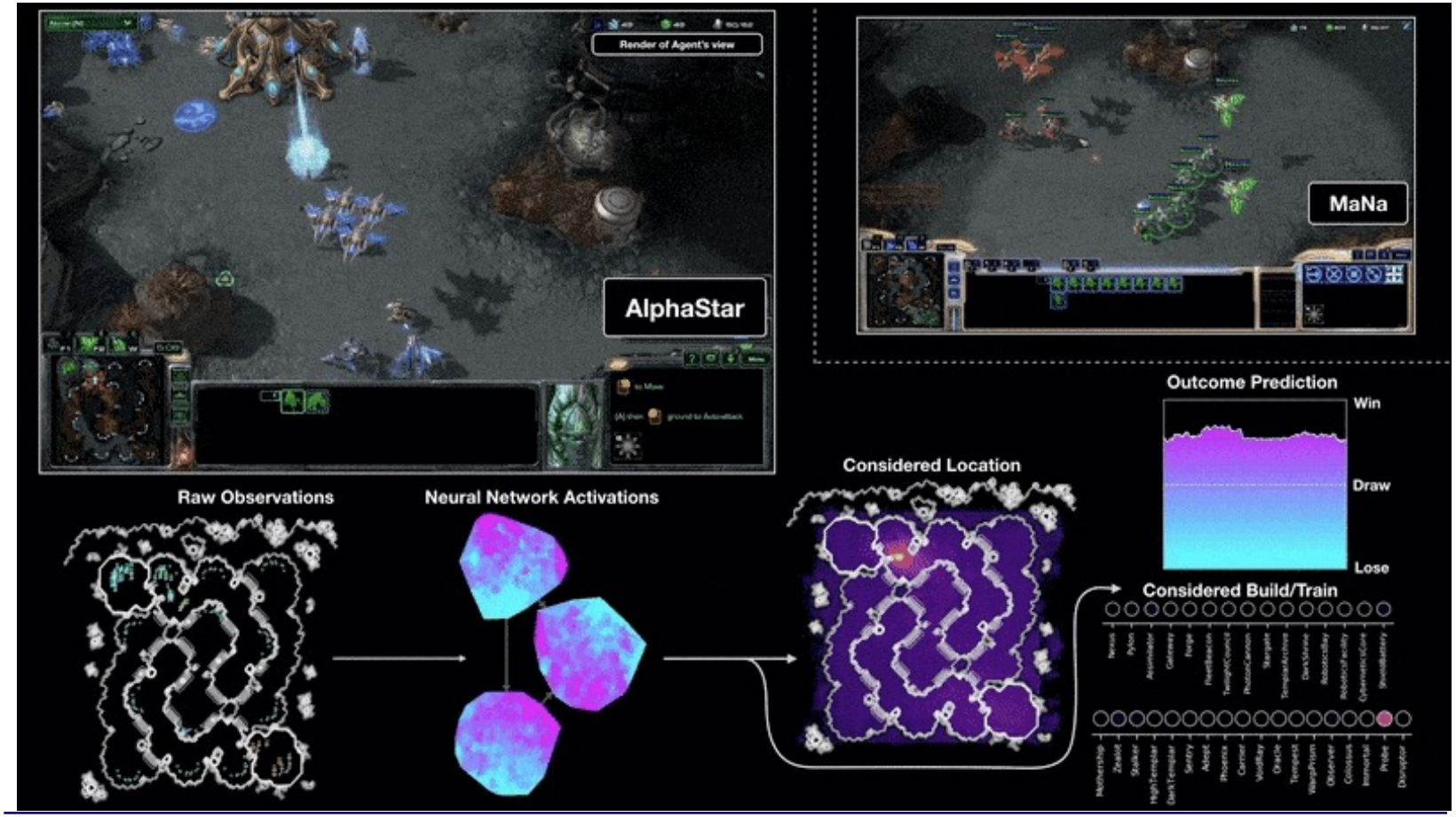


AlphaGo



AlphaStar

AI Player for StarCraft II designed by DeepMind.



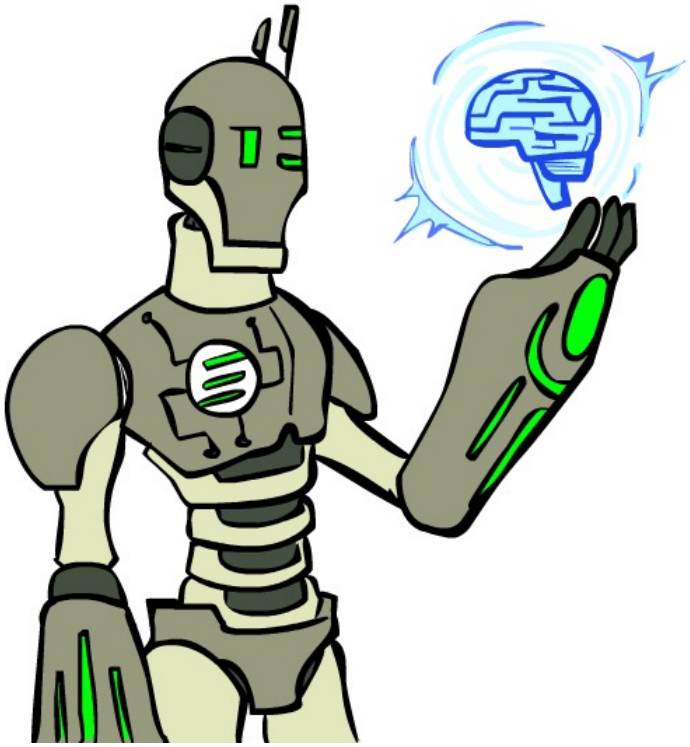
Atari Breakout

RL hacks Atari !

Today

Reinforcement Learning

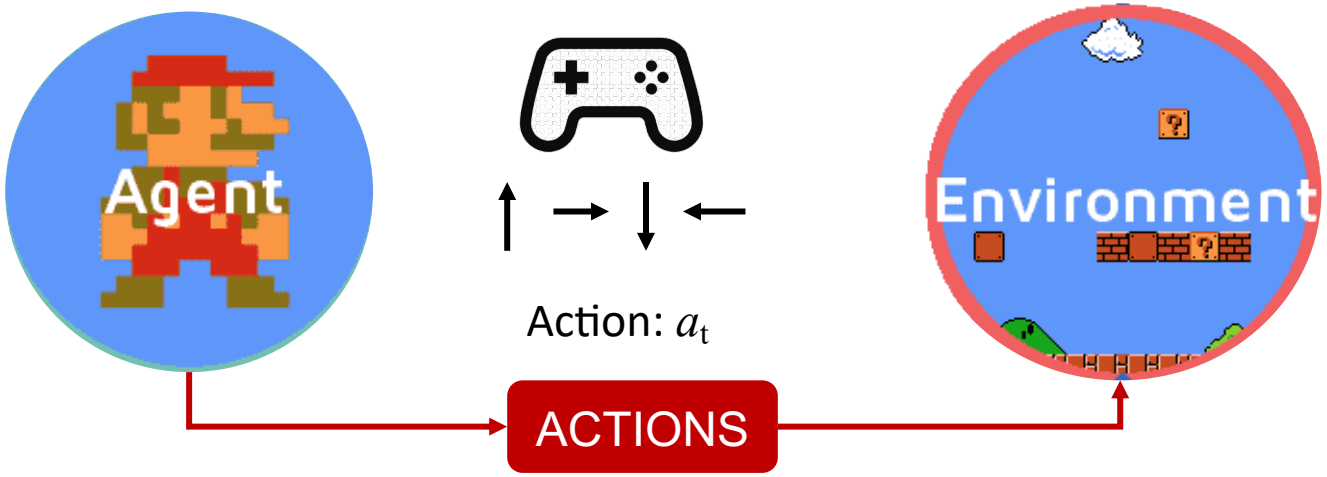
- Concept
- Deep Q Learning
- Policy Gradient
- Actor-Critic



Reinforcement Learning: Key Concepts

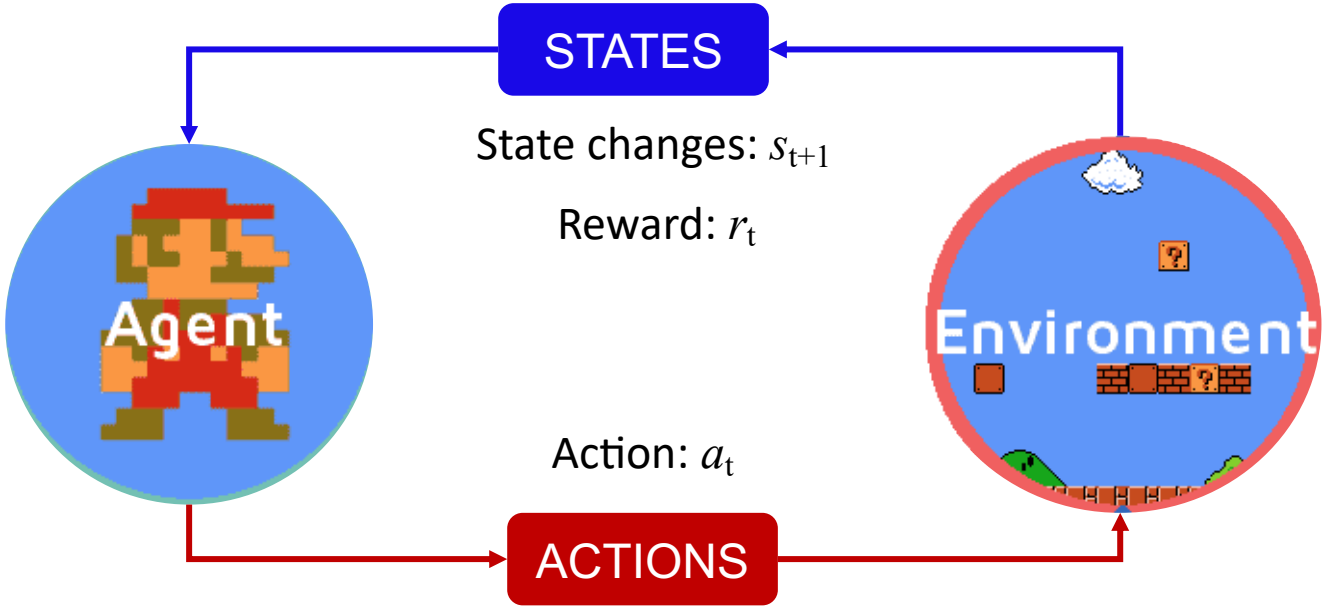
Agent: an actor which takes actions and learns knowledge
Environment: the world in which the agent exists and operates.

Reinforcement Learning: Key Concepts



Action: a move the agent can make in the environment.
Action space A: the set of possible actions an agent can make in the environment.

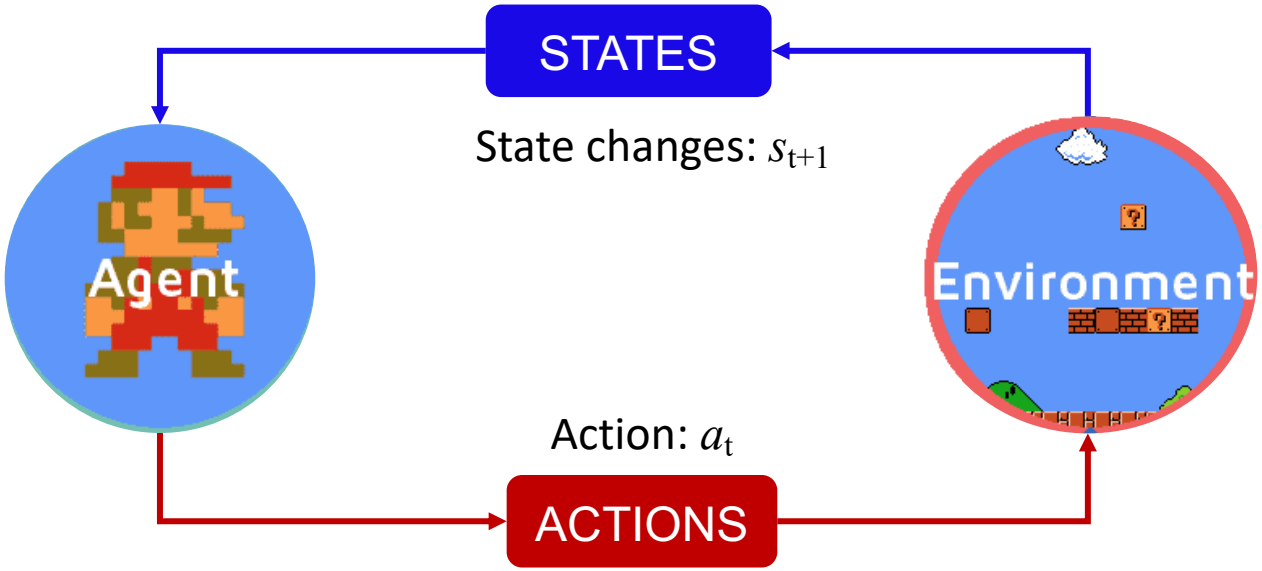
Reinforcement Learning: Key Concepts



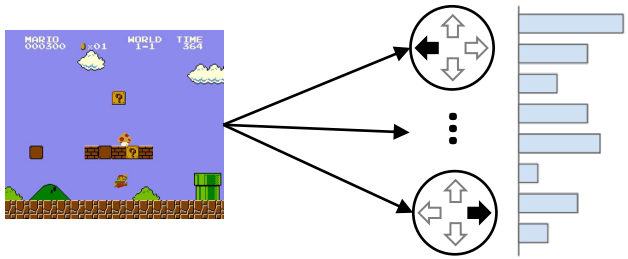
State: observations

Reward: feedback that scores the agent's action.

Reinforcement Learning: Key Concepts

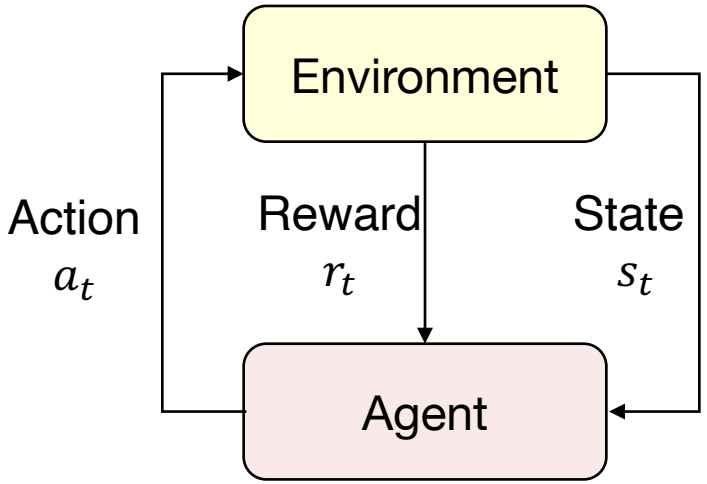


Policy π: $S \rightarrow A$: a function that maps from state to action



Reinforcement Learning Problem

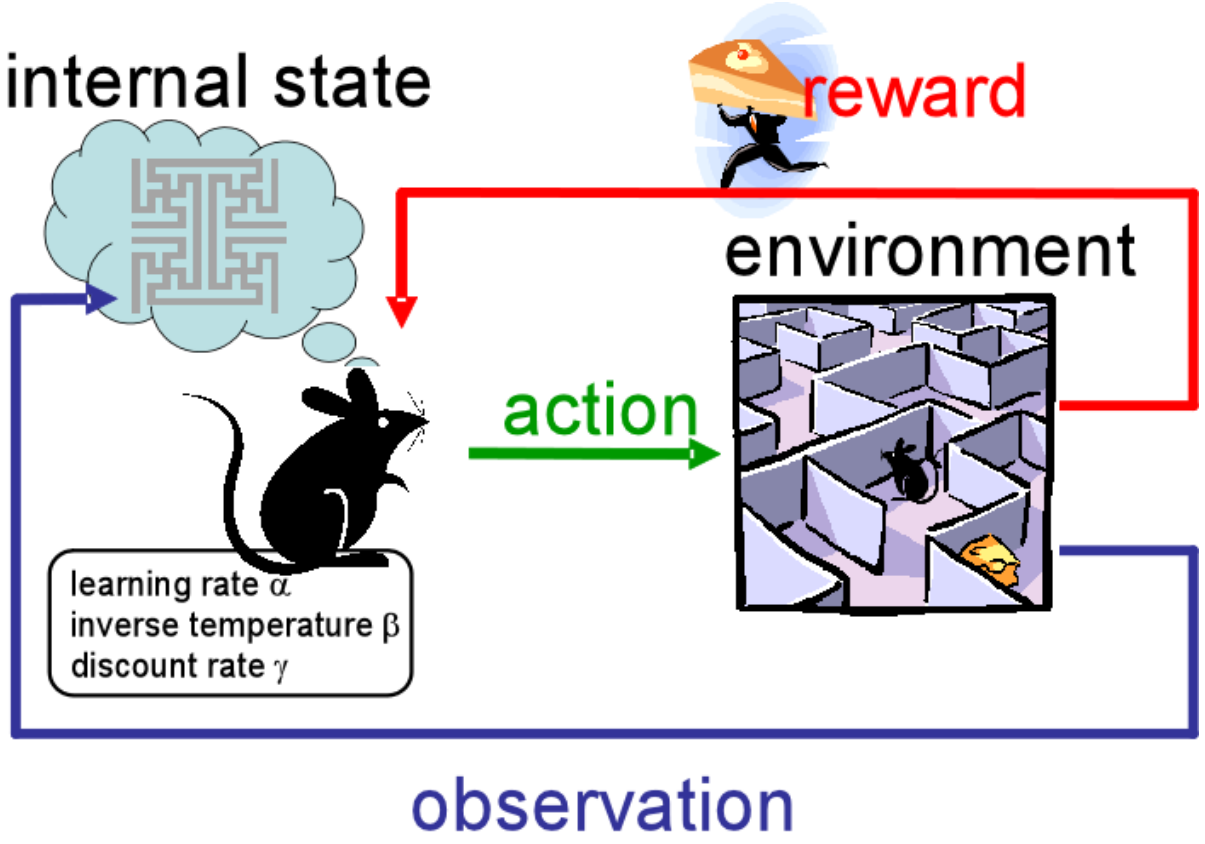
- Learning from **interacting** with an **environment.**



- a set of environment states S;
- a set of actions A;
- rules of transitioning between states;
- rules that determine the scalar immediate reward of a transition;
- rules that describe what the agent observes.

Objective: learning a policy (i.e., state-action mapping) that maximizes the long-term payoff.

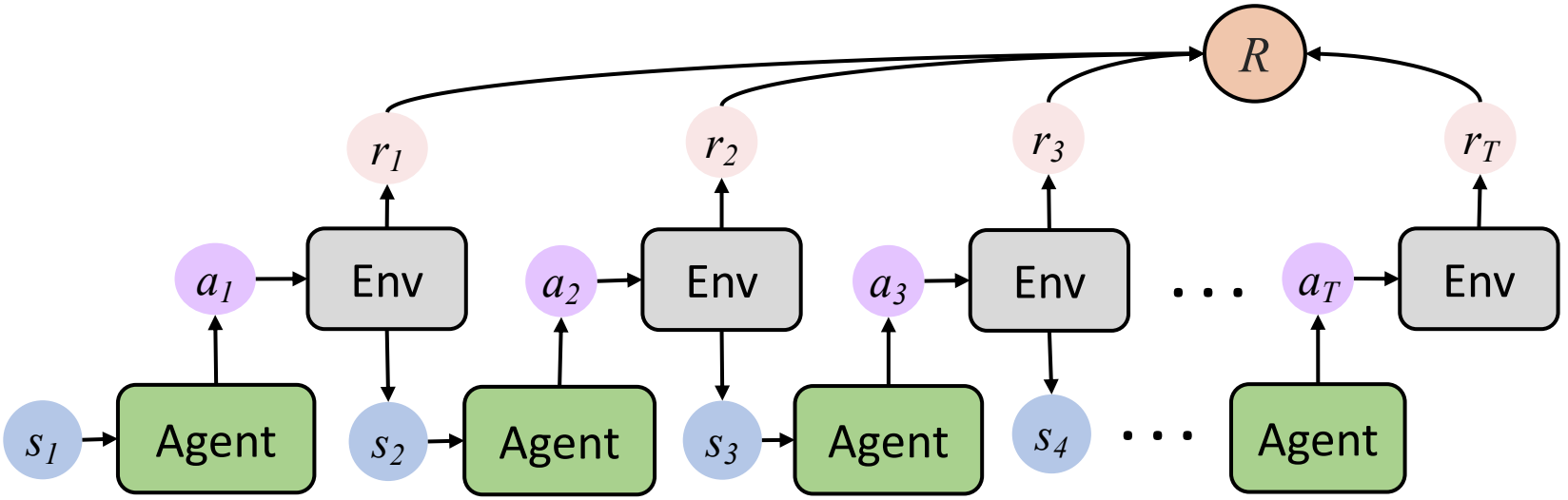
Example


internal state
reward
environment
action
learning rate α
inverse temperature β
discount rate γ
observation

Reinforcement Learning Process

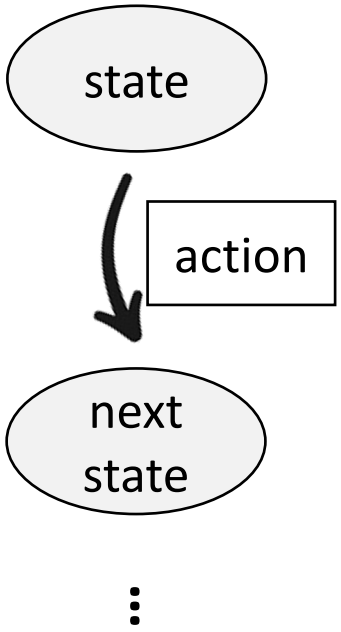
In the view of actions

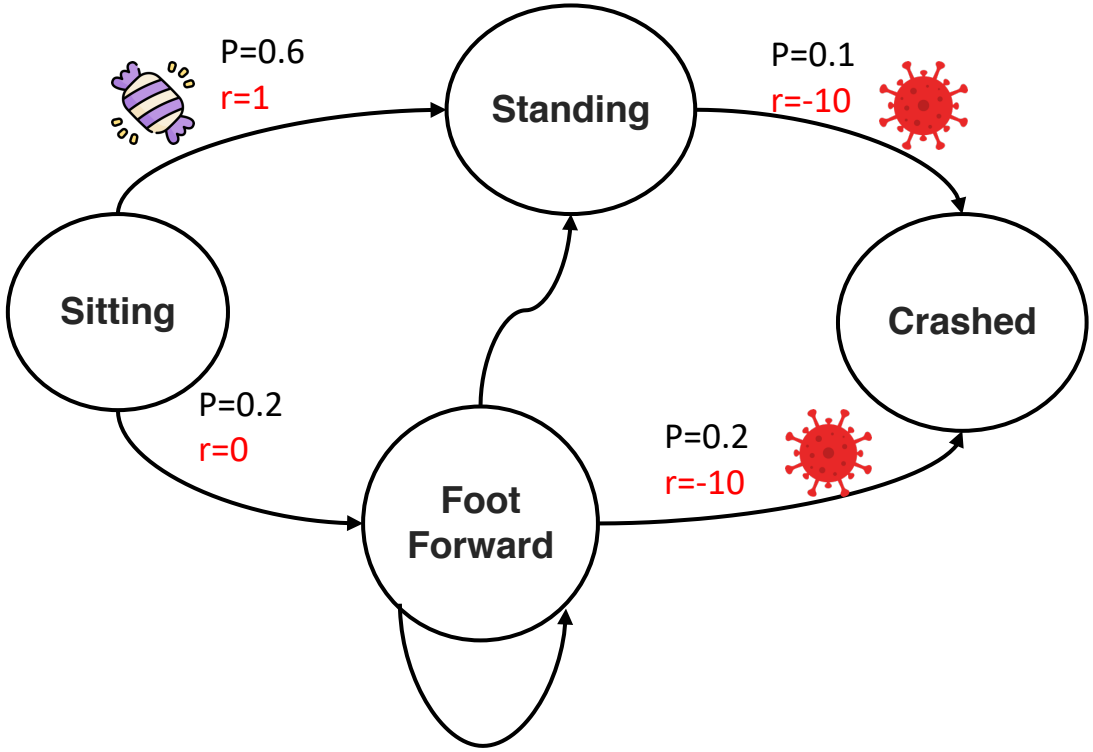
Long-term return: the sum of all rewards obtained from time t.



Reinforcement Learning Process

In the view of states

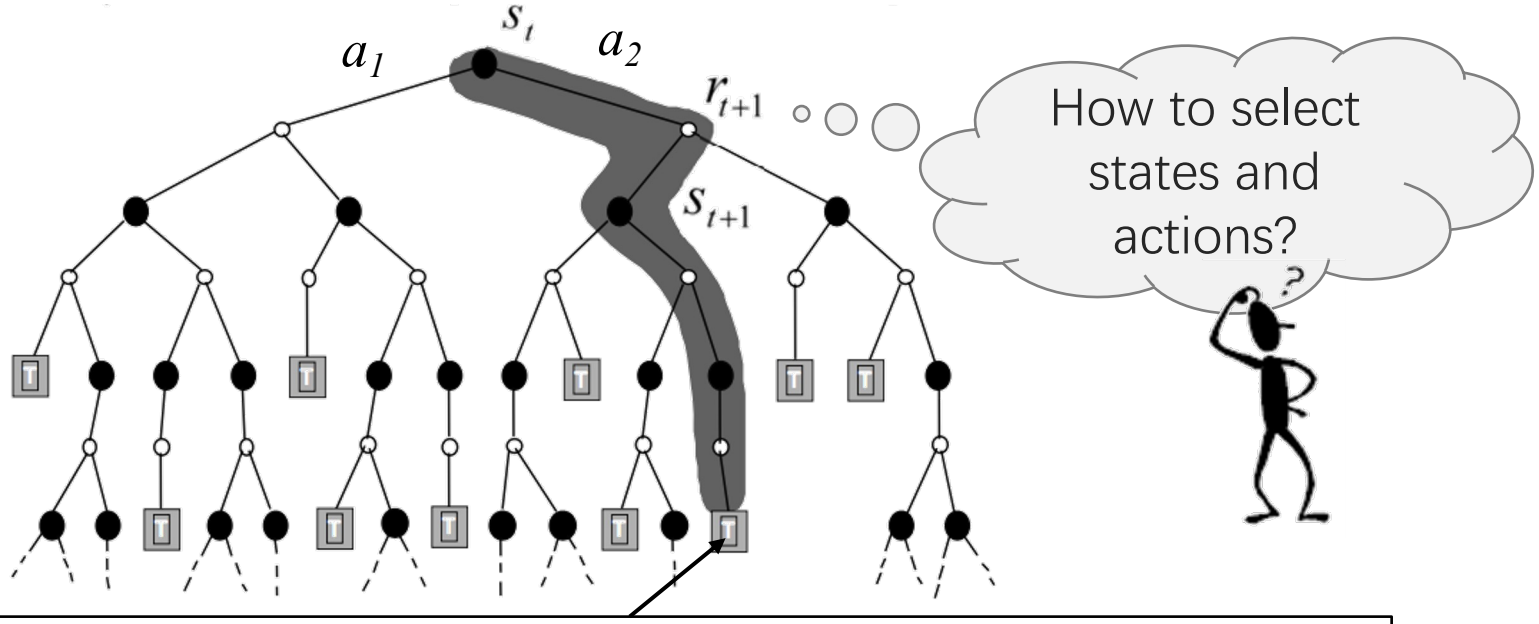




Markov Decision Process (MDP)

In the View of State Space

- The agent finds a path in the state space that potentially leads to the maximum long-term return $\boldsymbol{R}_t$.



Actual long-term return following s_t: $\boldsymbol{R}_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} + \ldots$

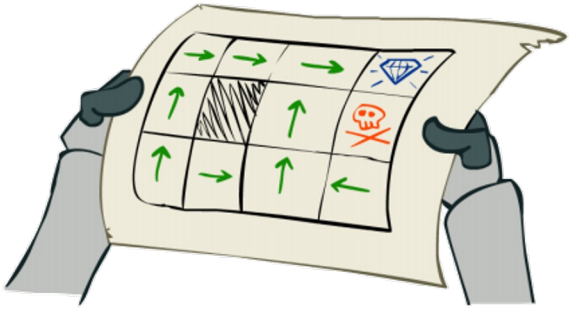
Discounted long-term return: $\boldsymbol{R}_t = \sum_{i=t}^{\infty} \gamma^i r_i = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$

Scoring (s, a) pairs: Q-function

- The **Q-function** captures the **expected total future reward** an agent in state s can receive by executing a certain action a

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

score(state, action)



$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i = r_t + \gamma r_{t+1} \ldots + \gamma^n r_{t+n} + \ldots$$

$$\begin{aligned} Q(s_t, a_t) &= \mathbb{E}[R_t | s_t, a_t] \\ &= \mathbb{E}[r_t + \gamma r_{t+1} \ldots + \gamma^n r_{t+n} + \ldots] \\ &= r_t + \gamma\, \mathbb{E}(r_{t+1} \ldots + \gamma^{n-1} r_{t+n} + \ldots)] \\ &= r_t + \gamma\, Q(s_{t+1}, a_{t+1}) \end{aligned}$$

Recursion

Derive the Policy

- How to take actions given a Q-function?

$$Q(s_t, a_t) = \mathbb{E}[R_t \,|s_t, a_t]$$

- Having obtained the Q-values, the agent derives the optimal **policy π(s)**, to infer the **best action to take** at state s

$$\pi^\star(s) = \arg\max_a Q(s, a)$$

The policy should choose an action that maximizes future reward.

Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

$a = \text{argmax}_a Q(s, a)$

Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

$a = \arg\max_a Q(s, a)$

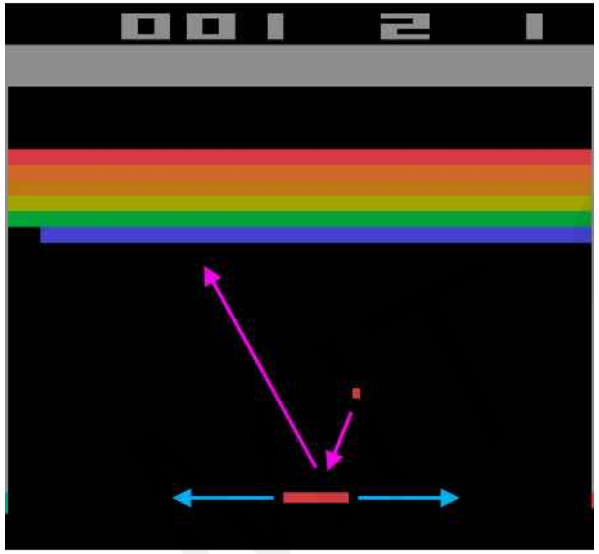
Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Value Learning (Q-Learning)

- Learn the Q-score for each <*s*, *a*> pair.



Filling in a lookup table of all Q-scores

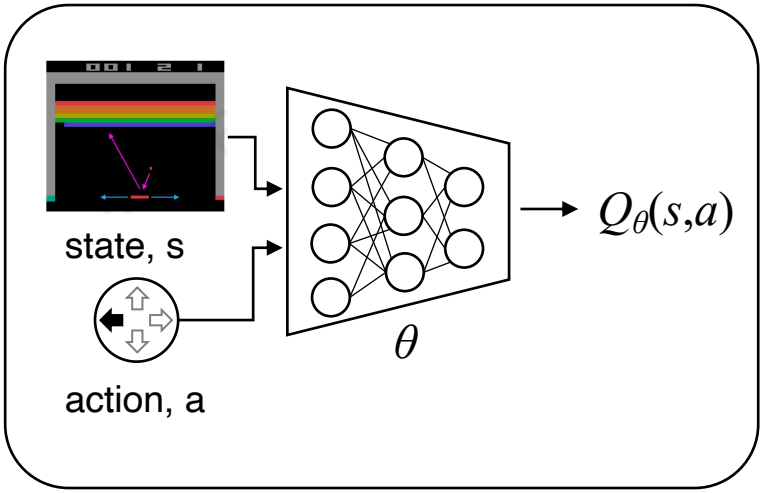
State			Action	Q(s,a)	
x	y	dir			
0.5	0.4	90	←	15	
0.5	0.4	90	→	34	√
0.7	0.9	25	→	12	
0.7	0.9	25	←	22	√
0.2	0.4	108	→	58	√
0.2	0.4	108	←	38	
…	…	…	…	…	

which (state, action) pair has the best Q-score?

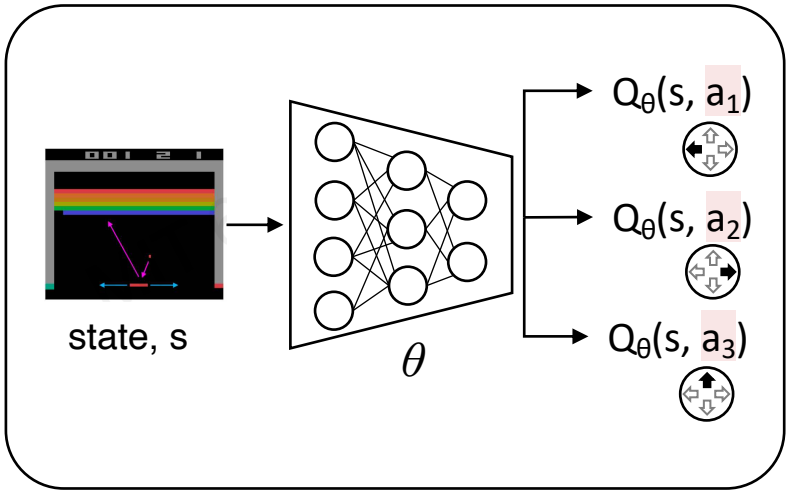
Problem: The space of all <s,a> pairs is huge and intractable!

Deep Q Networks (DQN)

- **Idea:** use deep neural networks to model the Q-function



Action + State → Expected return

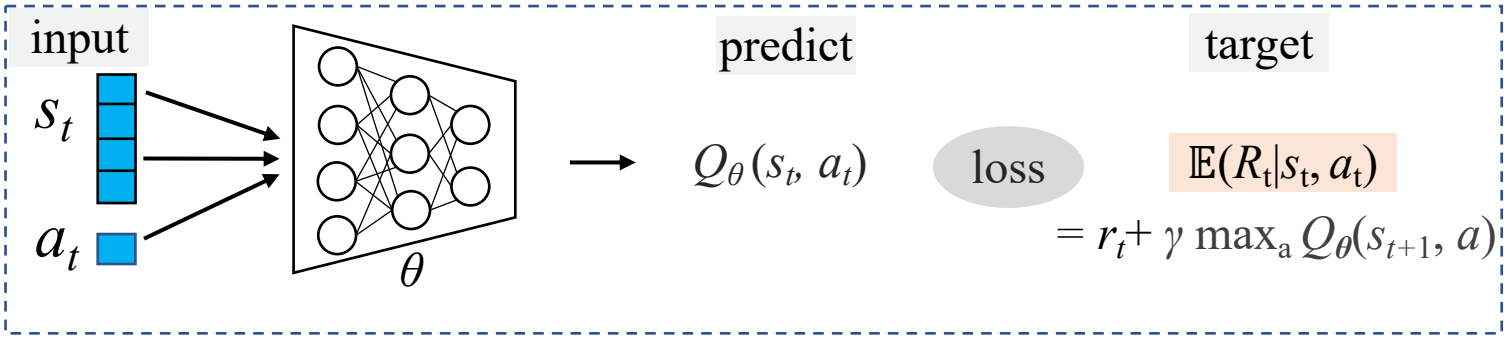


State → Expected return for each action

What happens if we take all the best actions?
Maximize the target return → Train the agent

Deep Q Networks (DQN)

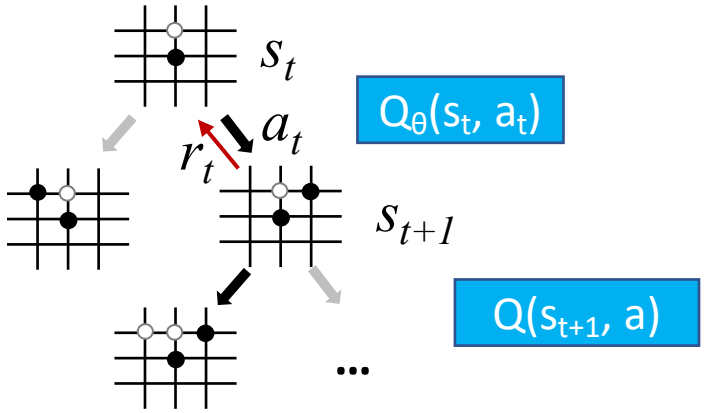
Use deep neural networks (e.g., CNN) to model the Q-function.



- Train:
  - estimate the parameters θ in the $Q_\theta(s, a)$ network using agent play experiences.
- Test:
  - calculate $Q_\theta(s, a)$ for all a's under s and choose $a = \text{argmax}_a Q_\theta(s, a)$.

Training DQN

- **Episode**: run agent and obtain $(s_1, a_1, r_1, s_2, a_2, r_2, \ldots s_T, a_T, r_T)$

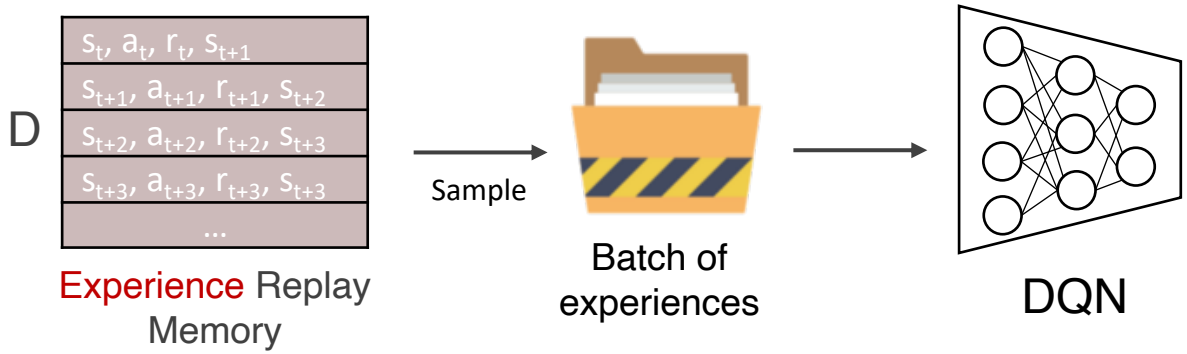


- For any given input s_t and a_t, the model estimates a score $Q_\theta(s_t, a_t)$. Let r_t be the reward and s_{t+1} be the next state. The expected **maximum** long-term reward is: $r_t + \gamma \max_{a \in A} Q(s_{t+1}, a)$.
- Our goal is to minimize their difference, i.e., the MSE loss:

$$\ell(\boldsymbol{\theta} \mid s_t, a_t, r_t) = \| (\underbrace{r_t + \gamma \max_a Q_{\boldsymbol{\theta}}(s_{t+1}, a)}_{\text{target}} - \underbrace{Q_{\boldsymbol{\theta}}(s_t, a_t)}_{\text{predicted}}) \|^2$$

Training DQN

- **Experience Replay**: run agent multiple episodes and build a dataset using agent's own experiences: $D = \{(s^{(1)}, a^{(1)}, r^{(1)}, s'^{(1)}), \ldots, (s^{(N)}, a^{(N)}, r^{(N)}, s'^{(N)})\}$



- For multiple $(s, a, r, s') \in D$:

$$L(\boldsymbol{\theta} \mid D) = \mathbb{E}_{(s,a,r,s')\sim D}[\| (r + \gamma \max_{a'} Q_{\theta}(s', a') - Q_{\theta}(s, a) \|^2]$$

mini-batch experience replay

Maximum possible Q-value for the next state (=Q_target)

Current predicted Q-value

Training DQN

$$L(\boldsymbol{\theta} \mid D) = \mathrm{E}_{(s,a,r,s')\sim D}[\| (r + \gamma \max_{a'} Q(s',a'; \theta) - Q(s, a; \theta)\|^2]$$

- Gradients:

$$\nabla_\theta L = \mathbb{E}_{(s,a,r,s')\sim D}\left(r + \gamma \max_{a'} Q(s', a'; \theta) - \boldsymbol{Q}(s, a; \theta)\right)\nabla_\theta \boldsymbol{Q}(s, a; \theta)$$

Q-target

Problem: non-stationary target?

- **Solution**: use some old, fixed parameters θ_{old} as a fixed Q-target (update every C steps)

$$\nabla_{\theta t} L = \mathbb{E}_{(s,a,r,s')\sim D}(r + \gamma \max_{a'} \boldsymbol{Q}(s',a'; \theta_{\text{old}}) - \boldsymbol{Q}(s, a;\theta t))\nabla_{\theta_t} \boldsymbol{Q}(s, a;\theta_t)$$

fixed, old Q-target

The Deep Q-Learning Algorithm

Algorithm 1 Deep Q-learning with Experience Replay

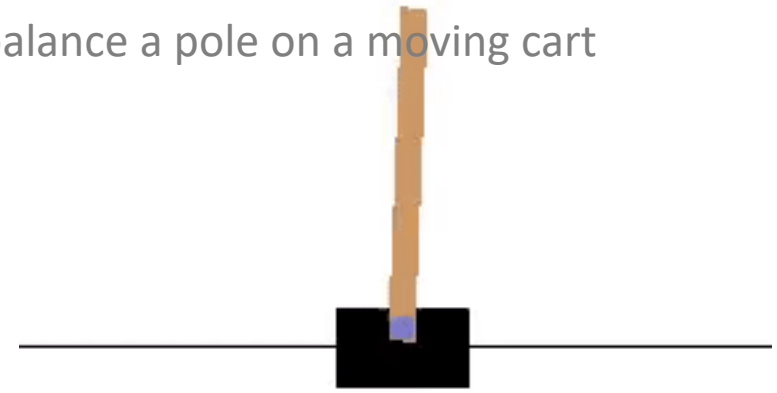
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for *episode* = 1,…, *M*
  Initialize state s_t
  for $t = 1, \ldots, T$
    With probability **ε** select a random action a_t — Sampling
    otherwise select $a_t = \max_a Q^*(s_t, a; \theta)$
    Execute action a_t and observe reward r_t and state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    $s_t \leftarrow s_{t+1}$
    Sample random minibatch of transitions (s_t, a_t, r_t, s_{t+1}) from D
    Set $y_j = r_j + \gamma \max_{a'} Q(s_{t+1}, a'; \theta)$ **if** s_{t+1} is non-terminal **else** r_j
    Perform a gradient descent step on $(y_j - Q(s_t, a_j; \theta))^2$ — Training
  end for
end for

Time for Coding

https://www.kaggle.com/code/dsxavier/dqn-openai-gym-cartpole-with-pytorch

CartPole

balance a pole on a moving cart



Reinforcement Learning Algorithms

Value Learning

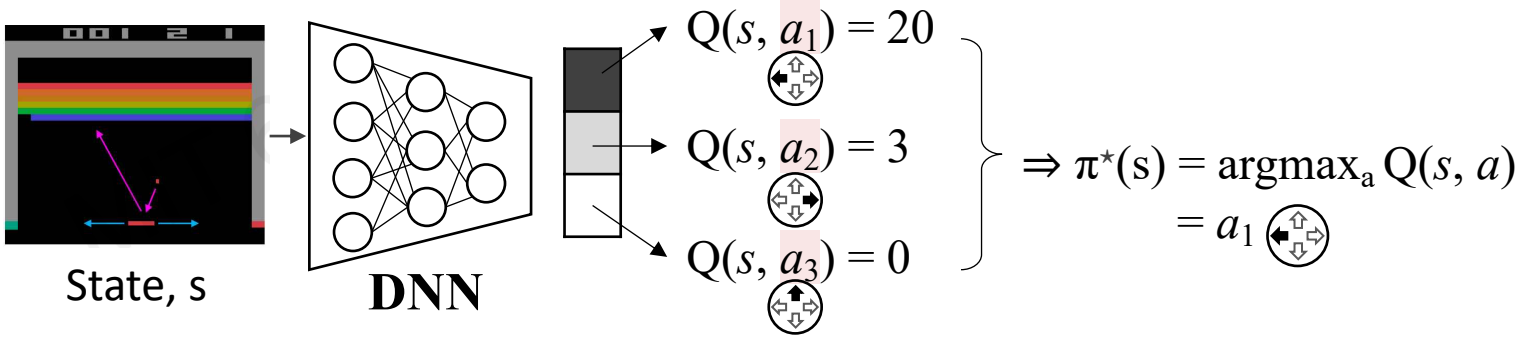
Find $Q(s, a)$
$a = \text{argmax}_a Q(s, a)$

Policy Learning

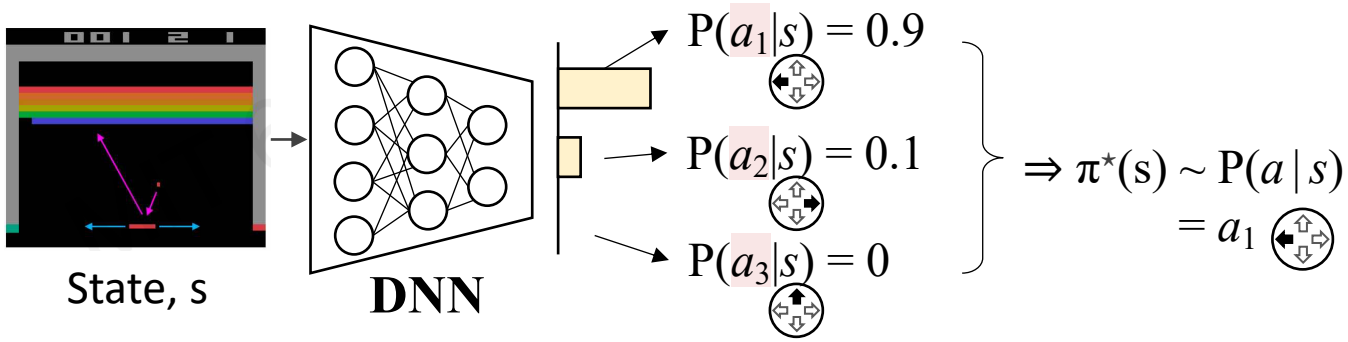
Find $\pi(s)$
Sample $a \sim \pi(s)$

Policy Gradient: Key Idea

DQN: Approximate a Q-function and use it to infer the optimal policy $\pi^*(s)$

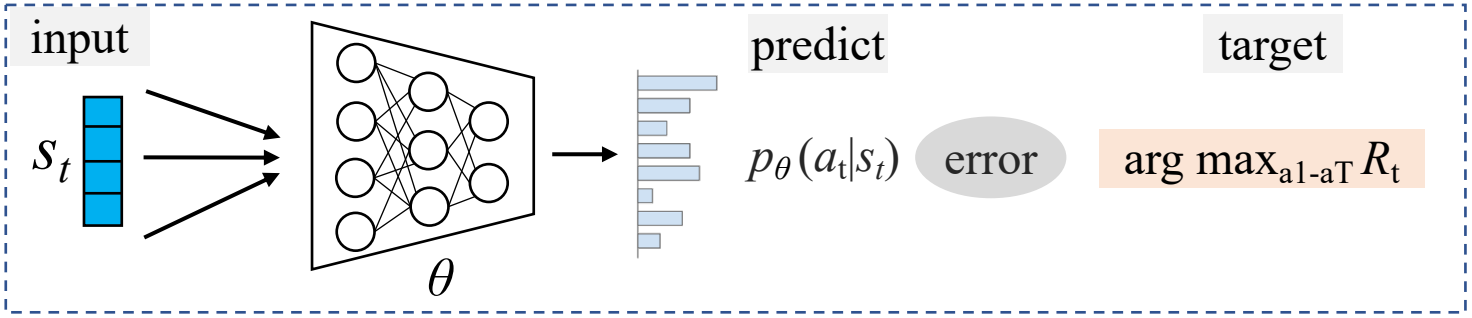


Policy Gradient: directly optimize the policy $\pi(s)$



Policy Network

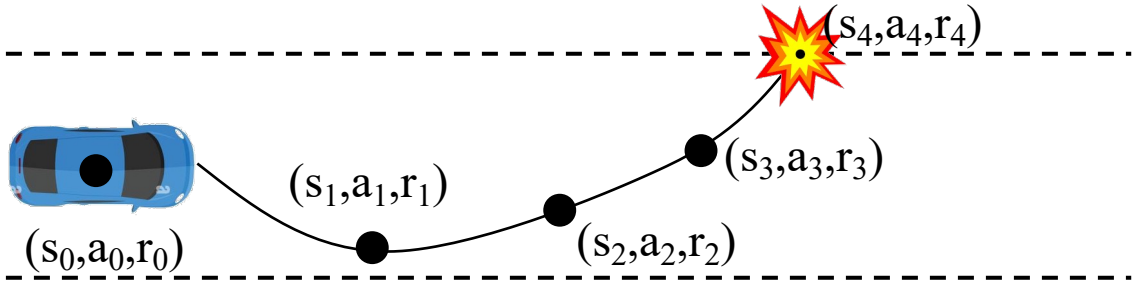
- Use deep neural networks to model the **policy** π_θ.



- Train:
  - Estimate the parameters θ in the $p_\theta(a|s)$ network using agent play experiences.
- Test:
  - choose $a \sim P_\theta(a \mid s)$.

Data and Learning Objective

- **Episode**: run a policy π_θ until termination, record the trajectory $\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_T, a_T, r_T)$.



- τ is stochastic with a probability

Trainable

$$p_\theta(\tau)=p(s_1)p_\theta(a_1|s_1)p(s_2|s_1, a_1)p_\theta(a_2|s_2)p(s_3|s_2,a_2)\ldots$$

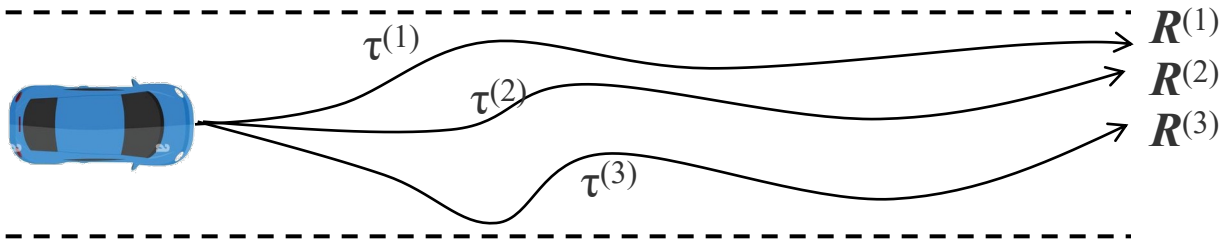
deterministic and assigned by the environment

- The reward for τ is:

$$\boldsymbol{R}(\tau) = r_t+\gamma r_{t+1}+\gamma^2 r_{t+2}+\ldots \gamma^{T-t} r_T$$

Data and Learning Objective

- Run the agent for multiple episodes under a policy π_θ, and sample multiple trajectories $\tau^{(1)}, \tau^{(2)}, \ldots, \tau^{(N)} \sim p_\theta(a|s)$



- Let $\boldsymbol{R}_\theta = \mathbb{E}_{\tau\sim p(\tau)}\boldsymbol{R}(\tau) = \Sigma_\tau \boldsymbol{R}(\tau) p_\theta(\tau)$ be the expected reward from the environment following the policy, our goal is to find θ that maximize $\boldsymbol{R}_\theta$.

$$\nabla_\theta \boldsymbol{R}_\theta = \Sigma_\tau R(\tau)\nabla p_\theta(\tau) = \sum_\tau R(\tau)\, p_\theta(\tau)\, \frac{\nabla p_\theta(\tau)}{p_\theta(\tau)}$$

$$= \sum_\tau R(\tau)\, p_\theta(\tau)\nabla \log p_\theta(\tau) = \mathbb{E}_{\tau\sim p(\tau)}[R(\tau)\nabla \log p_\theta(\tau)\,]$$

$$\approx \frac{1}{N}\sum_{l=1}^{N} R^{(l)}\nabla \log p_\theta(\tau^{(l)}) = \frac{1}{N}\sum_{l=1}^{N}\sum_{t=1}^{T} R^{(l)}\nabla \log p_\theta(a_t^{(l)}\,|\,s_t^{(l)})$$

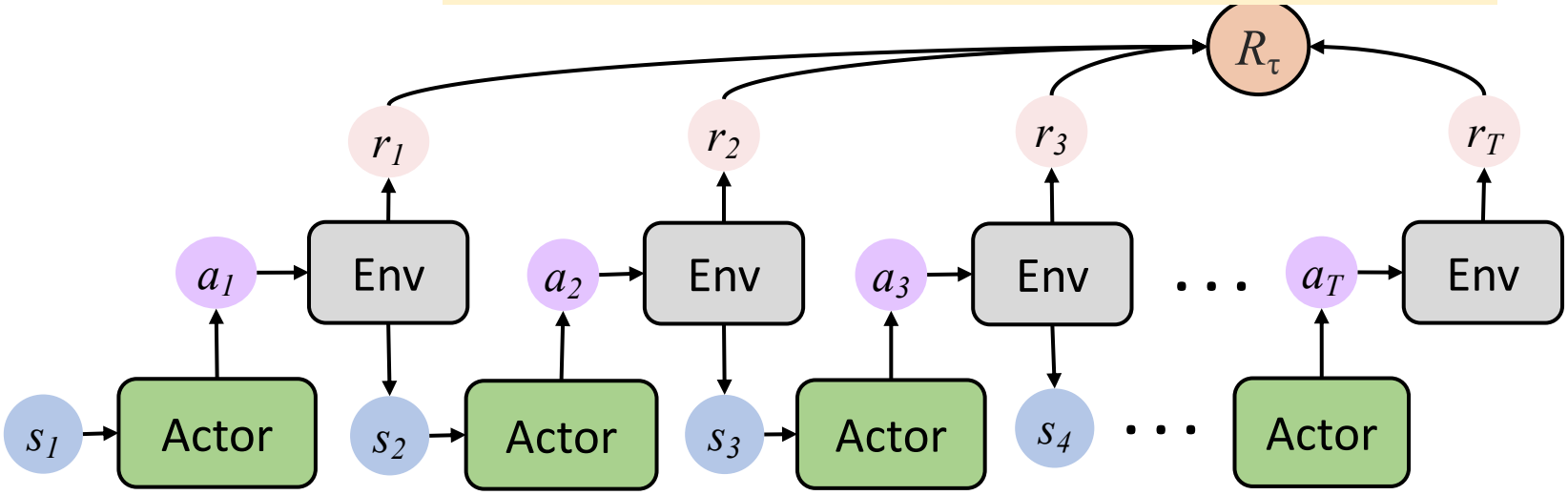
Policy Gradient vs Classification

Classification $L = -\Sigma_t \log p_\theta(a_t|s_t)$ $\nabla L = -\Sigma_t \nabla \log p_\theta(a_t|s_t)$

how likely the trajectory is under the current policy?

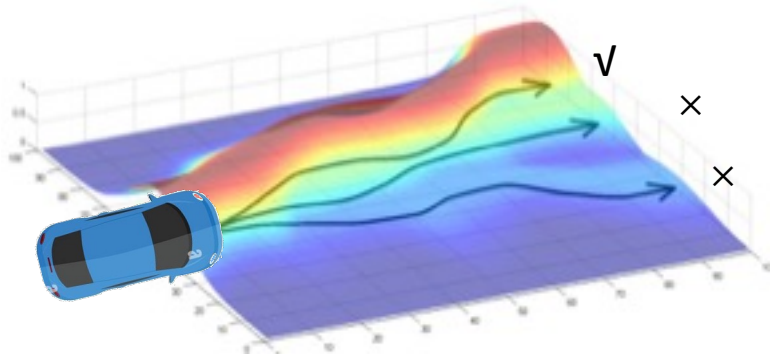
Policy gradient $L = -\Sigma_t \boldsymbol{R}_\tau \log p_\theta(a_t|s_t)$ $\nabla L = -\Sigma_t \boldsymbol{R}_\tau \nabla \log p_\theta(a_t|s_t)$

increase the likelihood of a policy if the trajectory results in a high positive reward and vise versa.



What Does Policy Gradient Actually Do?

- **Decrease** probability of actions that resulted in **low** reward
- **Increase** probability of actions that resulted in **high** reward



If going up the hill means higher rewards, we will change the model parameters (policy) to increase the likelihood of trajectories that move higher.

The Algorithm

REINFORCE (William,1992)

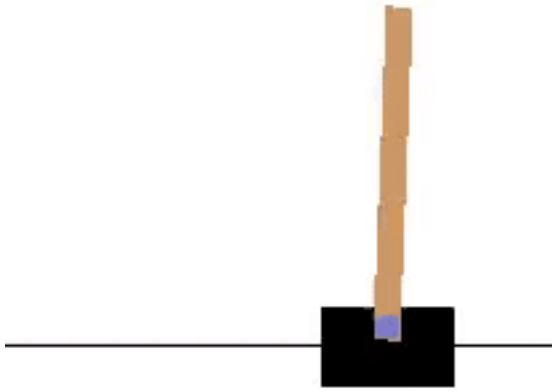
1. Sample $\{\tau^{(l)}\}$ from $p_\theta(a_t|s_t)$
2. Calculate $\nabla_\theta R_\theta$
   2.1 compute $\log p_\theta(a_t^{(l)}|s_t^{(l)})$,
   2.2 obtain the gradient $-\nabla$,
   2.3 multiply it by the weight, i.e., the total reward $R(\tau^{(l)})$
3. $\theta \leftarrow \theta + \eta\nabla_\theta R_\theta$
4. Repeat 1-3 until convergence

TIME for Coding

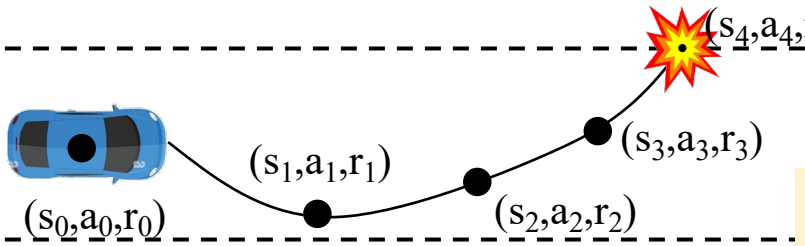
Tutorial: CartPole

https://www.kaggle.com/code/abedsultan/cartpole-balance-with-reinforce-policy-gradient-rl

balance a pole on a moving cart



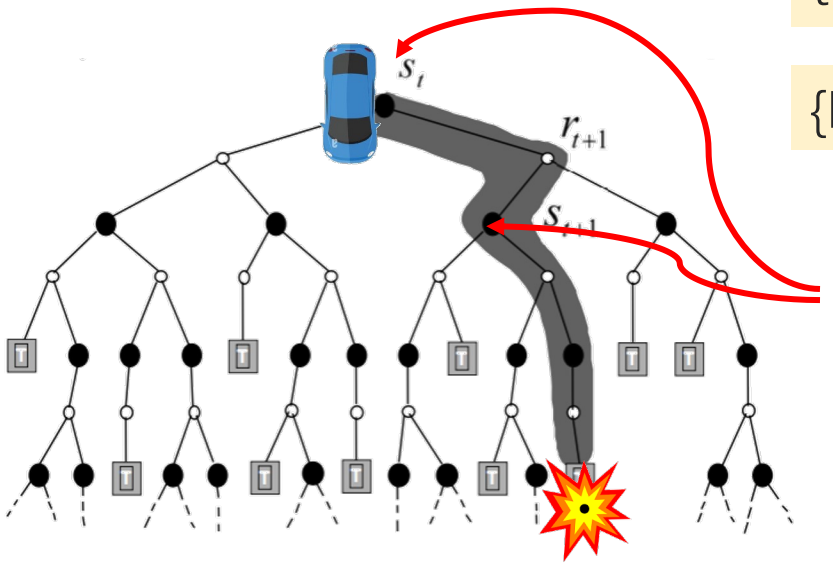
Limitations of Policy Gradient



$$R(\tau) = r_t+\gamma r_{t+1}+\gamma^2 r_{t+2}+\dots \gamma^{T-t} r_T$$

R_τ cannot be obtained until the termination of each episode.

$\{R_\tau\}_{\tau\sim\pi(\cdot)}$ have a large variance.



Can we estimate $\mathbb{E}(\boldsymbol{R}_\tau)$ at each s_t?

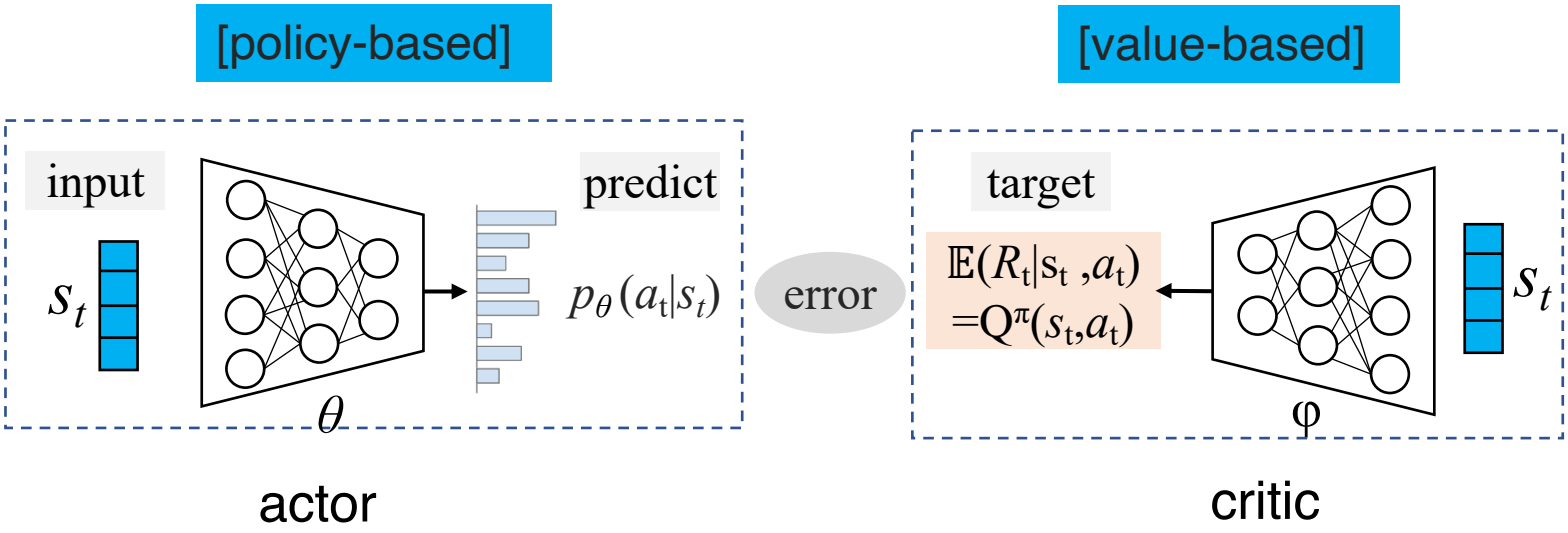
$$R(\tau) = r_t+\gamma r_{t+1}+\gamma^2 r_{t+2}+\dots \gamma^{T-t} r_T$$

Actor-Critic

- **Combining policy-based with value-based models**

  Actor: predicts the action at under each state s_t

  Critic: estimates the long-term return $\mathbb{E}(\boldsymbol{R}_t \mid s_t, a_t)$ for the actor



Training Actor-Critic

Loss:

Policy Gradient: $L_{act} = -\Sigma_t \, \mathbf{R} \log p_\theta(a_t|s_t)$

Actor-critic: $L_{act} = -\Sigma_t \, \mathbf{Q}(s_t, a_t) \log p_\theta(a_t|s_t)$

$Q(s_t, a_t)$ have large values
Let's subtract its average, $V(s_t)$

$L_{act} = -\Sigma_t \, [\mathbf{Q}(s_t, a_t) - \mathbf{V}(s_t)] \log p_\theta(a_t|s_t)$

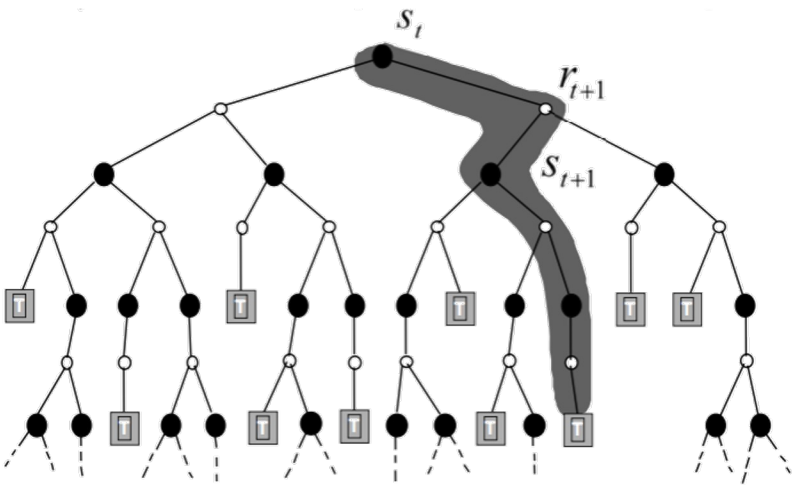
What's this?

State Value and TD Error

- The state-value function $V^{\pi}(s)$ captures the expected *cumulated* reward that can be obtained after seeing state s when using actor π

$$V^{\pi}(s) = \mathbb{E}(R_t|s)$$

score(state)



$V^{\pi}(s)$ is large

$V^{\pi}(s)$ is small

Temporal-difference error

$$V^{\pi}(s_t) \approx r + \gamma V^{\pi}(s_{t+1})$$

smaller variance — may be biased

$$\delta = r + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$$

Training Actor-Critic

Actor: similar to policy gradient

$$L = -\Sigma_t \, [\boldsymbol{Q}(s_t, a_t) - \mathbf{V}(s_t)] \log p_\theta(a_t|s_t)$$

$$= -\Sigma_t \, \underbrace{[\mathbf{r} + \gamma \mathbf{V}(s_{t+1}) - \mathbf{V}(s_t)]}_{\text{TD error } \delta} \log p_\theta(a_t|s_t)$$

$$\nabla L = -\Sigma_t \, [\mathbf{r} + \gamma \mathbf{V}(s_{t+1}) - \mathbf{V}(s_t)] \, \nabla \log p_\theta(a_t|s_t)$$

Critic: similar to Q-learning

$$L(\varphi \mid D) = \mathbb{E}_{(s,r,s')\sim D}[\| (r + \gamma V(s'; \varphi) - V(s; \varphi)\|^2]$$

$$\nabla_{\varphi t} L = \mathbb{E}_{(s,r,s')\sim D}(\underbrace{r + \gamma V(s'; \varphi_{\text{old}})}_{\textbf{fixed, old V-target}} - V(s; \varphi_t)) \nabla_{\varphi_t} V(s; \varphi_t)$$

The Algorithm

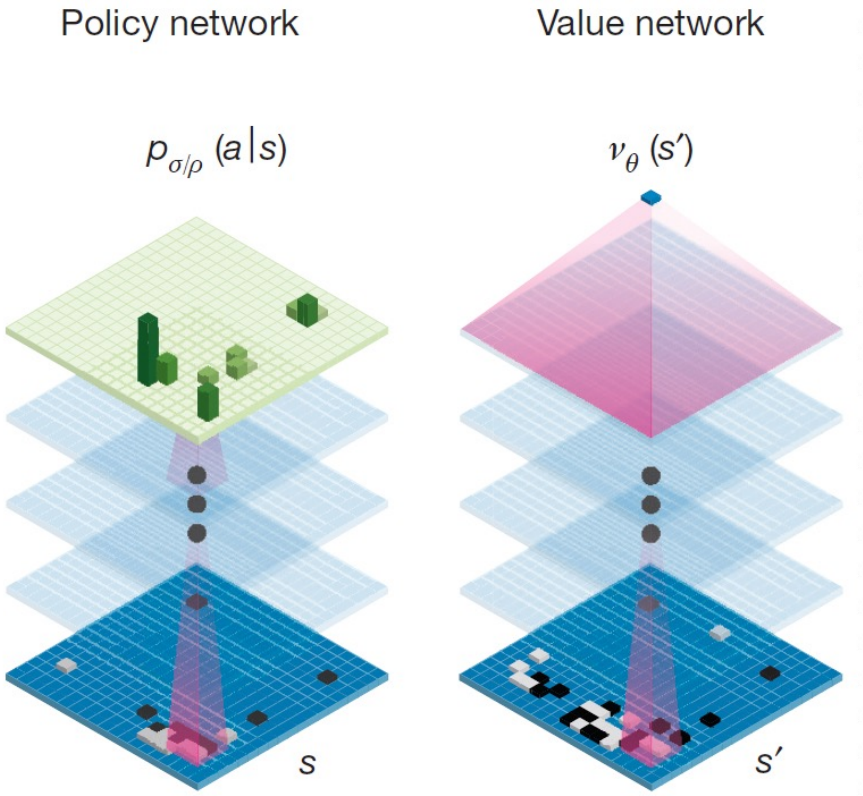
Algorithm TD Advantage Actor-Critic

Randomly initialize critic network $V_\pi^U(s)$ *and actor network* $\pi^\theta(s)$ *with weights* U *and* θ
Initialize environment E
for *episode = 1, M* ***do***
    Receive initial observation state s_0 *from E*
    for *t=0, T* ***do***
        Sample action $a_t \sim \pi(a|\mu,\sigma) = \mathcal{N}(a|\mu,\sigma)$ *according to current policy*
        Execute action a_t *and observe reward* r *and next state* s_{t+1} *from E*
        Set TD target $y_t = r + \gamma \cdot V_\pi^U(s_{t+1})$
        Update critic by minimizing loss: $\delta_t = \left(y_t - V_\pi^U(s_t)\right)^2$
        Update actor policy by minimizing loss:

$$Loss = -log\left(\mathcal{N}(a \mid \mu(s_t), \sigma(s_t))\right) \cdot \delta_t$$

        Update $s_t \leftarrow s_{t+1}$
    end for
end for

Case: AlphaGo

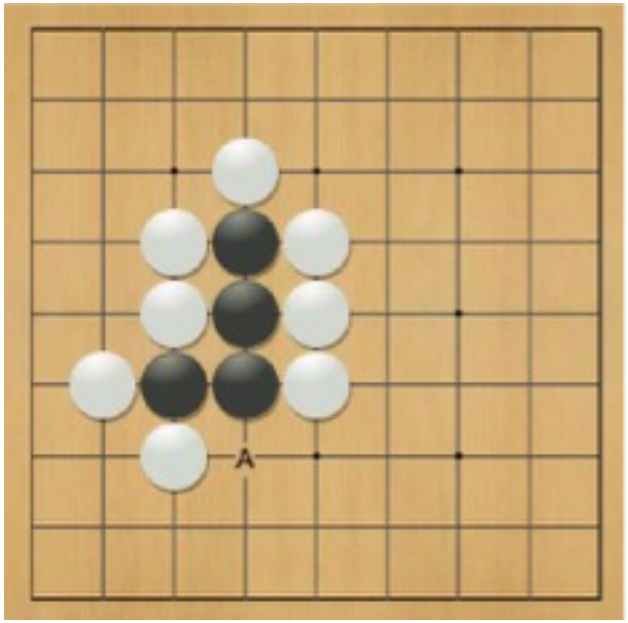


Mastering the game of Go with deep neural networks and tree search. Nature 2016.

Go Overview

- Originated in ancient China 2500 years ago
- 19×19 grid board, playing pieces "stones"
- **Goal**: surround more territory than the opponent
- **Turn** = place a stone or pass
- **Termination**: both players pass

Two Basic Rules of Go



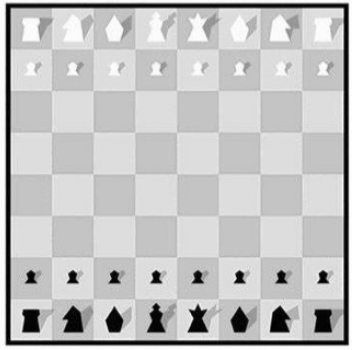
Capture: stones that have no liberties → captured and removed from board

Territory:

Why is Go hard for computers to play?

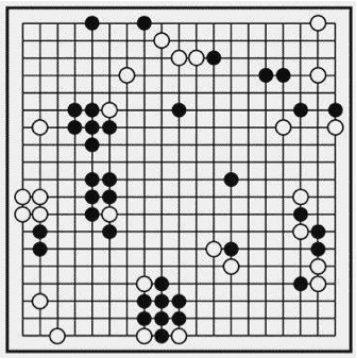
- Possible configuration of board extremely high $\approx 10^{700}$
- Impossible to use brute force exhaustive search

Chess



branch≈35,
depth≈80

Go



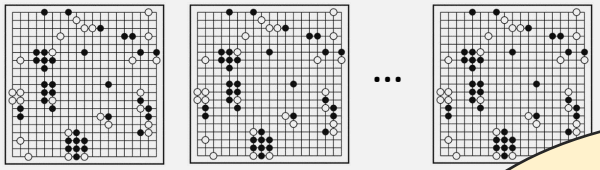
branch≈250,
depth≈150

Game tree complexity = b^d

In a Reinforcement Set-Up

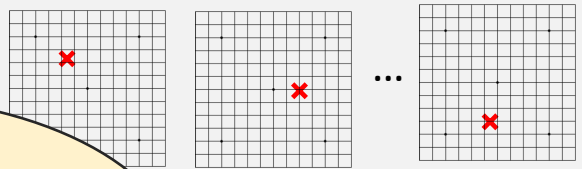
States

configurations of board



Actions

The position that a player will place a "stone" at



Goal:
Find a policy that maximize the expected total payoff

Rewards

Immediate reward:

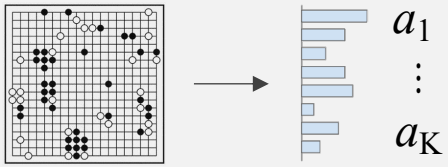
$$r_t = \begin{cases} 0 & \text{if } s_t \text{ is non-terminal} \\ \pm 1 & \text{otherwise} \end{cases}$$

Long-term payoff:

$$z_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$

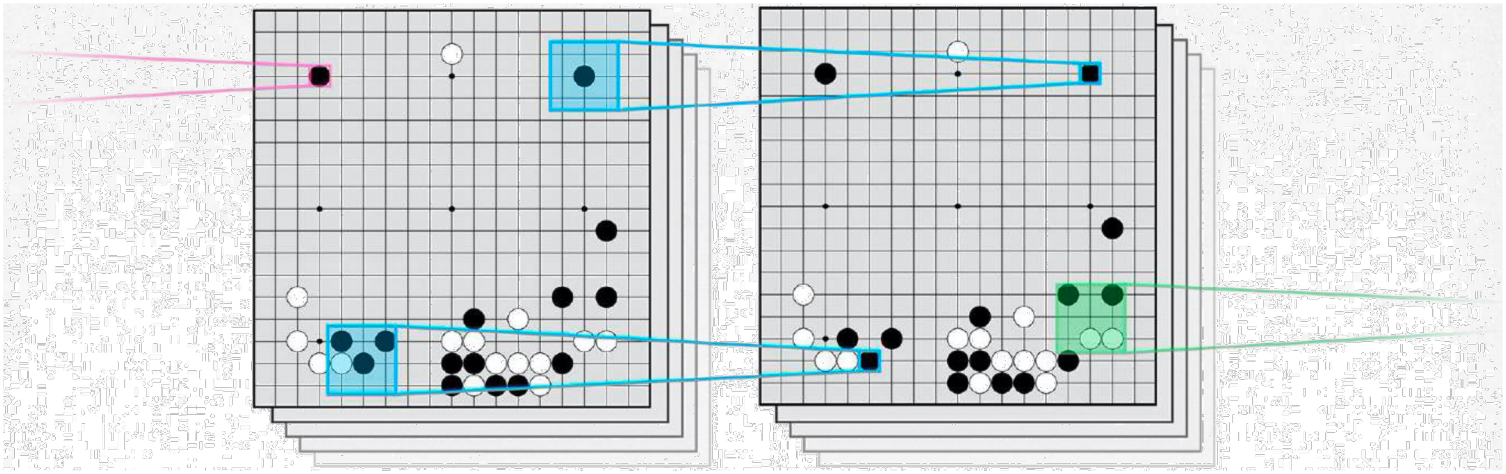
Policy

Which action should be taken under each state?



Representing States

Convolutional Neural Networks

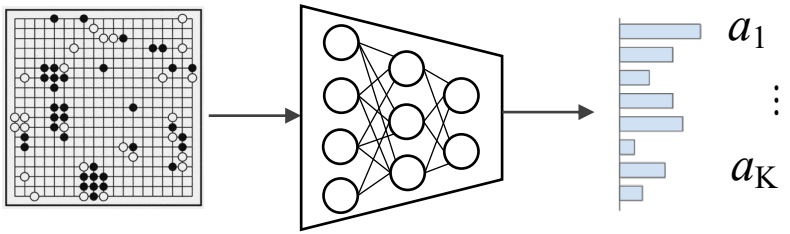


Configurations of board can be seen as images !

Representing Policy

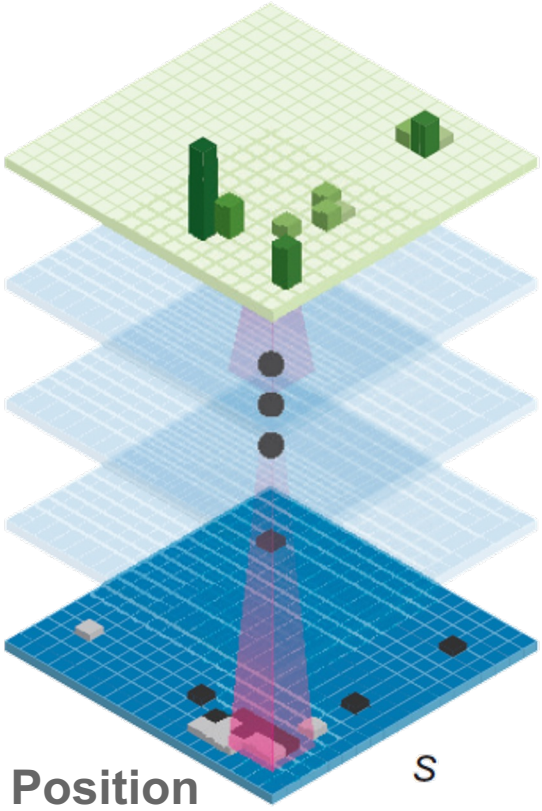
Policy Network

Which action should be taken for state s?



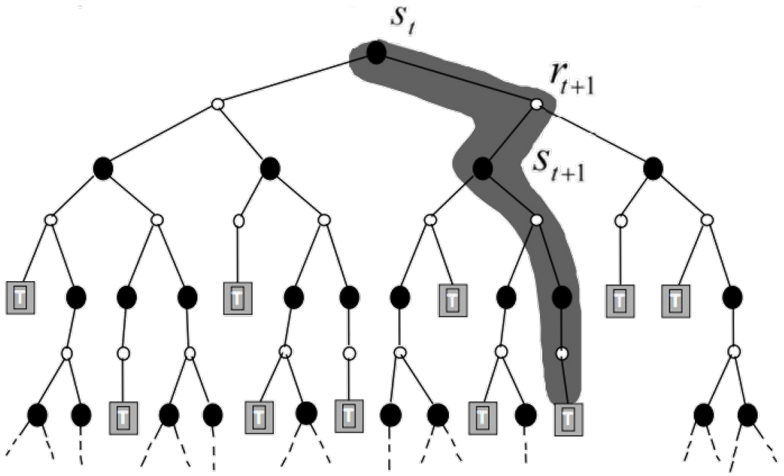
Move probabilities

$p_{\sigma/\rho}(a|s)$



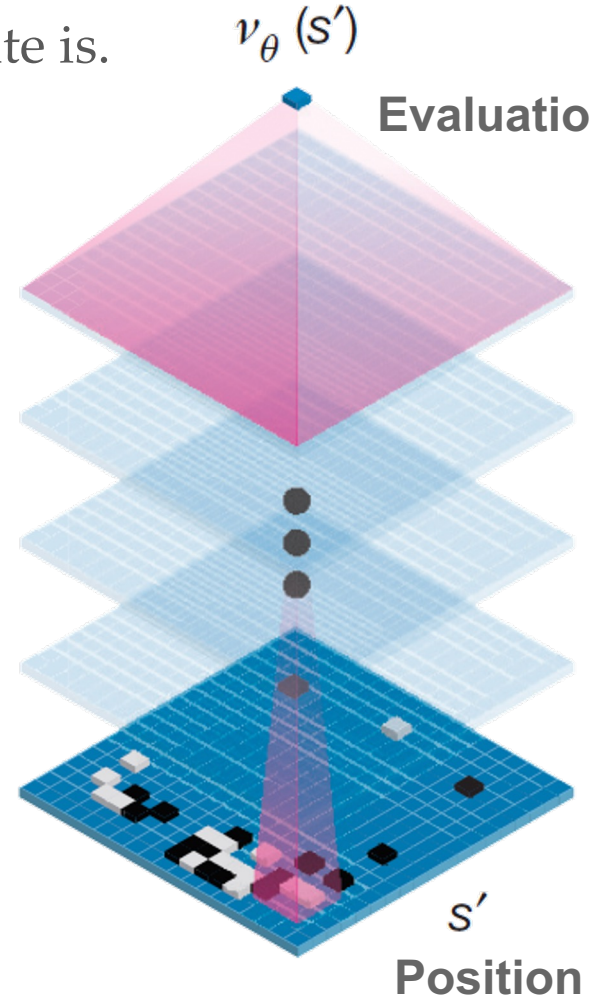
Evaluating States

Value Network: evaluate how good a state is.

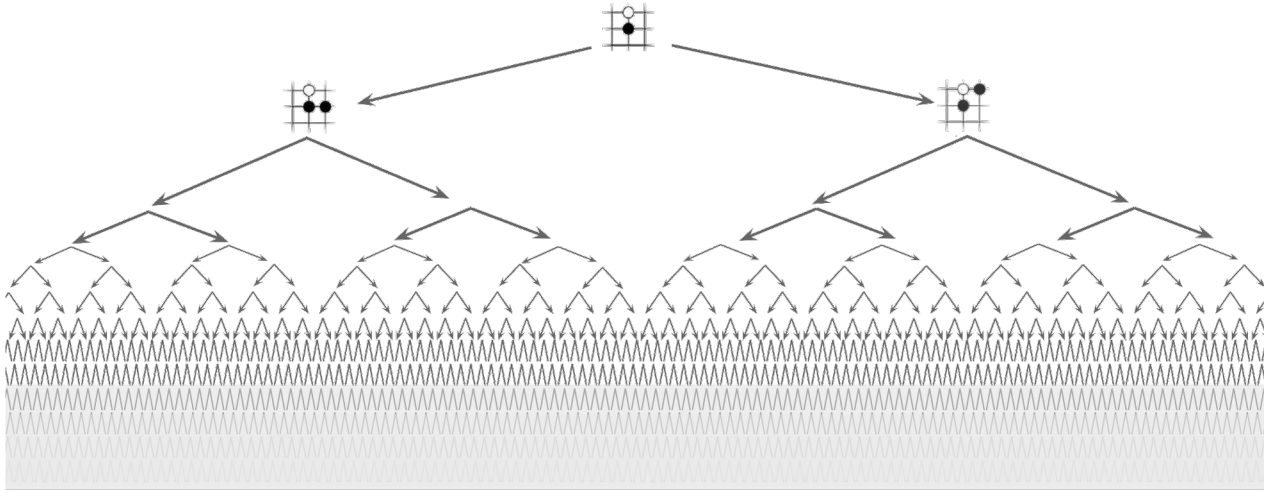


$$V(s_t) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots]$$
$$= \mathbb{E}[z_t] \approx z_t$$

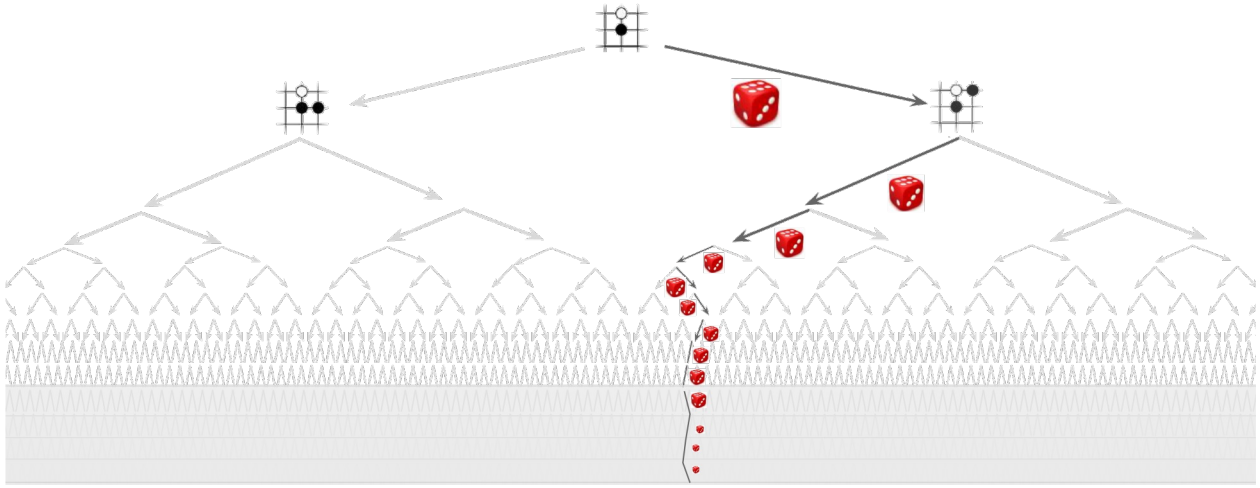
$$L = ||V(s_t) - V_\theta(s_t)||^2 \approx || z_t - V_\theta(s_t)||^2$$



Monte-Carlo Rollouts (蒙特卡洛预演)

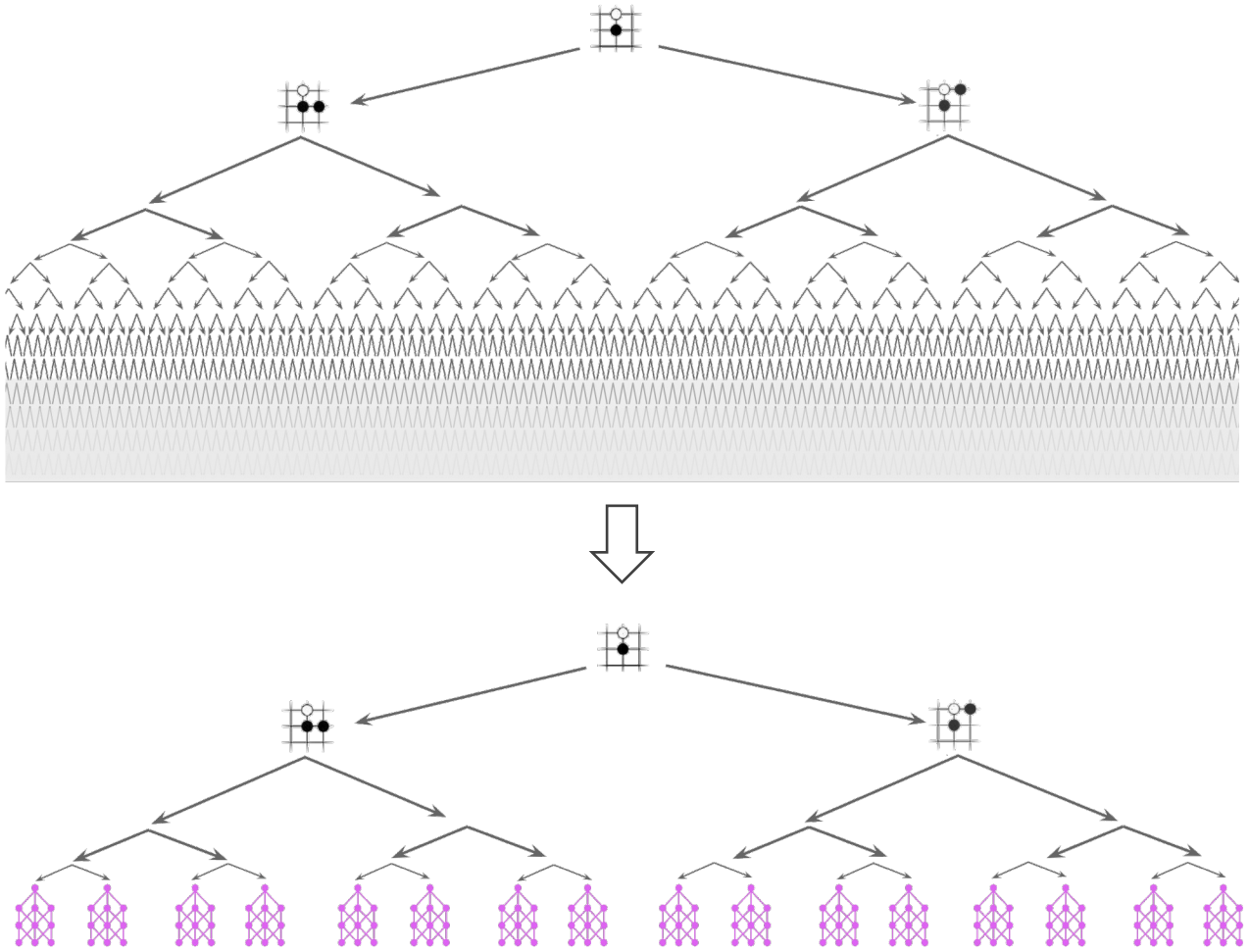


Exhaustive Search

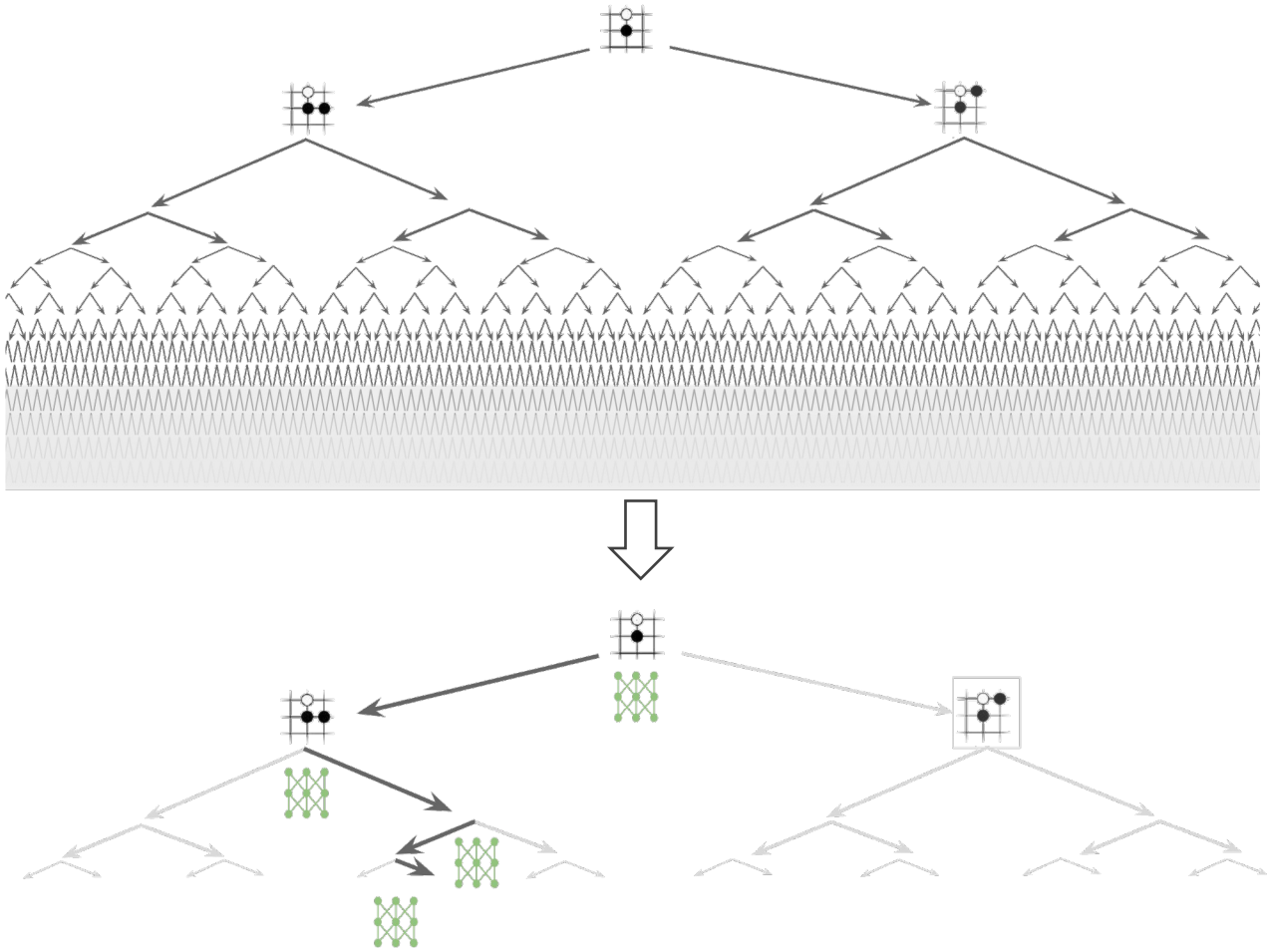


Monte-Carlo Rollouts

Reducing Depth with a Value Network



Reducing Breadth with Policy Network

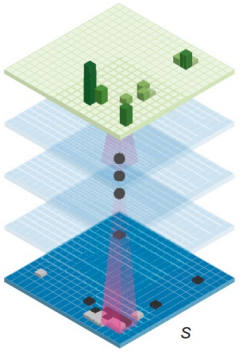


Training AlphaGo

Human expert positions

Supervised Learning

classification

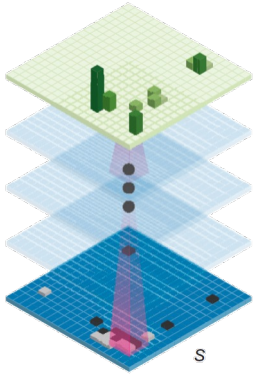


Supervised Learned Policy Network

Self Play

Self-Played Data

Reinforcement Learning



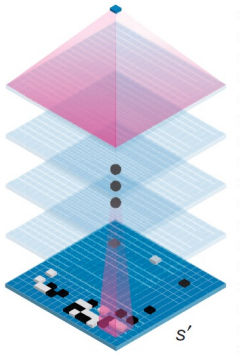
Reinforcement Learned Policy Network

Self Play

Self-Played Data

Reinforcement Learning

regression



Value Network

Supervised Learning of Policy Networks

- **Policy network:** 12-layer convolutional neural network
- **Training data:** 30M positions from human expert games (KGS 5+ dan)
- **Training algorithm:** maximize likelihood by gradient descent

$$\sigma \leftarrow \sigma + \eta \nabla_\sigma \log p_\sigma(a|s)$$

- **Training time:** 4 weeks on 50 GPUs using Google Cloud
- **Results:** 57% accuracy on held out test data (state-of-the art was 44%)

Reinforcement Learning of Policy Networks

- **Policy network**: 12 layer convolutional neural network
- **Training data**: games of self-play between policy network
- **Training algorithm**: maximize wins r by policy gradient:

$$\sigma \leftarrow \sigma + \eta \nabla_\sigma \log p_\sigma(a \mid s)\, z$$

- **Training time**: 1 week on 50 GPUs using Google Cloud
- **Results**: 80% vs supervised learning. Raw network ~3 amateur dan.

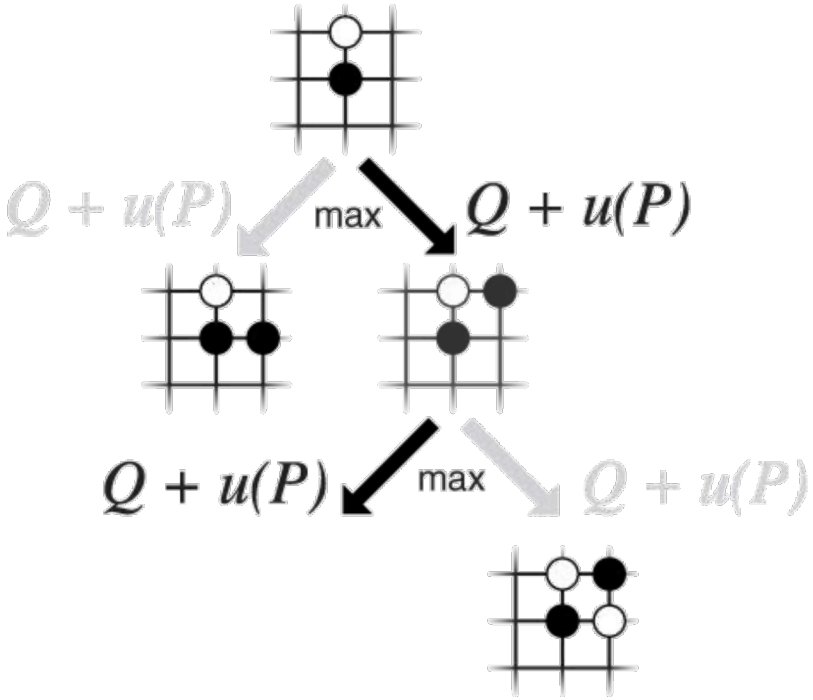
Reinforcement Learning of Value Networks

- **Value network**: 12 layer convolutional neural network
- **Training data**: 30 million games of self-play
- **Training algorithm**: minimize MSE by gradient descent

$$\theta \leftarrow \theta + \eta \nabla_{\theta} V_{\theta}(s) \, (z - V_{\theta}(\mathrm{s}))$$

- **Training time**: 1 week on 50 GPUs using Google Cloud
- **Results**: First strong position evaluation function - previously thought impossible

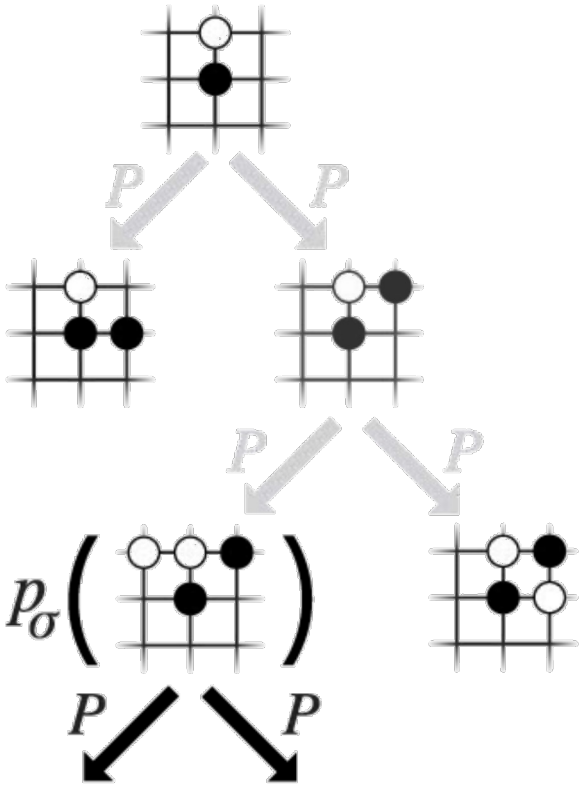
Monte-Carlo Tree Search: Selection



P prior probability
Q action value
N visit count

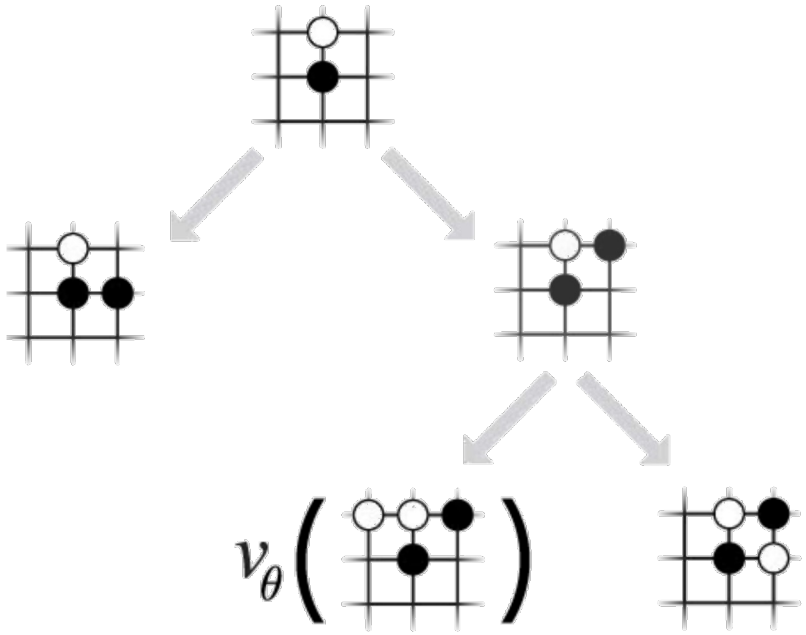
$u(P) \propto P/N$

Monte-Carlo Tree Search: Expansion



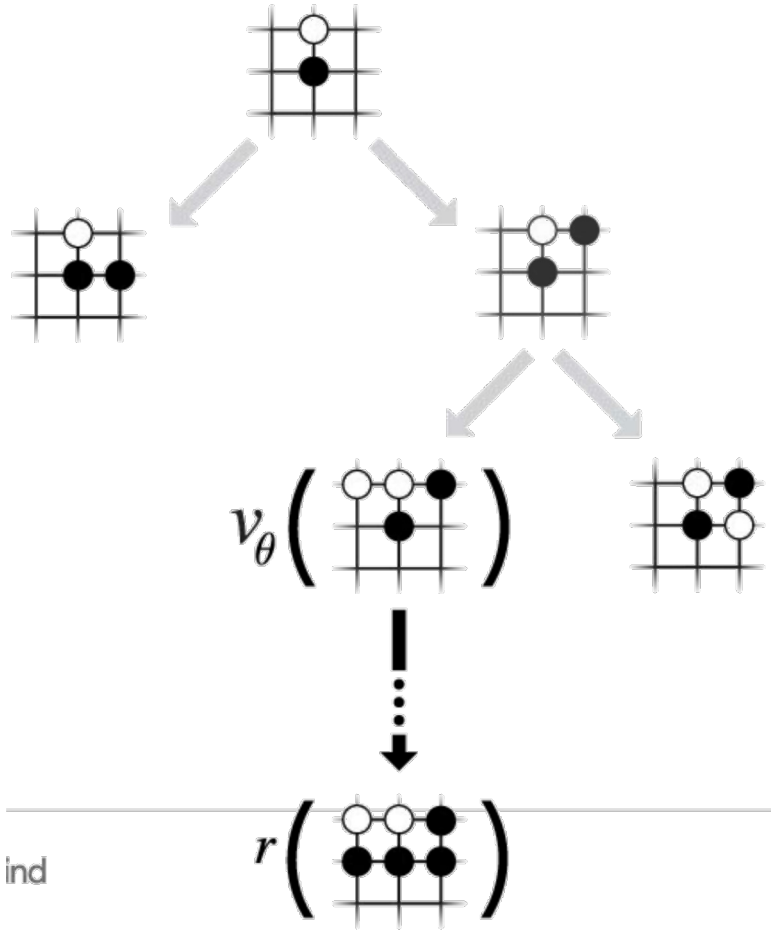
p_σ Policy network
P prior probability

Monte-Carlo Tree Search: Evaluation



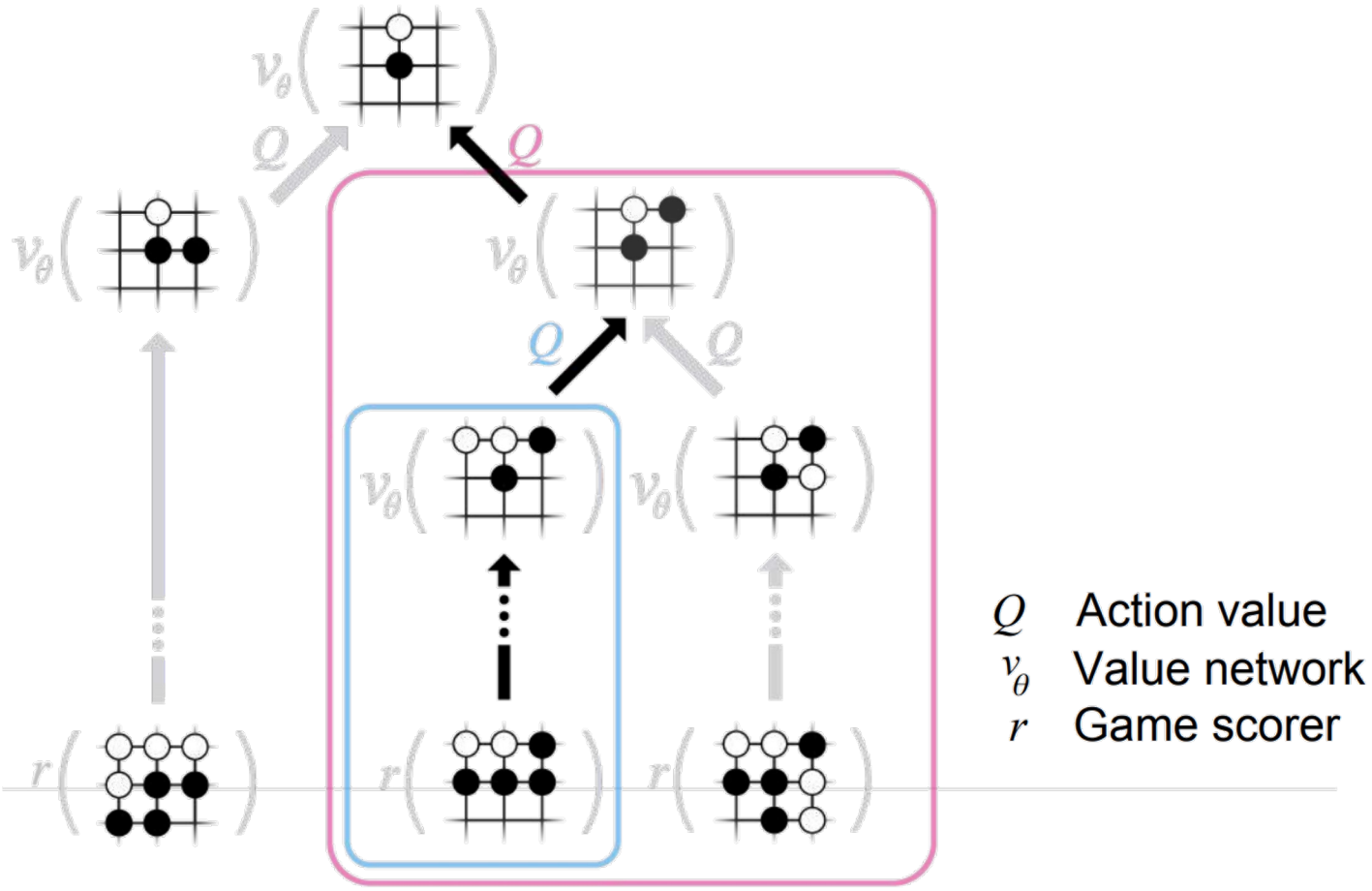
V_θ Value network

Monte-Carlo Tree Search: Rollout (预演)

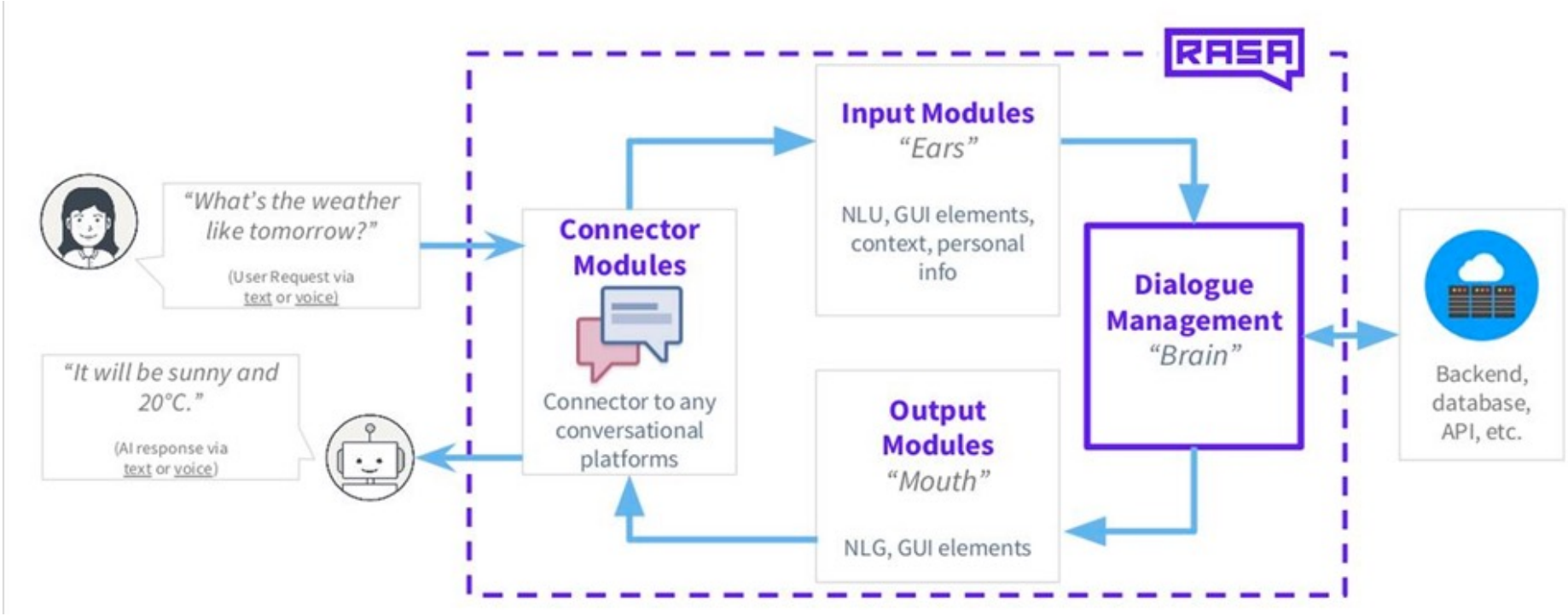


V_θ Value network
r Game score

Monte-Carlo Tree Search: Backup (回溯)



Case: RL for Texts



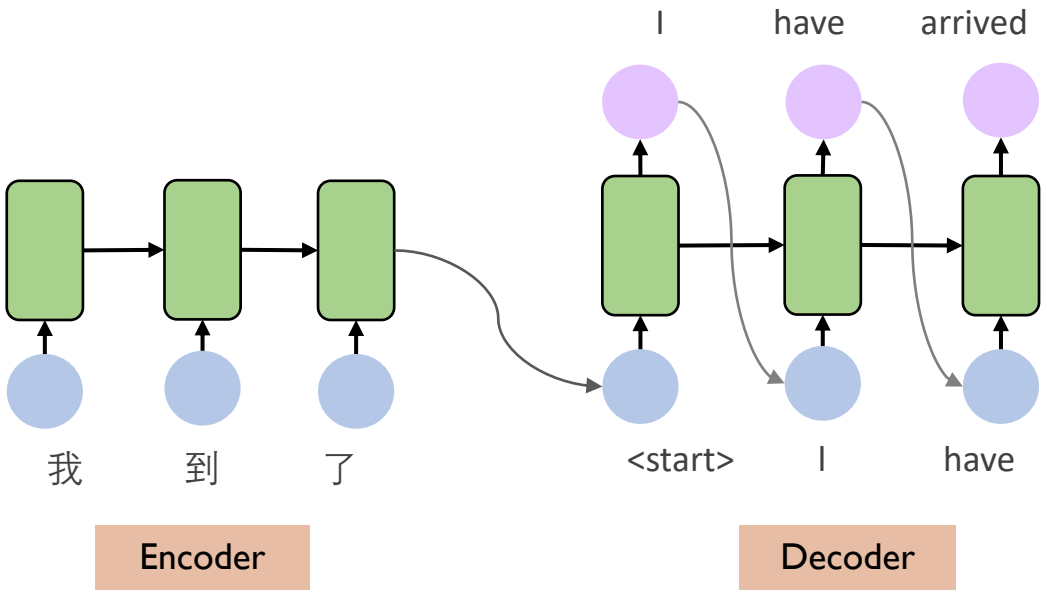
Keneshloo, Yaser, et al. "Deep reinforcement learning for sequence-to-sequence models." *IEEE transactions on neural networks and learning systems* 31.7 (2019): 2469-2489.

A DEEP REINFORCED MODEL FOR ABSTRACTIVE SUMMARIZATION

SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient

Recall: Neural Language Generation (NLG)

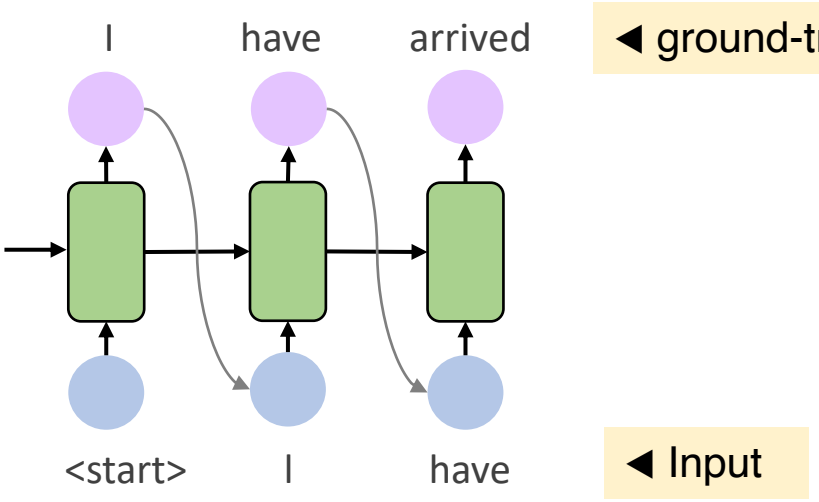
- Given input sequence $x = \{x_1, x_2, \cdots, x_N\}$, output $y = \{y_1, y_2, \cdots, y_T\}$.



- **Applications**: machine translation, question answering, dialogue system, summarization, …

Recall: Training NLG

- Teacher-forcing training



- **Training objective**: minimize cross-entropy (= maximum likelihood) :

$$l(\theta|x, y) = -\sum_{t=1}^{T} \log p_\theta\left(y_t|y_1, \ldots, y_{t-1}, x\right)$$

Problems with MLE Training

- **Different training and testing objectives**

Training	discrepancy	Testing
maximize likelihood of predicted words	⟷	correctness, coherence, fluency, diversity, …
continuous, differentiable, short-term		discrete, non-differentiable, long-term

- Example: problems with MLE training in dialog generation

Generic response:

A: What's the weather like?
B: I don't know
A: where are you going?
B: I don't know
…

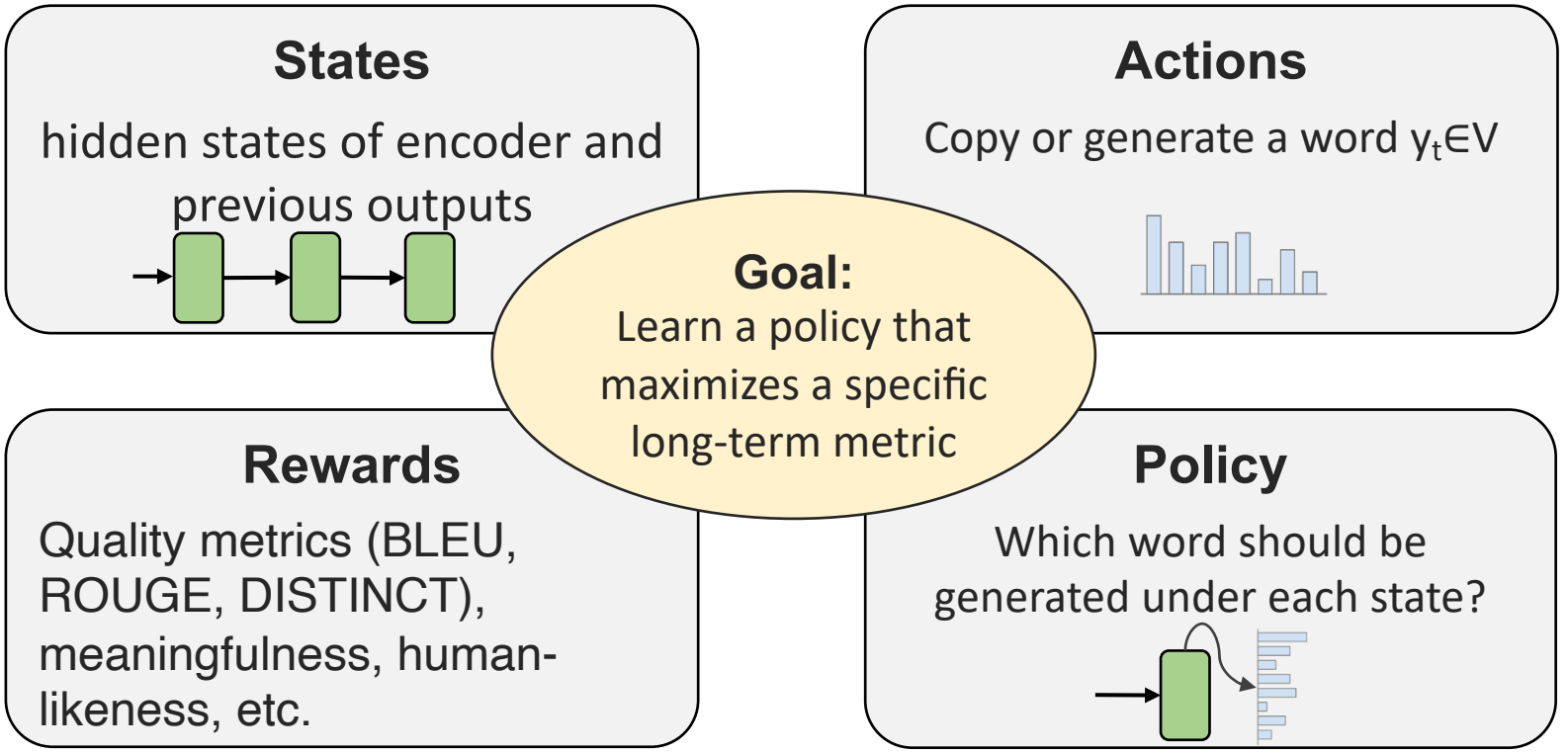
Repetition:

A: Where are you going ? (1)
B: I 'm going to the restroom . (2)
A: See you later . (3)
B: See you later . (4)
A: See you later . (5)

Li et al. Deep Reinforcement Learning for Dialogue Generation.

Deep Reinforced Model for NL Generation

- Integrate developer-defined rewards that better mimic the true goal of text generation.
- model the long term influence of a generated text



Deep Reinforced Model for NL Generation

Reward #1 – Ease of answering:

avoid utterance with a dull response.

$$r_1 = -\frac{1}{N_{\mathbb{S}}} \sum_{s \in \mathbb{S}} \frac{1}{N_s} \log p_{\text{seq2seq}}(s|a)$$

where $\mathbb{S}$ is a list of dull answers such as "I don't know", "I have no idea", etc

Deep Reinforced Model for NL Generation

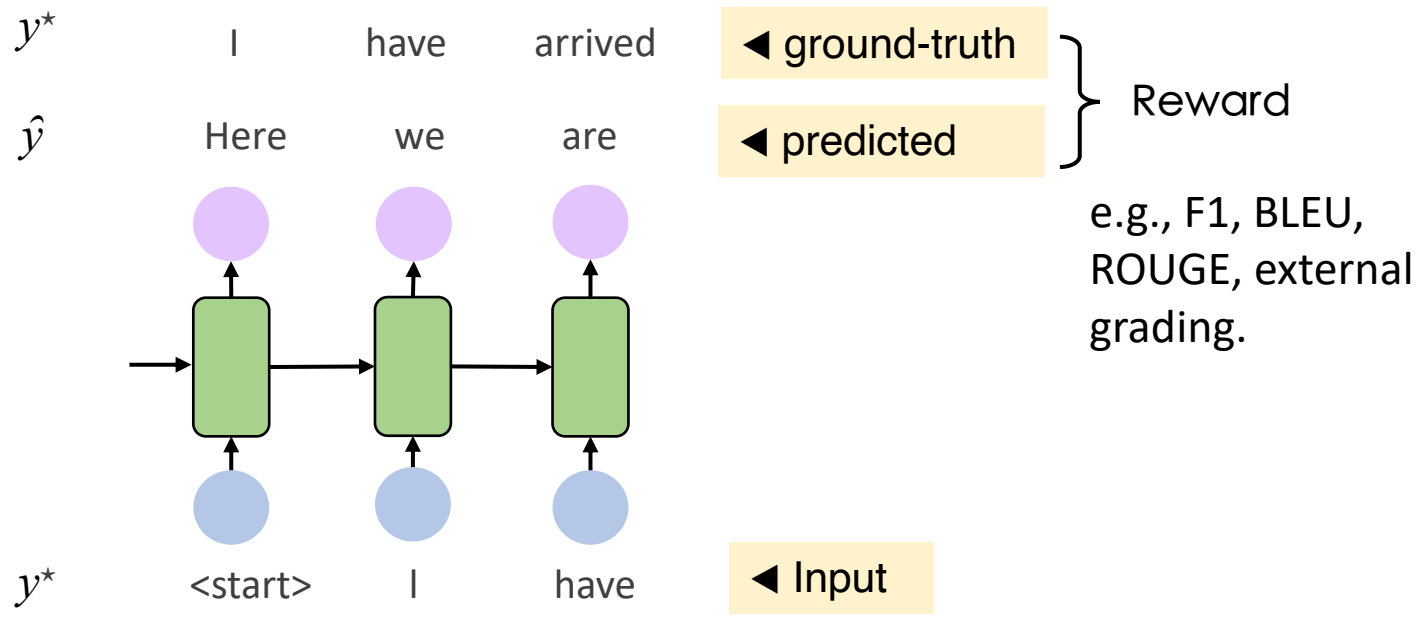
Reward #2 – Information Flow:

penalize semantic similarity between consecutive turns from the same agent..

$$r_2 = -\log\cos(h_{p_i}, h_{p_{i+1}}) = -\log\cos\frac{h_{p_i} \cdot h_{p_{i+1}}}{\|h_{p_i}\|\|h_{p_{i+1}}\|}$$

where h_{p_i} and $h_{p_{i+1}}$ denote representations obtained from the encoder for two consecutive turns p_i and p_{i+1}

Training RL-NLG



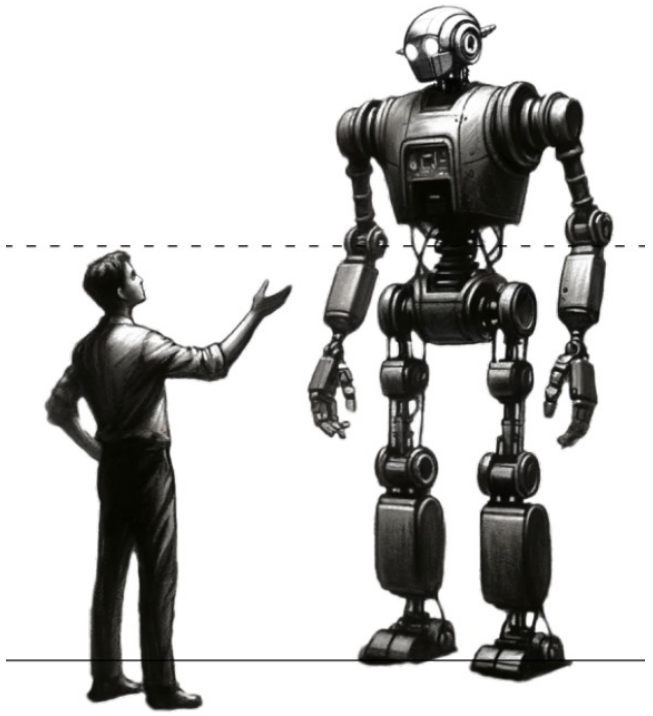
- **Training objective**: policy gradient

$$l(\theta) = -\sum_{t=1}^{T} \boldsymbol{R} \log p_\theta(\hat{y}_t | \hat{y}_1, \ldots, \hat{y}_{t-1}, x)$$

$$\nabla l(\theta) = -\sum_{t=1}^{T} \boldsymbol{R} \nabla \log p_\theta(\hat{y}_t | \hat{y}_1, \ldots, \hat{y}_{t-1}, x)$$

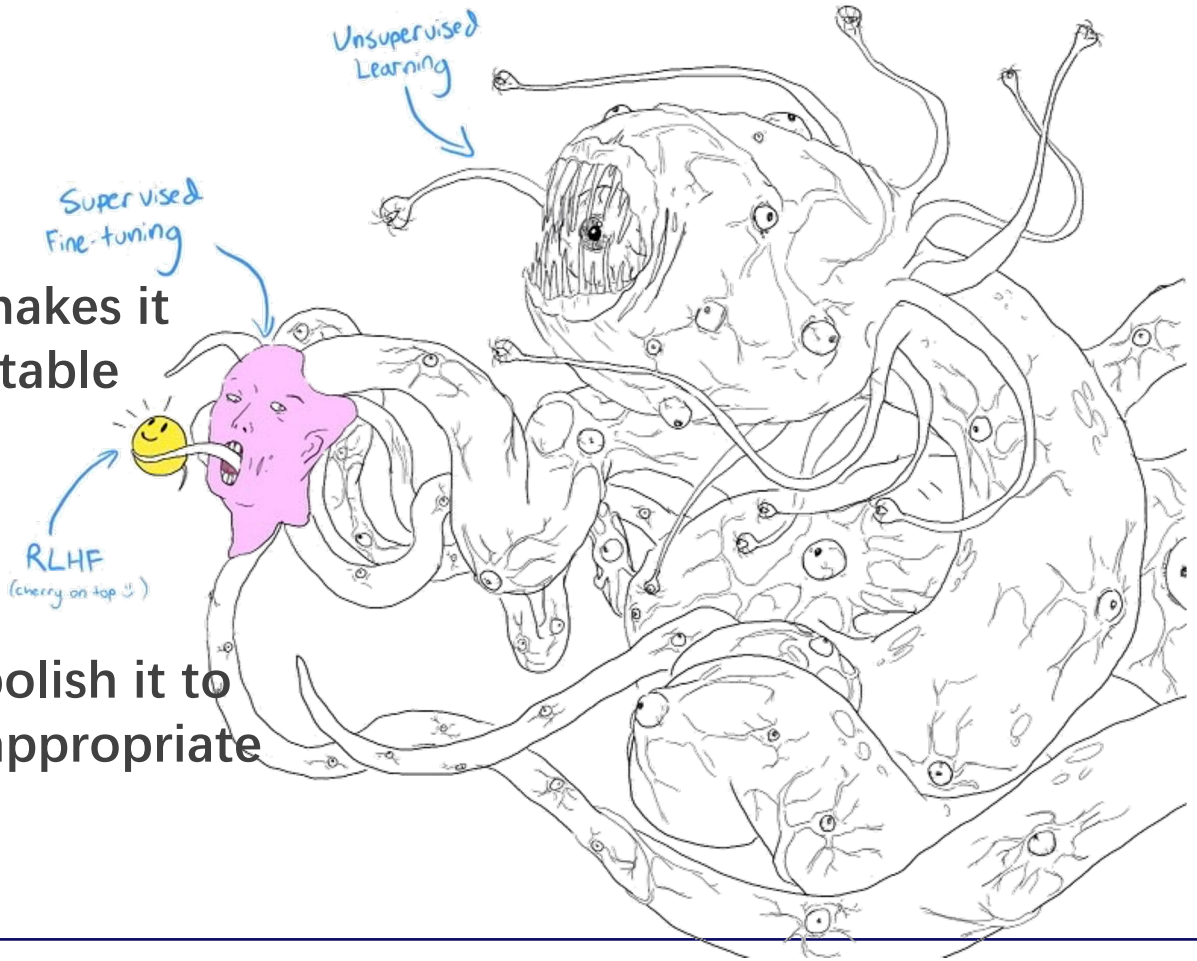
Case: RL for LLMs

RLHF – Reinforcement Learning from Human Feedback



Why RLHF?

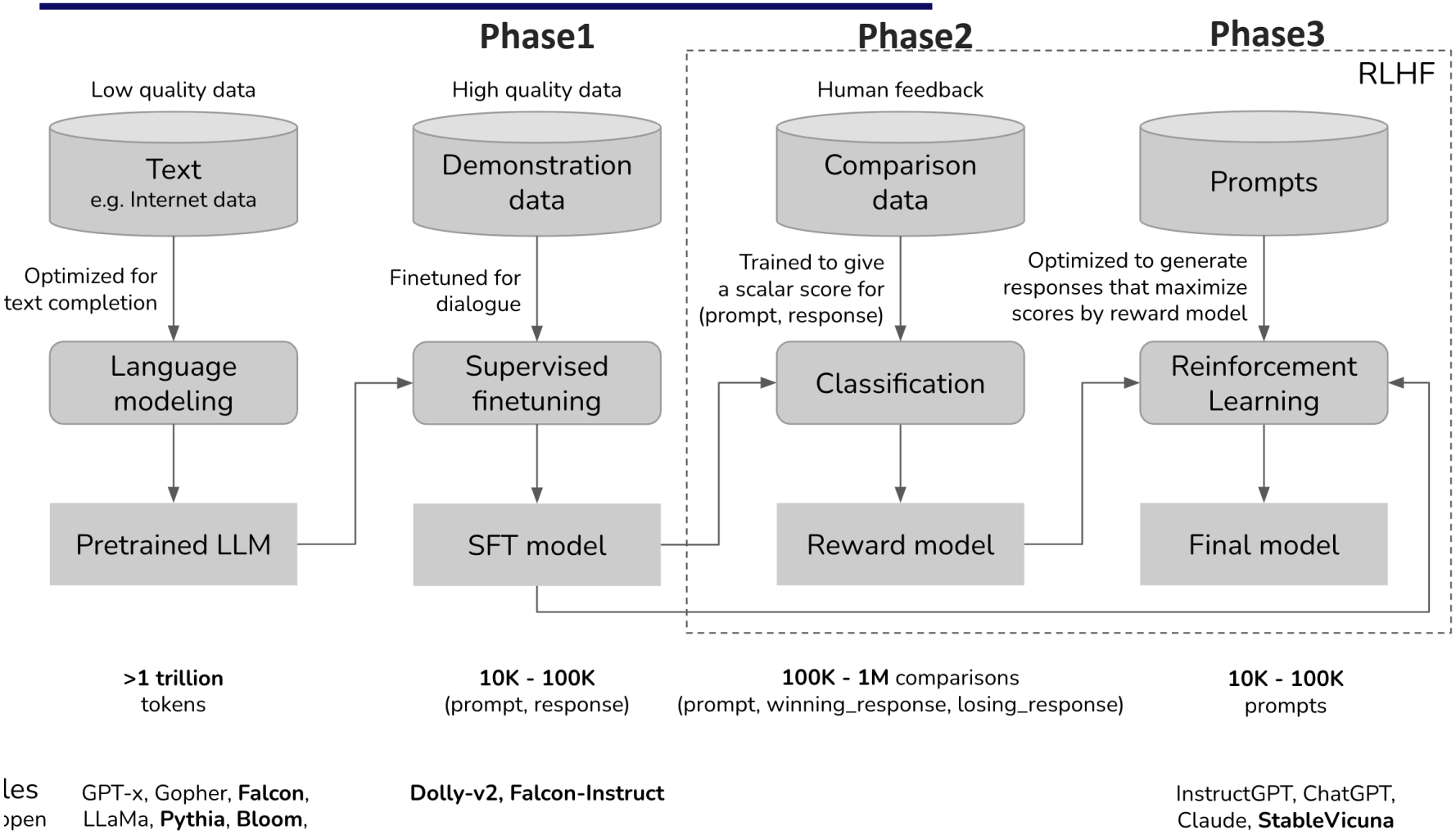
LLM is an untamed monster



Fine-tuning makes it socially acceptable

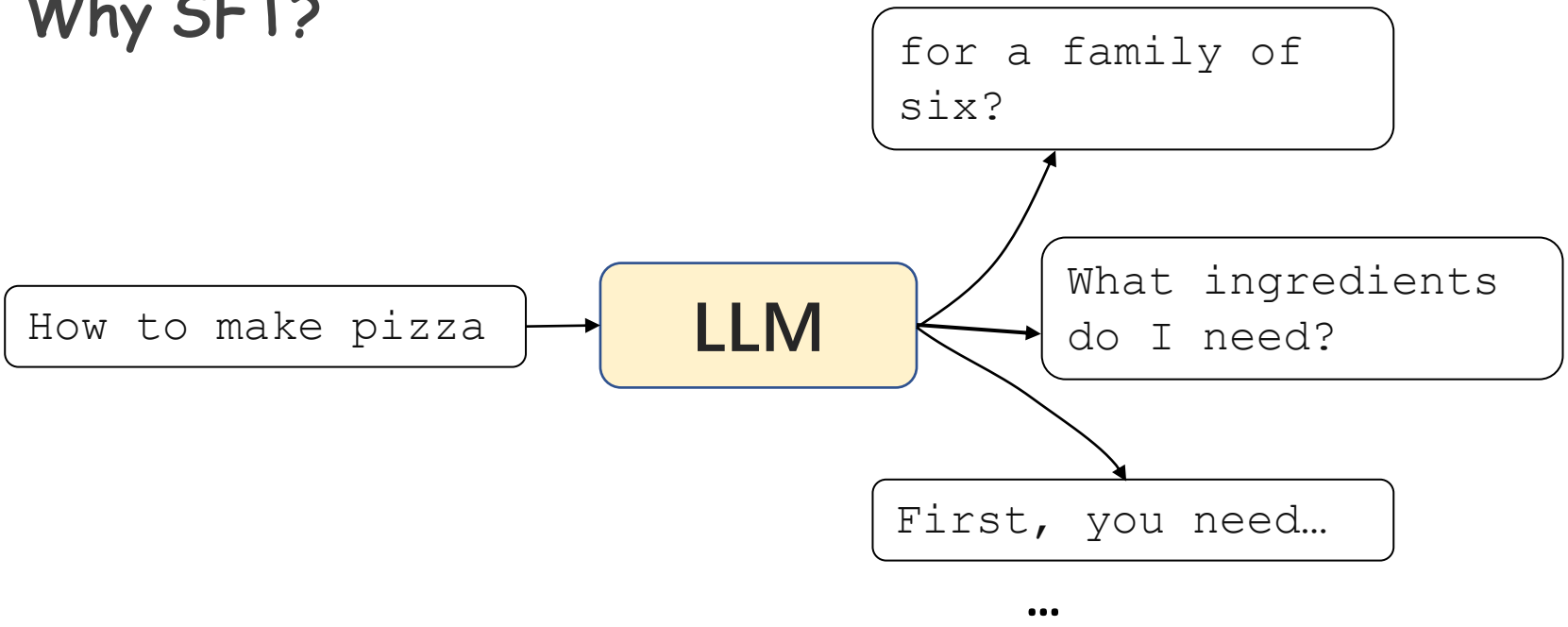
RLHF further polish it to be customer-appropriate

RLHF Overview



Phase1: Supervised finetuning (SFT)

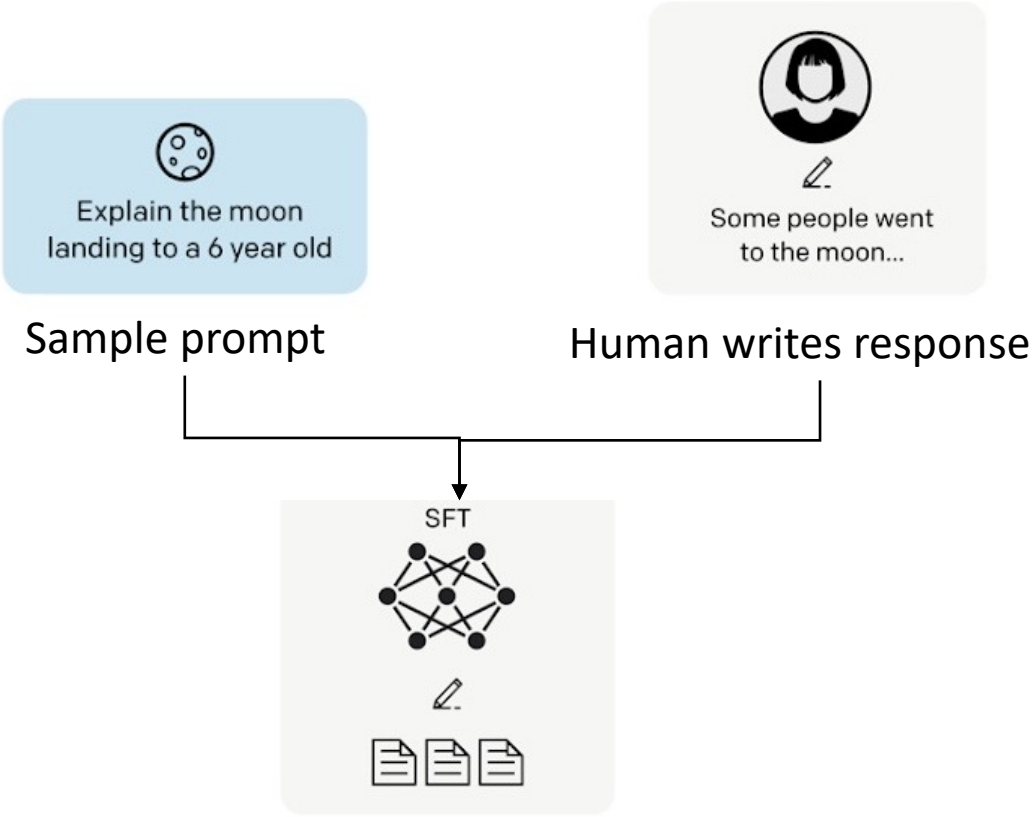
Why SFT?



SFT goal: optimize the LLM to generate the responses that users are looking for.

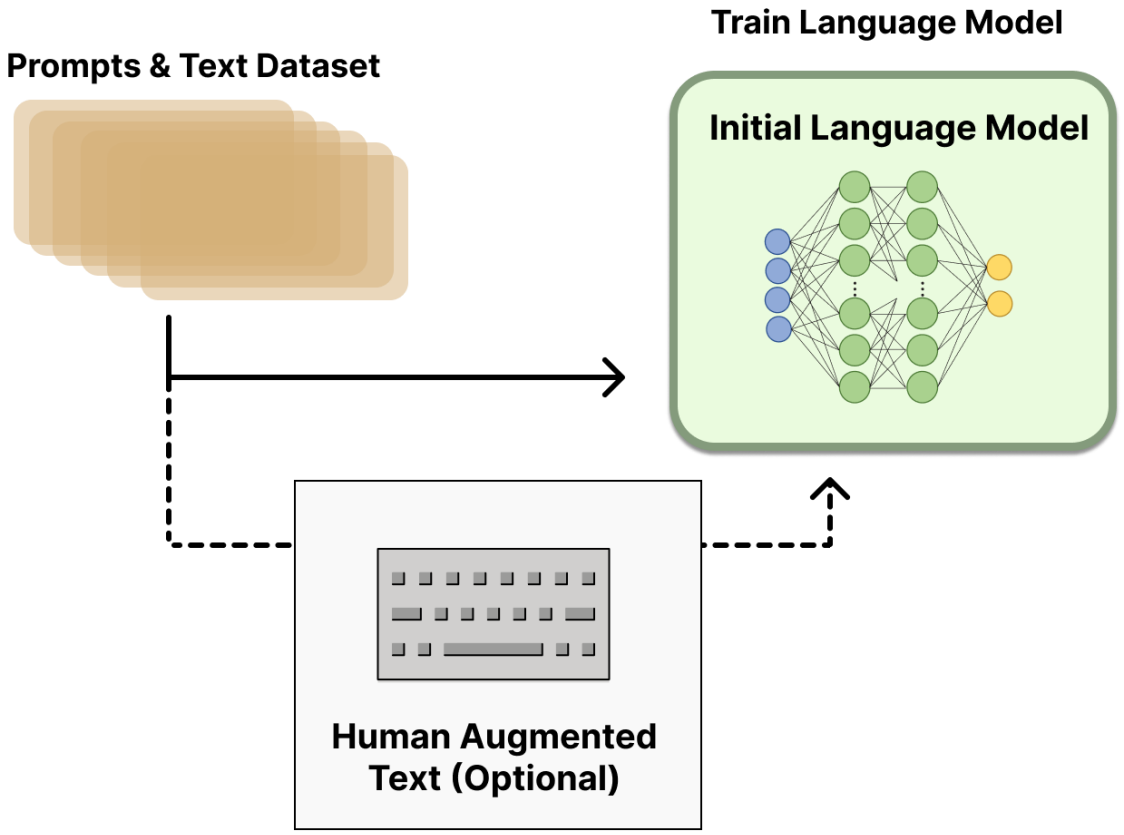
Phase1: Supervised Finetuning (SFT)

- Fine-tune LLM with human-written (prompt, response) pairs.

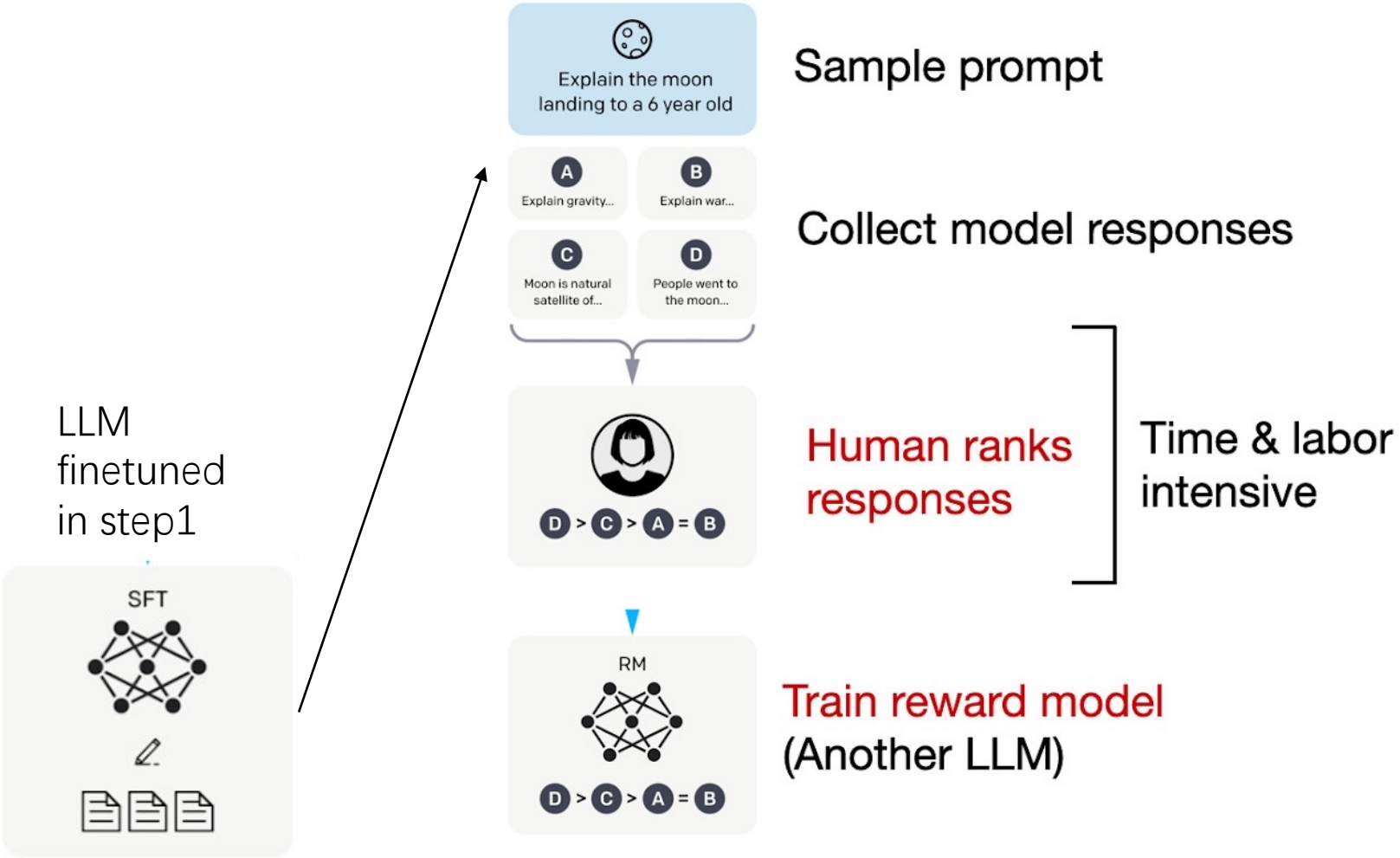


Phase1: Supervised Finetuning (SFT)

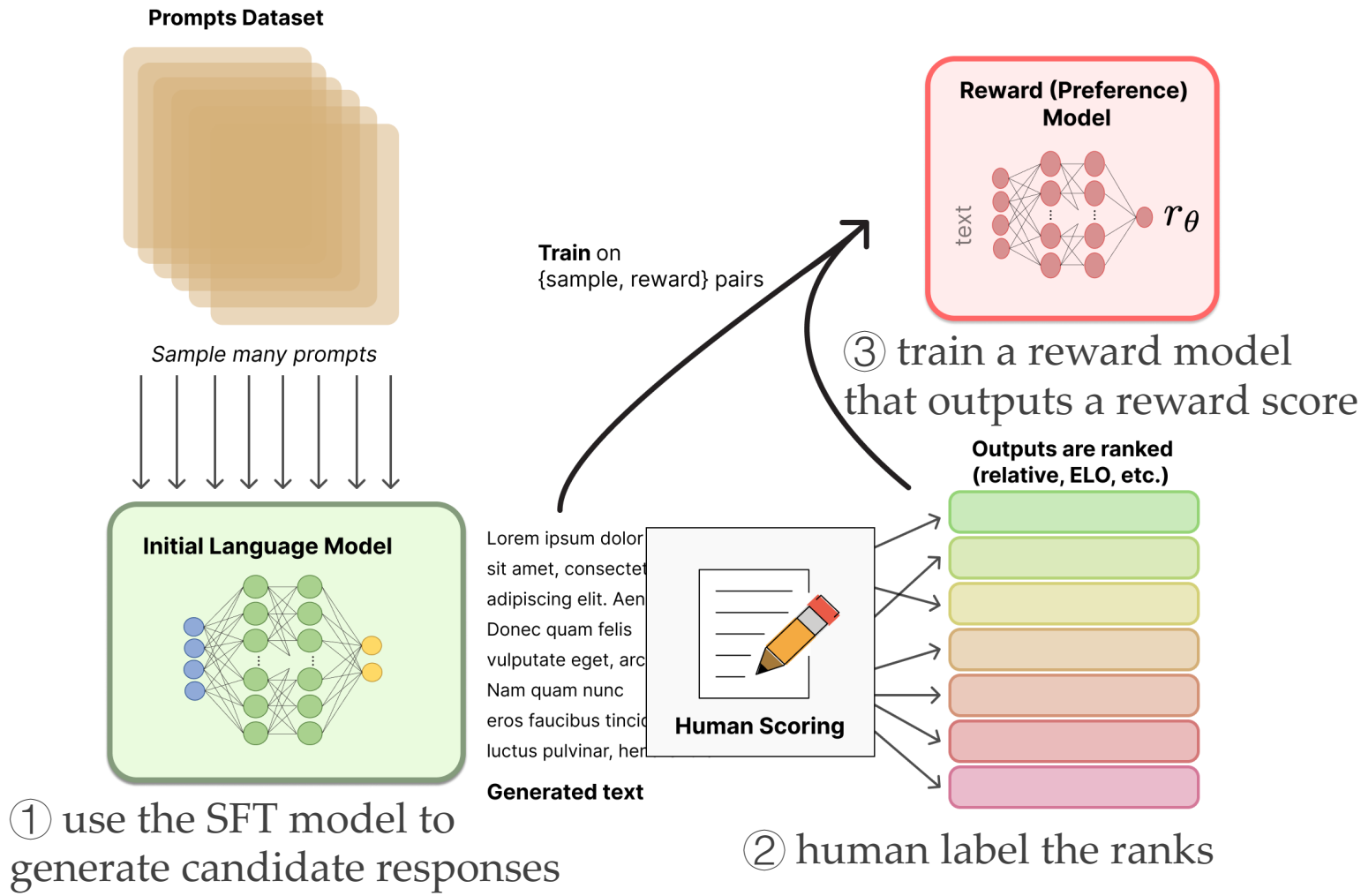
- Finetune LLM with human-written (prompt, response) pairs.



Phase2: Create a Reward Model

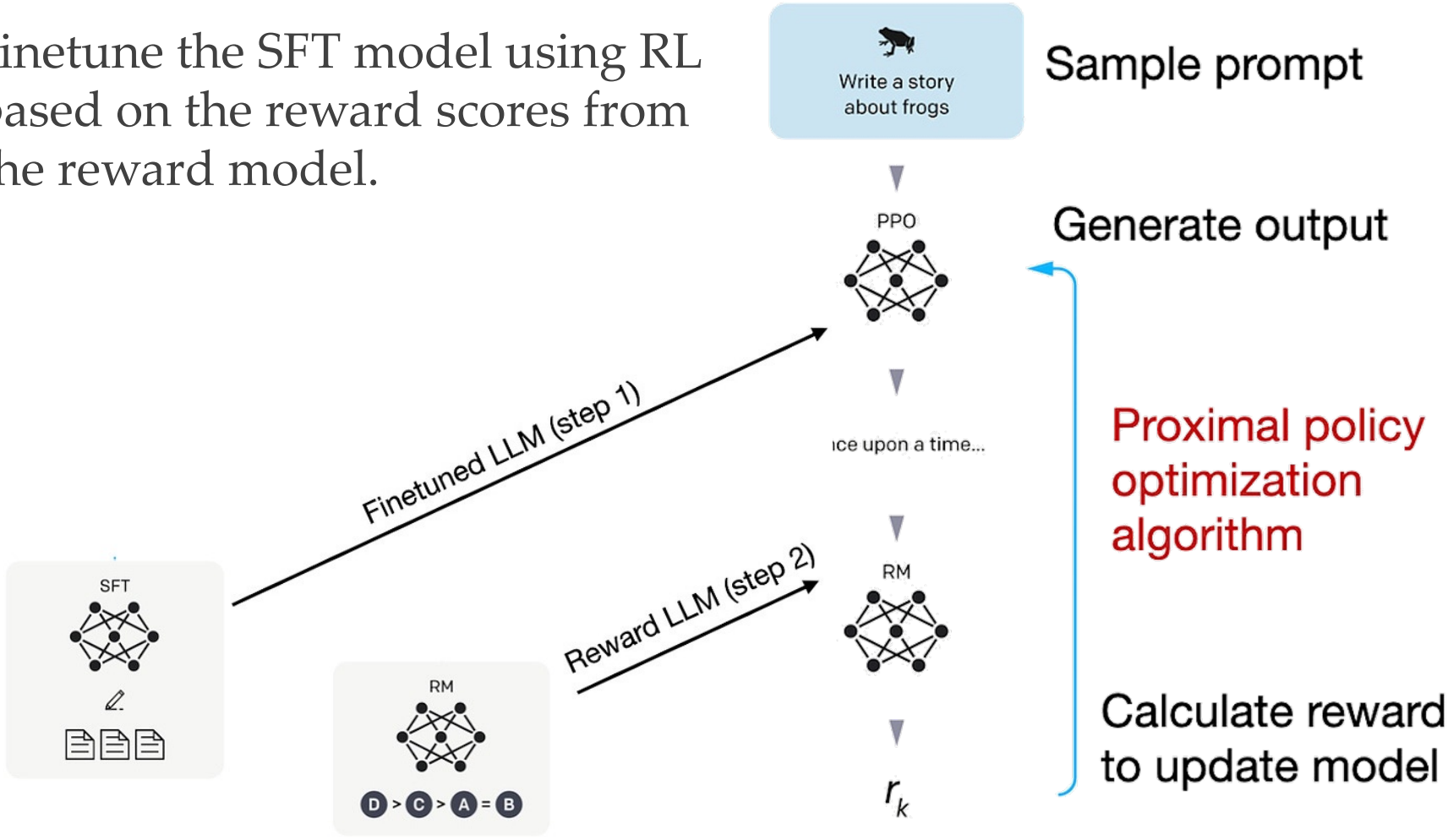


Phase2: Create a Reward Model

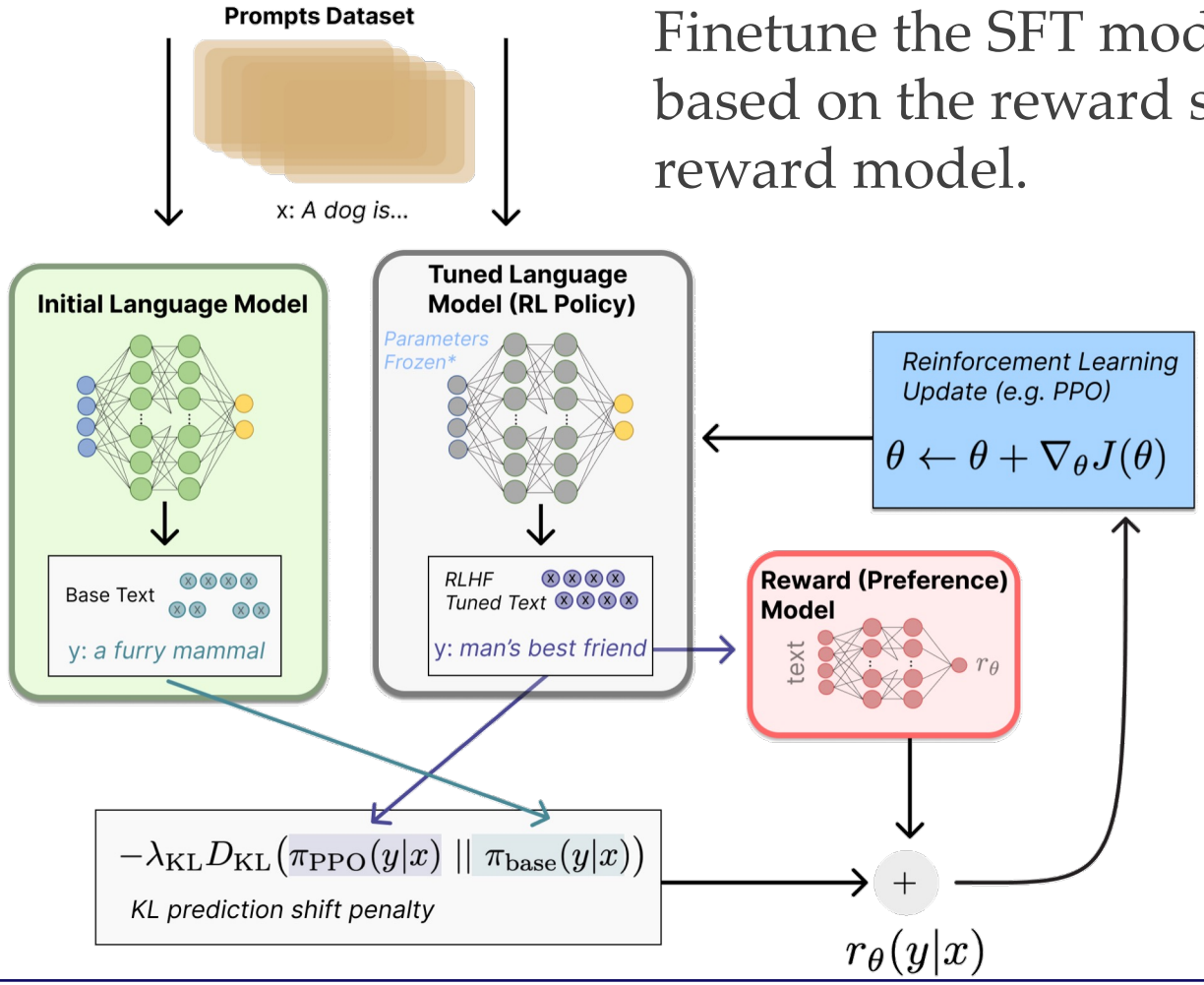


Phase3: Finetuning via RL

- Finetune the SFT model using RL based on the reward scores from the reward model.



Phase3: Finetuning via RL



Finetune the SFT model using RL based on the reward scores from the reward model.