Summer School On Recent Advances
in Deep Learning, VIASM 2023

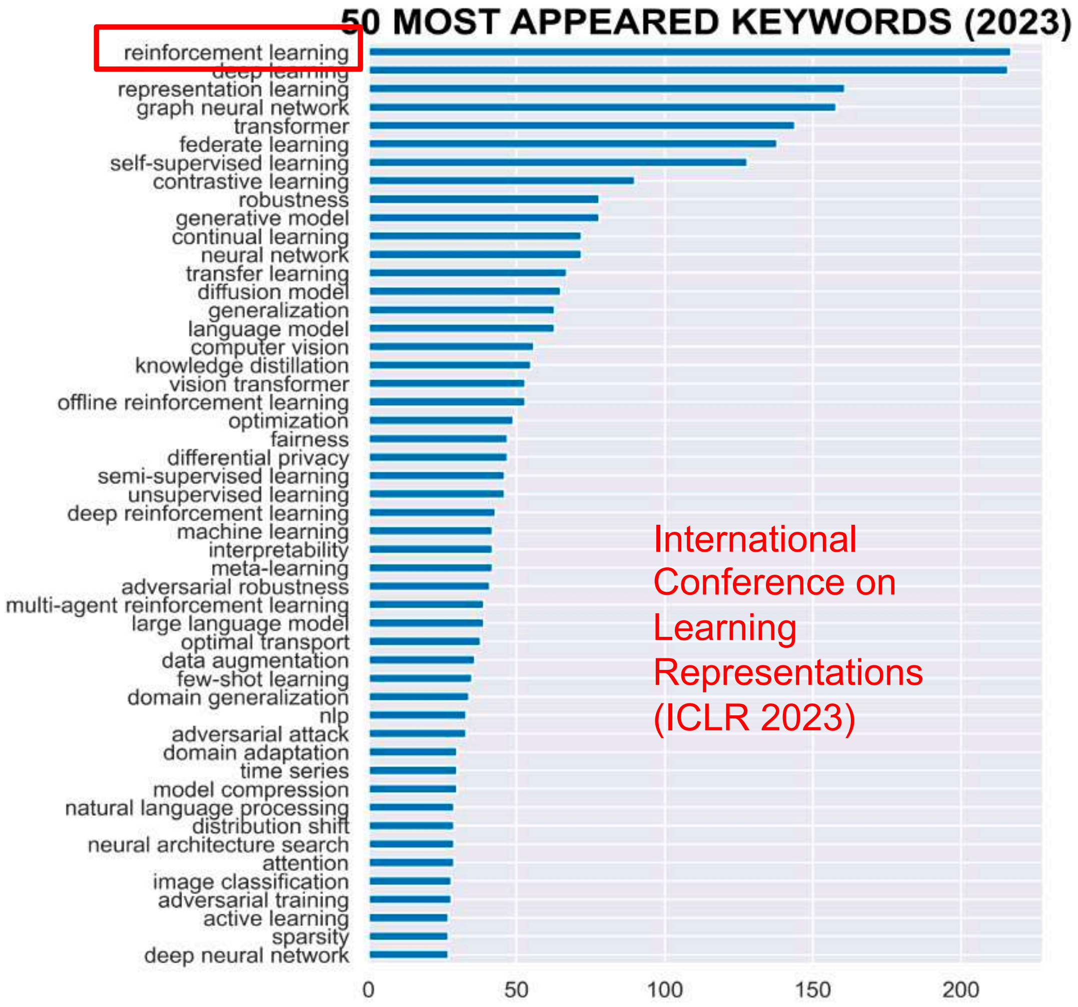
Reinforcement Learning and Deep RL

Lê Anh Cường

Khoa CNTT, Trường ĐH Tôn Đức Thắng

Why is RL important?

- AlphaGo (2015) and AlphaGo Zero (2017)
- ChatGPT (2022)
- Diffusion model (2023)



Features of RL

- Versatility (linh hoạt)
- Autonomous Decision-Making and Handling Uncertainty
- Sequential Decision-Making
- Exploration and Exploitation
- Continuous Learning

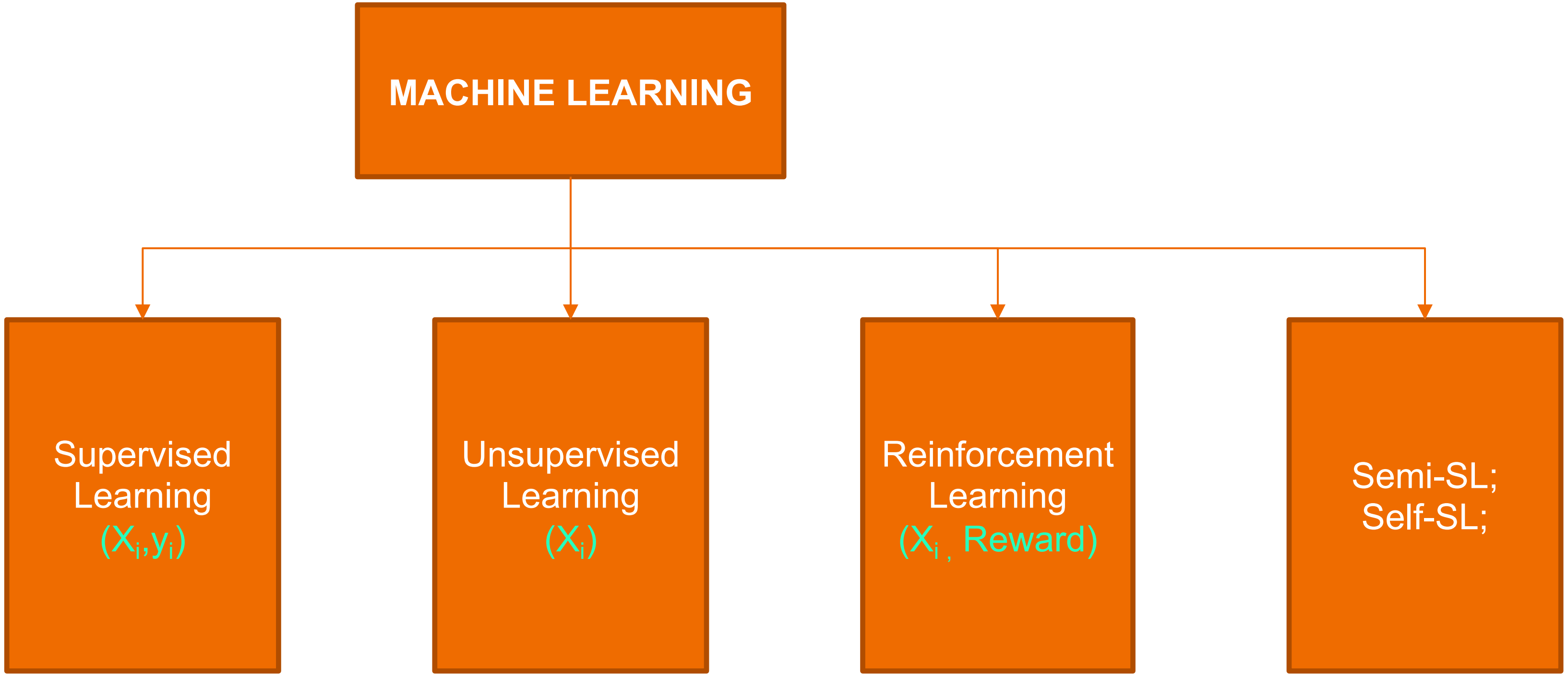
Applications of RL

- Game Playing
- Robotics Control
- Autonomous Vehicles
- Resource Management
- Trading policy
- Conversational agents
- …

Outline

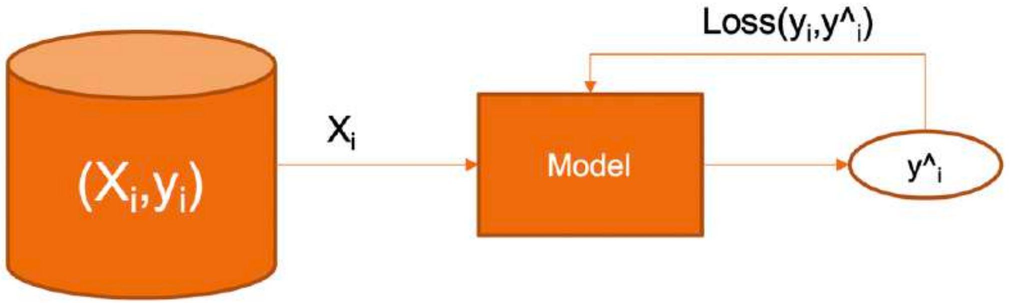
1. Introduction to Reinforcement Learning (RL)
    - Distinguish RL with SP
    - Formulate RL problem
    - Q-Learning Algorithm
2. Deep Reinforcement Learning
    - Why Deep RL?
    - Deep Q-Learning Algorithm
    - Policy Gradient
    - AlphaGo & AlphaGo Zero
    - Deep RL in ChatGPT

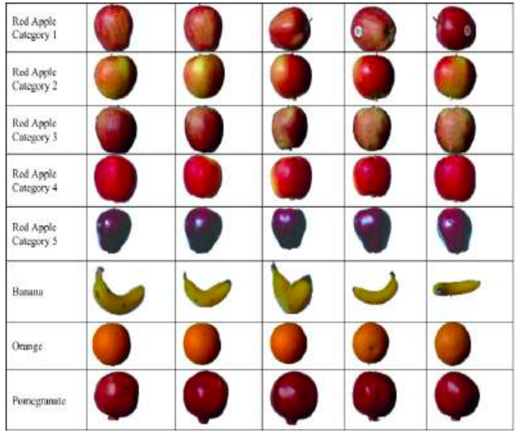
Machine Learning types


MACHINE LEARNING
Supervised Learning (X_i, y_i)
Unsupervised Learning (X_i)
Reinforcement Learning (X_i, Reward)
Semi-SL; Self-SL;

Supervised Learning?

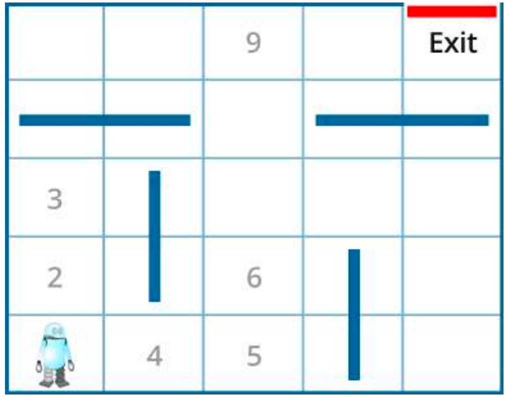
- Supervised Learning (SP): The primary goal of supervised learning is to learn a mapping between inputs and corresponding outputs based on the labeled training data.

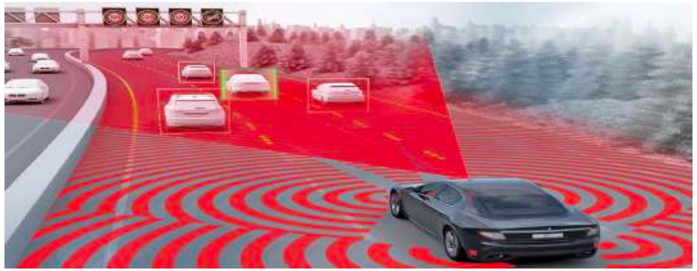




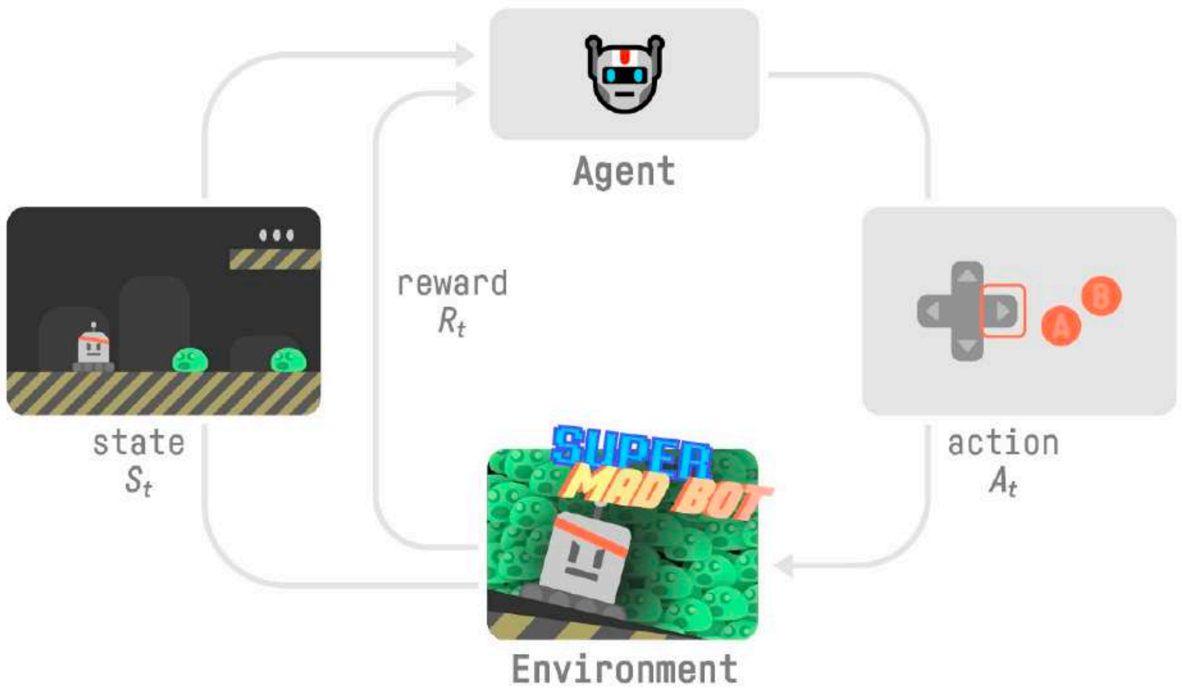
Examples of Reinforcement Learning

- RL is used for decision making problem;
- The model learns to make a sequence of decisions that lead to desirable outcomes over time.





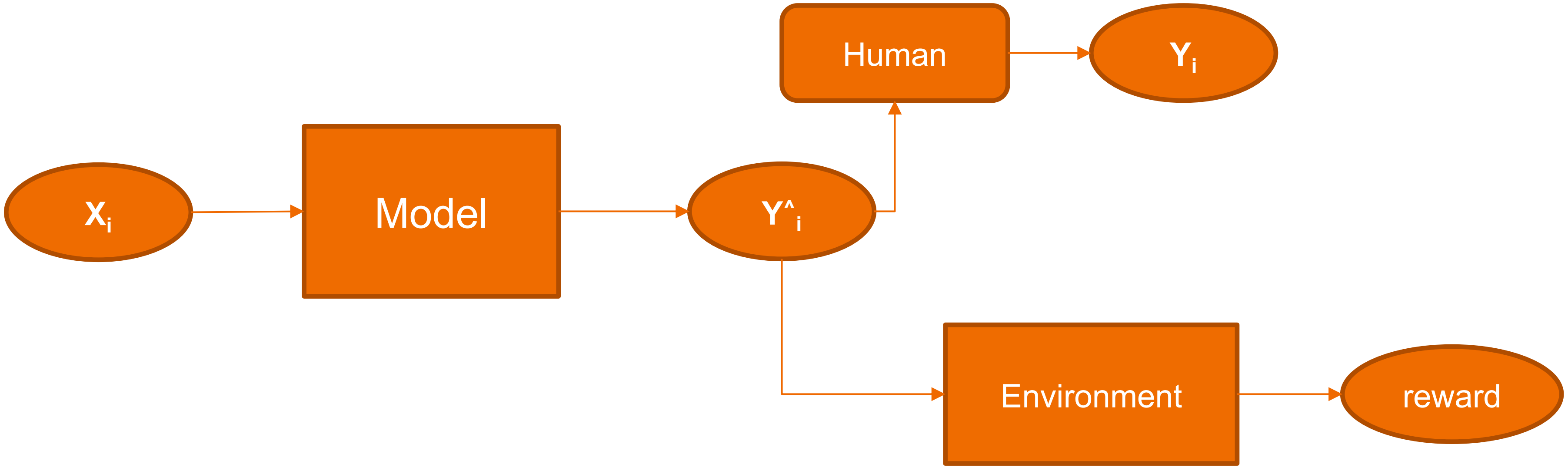
Examples of Reinforcement Learning



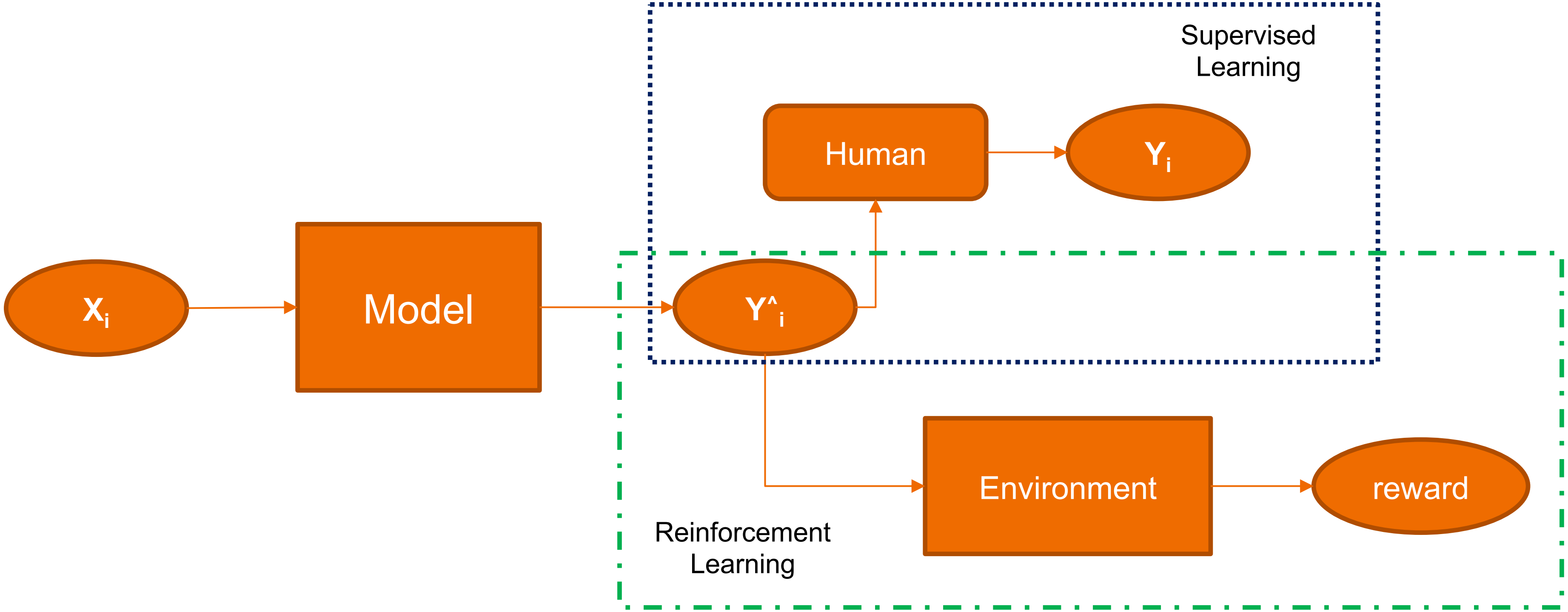
Supervised Learning vs Reinforcement Learning

- Supervision/Instruction
  - Supervised Learning: labels by human
  - Reinforcement Learning: rewards by the environment
- Goal
  - Supervised Learning: Learn model to map X to Y
  - RL: Learn a playing policy (sequence of decisions) to win game
    - Adjust, not fully supervised
    - Gradually adjust to the dynamic changes (of the environment)

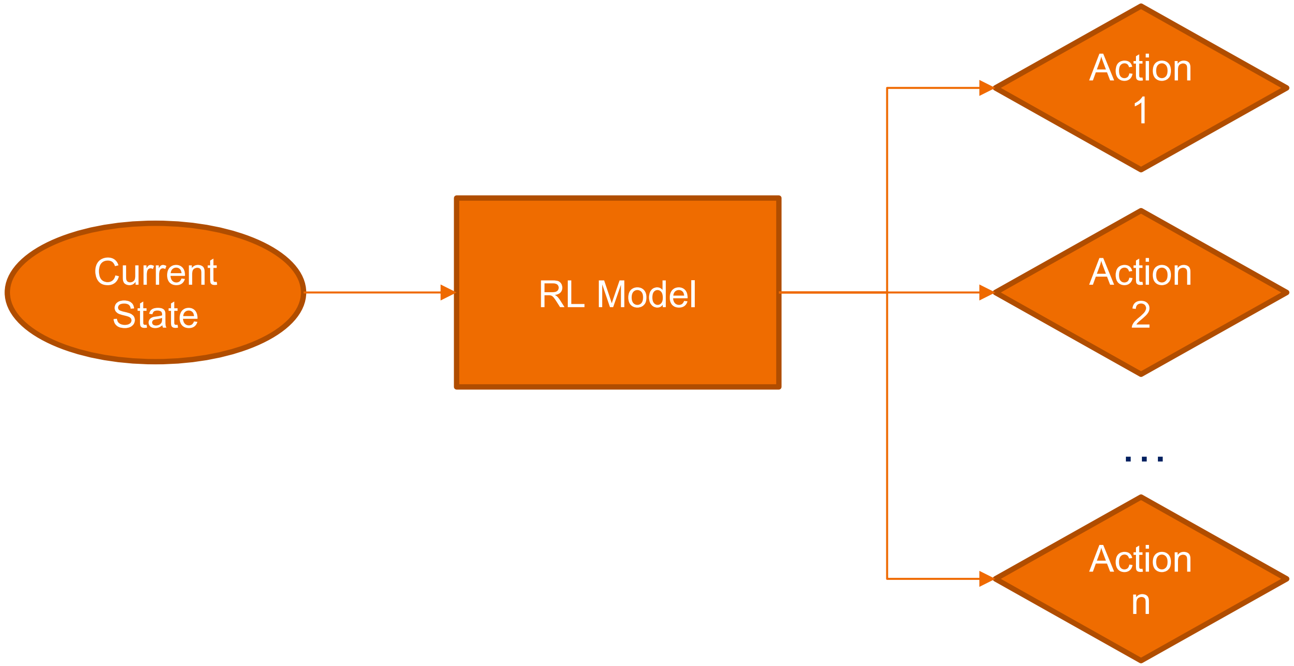
Supervised Learning vs Reinforcement Learning



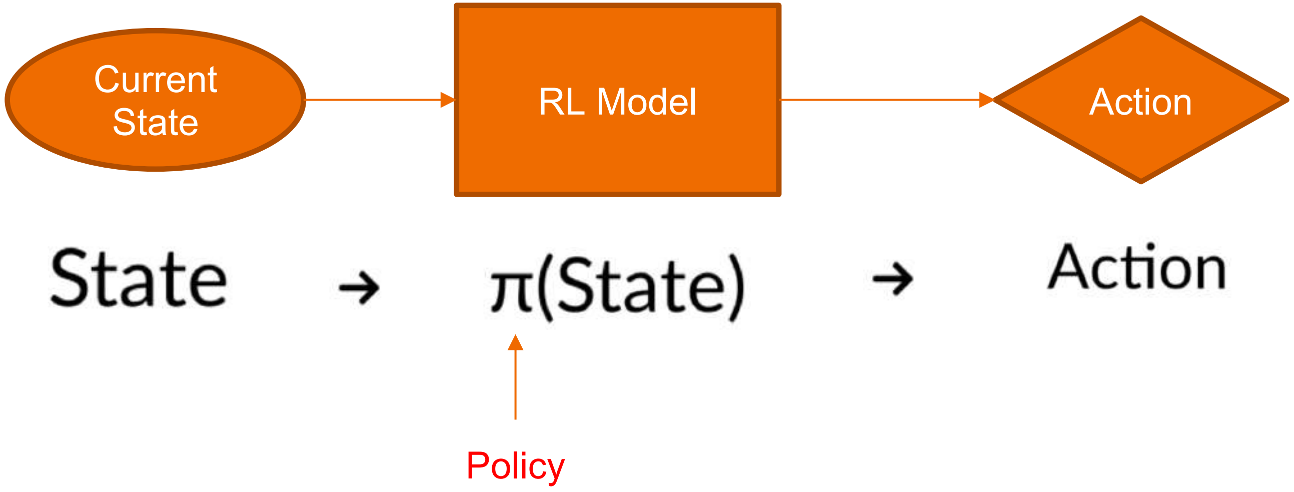
Supervised Learning vs Reinforcement Learning



What is the mission of RL model



What is the mission of RL model



Policy Function of RL

There are two types of policy of determining the next action:

•*Deterministic*: a policy at a given state **will always return the same action.**

$$a = \pi(s)$$

•*Stochastic*: output **a probability distribution over actions.**

$$\pi(a|s) = \boxed{P[A|s]}$$

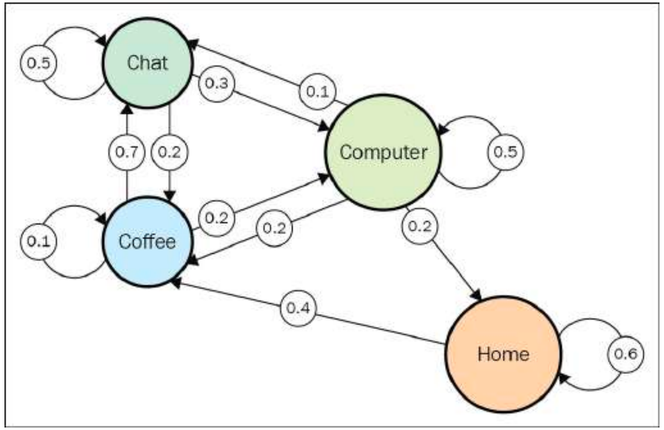
Markov Decision Process (MDP)

A Markov decision process is a 4-tuple (S, A, P_a, R_a), where:

- S is a set of states called the *state space*,
- A is a set of actions called the *action space* (alternatively, A_s is the set of actions available from state s),
- $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t + 1$,
- $R_a(s, s')$ is the immediate reward (or expected immediate reward) received after transitioning from state s to state s', due to action a

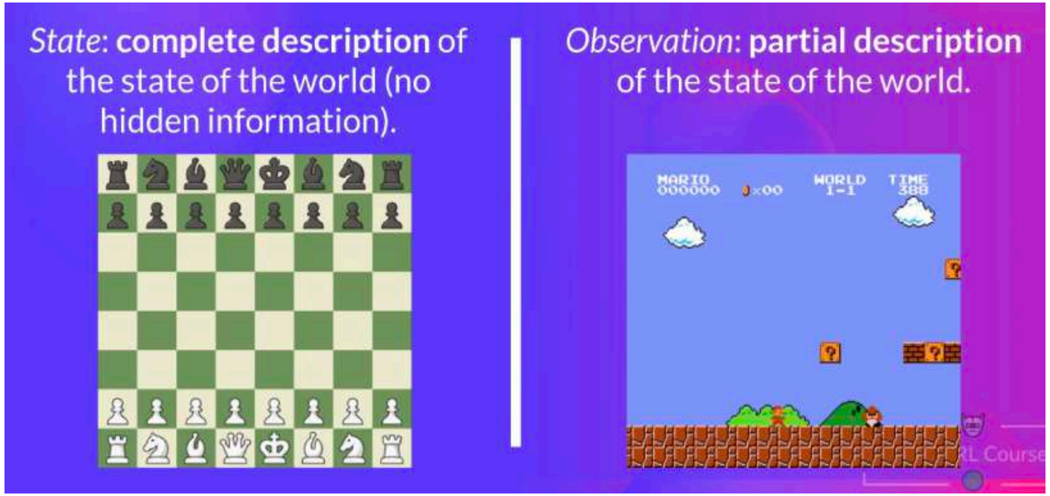
The state and action spaces may be finite or infinite, for example the set of real numbers. Some processes with countably infinite state and action spaces can be reduced to ones with finite state and action spaces.[3]

A policy function π is a (potentially probabilistic) mapping from state space (S) to action space (A).



What is the State?

- Observations/States are the **information the Agent/Model gets from the environment.**


State: **complete description** of the state of the world (no hidden information).
Observation: **partial description** of the state of the world.
MARIO 000000
x00
WORLD 1-1
TIME 388
RL Course

What are the Action?

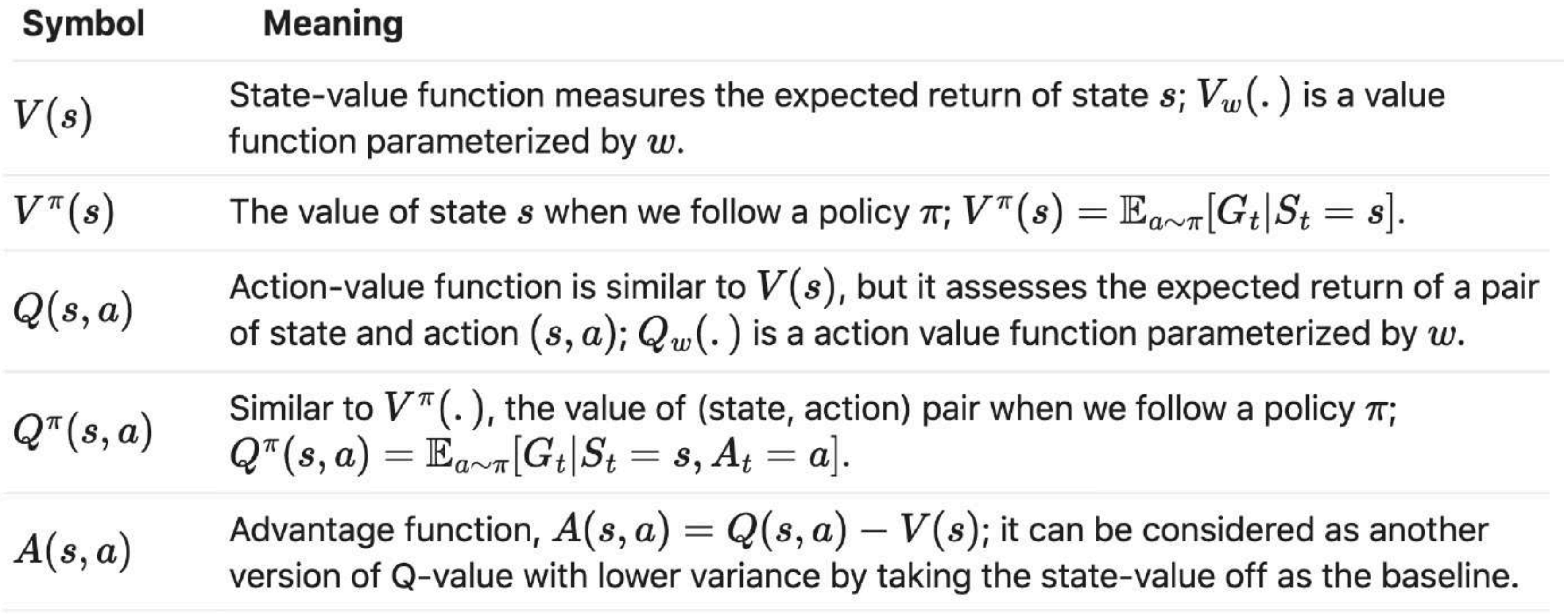
- The Action space is the set of **all possible actions in an environment.**
- The actions can come from a *discrete* or *continuous space*:

Action Space
Discrete: finite number of possible actions
Continuous: infinite number of possible actions
MARIO 000000
x00
WORLD 1-1
TIME 388
Deep RL Course

Notations

Symbol	Meaning
$s \in \mathcal{S}$	States.
$a \in \mathcal{A}$	Actions.
$r \in \mathcal{R}$	Rewards.
S_t, A_t, R_t	State, action, and reward at time step t of one trajectory. I may occasionally use s_t, a_t, r_t as well.
γ	Discount factor; penalty to uncertainty of future rewards; $0 < \gamma \leq 1$.
G_t	Return; or discounted future reward; $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$.
$P(s', r \vert s, a)$	Transition probability of getting to the next state s' from the current state s with action a and reward r.
$\pi(a \vert s)$	Stochastic policy (agent behavior strategy); $\pi_\theta(.)$ is a policy parameterized by θ.

Notations



Symbol	Meaning
$V(s)$	State-value function measures the expected return of state s; $V_w(.)$ is a value function parameterized by w.
$V^\pi(s)$	The value of state s when we follow a policy π; $V^\pi(s) = \mathbb{E}_{a\sim\pi}[G_t \| S_t = s]$.
$Q(s,a)$	Action-value function is similar to $V(s)$, but it assesses the expected return of a pair of state and action (s,a); $Q_w(.)$ is a action value function parameterized by w.
$Q^\pi(s,a)$	Similar to $V^\pi(.)$, the value of (state, action) pair when we follow a policy π; $Q^\pi(s,a) = \mathbb{E}_{a\sim\pi}[G_t \| S_t = s, A_t = a]$.
$A(s,a)$	Advantage function, $A(s,a) = Q(s,a) - V(s)$; it can be considered as another version of Q-value with lower variance by taking the state-value off as the baseline.

What is the Objective Function of RL?

- The Cumulative Reward at each time step **t** can be written as:

$$
\begin{aligned}
G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots + \gamma^{T-1} r_T \\
&= r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \dots + \gamma^{T-2} r_T) \\
&= r_{t+1} + \gamma(G_{t+1})
\end{aligned}
$$

- The rewards that come sooner (at the beginning of the game) **are more likely to happen** since they are more predictable than the long-term future reward.

Q-Learning Algorithm

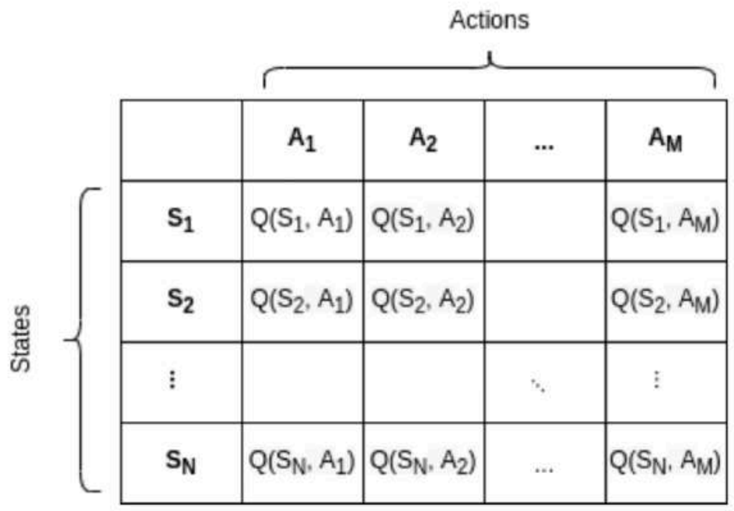
- Q-learning was introduced by Chris Watkins in 1989. A convergence proof was presented by Watkins and Peter Dayan in 1992.

Bellman Equation

$$Q(S_t, A_t) = R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$$

Q-Learning Algorithm

- Q-learning was introduced by Chris Watkins in 1989.
- Q-learning finds an optimal policy in the sense of maximizing the expected value.
- Q-learning is an off-policy Temporal Difference (TD) control algorithm


Actions
States

	A_1	A_2	...	A_M
S_1	$Q(S_1, A_1)$	$Q(S_1, A_2)$		$Q(S_1, A_M)$
S_2	$Q(S_2, A_1)$	$Q(S_2, A_2)$		$Q(S_2, A_M)$
$\vdots$			$\ddots$	$\vdots$
S_N	$Q(S_N, A_1)$	$Q(S_N, A_2)$	...	$Q(S_N, A_M)$

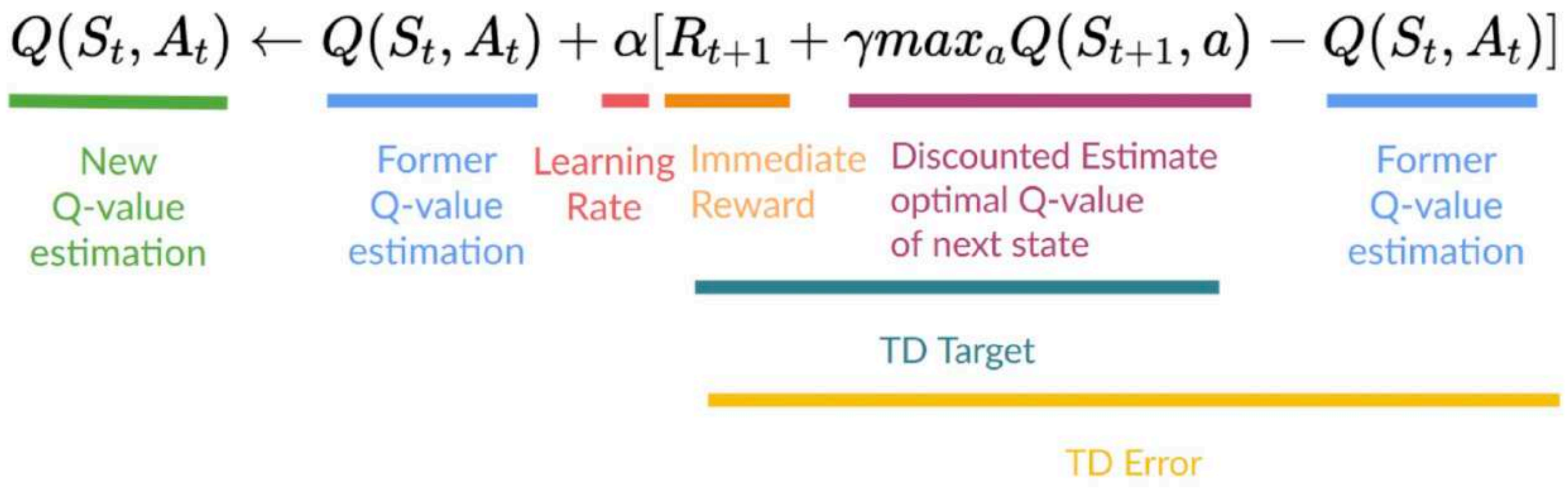
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Q-Learning Algorithm

$$\overset{\text{new}}{Q(S_t, A_t)} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[\underbrace{\underbrace{R_{t+1} + \gamma \max_a Q(S_{t+1}, a)}_{\text{new}} - \underbrace{Q(S_t, A_t)}_{\text{old}}}_{\text{Difference}} \right]$$

Q-Learning Algorithm: update the Q-Table



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

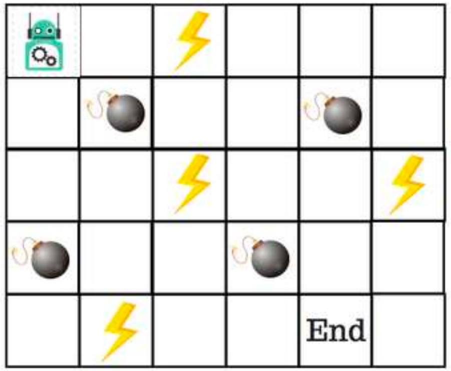
New Q-value estimation
Former Q-value estimation
Learning Rate
Immediate Reward
Discounted Estimate optimal Q-value of next state
Former Q-value estimation
TD Target
TD Error

https://huggingface.co/learn/deep-rl-course/unit3/deep-q-algorithm?fw=pt

Q-Learning Algorithm: an Example

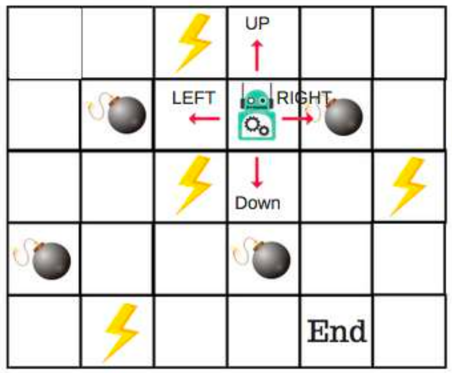
The scoring/reward system is as below:

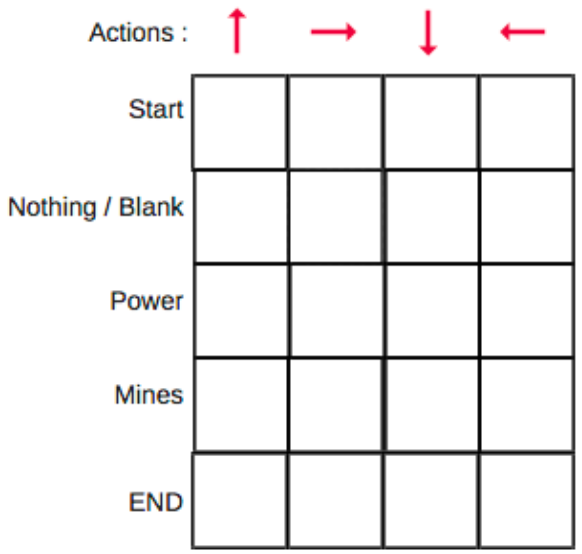
1. The robot loses 1 point at each step. This is done so that the robot takes the shortest path and reaches the goal as fast as possible.
2. If the robot steps on a mine, the point loss is 100 and the game ends.
3. If the robot gets power ⚡, it gains 1 point.
4. If the robot reaches the end goal, the robot gets 100 points.



https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/

Q-Learning Algorithm: Q-Table



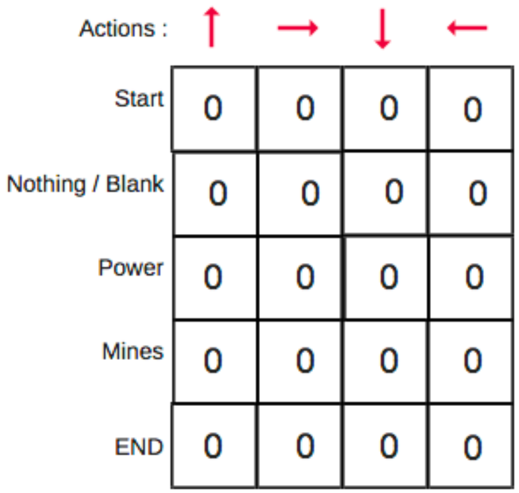


Actions :	↑	→	↓	←
Start				
Nothing / Blank				
Power				
Mines				
END				

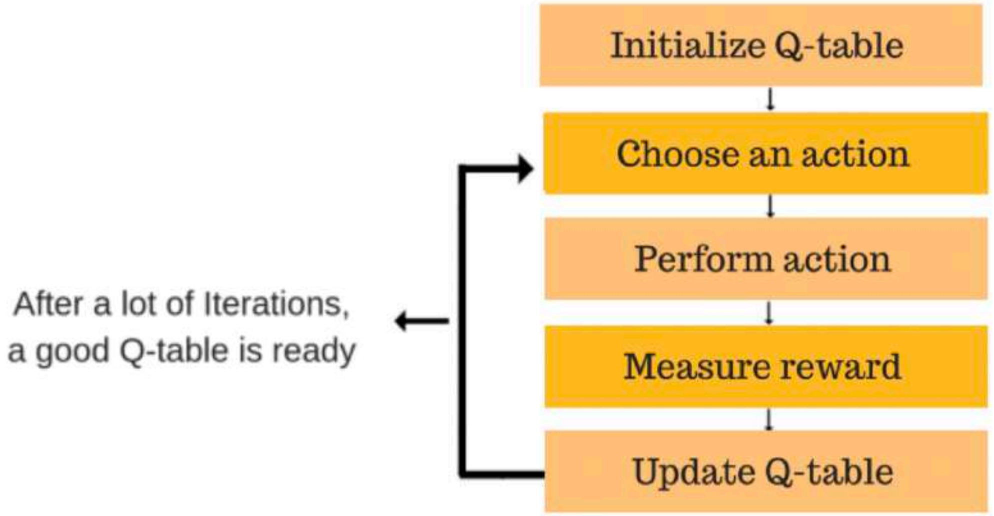
Q-Learning Algorithm:

Step 1: initialize the Q-Table

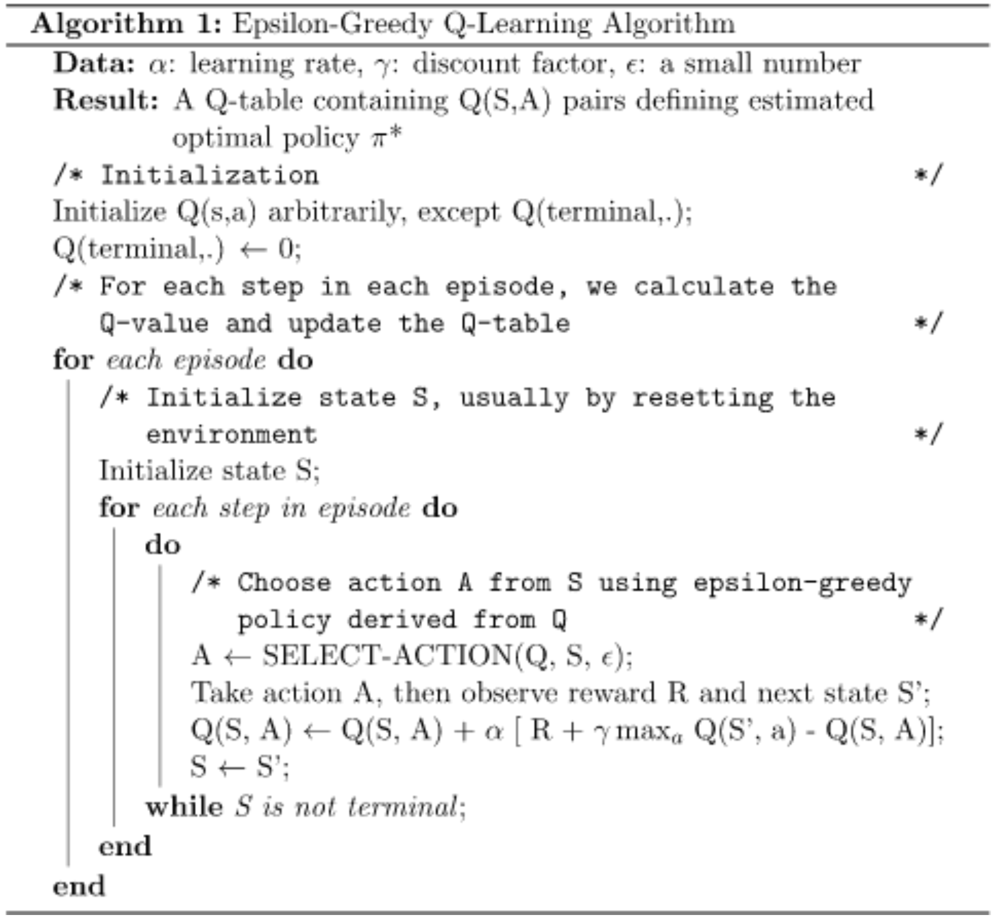
We will initialise the values at 0.



Actions :	↑	→	↓	←
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0



Q-Learning Algorithm:



```
Algorithm 1: Epsilon-Greedy Q-Learning Algorithm
Data: $\alpha$: learning rate, $\gamma$: discount factor, $\epsilon$: a small number
Result: A Q-table containing Q(S,A) pairs defining estimated
        optimal policy $\pi$*
/* Initialization                                              */
Initialize Q(s,a) arbitrarily, except Q(terminal,.);
Q(terminal,.) $\leftarrow$ 0;
/* For each step in each episode, we calculate the
   Q-value and update the Q-table                              */
for each episode do
    /* Initialize state S, usually by resetting the
       environment                                             */
    Initialize state S;
    for each step in episode do
        do
            /* Choose action A from S using epsilon-greedy
               policy derived from Q                           */
            A $\leftarrow$ SELECT-ACTION(Q, S, $\epsilon$);
            Take action A, then observe reward R and next state S';
            Q(S, A) $\leftarrow$ Q(S, A) + $\alpha$ [ R + $\gamma$ max$_a$ Q(S', a) - Q(S, A)];
            S $\leftarrow$ S';
        while S is not terminal;
    end
end
```

Outline

Why Deep RL?

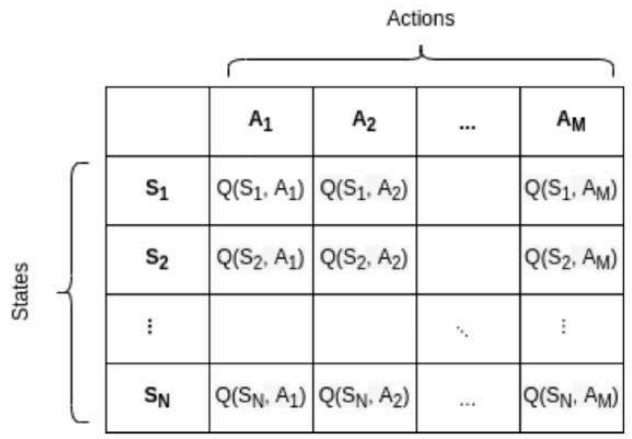
How to do with the state?

- State is an image
- State is very complicated
- The state space is very large, infinite

How to do with the action?

- Action space is very large
- Action in Generative Models

Conventional RL


Actions
States

	A_1	A_2	...	A_M
S_1	$Q(S_1, A_1)$	$Q(S_1, A_2)$		$Q(S_1, A_M)$
S_2	$Q(S_2, A_1)$	$Q(S_2, A_2)$		$Q(S_2, A_M)$
$\vdots$			$\ddots$	$\vdots$
S_N	$Q(S_N, A_1)$	$Q(S_N, A_2)$	...	$Q(S_N, A_M)$

RL vs Deep RL

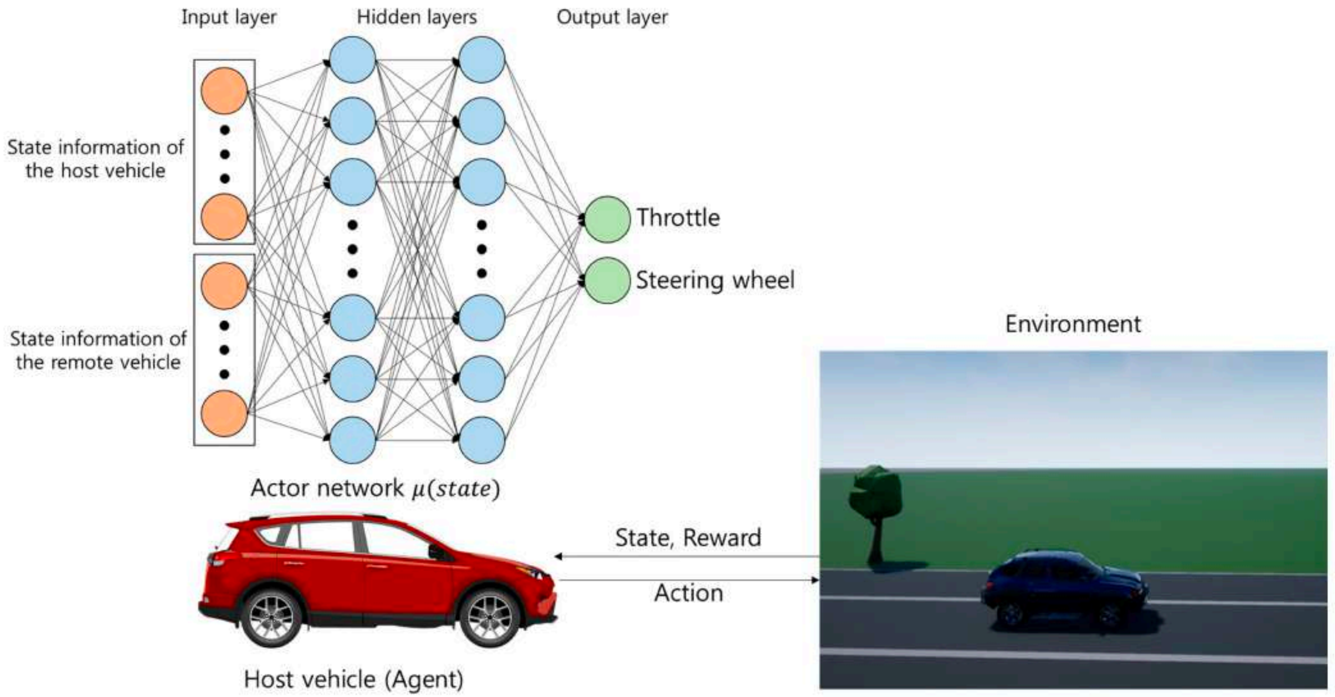
- In many practical decision-making problems, the **states** of the Markov Decision Process (MDP) are high-dimensional (e.g., images or texts)
  - We need **a model** to represent the states and actions
- Deep Reinforcement Learning algorithms incorporate deep learning to solve such MDPs:

Representing the policy as a deep neural network. $\pi(a|s)$

Deep RL paper (2013): Playing Atari with Deep Reinforcement Learning
https://arxiv.org/abs/1312.5602

Example

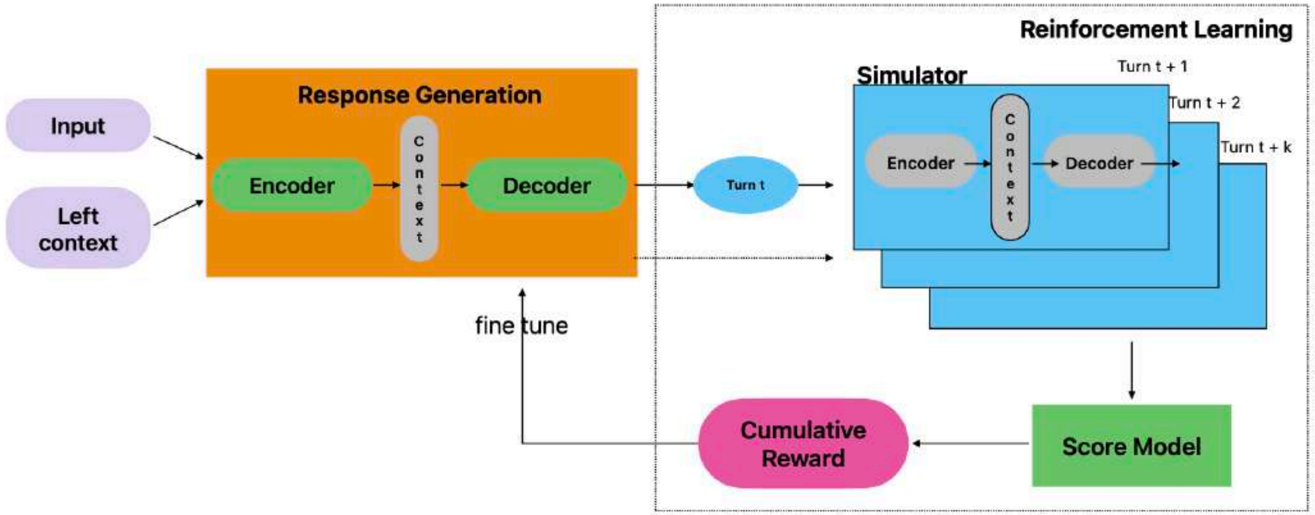
Autonomous Driving



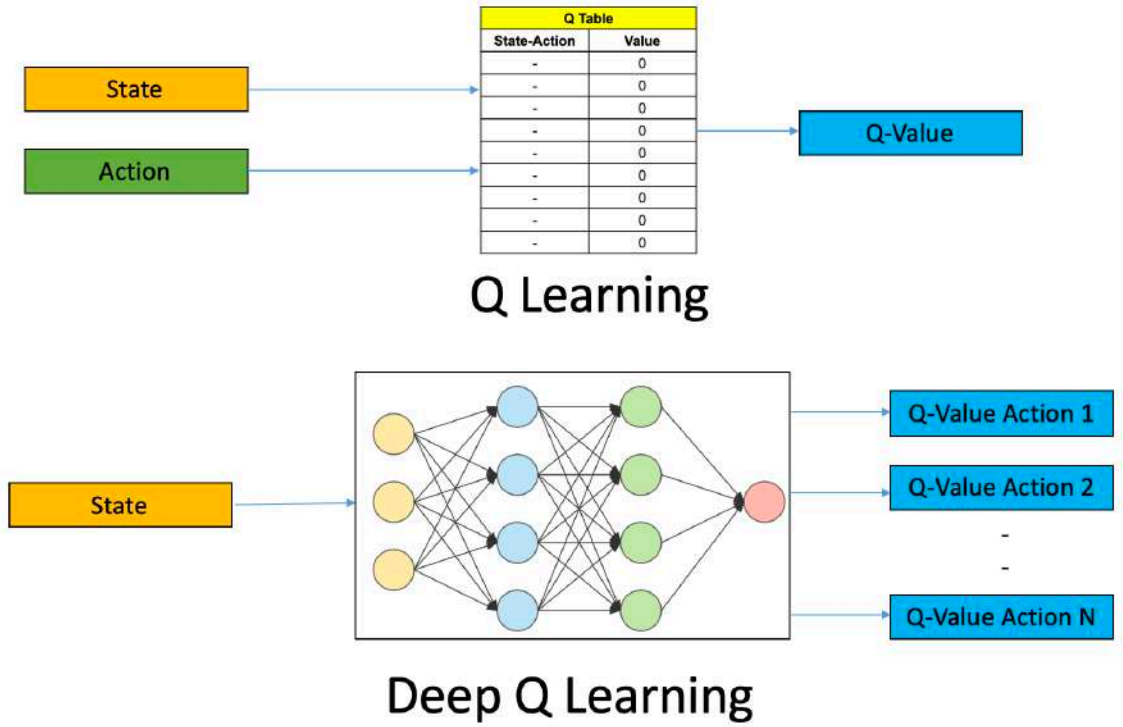
https://www.mdpi.com/2079-9292/8/5/543

Example

Conversational Agent



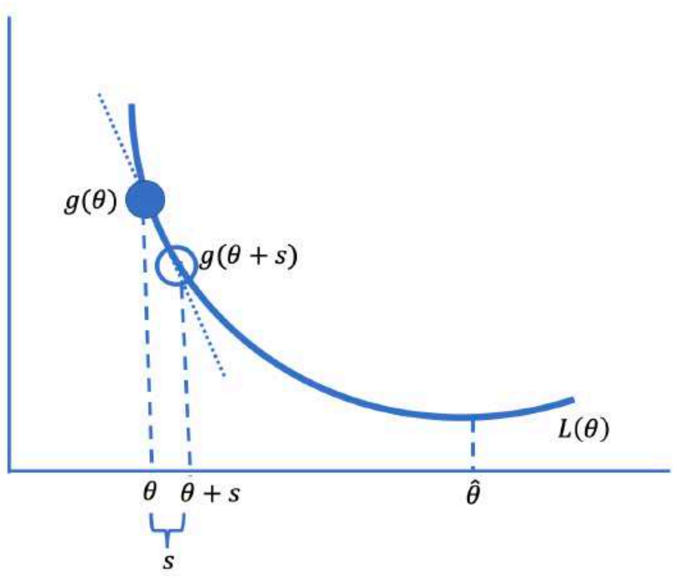
Deep RL



https://viblo.asia/p/gioi-thieu-ve-hoc-tang-cuong-va-ung-dung-deep-q-learning-choi-game-cartpole-Az45bYy6lxY

Gradient Descent ALgorithm

$$Objective\ function\ : J(\theta)$$
$$Gradient : \nabla_\theta J(\theta)$$
$$Update : \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$



Deep Q-Learning Algorithm (1995)

- **Temporal Difference**: $Q(S_t, A_t) = R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha TD_t(a,s)$$

$$TD(a,s) = R(s,a) + \gamma \max_{a'} Q(s',a') - Q_{t-1}(s,a)$$

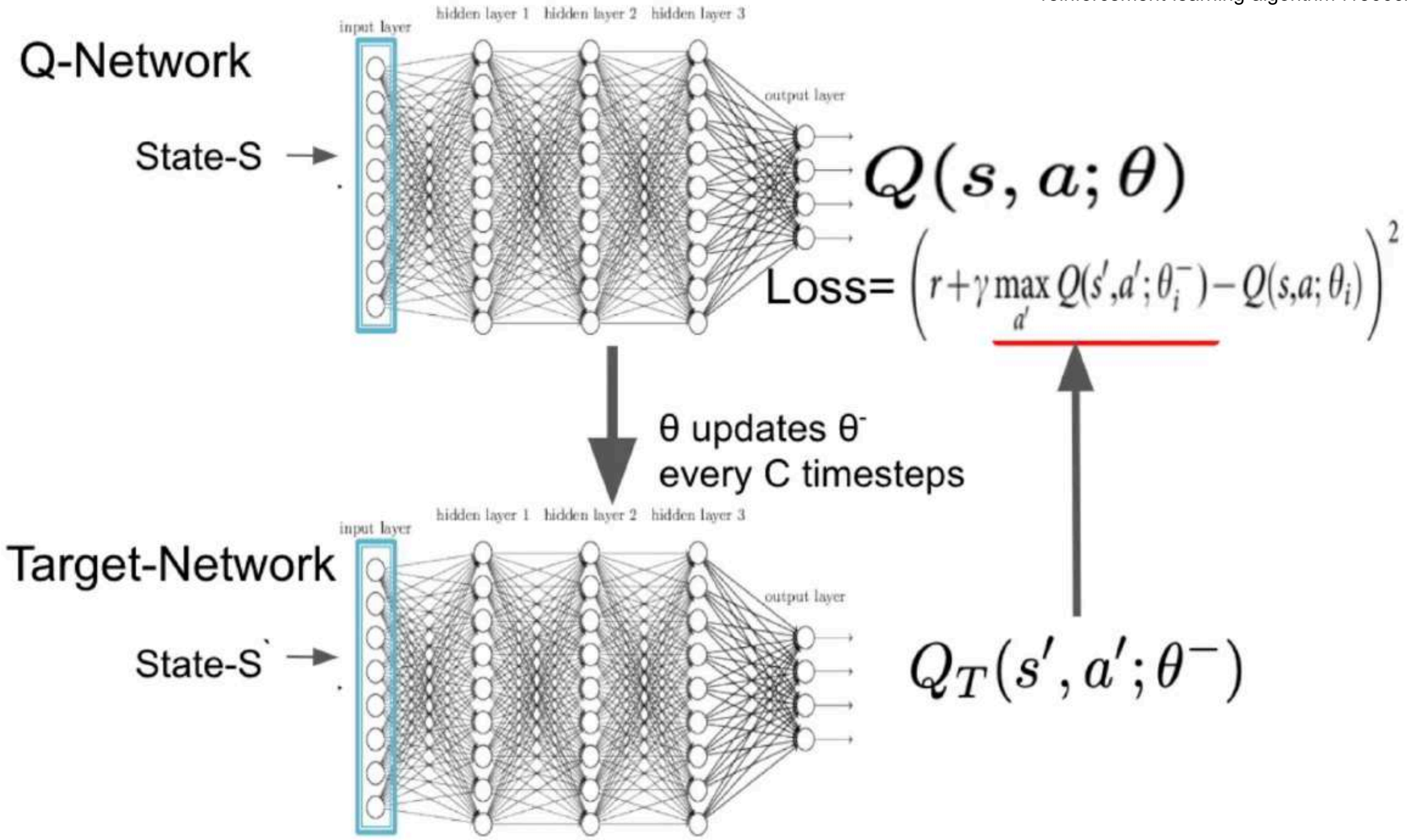
- **Loss function?**

In Deep Q-Learning, we create a loss function that compares our Q-value prediction and the Q-target and uses gradient descent to update the weights of our Deep Q-Network to approximate our Q-values better.

Deep Q-Learning Algorithm

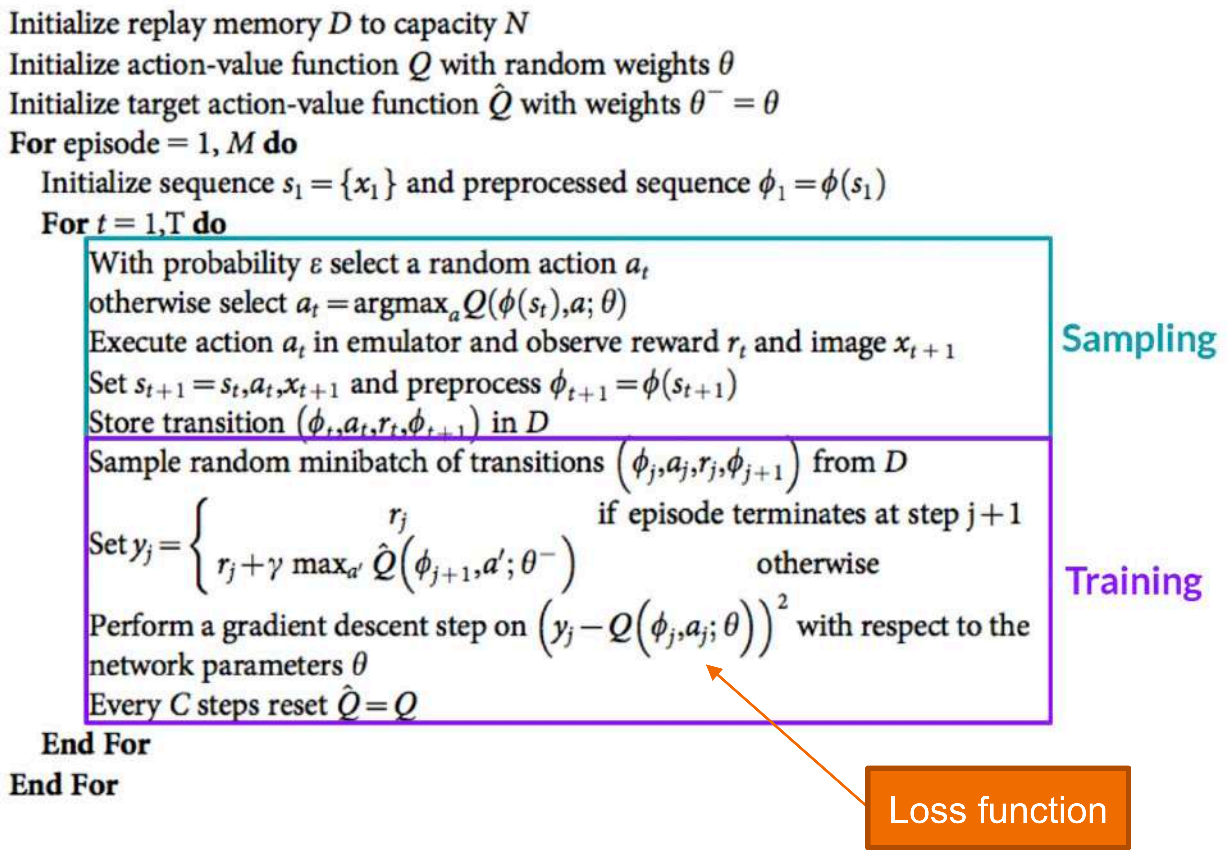
$$\underbrace{Q(S_t, A_t)}_{\text{New Q-value estimation}} \leftarrow \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}} + \underbrace{\alpha}_{\text{Learning Rate}} \underbrace{[\underbrace{\underbrace{R_{t+1}}_{\text{Immediate Reward}} + \underbrace{\gamma max_a Q(S_{t+1}, a)}_{\text{Discounted Estimate optimal Q-value of next state}}}_{\text{TD Target}} - \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}}]}_{\text{TD Error}}$$

https://huggingface.co/learn/deep-rl-course/unit3/deep-q-algorithm?fw=pt https://arshren.medium.com/deep-q-learning-a-deep-reinforcement-learning-algorithm-f1366cf1b53d

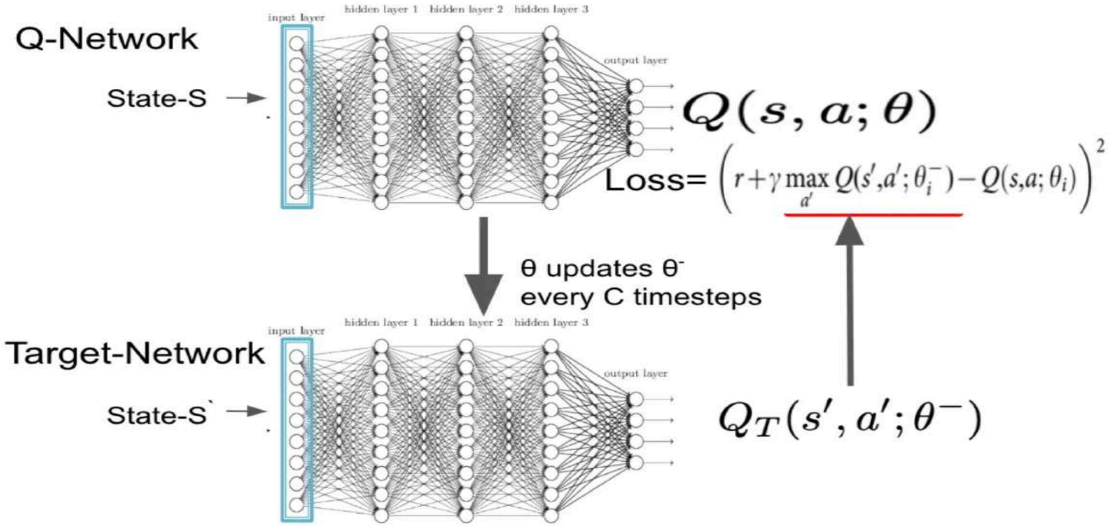


Deep Q-Learning Algorithm

https://huggingface.co/learn/deep-rl-course/unit3/deep-q-algorithm?fw=pt



```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ do
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    For $t = 1$,T do
        With probability $\varepsilon$ select a random action $a_t$
        otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
        network parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    End For
End For
```

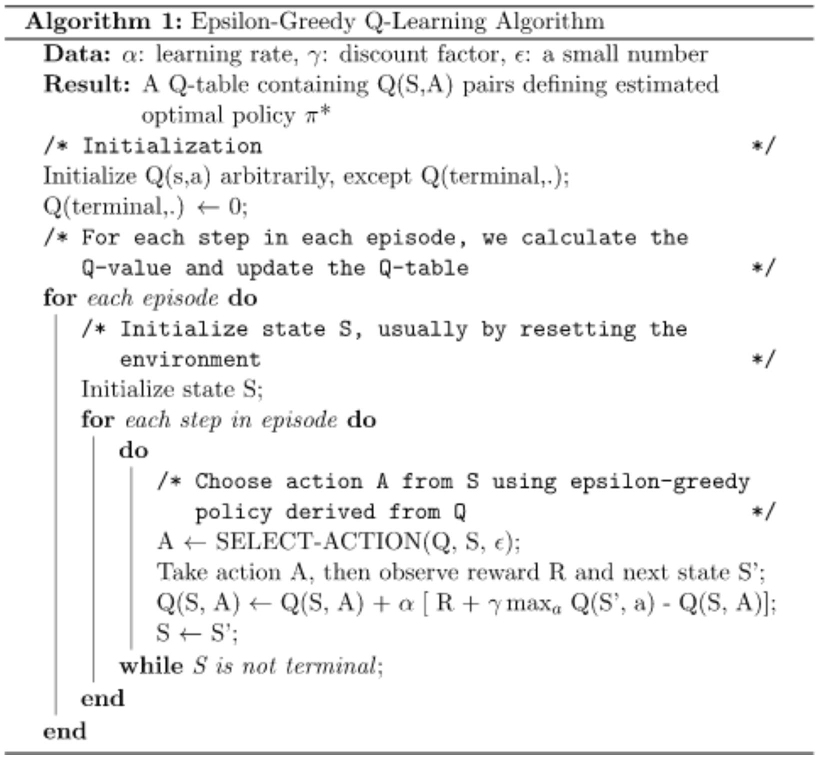


Deep Q Network(DQN)

$$\theta \leftarrow \theta + \alpha\left(r + \gamma \max_{a'} Q_T(s', a'; \theta^-) - Q(s, a; \theta)\right) \nabla_\theta Q(s, a; \theta)$$

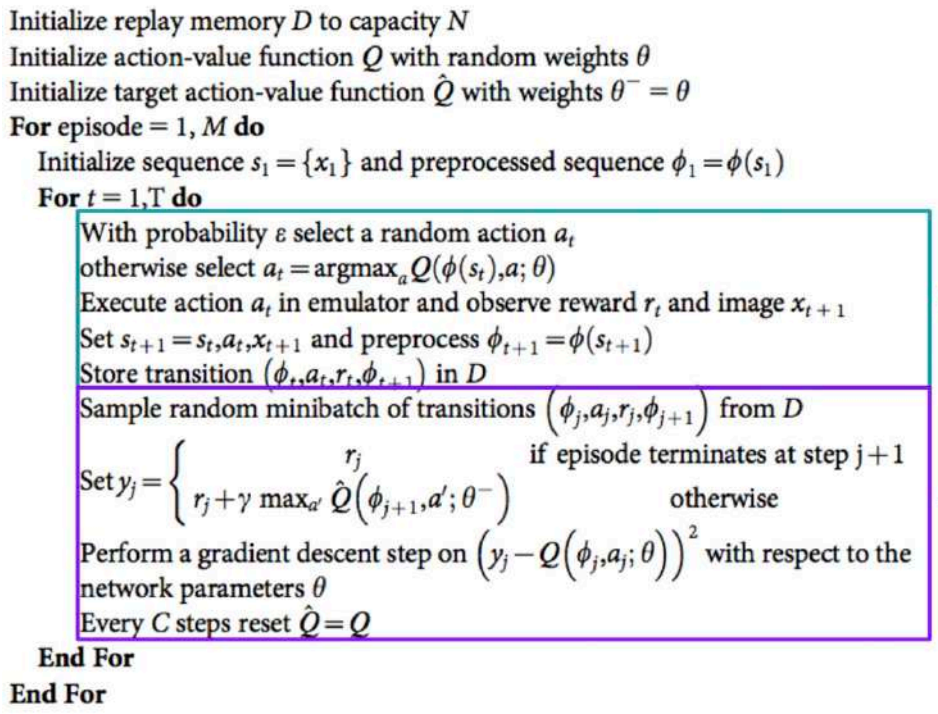
Q Network weight update

Q-Learning



```
Algorithm 1: Epsilon-Greedy Q-Learning Algorithm
  Data: $\alpha$: learning rate, $\gamma$: discount factor, $\epsilon$: a small number
  Result: A Q-table containing Q(S,A) pairs defining estimated
          optimal policy $\pi$*
  /* Initialization                                                  */
  Initialize Q(s,a) arbitrarily, except Q(terminal,.);
  Q(terminal,.) $\leftarrow$ 0;
  /* For each step in each episode, we calculate the
     Q-value and update the Q-table                                  */
  for each episode do
    /* Initialize state S, usually by resetting the
       environment                                                   */
    Initialize state S;
    for each step in episode do
      do
        /* Choose action A from S using epsilon-greedy
           policy derived from Q                                     */
        A $\leftarrow$ SELECT-ACTION(Q, S, $\epsilon$);
        Take action A, then observe reward R and next state S';
        Q(S, A) $\leftarrow$ Q(S, A) + $\alpha$ [ R + $\gamma \max_a$ Q(S', a) - Q(S, A)];
        S $\leftarrow$ S';
      while S is not terminal;
    end
  end
```

Deep Q-Learning



```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode = 1, $M$ do
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t$ = 1,T do
    With probability $\varepsilon$ select a random action $a_t$
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
    network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
```

Policy Gradient

- We have our policy π that has a parameter θ. This π outputs a probability distribution of actions.

$$\pi_\theta(a|s) = P[a|s]$$

- Remember that policy can be seen as an optimization problem. We must find the best parameters (θ) to maximize a score function, J(θ).

$$J(\theta) = E_{\pi\theta}[\sum \gamma r]$$

Policy Gradient

The goal of reinforcement learning is to find an optimal behavior strategy for the agent to obtain optimal rewards. The **policy gradient** methods target at modeling and optimizing the policy directly. The policy is usually modeled with a parameterized function respect to θ, $\pi_\theta(a|s)$. The value of the reward (objective) function depends on this policy and then various algorithms can be applied to optimize θ for the best reward.

The reward function is defined as:

$$J(\theta) = \sum_{s \in \mathcal{S}} d^\pi(s) V^\pi(s) = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s, a)$$

Policy Gradient (1999)

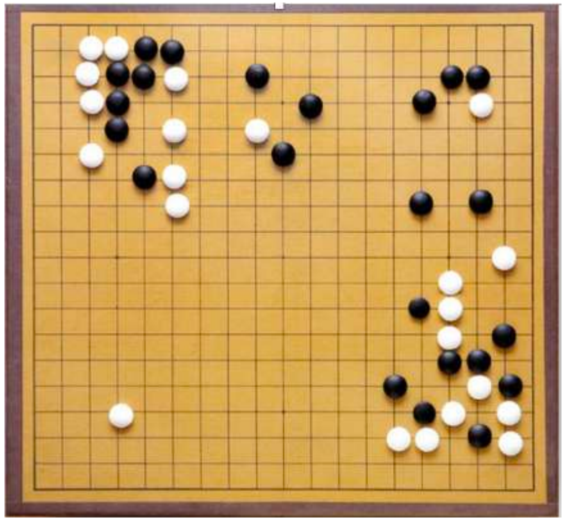
- To measure how good our policy is, we use a function called the objective function (or Policy Score Function) that calculates the expected reward of policy.

$$J_1(\theta) = E_\pi[\underbrace{G_1 = R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots}_{\text{Cumulative discounted rewards starting at start state}}] \underset{\text{Equivalent}}{=} \underbrace{E_\pi(V(s_1))}_{\text{Value of state 1}}$$

$$J_{avR}(\theta) = E_\pi(r) = \underbrace{\sum_s d(s)}_{\text{Probability that I'm in state s}} \underbrace{\sum_a \pi\theta(s,a)}_{\text{Probability that I take this action a from that state under this policy}} \underset{\text{Immediate reward that I'll get}}{R_s^a}$$

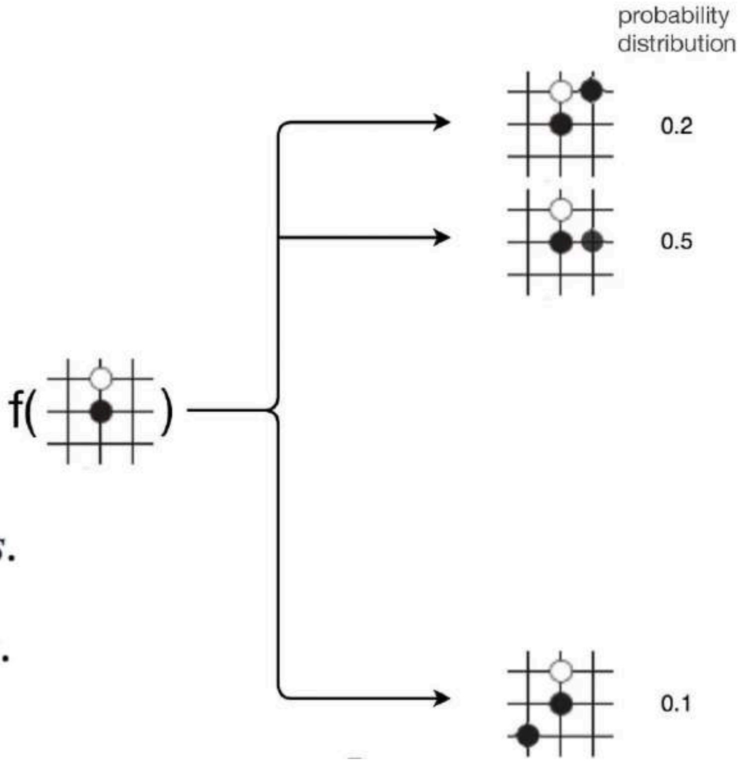
AlphaGo

- “In October 2015, AlphaGo played its first match against the reigning three-time European Champion, Mr Fan Hui. AlphaGo won the first ever game against a Go professional with a score of 5-0”.
- “AlphaGo then competed against legendary Go player Mr Lee Sedol, the winner of 18 world titles, who is widely considered the greatest player of the past decade. AlphaGo's 4-1 victory in Seoul, South Korea, on March 2016 was watched by over 200 million people worldwide.”



AlphaGo Algorithm

$$prob(y) = f(board)$$

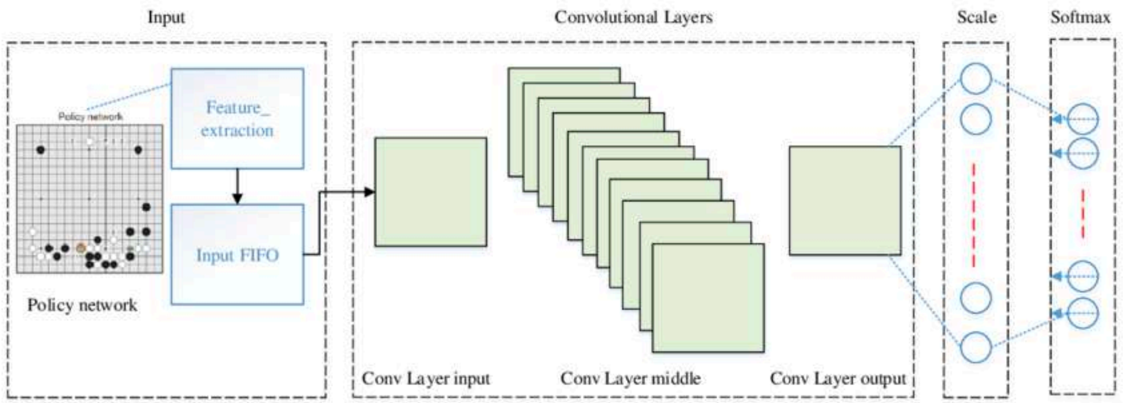


p(a|s): Policy for taken action *a* given the state *s*.

$\mathbf{p}_\sigma(\mathbf{a}|\mathbf{s})$: is the policy modeled by parameter σ.

SL policy network

- A Go board has a 19 × 19 grid. It takes a 19× 19 × 48 input feature to represent the board.

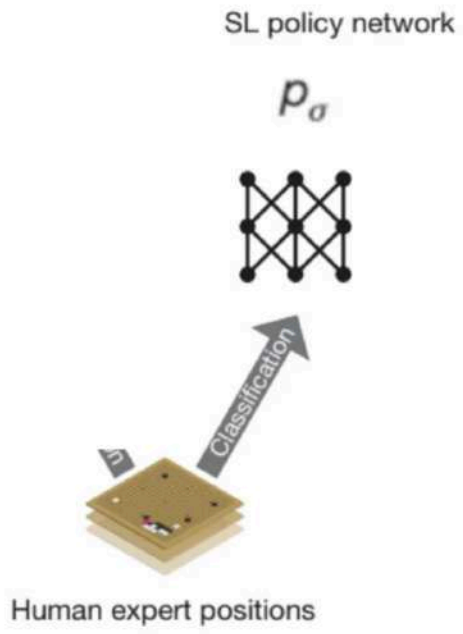


Training the SL policy network

- To train the Supervised Learning (SL) policy network, AlphaGo collects moves for 30 million board positions. Then it applies the backpropagation in deep learning to train the model parameters σ. (This is the same way you train a deep network classifier.)

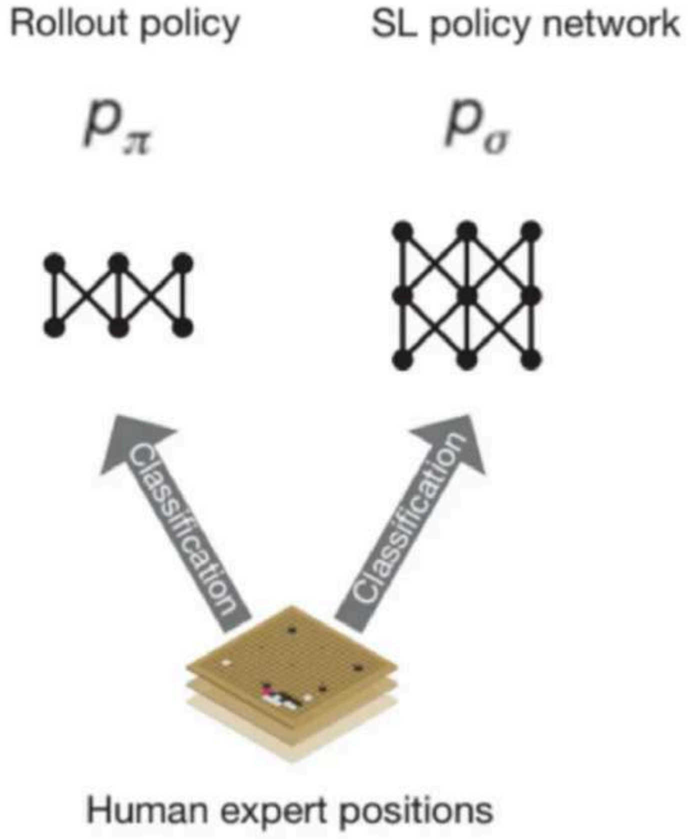
$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial\sigma}$$

- The SL policy network achieves a 57% accuracy. It sounds not too accurate but this policy network can beat an advanced amateur already.



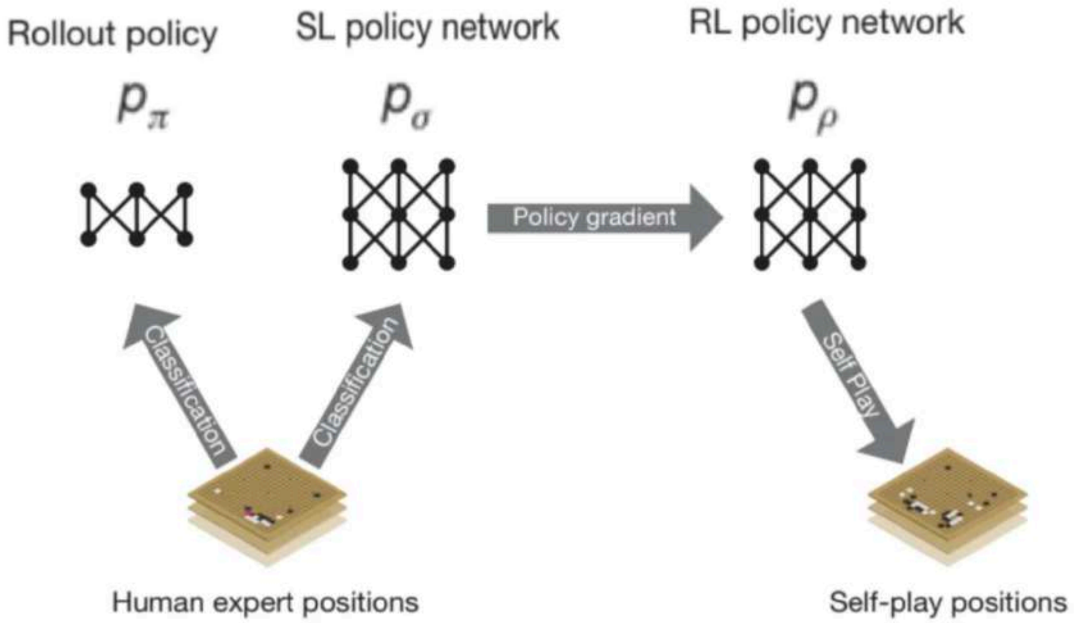
Rollout policy

- The authors create another policy called **rollout policy π** which use a more simple linear softmax classifier. This rollout policy has a lower 24.2% accuracy (compared to 55.7%) but it is 1500x faster.



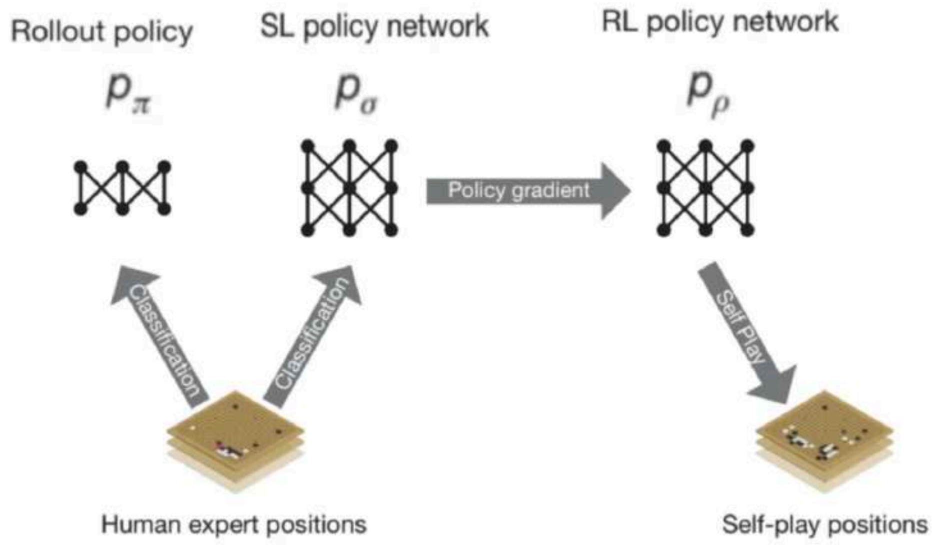
AlphaGo: start Reinforcement learning

- First, the authors duplicate SL policy network and call it **RL policy network *ρ***.



Training the RL policy network

- We play until the game is finished. For time step *t*, the game result $\mathbf{z}_t$ is equal to 1 if we win at the end or -1 if we lose.

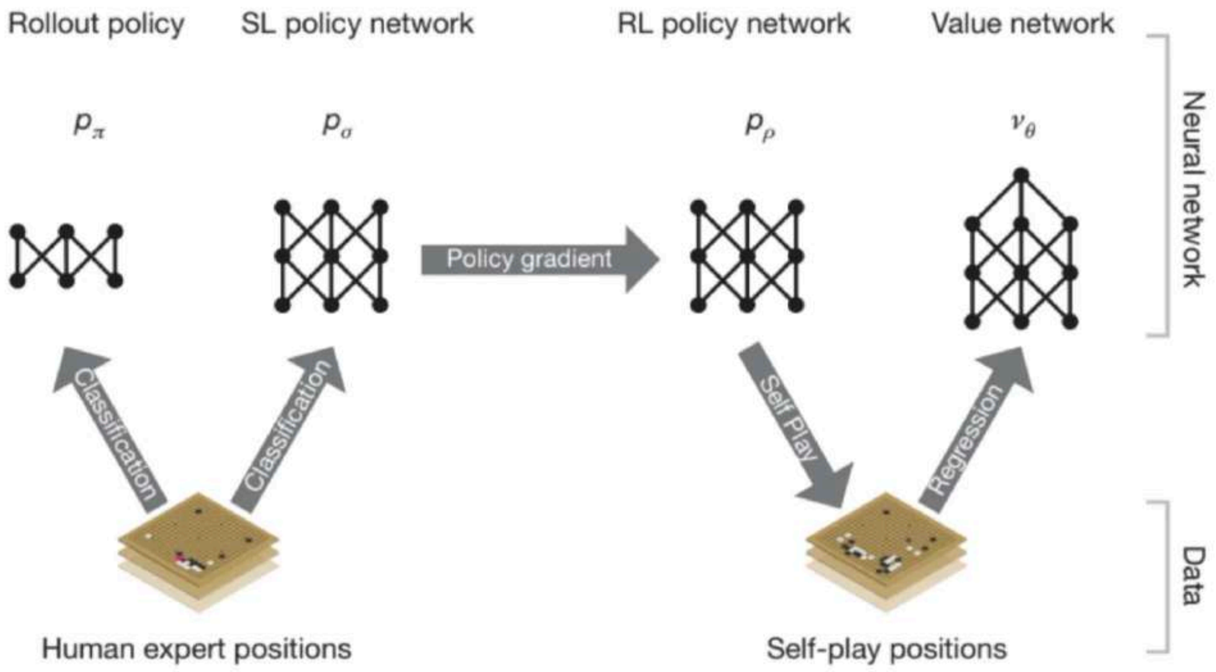


$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t$$

policy gradient RL

AlphaGo: Value Network

- In the last stage of AlphaGo training, we want to duplicate the human capability in evaluating a board position. We train a deep network to estimate the **value** of our positions.



Training the Value Network

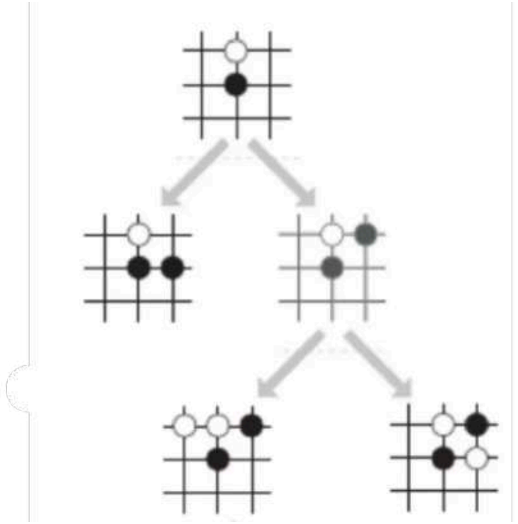
- To train the value network, we play games against each other using the same RL policy network. The design of the value network is similar to the policy network except it outputs a scalar value instead of a probability distribution. We compute the mean square error MSE of our value function ***v(s)*** with the game result ***z.***

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s))$$

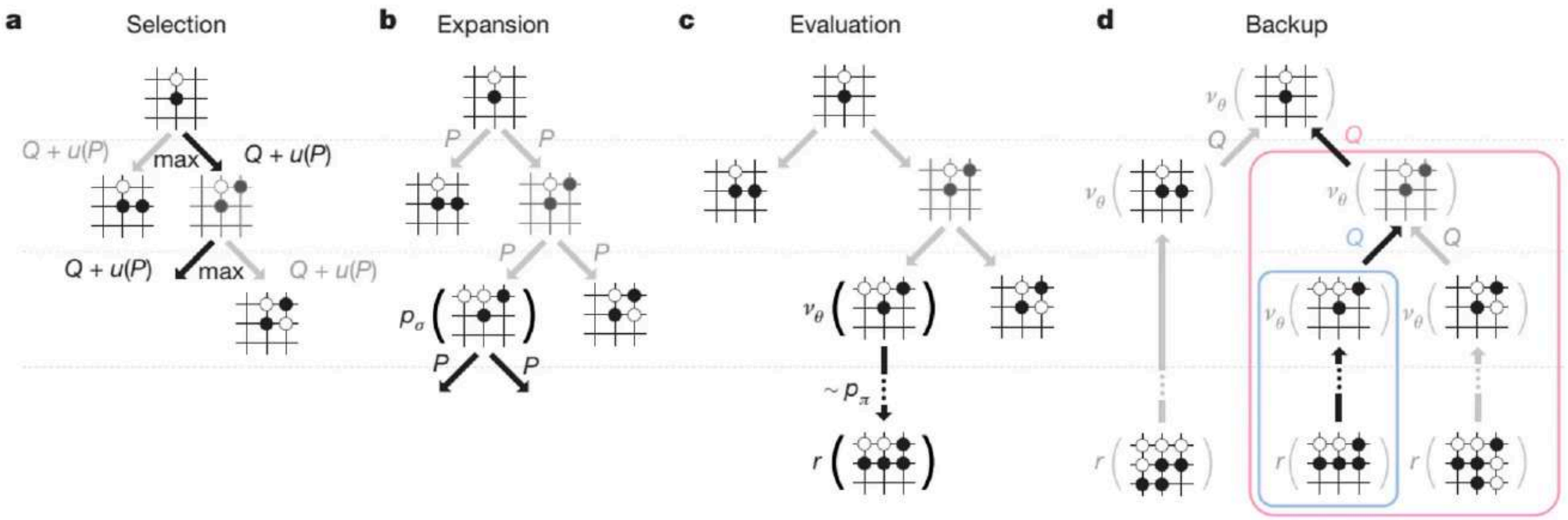
- We use the RL policy network to play more than 30 million games and collect only one position from each game into the training dataset. The AlphaGo value network is trained with 50 GPUs for one week.

Monte Carlo Tree Search (MCTS)

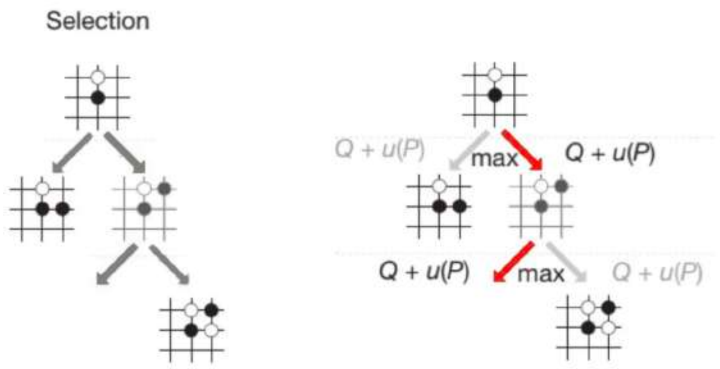
- We build a search tree recording sequences of moves



Monte Carlo Tree Search (MCTS)



Monte Carlo Tree Search (MCTS)



$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t, a) + u(s_t, a))$$

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t, a) + u(s_t, a))$$

$$u(s,a) \propto \frac{P(s,a)}{1 + N(s,a)}$$

$$P(s,a) = p_\sigma(a|s)$$

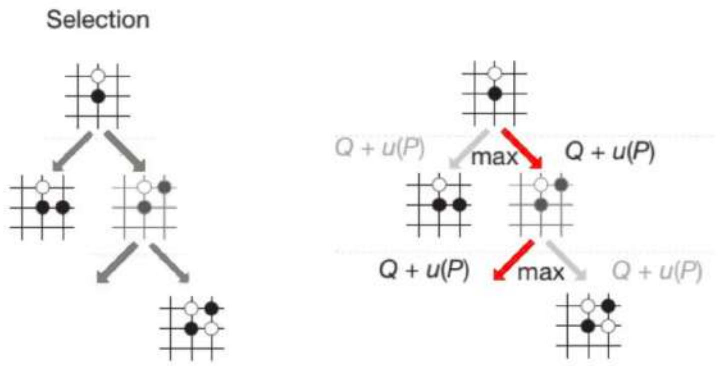
$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i)$$

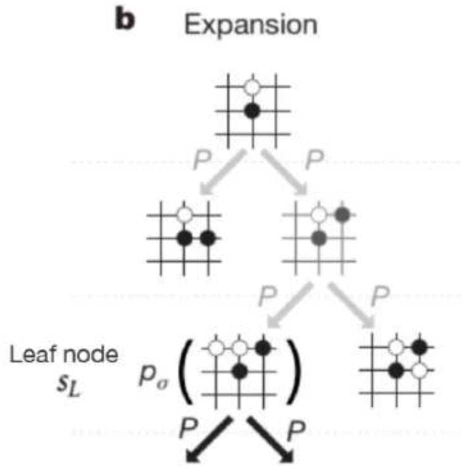
$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i)V(s_L^i)$$

- ***Q*** is for the exploitation and ***u*** is for the exploration.

Monte Carlo Tree Search (MCTS)





- We add a new leaf node (*SL*) to expand the tree. We will compute its value function from the value network.

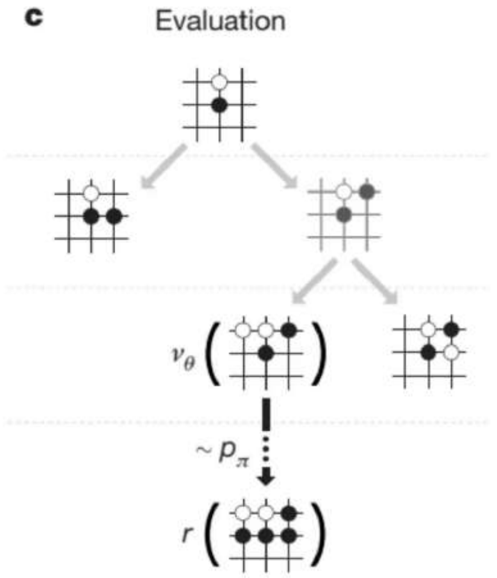
$$v_\theta(s_L)$$

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) V(s_L^i)$$

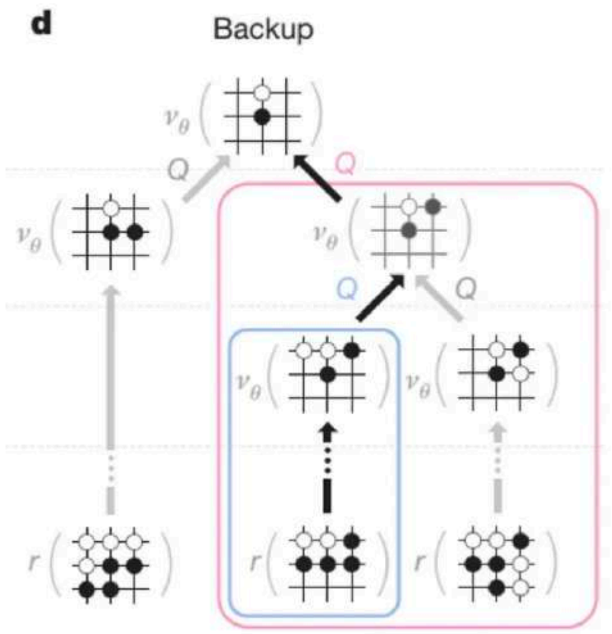
Monte Carlo Tree Search (MCTS)

- In the evaluation phase, we simulate the rest of the game using **Monte Carlo Rollout** starting from the leaf node. It means finish playing a game using a policy and find out whether you win or loss.
- In this case, we use the rollout policy instead of the SL policy or the RL policy.



Monte Carlo Tree Search (MCTS)

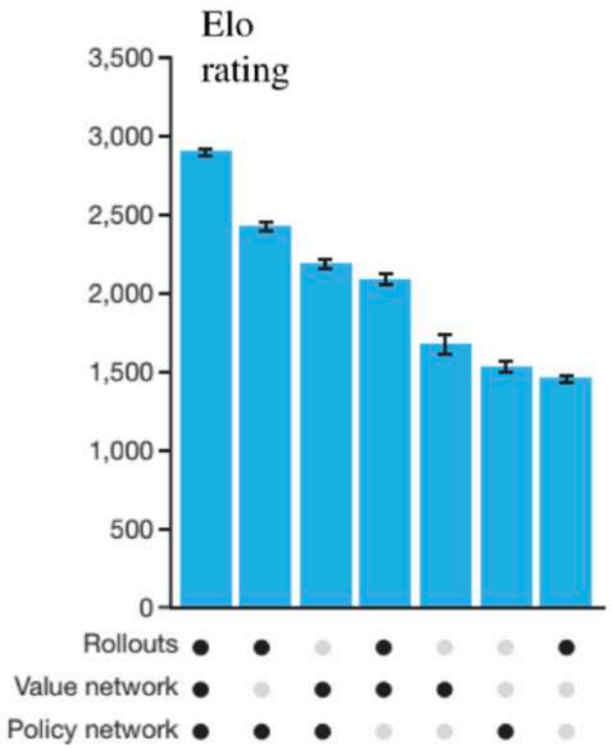
- After the evaluation, we know whether our moves win or lose the game. Now we compute ***Q*** to remember how well to make a move from the game results and the leaf node's value function.



https://jonathan-hui.medium.com/alphago-how-it-works-technically-26ddcc085319

Summary of AlphaGo

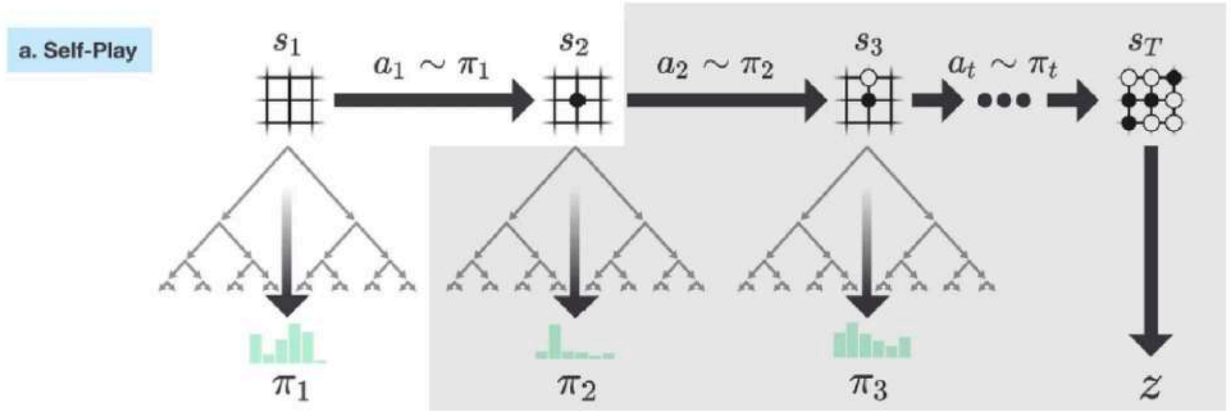
- Go players need to prioritize searches and to evaluate positions very well. Using supervised learning, we create a policy network to imitate expert moves. With this policy, we can play Go at the advanced amateur level.
- Using reinforcement learning, we apply the game results to refine the policy network further. We also train a value network to evaluate positions.
- To decide the next move, we simulate games to find the best move. But it is not that simple. We use the policy network and the value network to narrow down the search. To mitigate errors in our evaluation, we compute a weighted average from our game results and our board evaluation.



AlphaGo Zero

Self training using Monte Carlo Tree Search

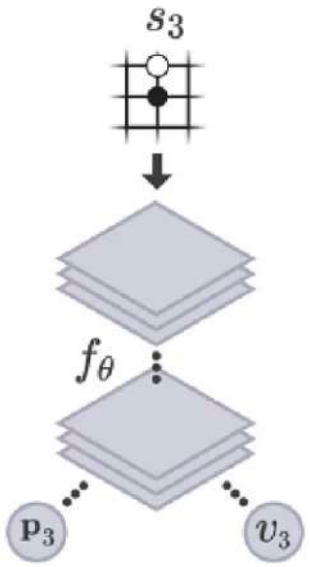
- After looking into MCTS, we come back on how ***f*** is trained. So we start with an empty board position *s1*. We use the MCTS to formulate a policy ***π1***. Then we sample a move *a1* from ***π1***. After taking the move *a1*, the board is in *s2*. We repeat the process again until the game is finished at which we determine who win ***z*** (***z***=1 if we win, 0 otherwise.).

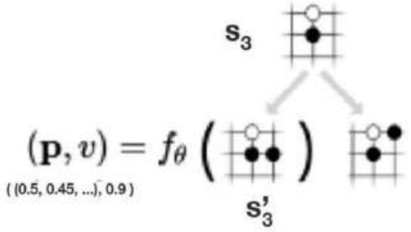


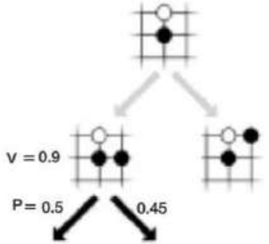
AlphaGo vs AlphaGo Zero

- In Go, our policy controls the moves (actions) to win a game. To model uncertainty, the policy is a probability distribution ***p(s, a)***: the chance of taking a move ***a*** from the board position ***s***.

- In AlphaGo Zero, we use a single deep network ***f***, composed of convolutional layers, to estimate both ***p*** and ***v***.

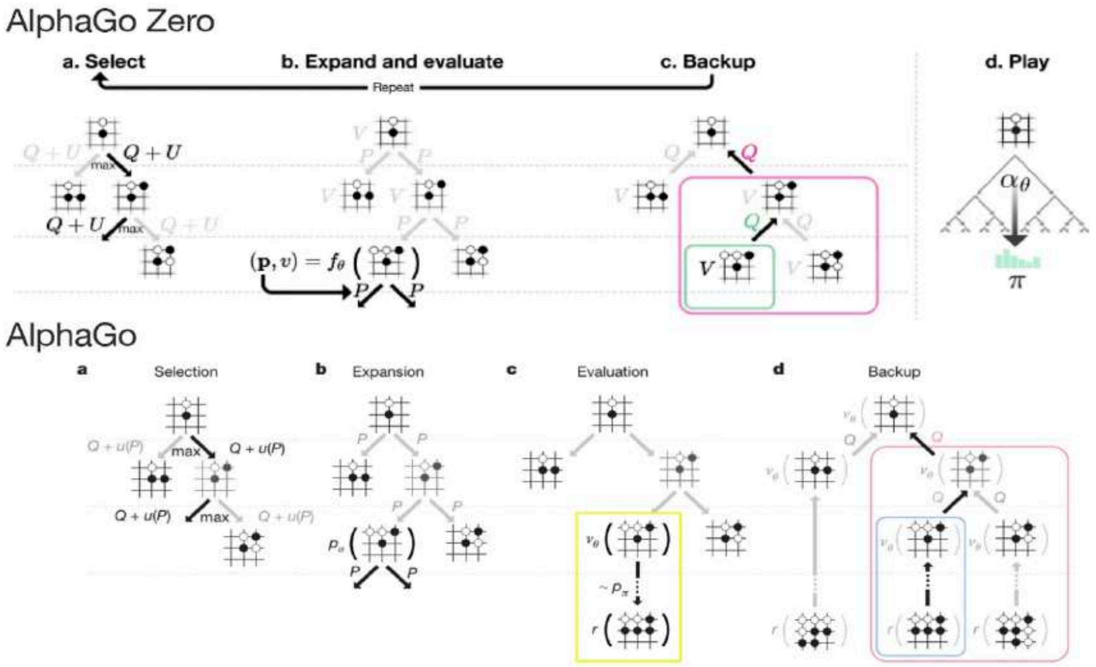






AlphaGo vs AlphaGo Zero

- AlphaGo does not use the Monte Carlo Rollout (in yellow).



AlphaGo vs AlphaGo Zero

- AlphaGo Zero uses the self-trained network ***ϑ*** to calculate the value function ***v*** while Alpha Go uses the SL policy network ***σ*** learned from real games. Even the equation in computing ***Q*** looks different, the real difference is AlphaGo has an extra term ***z*** (the game result) that is found by the Monte Carlo rollout which AlphaGo Zero skips.

AlphaGo Zero

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s,a) + u(s,a))$$

$$u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$$

$$(P(s,a), v) = f_\theta(s)$$

$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i)$$

$$W(s,a) = W(s,a) + v$$

$$Q(s,a) = \frac{W(s,a)}{N(s,a)}$$

$$\pi(a|s) = N(s,a)^{1/\tau} / \textstyle\sum_b N(s,b)^{1/\tau}$$

AlphaGo

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t,a) + u(s_t,a))$$

$$u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$$

$$P(s,a) = p_\sigma(a|s)$$

$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i)$$

$$v = (1-\lambda)v_\theta(s_L) + \boxed{\lambda z_L}$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i)\, v$$

$$\pi(a|s) = N(s,a)^{1/\tau} / \textstyle\sum_b N(s,b)^{1/\tau}$$

AlphaGo Zero performance

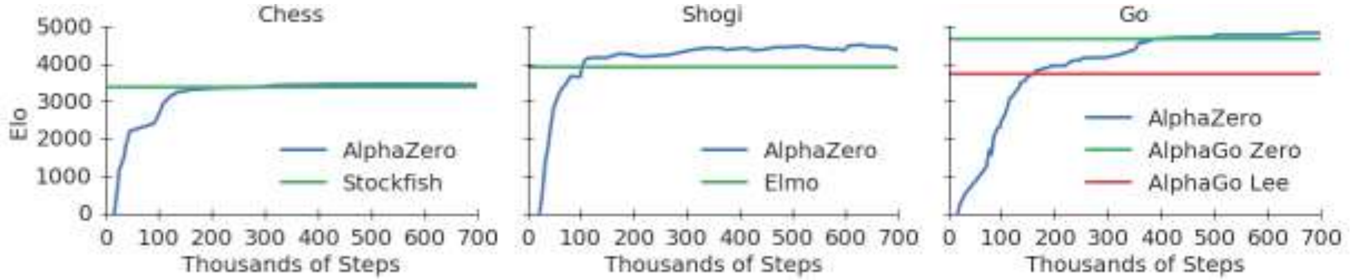
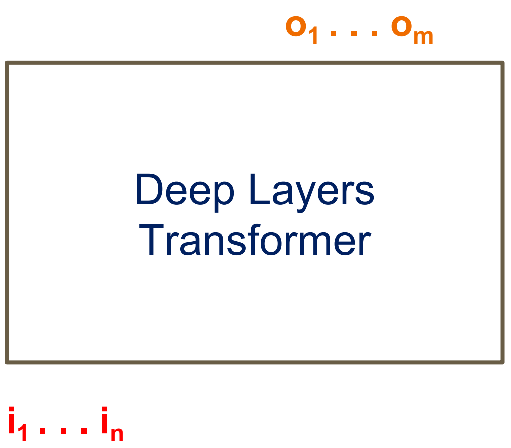


Figure 1: Training *AlphaZero* for 700,000 steps. Elo ratings were computed from evaluation games between different players when given one second per move. **a** Performance of *AlphaZero* in chess, compared to 2016 TCEC world-champion program *Stockfish*. **b** Performance of *AlphaZero* in shogi, compared to 2017 CSA world-champion program *Elmo*. **c** Performance of *AlphaZero* in Go, compared to *AlphaGo Lee* and *AlphaGo Zero* (20 block / 3 day) (29).

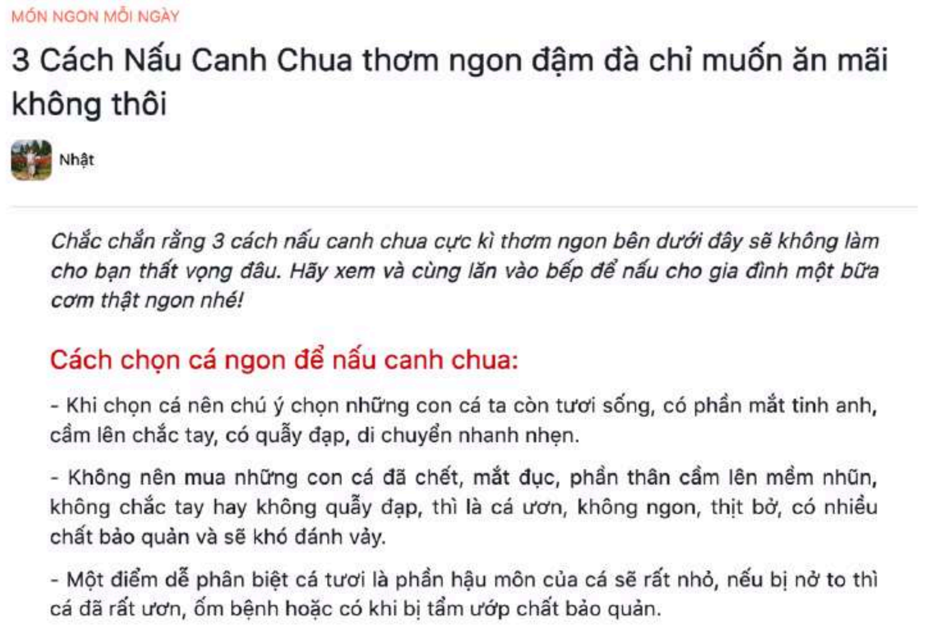
Reinforcement Learning with Human Feedback in ChatGPT

ChatGPT is a Large Language Model

Có 3 cách nấu...



Giới thiệu cách nấu canh chua?



MÓN NGON MỖI NGÀY

3 Cách Nấu Canh Chua thơm ngon đậm đà chỉ muốn ăn mãi không thôi

Nhật

Chắc chắn rằng 3 cách nấu canh chua cực kì thơm ngon bên dưới đây sẽ không làm cho bạn thất vọng đâu. Hãy xem và cùng lăn vào bếp để nấu cho gia đình một bữa cơm thật ngon nhé!

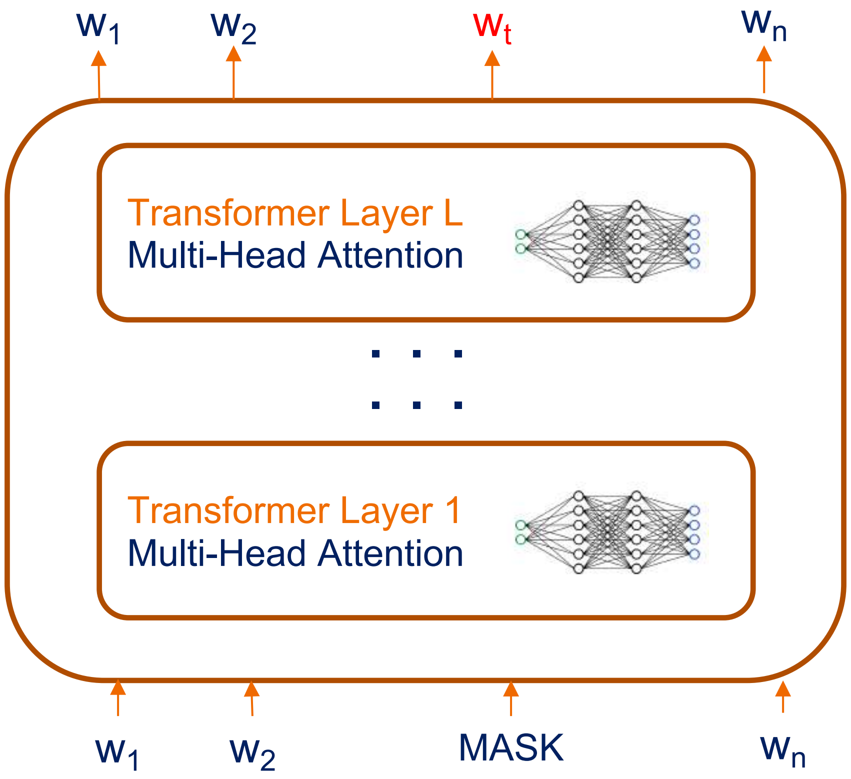
Cách chọn cá ngon để nấu canh chua:

- Khi chọn cá nên chú ý chọn những con cá ta còn tươi sống, có phần mắt tinh anh, cầm lên chắc tay, có quẫy đạp, di chuyển nhanh nhẹn.

- Không nên mua những con cá đã chết, mắt đục, phần thân cầm lên mềm nhũn, không chắc tay hay không quẫy đạp, thì là cá ươn, không ngon, thịt bở, có nhiều chất bảo quản và sẽ khó đánh vảy.

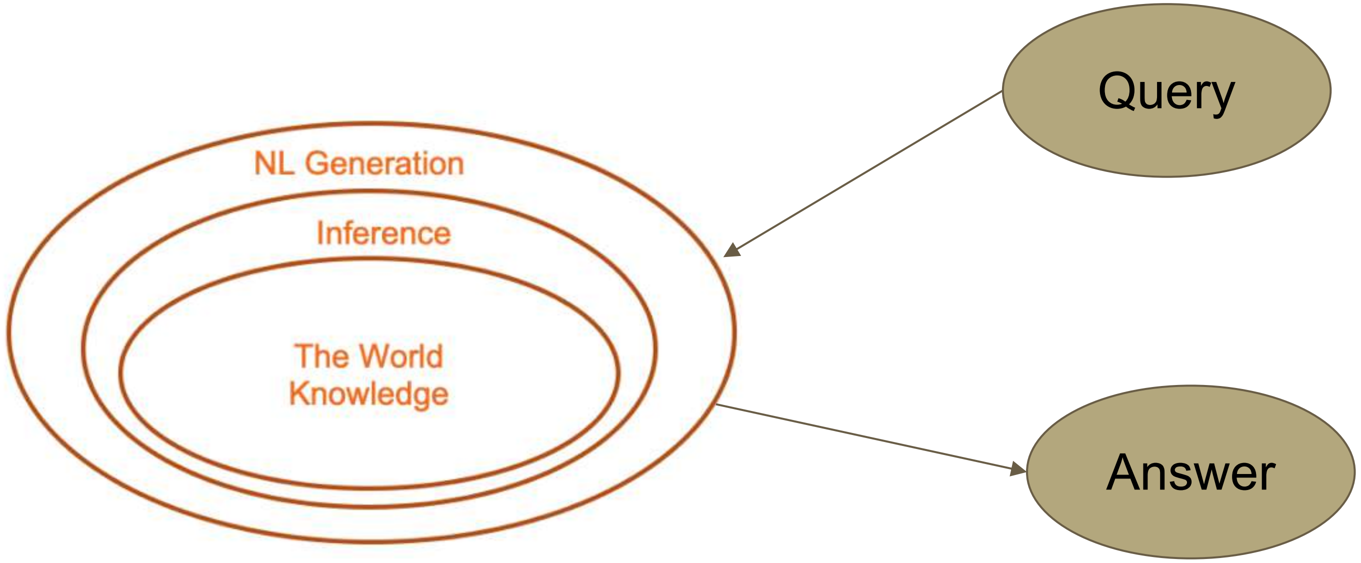
- Một điểm dễ phân biệt cá tươi là phần hậu môn của cá sẽ rất nhỏ, nếu bị nở to thì cá đã rất ươn, ốm bệnh hoặc có khi bị tẩm ướp chất bảo quản.

Transformer Model



- Encoding knowledege about concepts and relationship (not about Words)
- Information, knowledge is contained in the neural network (not in the form of strings and probabilities)

Problem: not align between Query and Answer



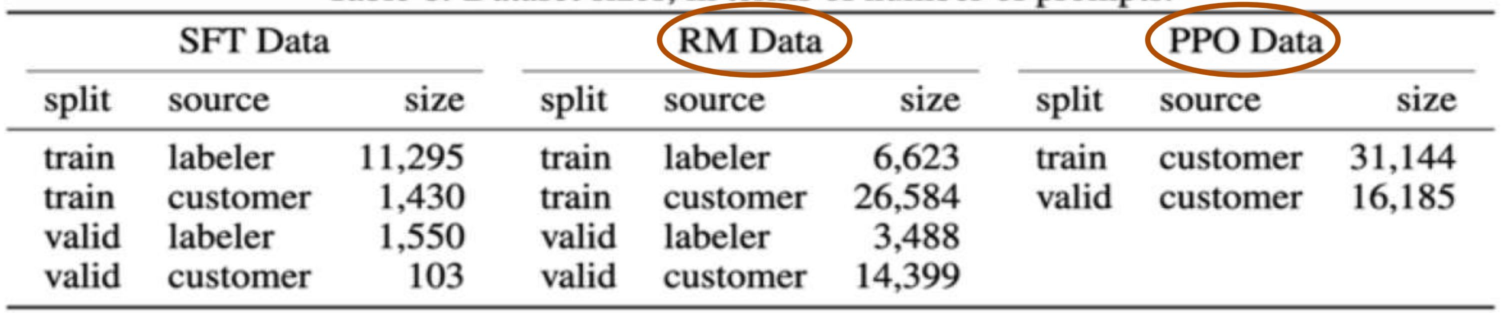
Question: Which bird is a mammal?

Answer: Lots of mammals including penguins....

ChatGPT

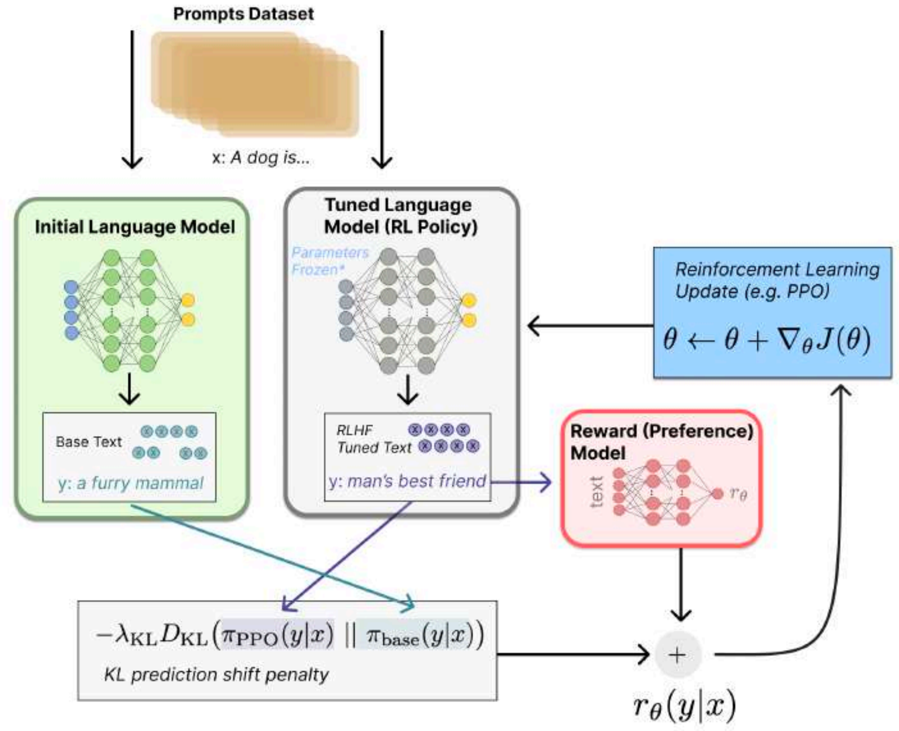
1. Very large text sets trained on a very large Model
2. Supervised Fine Tuning
3. Reinforcement Learning with Human Feedback(RLHF)

Table 6: Dataset sizes, in terms of number of prompts.



SFT Data			RM Data			PPO Data		
split	source	size	split	source	size	split	source	size
train	labeler	11,295	train	labeler	6,623	train	customer	31,144
train	customer	1,430	train	customer	26,584	valid	customer	16,185
valid	labeler	1,550	valid	labeler	3,488			
valid	customer	103	valid	customer	14,399			

Reinforcement Learning with Human Feedback (HLHF) in ChatGPT


Prompts Dataset
x: A dog is...
Initial Language Model
Base Text
y: a furry mammal
Tuned Language Model (RL Policy)
Parameters Frozen*
RLHF Tuned Text
y: man's best friend
Reinforcement Learning Update (e.g. PPO)
$\theta \leftarrow \theta + \nabla_\theta J(\theta)$
Reward (Preference) Model
text
r_θ
$-\lambda_{\mathrm{KL}} D_{\mathrm{KL}}(\pi_{\mathrm{PPO}}(y|x) \,||\, \pi_{\mathrm{base}}(y|x))$
KL prediction shift penalty
+
$r_\theta(y|x)$

Proximal Policy Optimization (PPO)

OpenAI (2017): “We’re releasing a new class of reinforcement learning algorithms, Proximal Policy Optimization (PPO), which **perform comparably or better than state-of-the-art approaches** while being much simpler to implement and tune. PPO has become the default reinforcement learning algorithm at OpenAI **because of its ease of use and good performance**.”

https://openai.com/research/openai-baselines-ppo

Why PPO?

- The policy $\pi_\theta(a|s)$ is stochastic, meaning that the parameters dictate the **sampling probability of actions** *a*, and thereby influence the probability of following trajectories $\tau=s_1,a_1,\ldots s_n,a_n$. As such, we can express the objective function dependent on θ:

$$J(\theta) = E_{\tau\sim\pi_\theta} R(\tau) = \sum_{\tau} P(\tau;\theta)R(\tau)$$

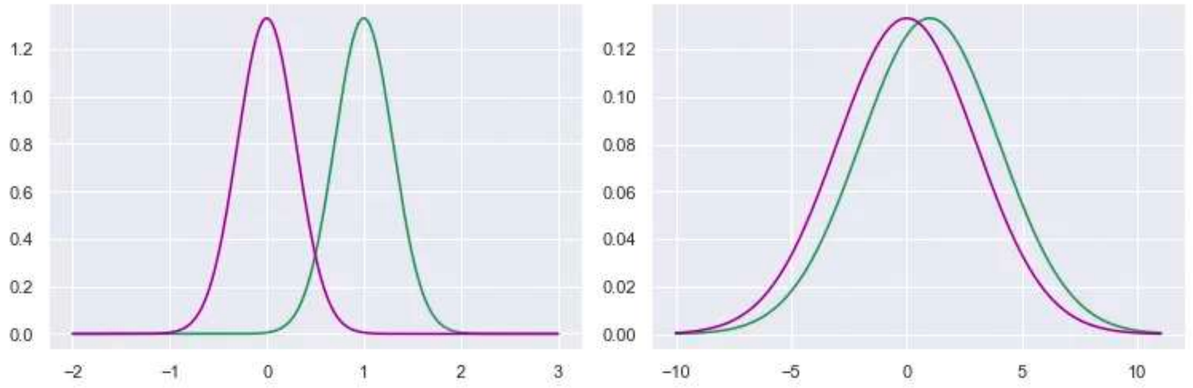
- Traditional policy gradient update function, updating policy weights θ based on objective function gradient ∇_θJ(θ) and step size α

$$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$$

Why PPO?

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

- Consider weight updates of equal magnitude to the two policies below. Clearly, the distribution on the left is affected much more than the one on the right. However, the Euclidian distance (i.e., parameter distance) is the same!



Comparison of normal distribution pairs. The left has μ_1=0, μ_2=1 and σ_1=σ_2=0.3. The right has μ_1=0, μ_2=1 and σ_1=σ_2=3.0. Although the Euclidean distance between both pairs is 1, it is obvious the pair on the right is much more similar than the one on the left.

PPO

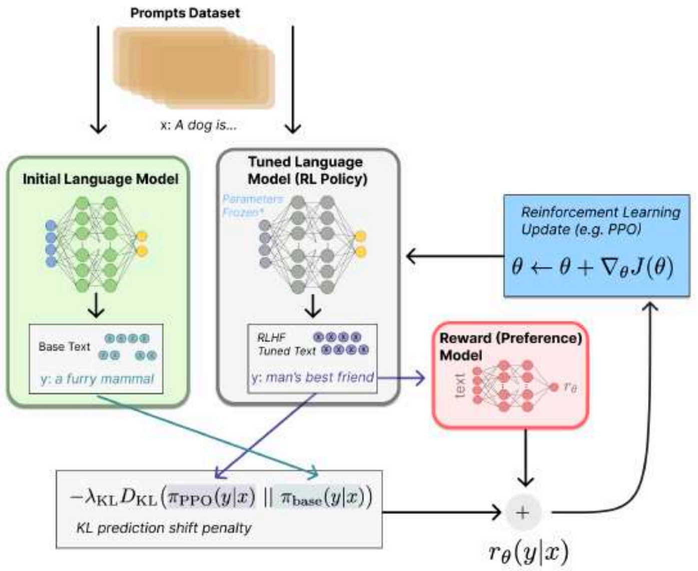
- The Kullback-Leibner (KL) divergence measures how much a policy *π_θ* changes due to an update. Remember that the policy is represented by a probability distribution, which determines the probability of selecting a given action. **KL divergence** is defined as follows:

$$\mathcal{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta+\Delta\theta}) = \sum_{x\in\mathcal{X}} \pi_\theta(x) \log\left(\frac{\pi_\theta(x)}{\pi_{\theta+\Delta\theta}(x)}\right)$$

Now, if we restrict the divergence of the update to be no more than ϵ

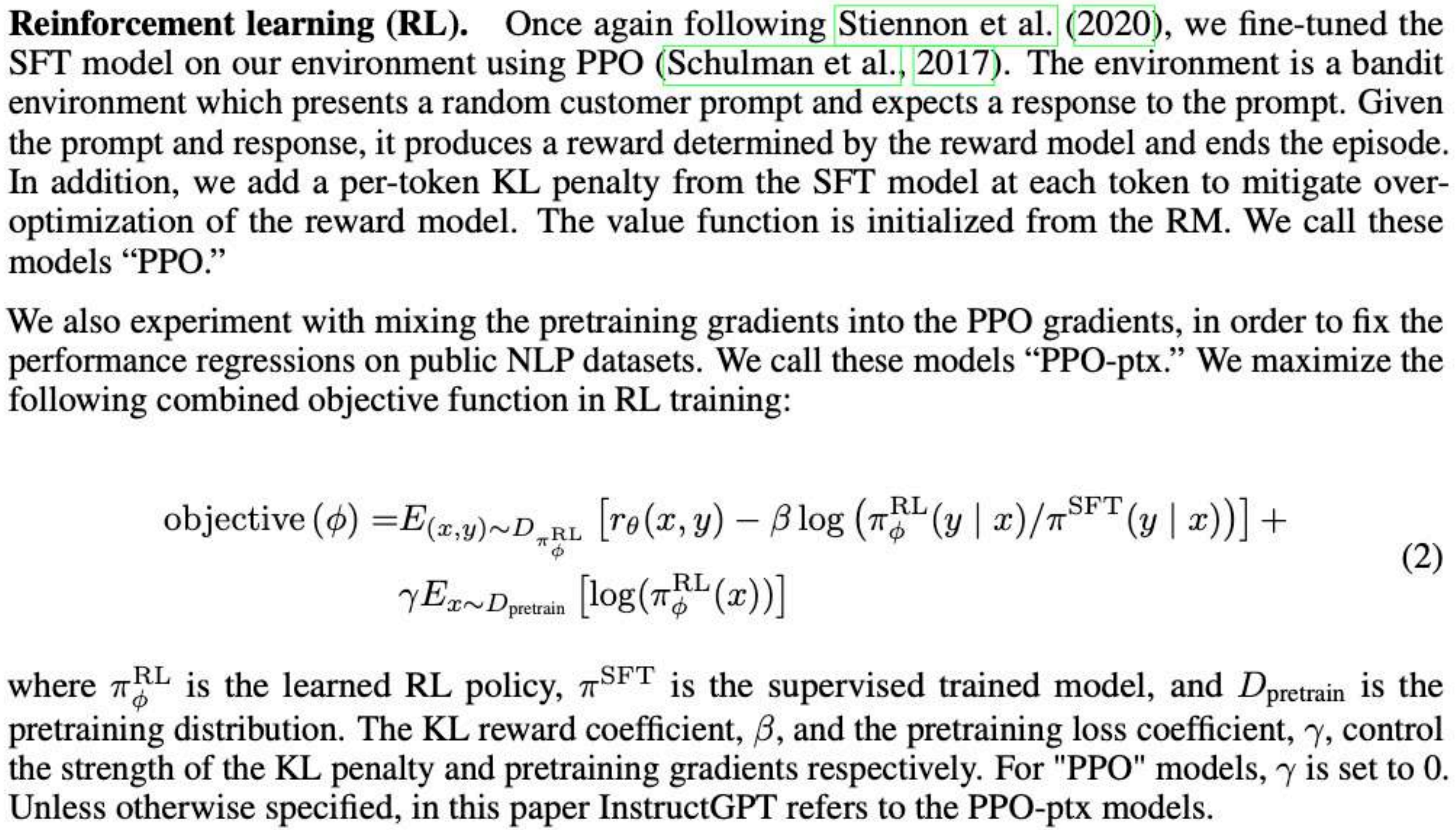
$$\Delta\theta^* = \underset{\mathcal{D}_{\mathrm{KL}}(\pi_\theta \| \pi_{\theta+\Delta\theta}) \le \epsilon}{\arg\max} \; J(\theta + \Delta\theta)$$

PPO and PPOptx


Prompts Dataset
x: A dog is...
Initial Language Model
Tuned Language Model (RL Policy)
Parameters Frozen*
Base Text
y: a furry mammal
RLHF Tuned Text
y: man's best friend
Reinforcement Learning Update (e.g. PPO)
$\theta \leftarrow \theta + \nabla_\theta J(\theta)$
Reward (Preference) Model
text
r_θ
$-\lambda_{KL} D_{KL}(\pi_{PPO}(y|x) \,||\, \pi_{base}(y|x))$
KL prediction shift penalty
+
$r_\theta(y|x)$

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

RLHF in ChatGPT



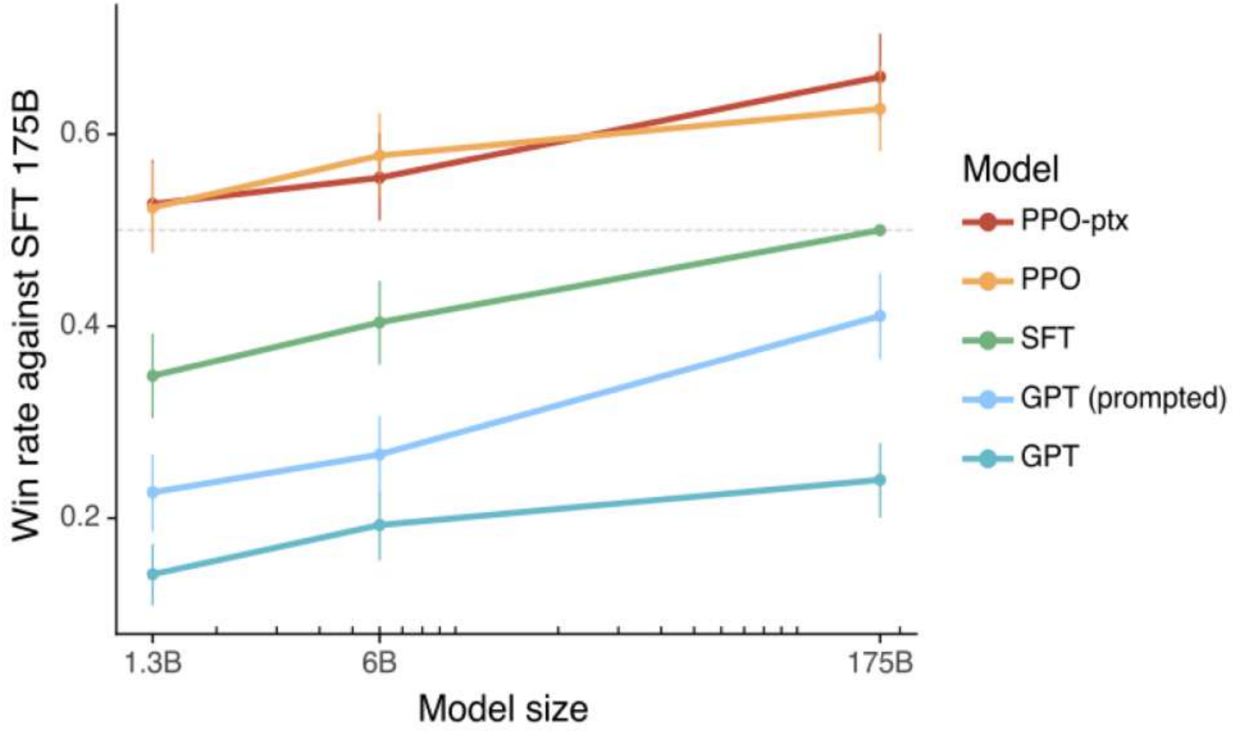
Reinforcement learning (RL). Once again following Stiennon et al. (2020), we fine-tuned the SFT model on our environment using PPO (Schulman et al., 2017). The environment is a bandit environment which presents a random customer prompt and expects a response to the prompt. Given the prompt and response, it produces a reward determined by the reward model and ends the episode. In addition, we add a per-token KL penalty from the SFT model at each token to mitigate over-optimization of the reward model. The value function is initialized from the RM. We call these models "PPO."

We also experiment with mixing the pretraining gradients into the PPO gradients, in order to fix the performance regressions on public NLP datasets. We call these models "PPO-ptx." We maximize the following combined objective function in RL training:

$$\begin{aligned}\text{objective}\,(\phi) = & E_{(x,y)\sim D_{\pi_\phi^{\mathrm{RL}}}}\left[r_\theta(x,y) - \beta\log\left(\pi_\phi^{\mathrm{RL}}(y\mid x)/\pi^{\mathrm{SFT}}(y\mid x)\right)\right] + \\ & \gamma E_{x\sim D_{\text{pretrain}}}\left[\log(\pi_\phi^{\mathrm{RL}}(x))\right]\end{aligned} \tag{2}$$

where π_ϕ^{RL} is the learned RL policy, π^{SFT} is the supervised trained model, and D_{pretrain} is the pretraining distribution. The KL reward coefficient, β, and the pretraining loss coefficient, γ, control the strength of the KL penalty and pretraining gradients respectively. For "PPO" models, γ is set to 0. Unless otherwise specified, in this paper InstructGPT refers to the PPO-ptx models.

Result



Win rate against SFT 175B
0.6
0.4
0.2
1.3B
6B
175B
Model size
Model
PPO-ptx
PPO
SFT
GPT (prompted)
GPT

Conclusion

1. Reinforcement learning is a powerful framework for training intelligent agents to make sequential decisions in dynamic environments.
2. Deep reinforcement learning, combining reinforcement learning with deep neural networks, has shown remarkable success in solving complex and high-dimensional problems.
3. Reinforcement learning has diverse applications, including robotics, autonomous driving, game playing, finance, healthcare, and natural language processing.
4. Challenges in reinforcement learning include sample efficiency, exploration-exploitation trade-offs, and generalization to unseen situations.

Thank you!

- Discussion

References

- https://arxiv.org/abs/1707.06347
- https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/
- https://huggingface.co/learn/deep-rl-course/unit3/deep-q-algorithm?fw=pt
- https://towardsdatascience.com/proximal-policy-optimization-ppo-explained-abed1952457b
- https://docs.google.com/presentation/d/1eI9PqRJTCFOIVihkig1voRM4MHDpLpCicX9lX1J2fqk
- https://jonathan-hui.medium.com/alphago-how-it-works-technically-26ddcc085319