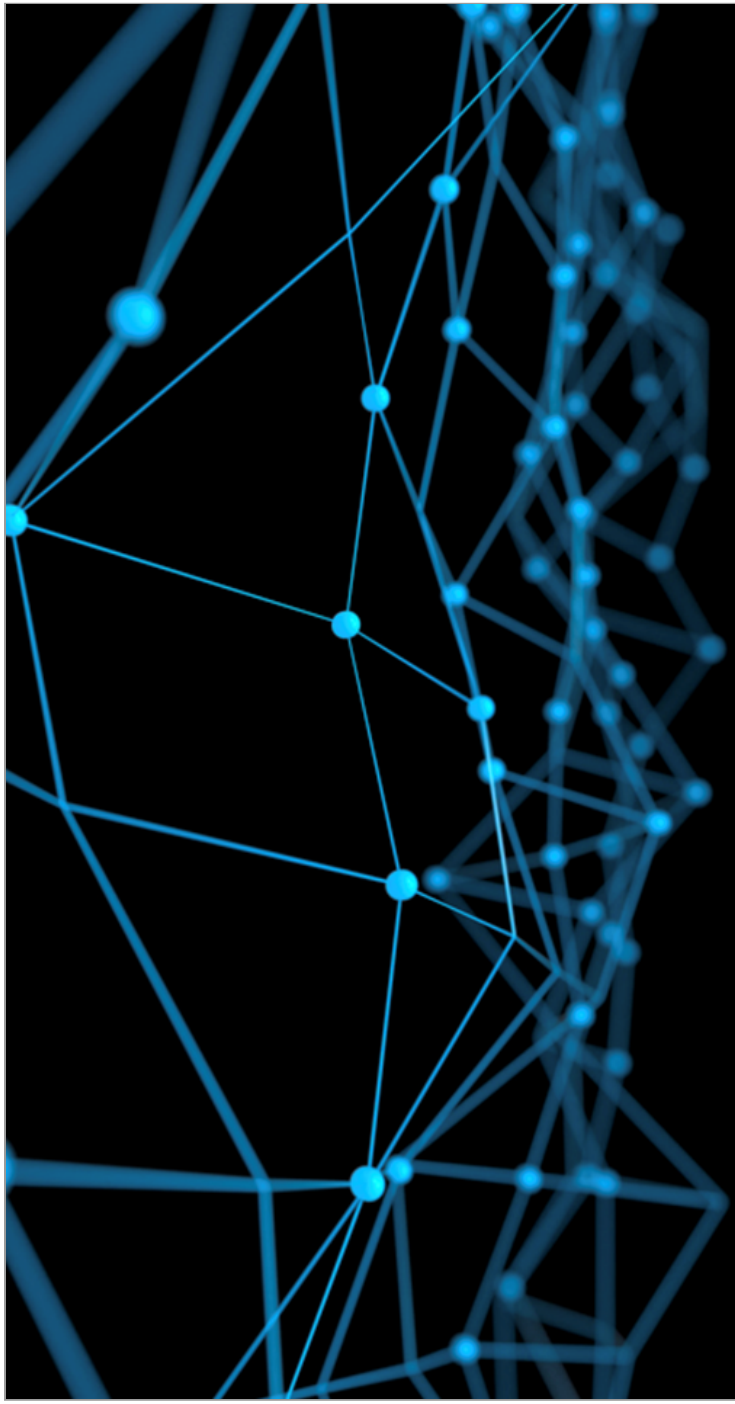


Reinforcement Learning

Mingsheng Long

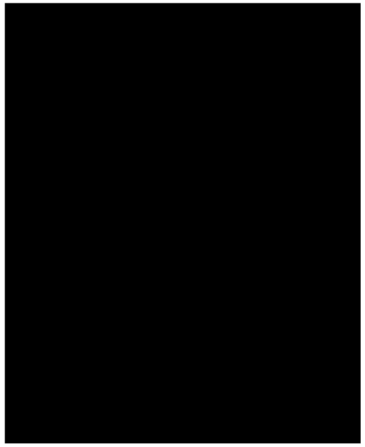
mingsheng@tsinghua.edu.cn

Fall 2022

Outline

- **Reinforcement Learning**
  - Markov Decision Process (MDP)
- Value-based RL
  - Q-Learning (DQN)
- Policy-based RL
  - Policy Gradient (REINFORCE)
- Games
  - AlphaGo, AlphaGo Zero, AlphaZero, MuZero
- Real World

What is Reinforcement Learning

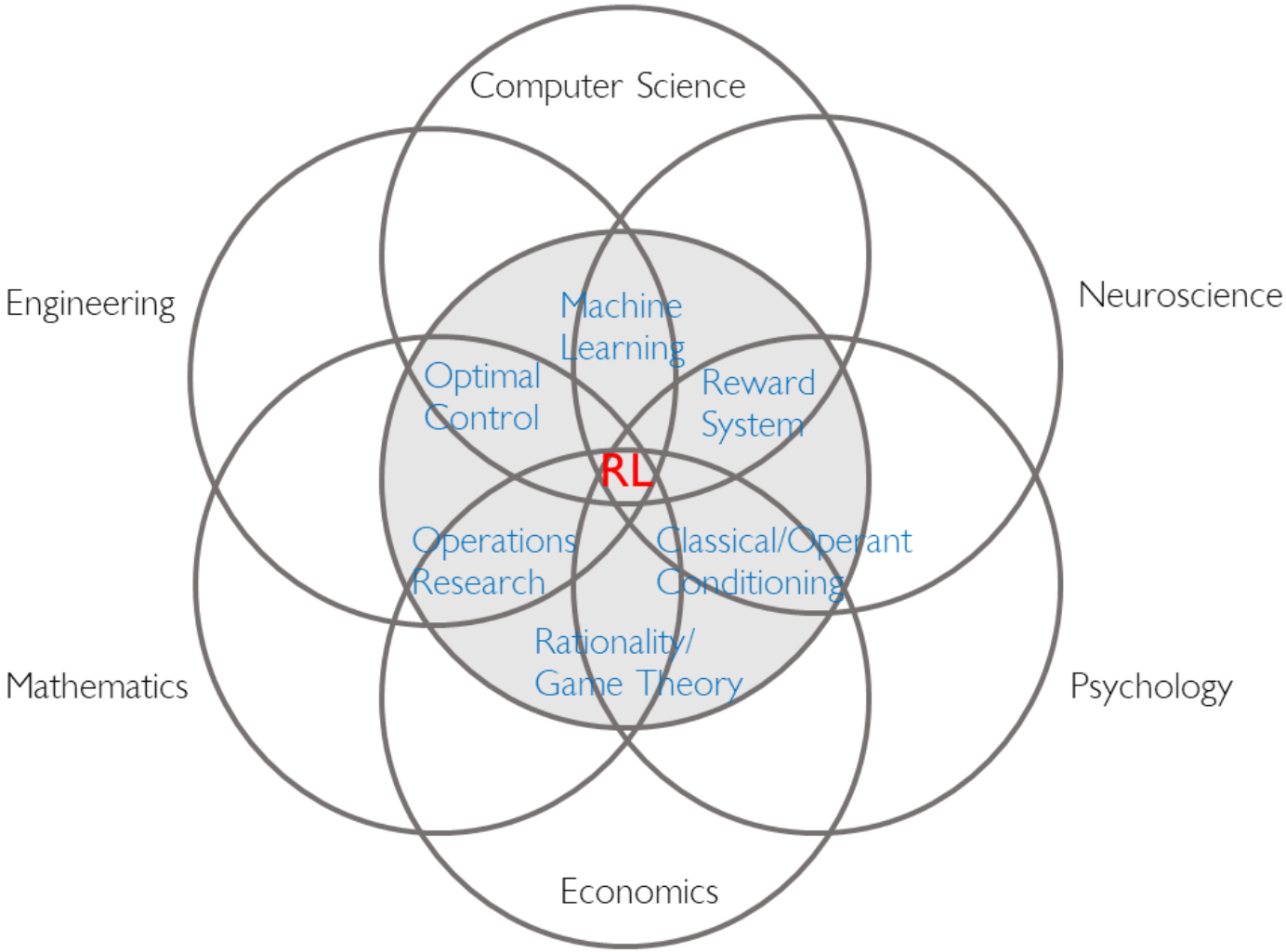


Play games: Atari, poker, Go, ...

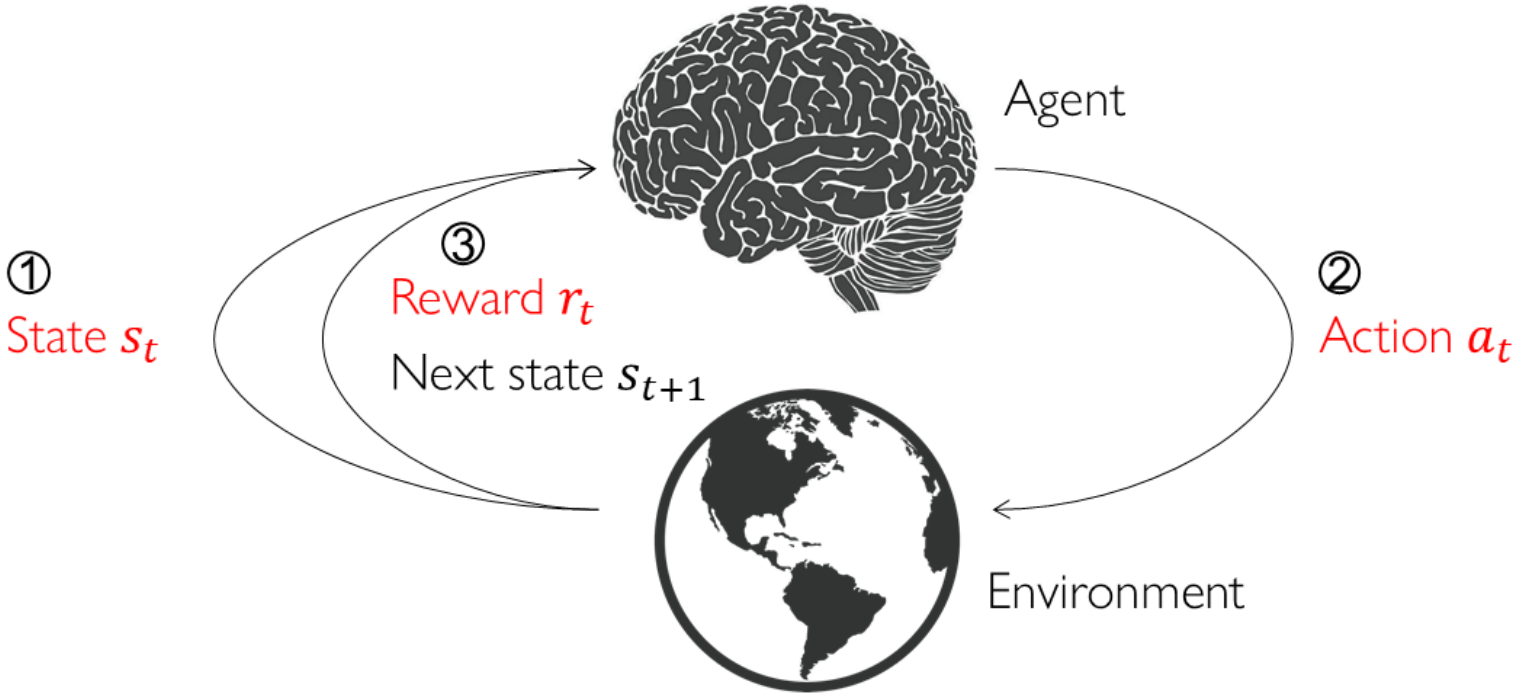
Explore worlds: 3D worlds, Labyrinth, ...

Control physical systems: manipulate, walk, swim, ...

Reinforcement Learning (RL)



Agent and Environment



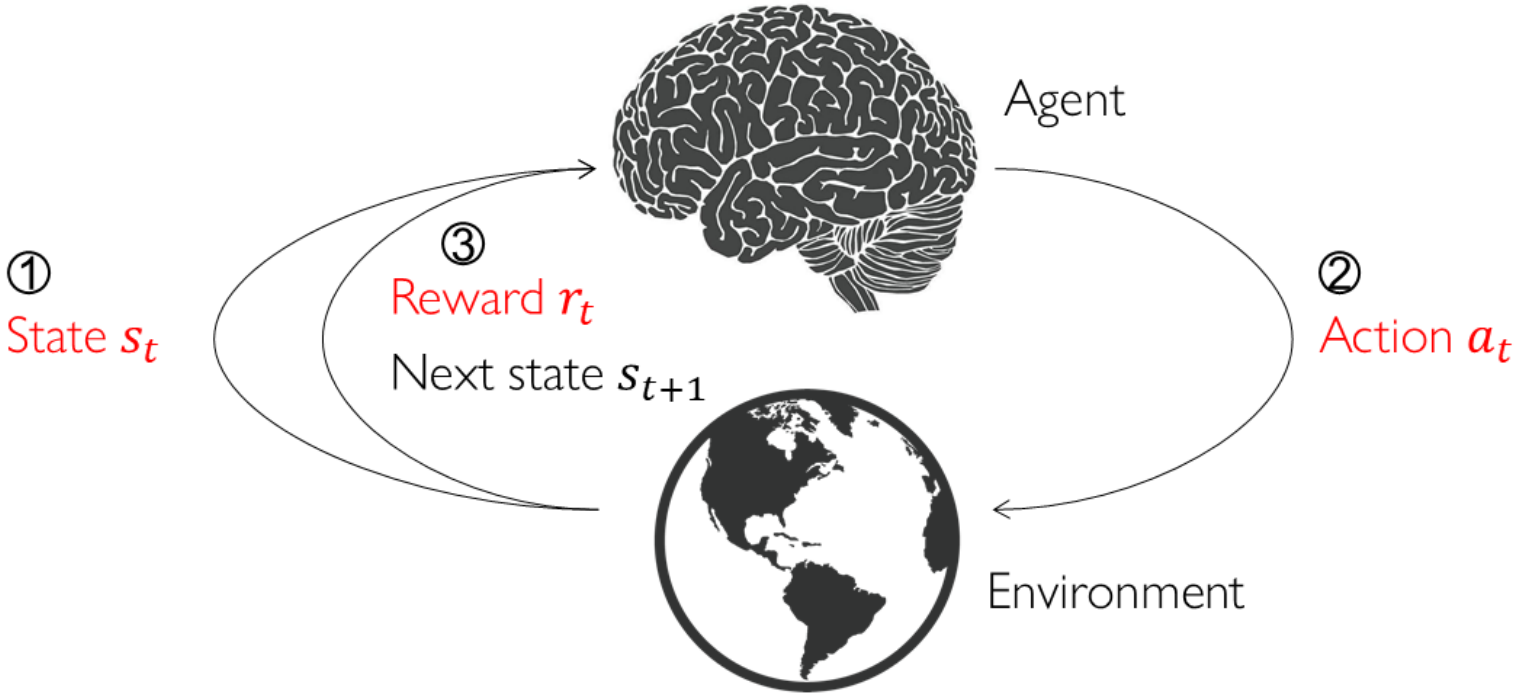
At each step t the agent:

- Executes action a_t
- Receives state s_t
- Receives scalar reward r_t

At each step t the environment:

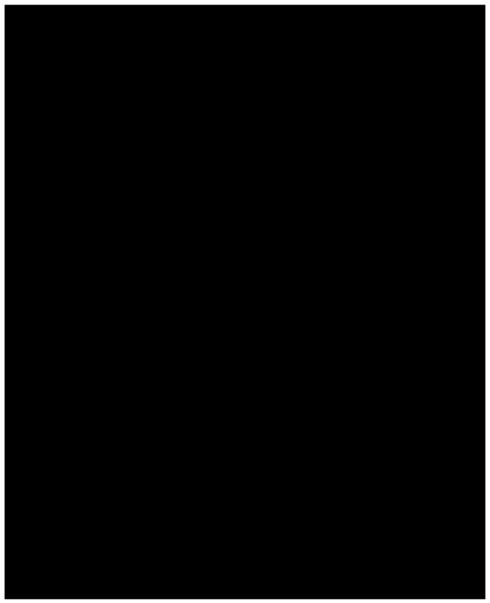
- Receives action a_t
- Emits state s_{t+1}
- Emits scalar reward r_{t+1}

Reinforcement Learning (RL)



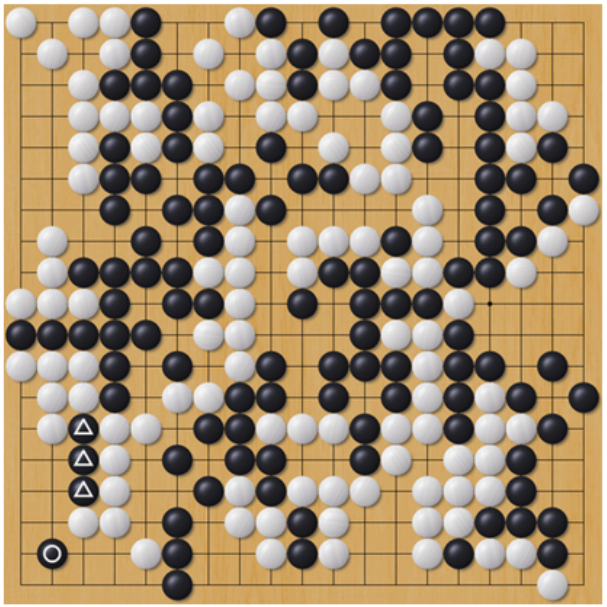
- **Problems:** An agent interacting with an environment, which provides numeric reward signals
- **Goal:** Learn how to take actions in order to maximize overall reward.

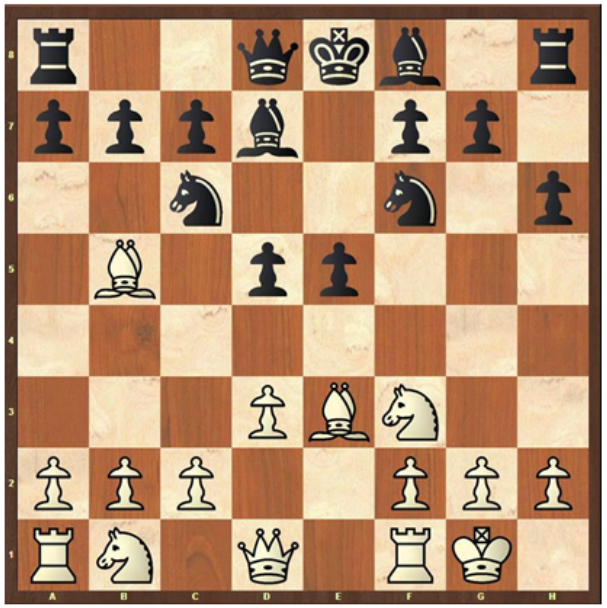
Atari Games



- Objective: Complete the game with the highest score
- State: Raw pixel inputs of the game state
- Action: Game controls e.g. Left, Right, Up, Down
- Reward: Score increase/decrease at each time step

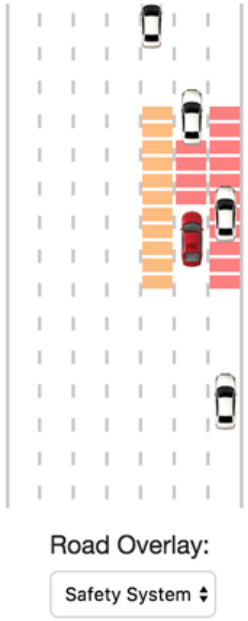
Board Games





- Objective: Win the game!
- State: Position of all pieces
- Action: Where to put the next piece down
- Reward: 1 if win at the end of the game, 0 otherwise

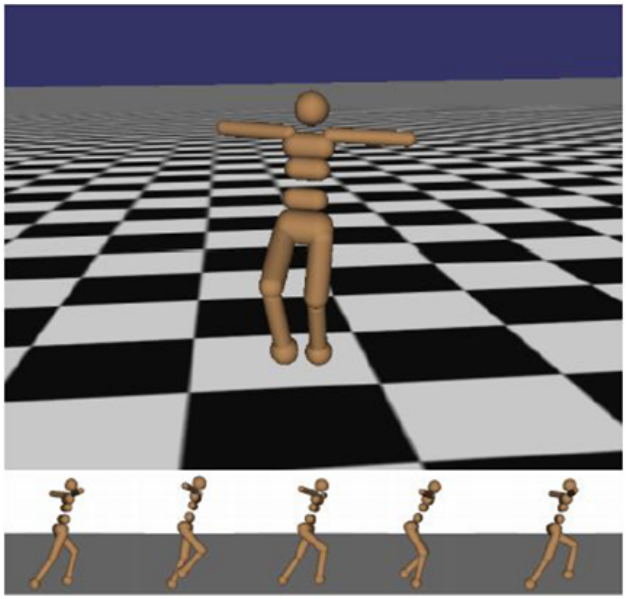
Deep Traffic



- Objective: Make the car move forward
- State: Position of all cars
- Action: Lane change, speed up and speed down
- Reward: 1 if the car move forward and do not collide with other cars

https://selfdrivingcars.mit.edu/deeptraffic/

Robot Locomotion



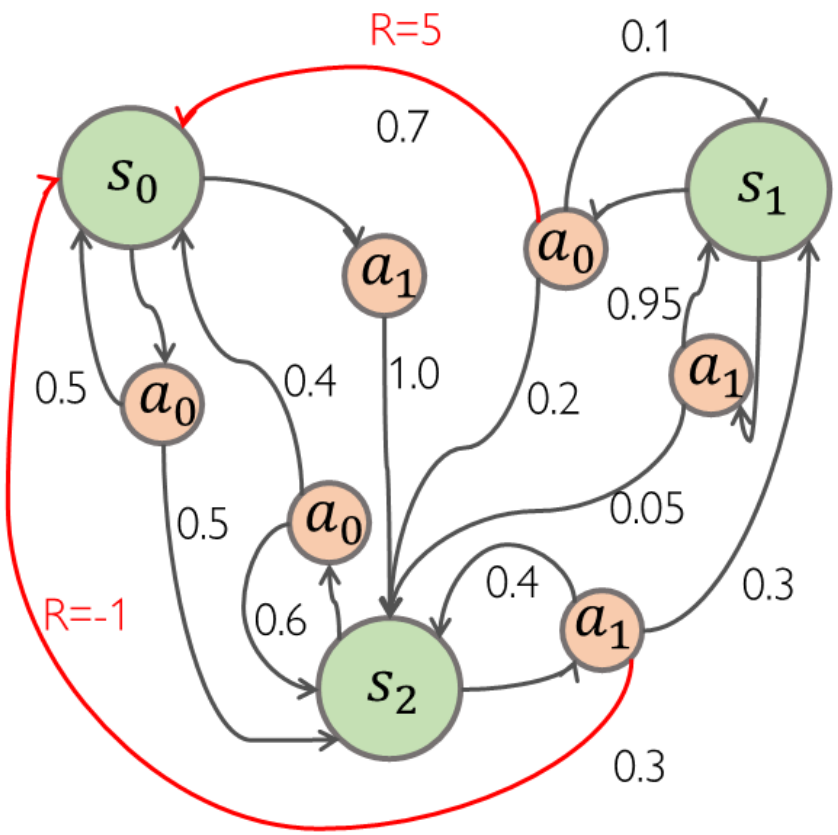
- Objective: Make the robot move forward
- State: Angle and position of the joints
- Action: Torques applied on joints
- Reward: 1 at each time step upright + forward movement

Outline

- **Reinforcement Learning**
  - **Markov Decision Process (MDP)**
- Value-based RL
  - Q-Learning (DQN)
- Policy-based RL
  - Policy Gradient (REINFORCE)
- Games
  - AlphaGo, AlphaGo Zero, AlphaZero, MuZero
- Real World

Markov Decision Process (MDP)

Mathematical formulation of the RL problem



A Markov Decision Process (MDP) is tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a set of states, $s \in \mathcal{S}$
- $\mathcal{A}$ is a set of actions, $a \in \mathcal{A}$
- $\mathcal{R}$ is distribution of reward r given (state, action) pair (s, a)
- $\mathcal{P}$ is a state transition probability matrix $P_{sas'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- γ is discount factor, the preference for present rewards over future rewards

Markov property

- Current state completely characterizes states of the world

Markov Decision Process (MDP)

- At time step $t = 0$, environment samples initial state $s_0 \sim p(s_0)$
- Then, for $t = 0$ until terminated (an episode):
  1. Agent selects action a_t
  2. Environment samples reward $r_t \sim \mathcal{R}(\cdot|s_t, a_t)$
  3. Environment samples next state $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$
  4. Agent receives reward r_t and next state s_{t+1}
- Sampling from MDP yields a trajectory $s_0, a_0, r_0, s_1, a_1, r_1 \ldots$
- A policy $\boldsymbol{\pi}: \mathcal{S} \rightarrow \mathcal{A}$ specifies what action to take in each state, $\pi(a|s)$
- Objective: find optimal $\boldsymbol{\pi}^*$ to maximize cumulative discounted reward

$$\sum_{t \geq 0} \gamma^t r_t$$

Policy

- A policy $\boldsymbol{\pi}: \mathcal{S} \rightarrow \mathcal{A}$ specifies what action to take in each state
  - Deterministic policy: $a = \pi(s)$ (for a state, only one action is taken)
  - Stochastic policy: $\pi(a|s) = P(a|s)$ (requires sampling to take action)
- Objective: find optimal $\boldsymbol{\pi}^*$ to maximize the expected sum of rewards

$$\pi^* = \text{argmax}\, \mathbb{E}[\sum_{t \geq 0} \gamma^t r_t | \pi]$$

$$\text{with } s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$$

• Policy in Reinforcement Learning		• Hypothesis in Supervised Learning
• Policy π	⟷	• Hypothesis f
• State s	⟷	• Feature x
• Action a	⟷	• Label y
• $\pi(a\|s) = P(a\|s)$		• $f(y\|x) = P(y\|x)$

$$\min_f \mathbb{E}\,[l(x, y); f]$$

Optimal Policy

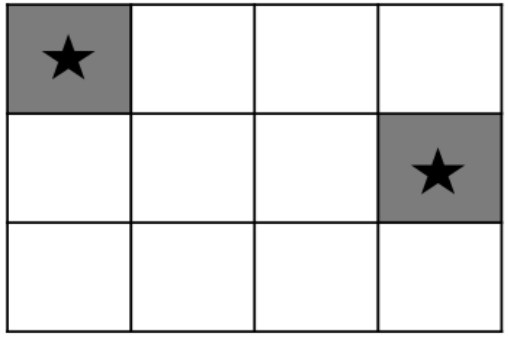
Actions = {

1. right
2. left
3. up
4. down

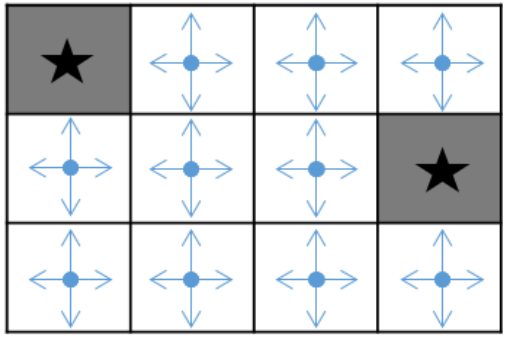
}

Set a negative reward for each transition (e.g. $r = -1$)

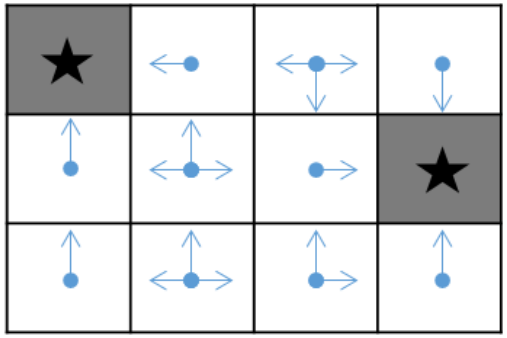
States



Random Policy



Optimal Policy



- Objective:

Reach one of terminal states (greyed out) in least number of actions

Value Function

- State value function is a prediction of the future reward
  - How much reward to get from state s under policy π?

$$V^{\pi}(s) = \mathbb{E}_{s_1,a_1,\ldots,s_t,a_t,\ldots}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, \pi\right] = \mathbb{E}_{\pi}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, \pi\right]$$

- State-action value function is a prediction of the future reward
  - How much reward to get from state $\mathbf{s}$ and action a under policy π

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

- Optimal Q-value function reaches maximum under optimal policy π^*

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a) \Rightarrow \pi^*(s) = \operatorname*{argmax}_{a} Q^*(s,a)$$

Bellman Equation

- Q-value function can be decomposed into a Bellman equation

$$Q^{\pi}(s,a) = \mathbb{E}_{s',a'}[r + \gamma Q^{\pi}(s',a')|s,a]$$

Proof:

$$Q^{\pi}(s,a) = \mathbb{E}_{s_{1:T},a_{1:T}}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, a_0 = a\right]$$

$$= \mathbb{E}_{s',a'}\left\{r + \gamma \mathbb{E}_{s_{2:T},a_{2:T}}\left[\sum_{t\geq 0}\gamma^t r_t\right]|s_1 = s', a_1 = a', s, a\right\}$$

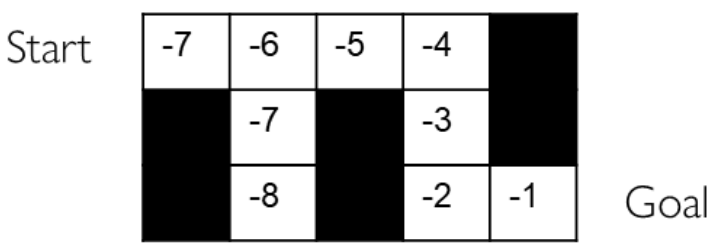
$$= \mathbb{E}_{s',a'}[r + \gamma Q^{\pi}(s',a')|s,a]$$

- Optimal Q-value function also decomposes into a Bellman equation

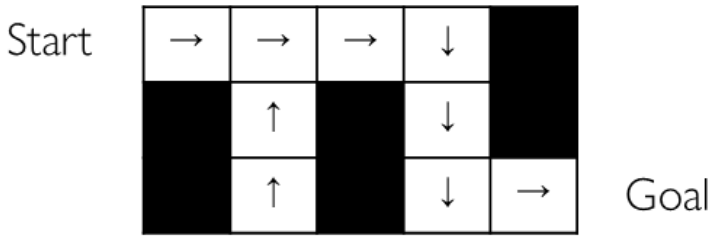
$$Q^{*}(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^{*}(s',a')|s,a\right]$$

Approaches to RL

- Value-based RL
  - Estimate the optimal Q-value function $Q^*(s,a)$
  - This is the maximum value achievable under any policy
- Policy-based RL
  - Search directly for the optimal policy $\pi^* = \pi^*(a|s)$
  - This is the policy achieving maximum future reward



Value-based RL



Policy-based RL

单选题 1分

以下关于RL的描述中，错误的是?

A RL可用MDP表示，旨在找到累积奖赏最大的策略

B 随机性策略中，选择各动作的概率之和为1

C Q-value函数的取值与所采用的具体动作无关

D Value-based和Policy-based是RL方法的两大类型

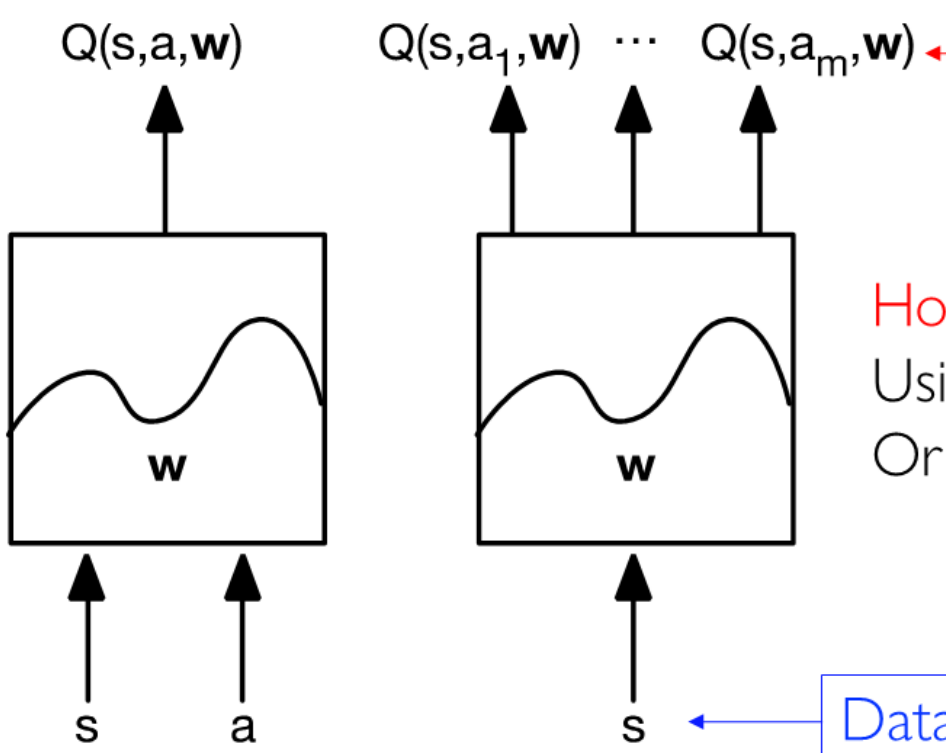
Outline

- Reinforcement Learning
  - Markov Decision Process (MDP)
- **Value-based RL**
  - **Q-Learning (DQN)**
- Policy-based RL
  - Policy Gradient (REINFORCE)
- Games
  - AlphaGo, AlphaGo Zero, AlphaZero, MuZero
- Real World

Value Network

- Represent the Q-value function by Q-network with weights w

$$Q(s,a,w) \approx Q^*(s,a)$$



How to get labels for training?
Using Monte Carlo estimation.
Or applying Bellman equation.

Q-Learning: Monte Carlo

- Monte Carlo estimation: $\mathbb{E}_p f(x) \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i)$. Data $\{x_i\}_1^n \sim p^n$.
- Recall $Q^\pi(s,a) = \mathbb{E}_{s_{1:T},a_{1:T}}[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, a_0 = a, \pi]$
  - When we cannot explicitly compute an expectation:
  - Use Monte Carlo sampling
- Every visit Monte Carlo:
- Sample $s_{1:T}^i, a_{1:T}^i$ for many times (i indicates index of simulation)
- Every time-step t that state s is visited in an episode:
  - Increment counter $N(s,a) \leftarrow N(s,a) + 1$
  - Increment total return $R(s,a) \leftarrow R(s,a) + R_t$
  - Value is estimated by mean return $Q(s,a) \leftarrow R(s,a)/N(s,a)$

Q-Learning: Temporal Difference

- Optimal Q-values should obey the Bellman equation

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

- Treat the right-hand side $r + \gamma \max_{a'} Q^*(s',a',w)$ as a target
- Minimize MSE loss by Stochastic Gradient Descent (SGD)

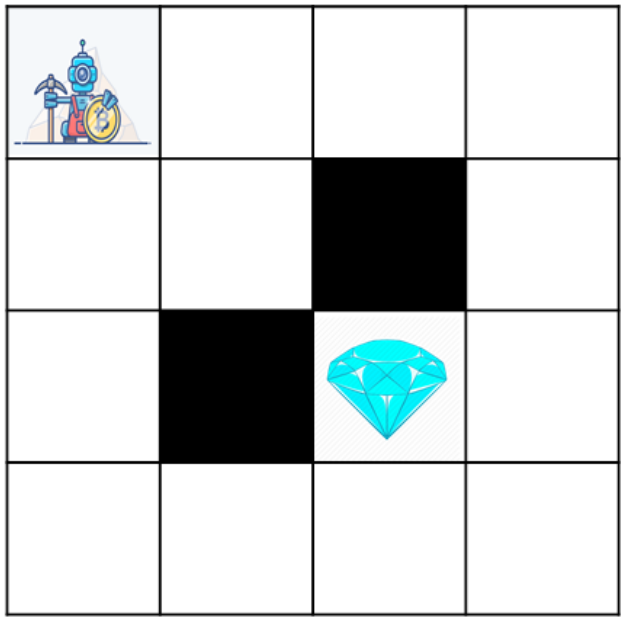
$$l = \left(r + \gamma \max_{a'} Q^*(s',a',w) - Q(s,a,w)\right)^2$$

- Converges to Q^* using table lookup representation (Q-table)

$$Q_{t+1}(s_t,a_t)$$
$$= Q_t(s_t,a_t) + \alpha(\overbrace{r_{t+1} + \gamma \max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t)}^{\text{Temporal Difference (TD) Error}})$$

Old state　　Learning Rate　　Old state

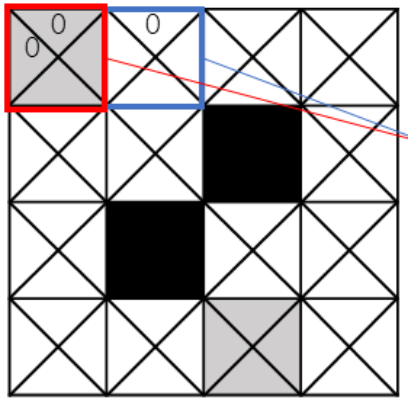
Q-Learning: Table Lookup



Setting:

- You lose -1 at each step (helps our agent to be fast).
- If you touch an black tile, you lose -100 points, and the episode ends.
- If you are in the diamond tile you win, you get +100 points.

Q-Learning: Table Lookup



We can take four actions (←↑→↓) at one tile.

Actions (4)

States (16)

left	right	up	down
0	?	0	?
?	?	0	?
…	…	…	…

We get an Q-table!

The value of each cell will be the maximum expected future reward for the given (s_t, a_t)

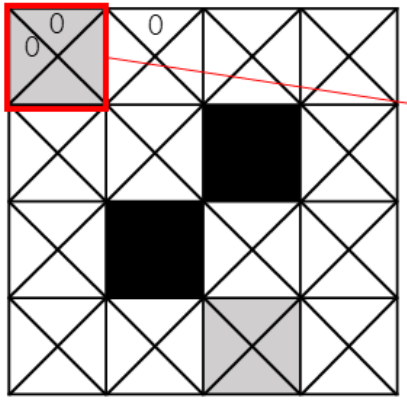
Step 1: Initialize Q-values (All 0)

Step 2: Iteratively update the Q-table until we manually stop the training.

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t))$$

Old state Learning Rate Old state

Q-Learning: Table Lookup



Actions (4)

States (16)

left	right	up	down
0	0	0	0
0	0	0	0
…	…	…	…

Q-table

left	right	up	down
0	-0.1	0	0
0	0	0	0
…	…	…	…

States (16)

Step 1: Initialize Q-values (All 0)

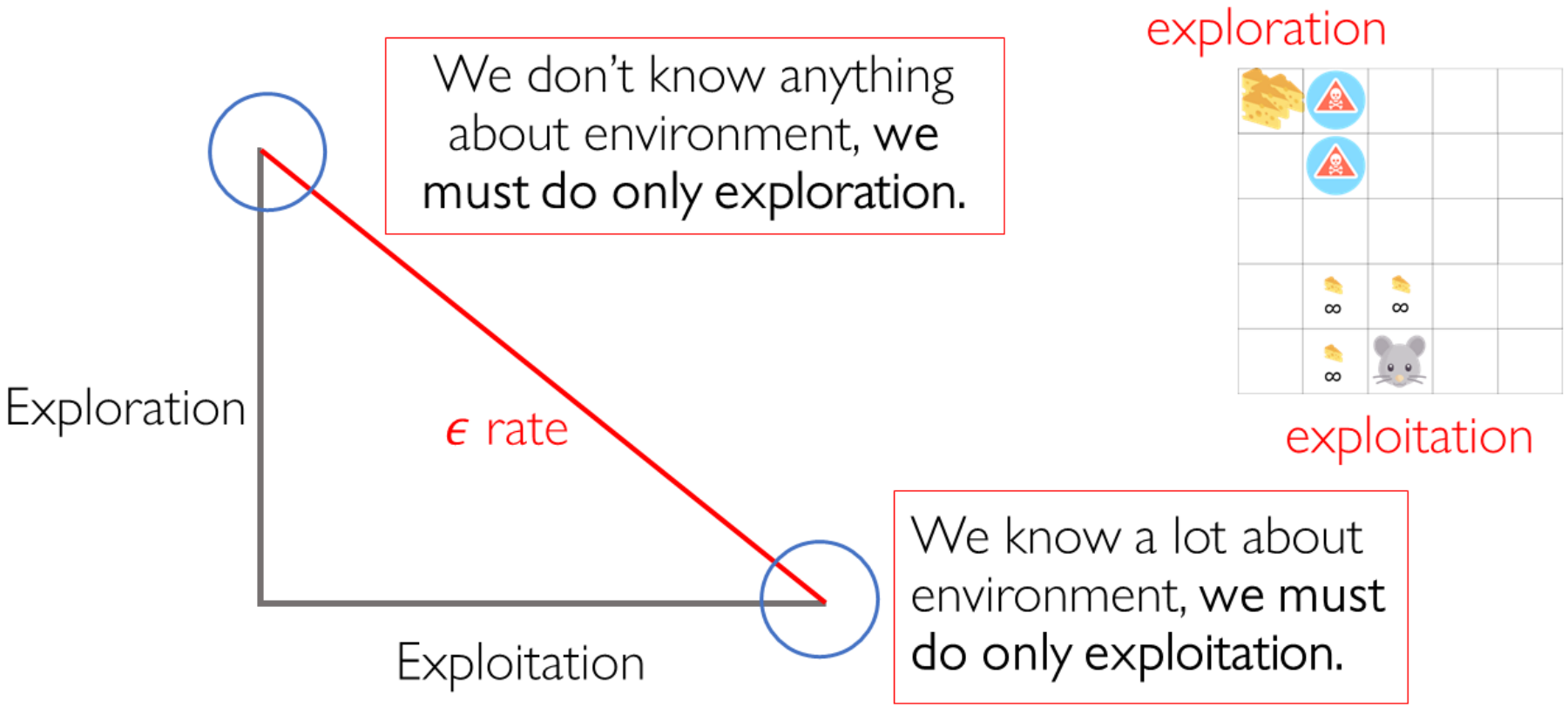
Step 2: Suppose we choose right and $\alpha = 0.1, \gamma = 0.9$:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t))$$

$$Q_{t+1}(s_t, a_t) = 0 + 0.1 * [-1 + 0.9 * 0 - 0] = -0.1$$

Exploration-Exploitation Dilemma

- **Greedy policy:** specify an exploration rate ϵ (ϵ-greedy strategy)
  - Set to 1 at the beginning.
  - Reduce it progressively through the training process.



Q-Learning: Neural Network

- Optimal Q-values should obey the Bellman equation

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

- Treat the right-hand side $r + \gamma \max_{a'} Q^*(s',a',w)$ as a target
- Minimize MSE loss by Stochastic Gradient Descent (SGD)

$$l = \left(r + \gamma \max_{a'} Q^*(s',a',w) - Q(s,a,w)\right)^2$$

- Converges to Q^* using table lookup representation (Q-table)
- Diverges when using neural networks due to:
  - Correlations between samples
  - Non-stationary targets

Deep Q-Network (DQN): Experience Replay

- To remove correlations, build dataset from agent's own experience

Correlations

$s_t, a_t, r_{t+1}, s_{t+1}$

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
...
$s_t, a_t, r_{t+1}, s_{t+1}$

s, a, r, s'

Forgetting

- Sample experiences from dataset and apply update

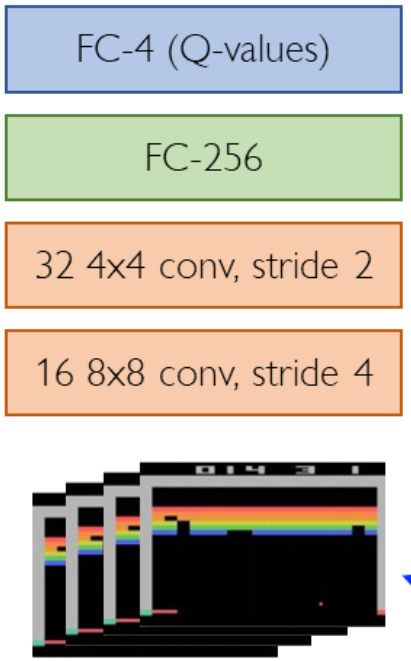
$$l = \left(r + \gamma \max_{a'} Q(s', a', w') - Q(s, a, w)\right)^2$$

- To deal with non-stationarity, target parameters w' are held fixed, which are different from parameters w for selecting actions

Deep Q-Network (DQN)

$Q(a, s; w)$:
neural network
with weights w

A single feedforward pass to compute Q-values for all actions from the current state → efficient!



Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1), Q(s_t, a_2), Q(s_t, a_3), Q(s_t, a_4)$

Number of actions between 4-18 depending on Atari game

Input: state s_t

Current state s_t: 84x84x4 stack of last 4 frames (temporal motion)
(after RGB->grayscale conversion, down-sampling, and cropping)

Improvements to DQN

- Double DQN: Remove upward bias caused by $\max_a Q(s, a, \boldsymbol{w})$
  - Current Q-network $\boldsymbol{w}$ is used to select actions
  - Older Q-network $\boldsymbol{w}'$ is used to evaluate actions

$$l = \left(r + \gamma Q(s', \underset{a'}{\operatorname{argmax}}\, Q(s', a', \boldsymbol{w}), \boldsymbol{w}') - Q(s, a, \boldsymbol{w})\right)^2$$

- Prioritized replay: Weight experience according to surprise
  - Store experience in priority queue according to DQN error

$$\left|r + \gamma \underset{a'}{\operatorname{argmax}}\, Q(s', a', \boldsymbol{w}') - Q(s, a, \boldsymbol{w})\right|$$

- Dueling network: Split Q-network into two channels (Reduce Variance)
  - Action-independent value function $V(s, v)$
  - Action-dependent advantage function $A(s, a, \boldsymbol{w})$

$$Q(s, a) = V(s, v) + A(s, a, w)$$

- Combined algorithm: 3x mean Atari score vs Nature DQN

Outline

- Reinforcement Learning
  - Markov Decision Process (MDP)
- Value-based RL
  - Q-Learning (DQN)
- **Policy-based RL**
  - **Policy Gradient (REINFORCE)**
- Games
  - AlphaGo, AlphaGo Zero, AlphaZero, MuZero
- Real World

Policy

- A policy $\boldsymbol{\pi}: \mathcal{S} \rightarrow \mathcal{A}$ specifies what action to take in each state
  - Deterministic policy: $a = \pi(s)$ (for a state, only one action is taken)
  - Stochastic policy: $\pi(a|s) = P(a|s)$ (requires sampling to take action)
- Objective: find optimal $\boldsymbol{\pi}^*$ to maximize the expected sum of rewards

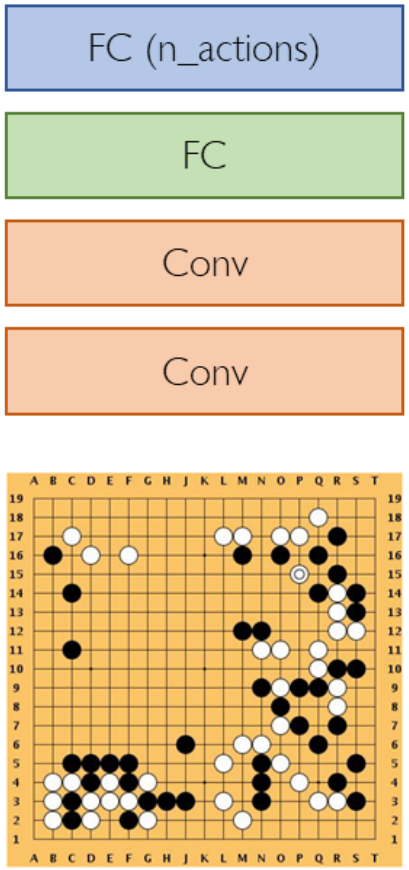
$$\pi^* = \text{argmax}\, \mathbb{E}[\sum_{t \geq 0} \gamma^t r_t | \pi]$$

$$\text{with } s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$$

• Policy in Reinforcement Learning		• Hypothesis in Supervised Learning
• Policy π	⟷	• Hypothesis f
• State s	⟷	• Feature x
• Action a	⟷	• Label y
• $\pi(a\|s) = P(a\|s)$		• $f(y\|x) = P(y\|x)$

$$\min_f \mathbb{E}\,[l(x, y); f]$$

Policy Network



Directly output the probability of actions $p(a_t|s_t)$, without learning the Q-value function.

Why?

- Guaranteed convergence to local minima
- High-dimensional (continuous) action spaces
- Stochastic policies (exploration vs. exploitation)

Question: How to train it?

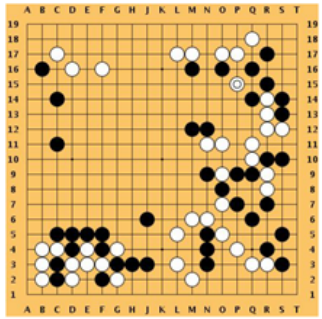
What is the training objective?

What is the loss function?

Input: states s_t

Training Objective: Increase the Reward

Case Study: Go



- Objective: Win the game!
- State: Position of all pieces
- Action: Where to put the next piece down
- Episode Eventual Reward: $J(\tau;\theta) = 1$ if win,
  $J(\tau;\theta) = 0$ otherwise

Episode: $\tau = (s_0, a_0, r_0, s_1, \dots)$

Case Study: Atari

- Objective: Bounce the ball past the other player
- State: Image at each timestep
- Action: Up / Down / Left / Right
- Discounted Reward: $J(\tau;\theta) = \mathbb{E}[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta]$

Want to find the optimal policy $\theta^* = \mathbf{argmax}\, J(\theta)$

Loss Function: from a supervised view

- In supervised learning, maximize likelihood: $\sum_i \log p(y_i|x_i)$
- Policy Gradient is exactly like supervised learning, except for:
  - No correct label → use fake label: sample action from policy
  - Training when an episode is finished
  - Scaled by the episode reward → increase the log probability for actions that work

Continuously changing dataset (episodes)

$$\text{Maximize} \sum_t J(\tau;\theta) \log p(a_t|s_t)$$

Episode reward

The state

The action we happened to sample

Loss Function: from a mathematical view

- Want to change the network's parameters θ so that action samples get higher rewards $J(\tau;\theta)$

Episode reward

Policy network

$$\nabla_\theta \mathbb{E}[J(\tau;\theta)] = \nabla_\theta \sum_t J(\tau;\theta)\, p(a_t|s_t;\theta)$$

$$= \sum_t \nabla_\theta J(\tau;\theta)\, p(a_t|s_t;\theta)$$

$$= \sum_t p(a_t|s_t;\theta)\, J(\tau;\theta) \frac{\nabla_\theta\, p(a_t|s_t;\theta)}{p(a_t|s_t;\theta)}$$

$$= \sum_t p(a_t|s_t;\theta)\, J(\tau;\theta)\, \nabla_\theta \log p(a_t|s_t;\theta)$$

$$= \mathbb{E}[J(\tau;\theta) \nabla_\theta \log p(a_t|s_t;\theta)]$$

trajectory

$$\tau = (s_0, a_0, r_0, s_1, \ldots)$$

Log-trick:

$$\nabla_\theta \log x = \frac{1}{x} \nabla_\theta x$$

Training Protocol

For episode in range(max_episodes):

observation = env.reset()

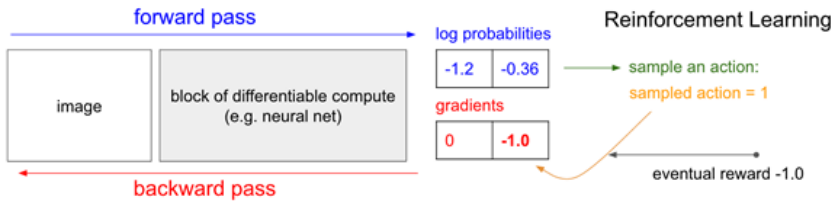
While true (for each timestep) :

1. action = choose_action(observation)
2. observation_, reward, done = env.step(action)
3. store(observation, action, reward)
4. if done:

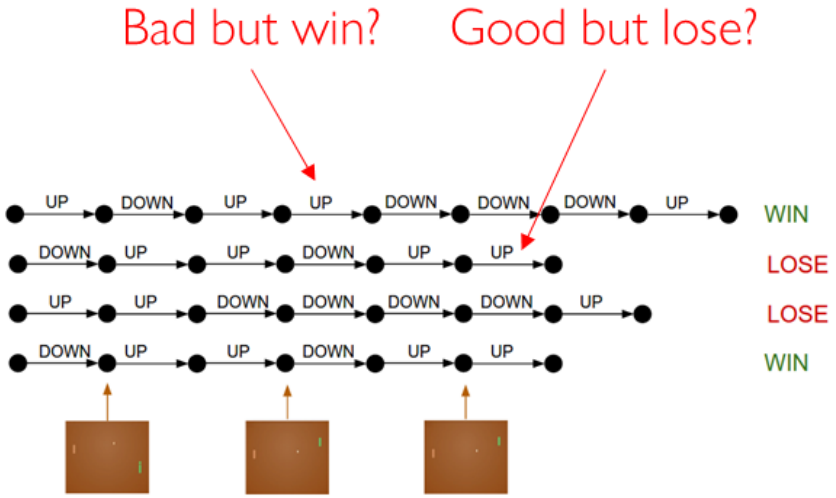
   feed-forward policy network

   maximize $\sum_t J(\tau;\theta) \log p(a_t|s_t)$

   break
5. observation = observation_

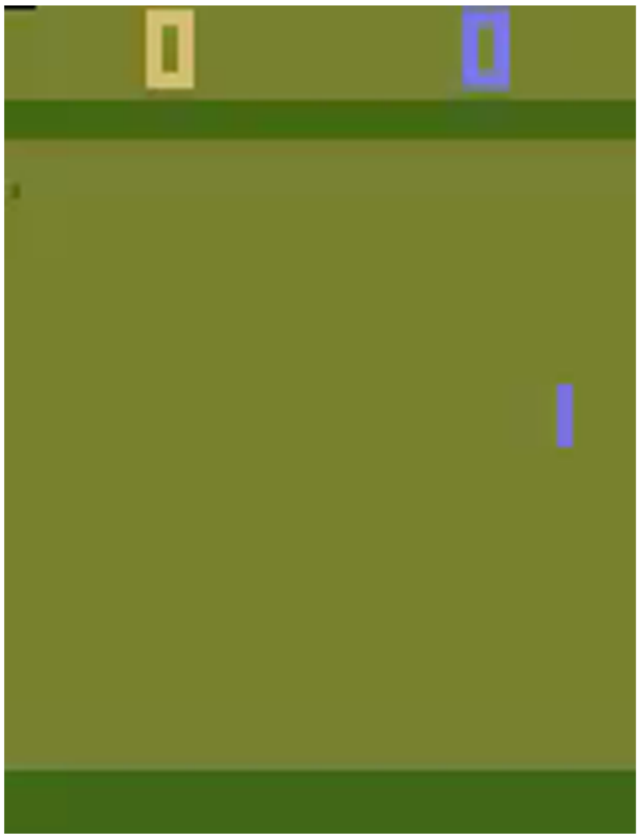


Don't care about temporary win/lose.
Focus on the average performance.



Interesting Facts

- Policy Gradient would quite easily defeat a human, where:
  - Frequent reward signals
  - Not too much long-term planning



Pong agent develops a strategy that:

- waits for the ball and then rapidly dashes to catch it just at the edge;
- launches it quickly and with high vertical velocity.

Q-Learning vs. Policy Gradient

- Policy Gradient
  - Very general but suffers from high variance so requires lots of samples.
  - Challenge: sample-efficiency
- Q-learning
  - Does not always work but when it works, usually more sample-efficient.
  - Challenge: exploration
- Guarantees
  - Policy Gradient: Converges to a local minima of $J(\theta)$, often good enough.
  - Q-learning: No guarantees since you are approximating the Bellman equation with a complex neural function approximator.
  - Guarantee if using table lookup or linear function approximator.

Q-Learning + Policy Gradient

- Monte-Carlo policy gradient still has high variance.
- We use a parameterized critic to approximate the reward:

$$J(\tau;\boldsymbol{\theta}) = Q(s,a,\boldsymbol{w})$$

- Actor-critic algorithms maintain two sets of parameters:
  - Critic updates action-value function parameters $\boldsymbol{w}$
  - Actor updates policy parameters $\boldsymbol{\theta}$, in direction suggested by critic.
- Actor-critic algorithms follow an approximate policy gradient:

$$Q(s,a,\boldsymbol{w})\nabla_\theta \log p(a_t|s_t;\boldsymbol{\theta})$$

  - Learning and optimization of $\boldsymbol{w}$ is same as value-based methods.
- Actor-critic methods will bring lower variance.

Actor-Critic Method

- Simple actor-critic algorithm based on action-value critic.
- Q-Learning and Policy Gradient iteratively.

function QAC

Initialise s, θ

Sample $a \sim \pi_\theta$

for each step **do**

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

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

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

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

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

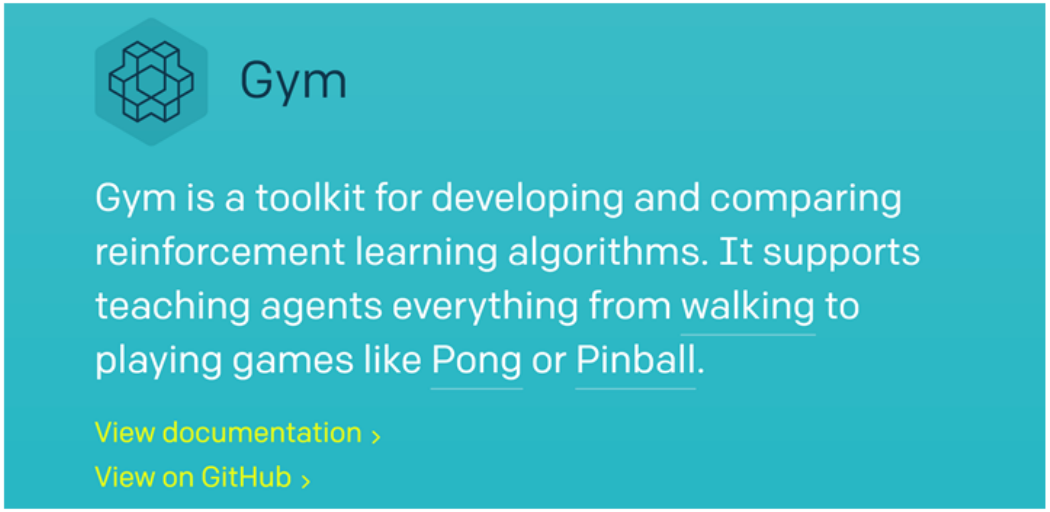
$a \leftarrow a', s \leftarrow s'$

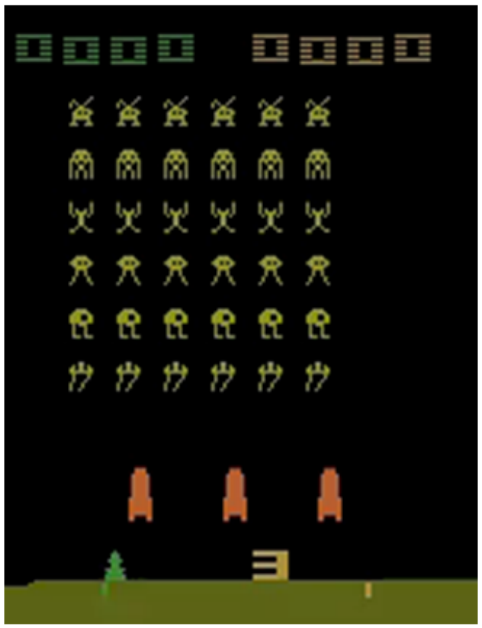
end for

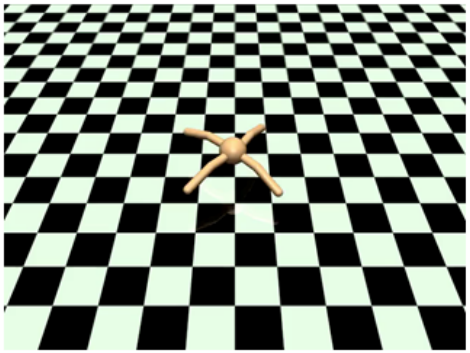
end function

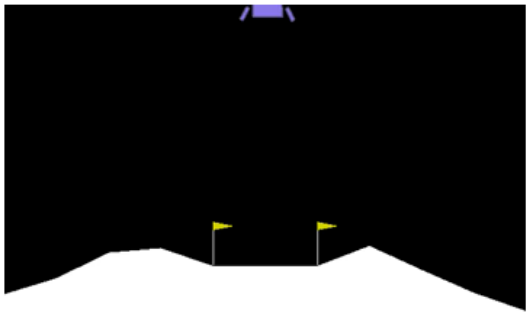
Awesome RL

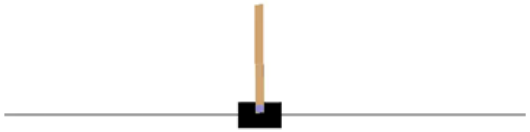
https://gym.openai.com/ https://github.com/aikorea/awesome-rl











https://morvanzhou.github.io/tutorials/machine-learning/reinforcement-learning/

单选题 1分

以下关于RL算法的描述中，错误的是?

A 时序差分学习结合了动态规划和蒙特卡罗的思想

B Q-Learning中，最优Q值无需满足Bellman方程

C RL面临探索-利用的窘境，可用ϵ-贪心法进行折中

D AC方法结合了时序差分学习和策略梯度学习

Outline

- Reinforcement Learning
  - Markov Decision Process (MDP)
- Value-based RL
  - Q-Learning (DQN)
- Policy-based RL
  - Policy Gradient (REINFORCE)
- **Games**
  - **AlphaGo, AlphaGo Zero, AlphaZero, MuZero**
- Real World

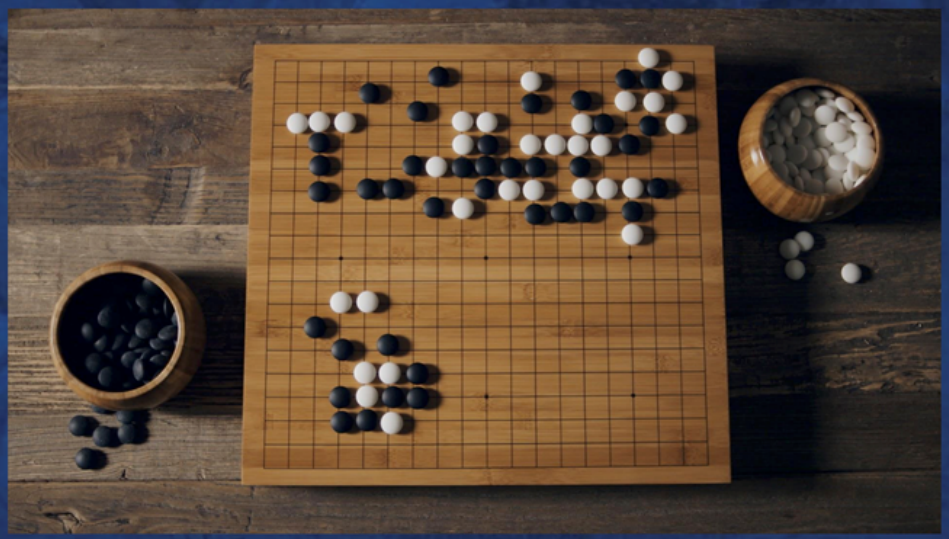
AlphaGo

Why is Go hard for computers to play?

Game tree complexity = b^d

Brute force search intractable:

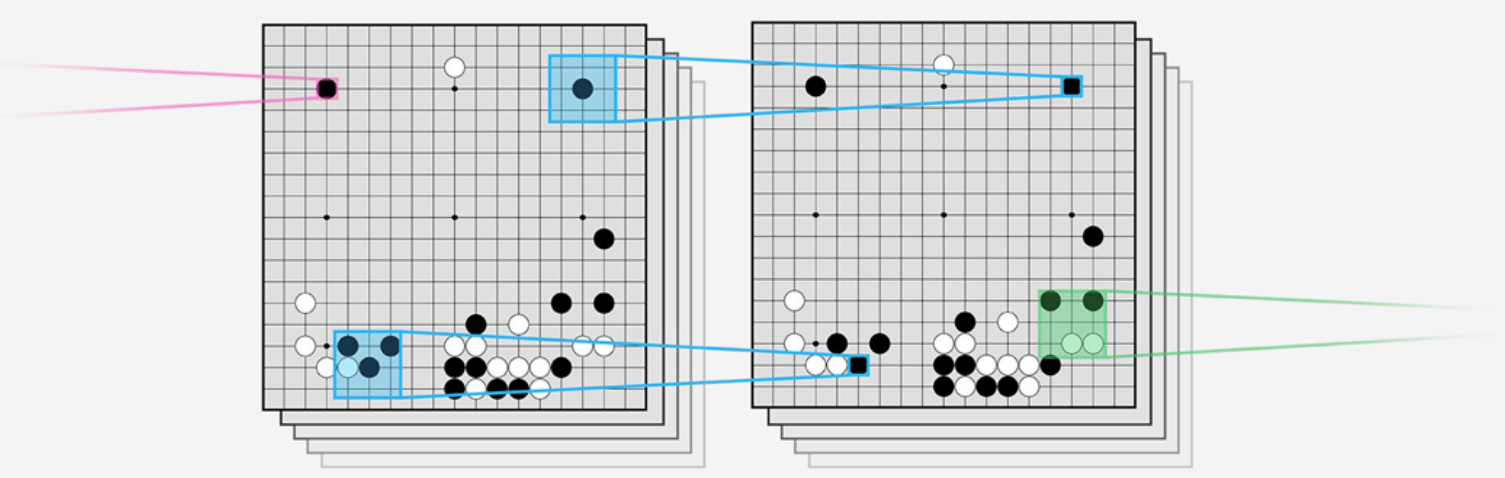
1. Search space is huge
2. "Impossible" for computers to evaluate who is winning

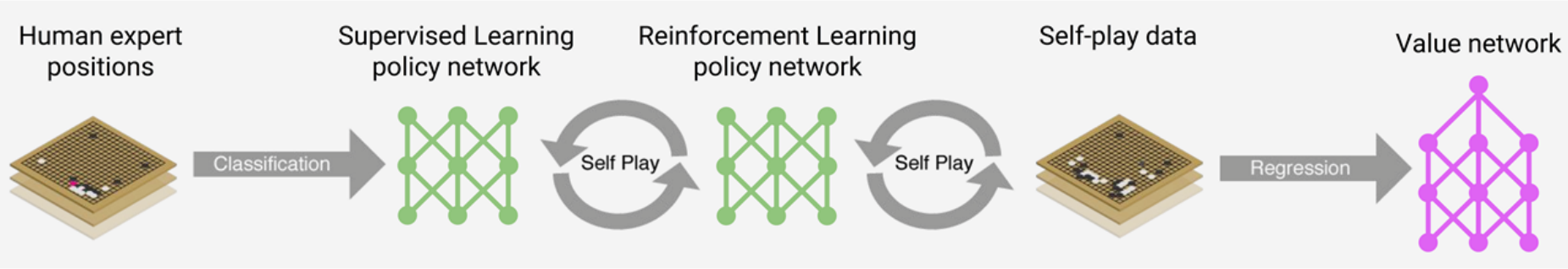


Google DeepMind

AlphaGo Overview

CNN in AlphaGo





Training: Policy Networks

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: maximise likelihood by stochastic gradient descent

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

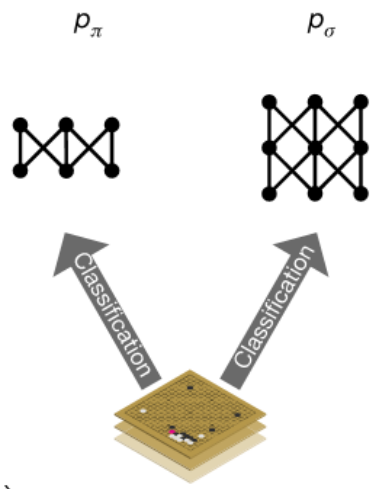
Training time: 4 weeks on 50 GPUs using Google Cloud

Results: 57% accuracy on held out test data (state-of-the art was 44%)

Google DeepMind

Training: Policy Networks

- There are two kinds of policy networks in AlphaGo.
- First one: 12-layer CNN. (p_σ)
  - Take 19x19x48 as inputs and outputs 19x19x1 as probability.
  - Many hand-crafted features are added as input features.
  - More accurate and is used in the playing procedure.
- Second one: Linear Logistic Regression (p_π)
  - Less accurate: Could only get ~24% accuracy.
  - But faster: $2\ \mu s$ for each actions.
  - Can be deployed on CPU paralleled with p_σ on GPU.
  - Used in evaluating the winning (Will be explained later).



Training: Policy Networks

Reinforcement learning of policy networks

$p_\rho(a_t|s_t)$

s

Policy network: 12 layer convolutional neural network

Training data: games of self-play between policy network

Training algorithm: maximise wins z by policy gradient reinforcement learning

Initialize with p_σ

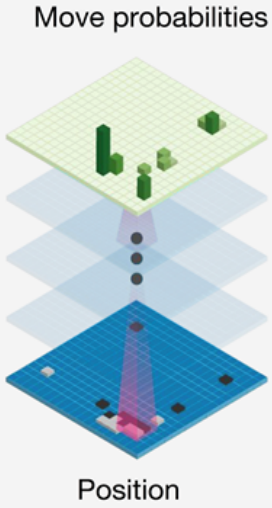
$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t$$

Self-play and training with replay

Training time: 1 week on 50 GPUs using Google Cloud

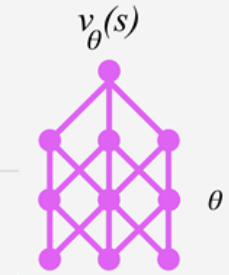
Results: 80% vs supervised learning. Raw network ~3 amateur dan.

Google DeepMind



Training: Value Networks

Reinforcement learning of value networks

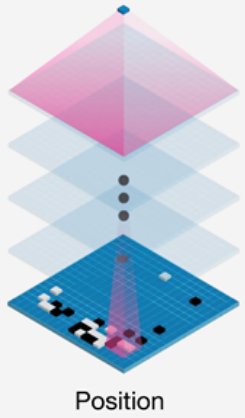


Value network: 12 layer convolutional neural network

Training data: 30 million games of self-play

Training algorithm: minimise MSE by stochastic gradient descent

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



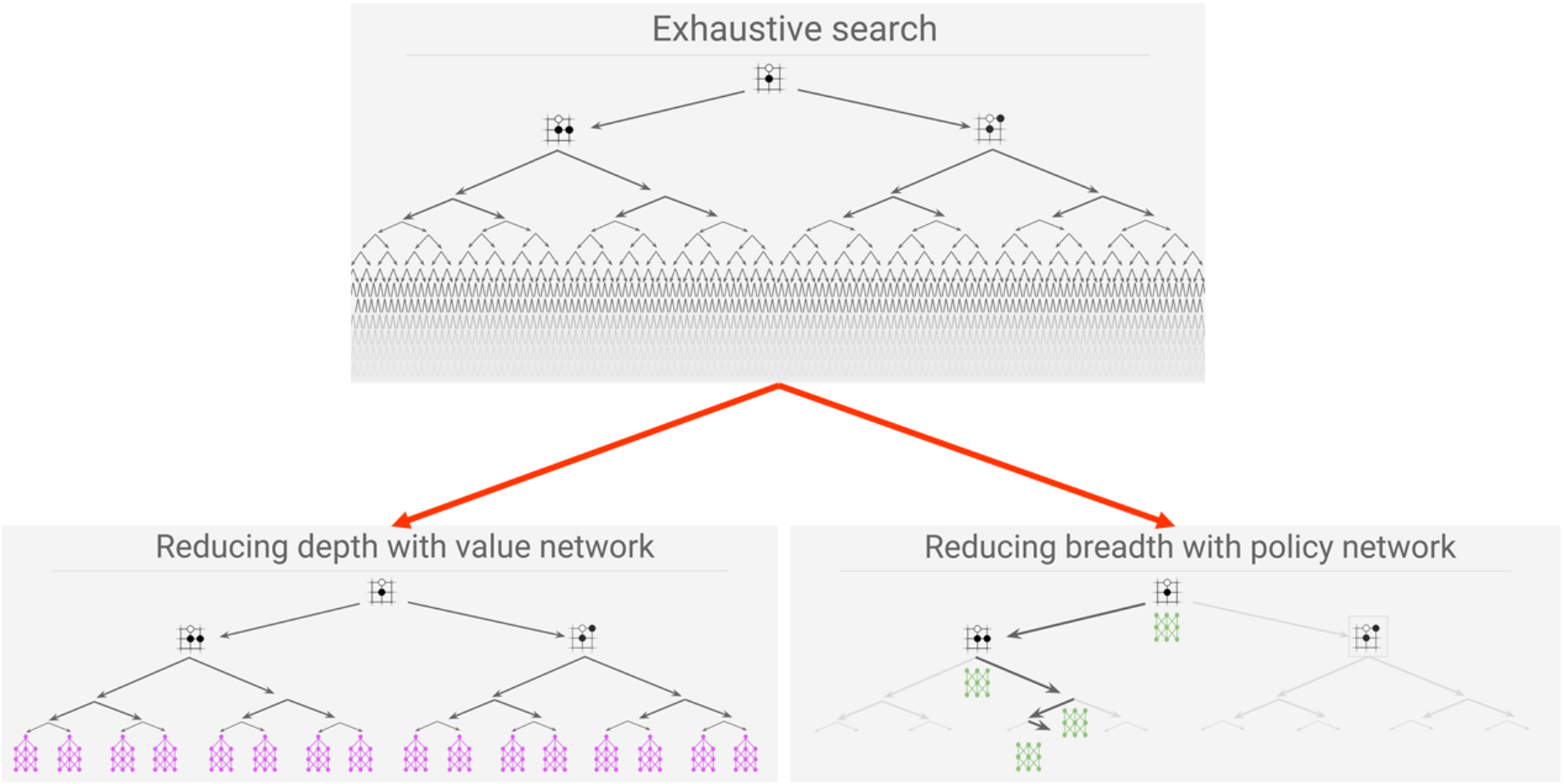
Training time: 1 week on 50 GPUs using Google Cloud

Results: First strong position evaluation function - previously thought impossible

Google DeepMind

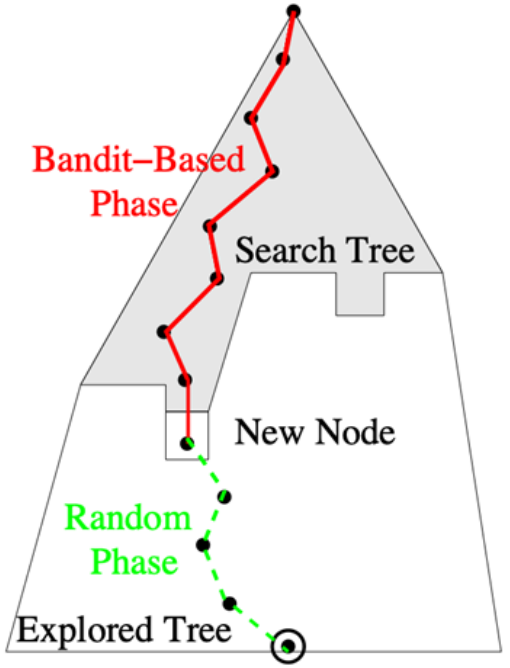
Testing: Monte Carlo Tree Search

AlphaGo = MCTS + Policy Network (Actor) + Value Network (Critic)

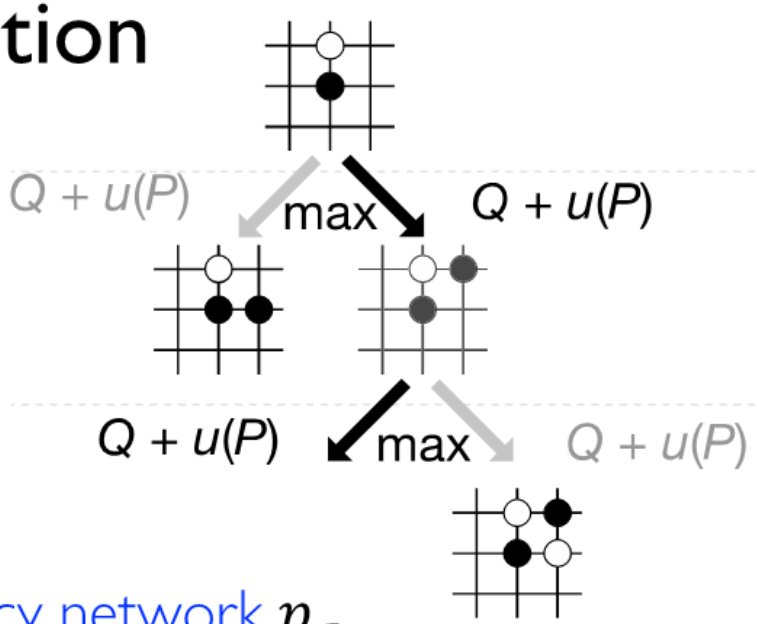


Monte Carlo Tree Search (MCTS)

- An extension of Monte Carlo method: Popular in games like Go.
  - No reward until the game is over. Discrete state only.
- Goal: Estimate $Q(s, a)$ on the current state s.
  - Simulate the complete game start from (s, a) many times and take average of final rewards.
- Three steps in each simulation:
  - Bandit Phase until a leaf node (Searched).
    - »Better version of ϵ-greedy.
  - Add new node when this node is explored.
  - Random Phase to the end of the game and compute reward.



MCTS: Selection



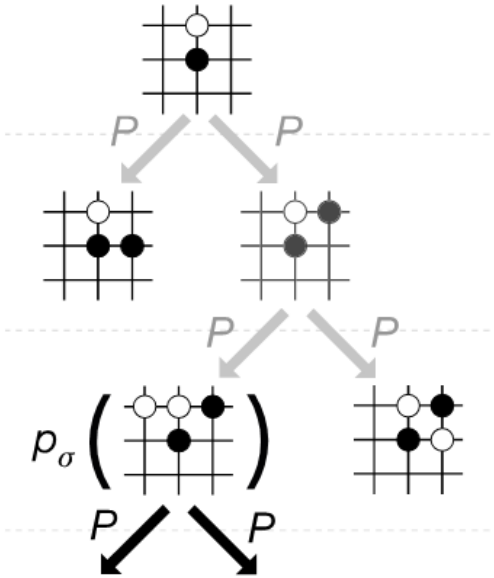
- In each step on the searched node:
  - How to choose action?
  - Exploration-Exploitation Dilemma
    - »For example, ϵ-greedy.
- AlphaGo uses a better method from policy network p_σ.

$$a_t = \underset{a}{\text{argmax}}(Q(s_t, a) + u(s_t, a)) \qquad u(s, a) \propto \frac{P(s, a)}{1 + N(s, a)}$$

  - $Q(s_t, a)$: average reward collected so far from MC simulations.
  - $P(s_t, a)$: prior expert probability provided by SL policy p_σ.
  - $N(s_t, a)$: number of times we have visited the parent node.
  - Larger $N(s_t, a)$: Better Action, but Less Search! (default in MCTS)

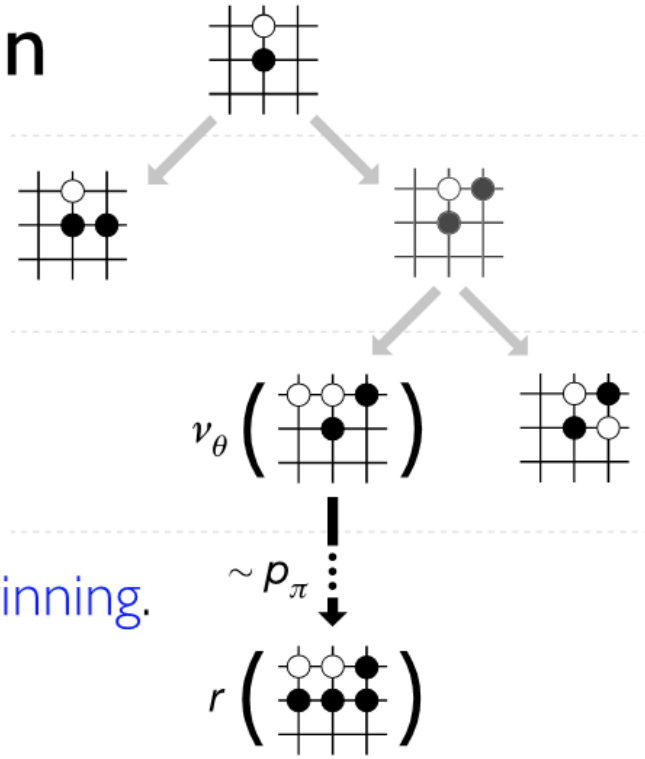
MCTS: Expansion

- Although $Q(s_t, a)$ is computed.
  - AlphaGo uses more stable criterion $N(s_t, a)$.
    - »To measure the quality of an action.
- Larger $N(s_t, a)$ will mean:
  - Both larger $Q(s_t, a)$ and $P(s_t, a)$.
  - Has been searched for enough times.



- When a leaf node gets a larger $N(s_t, a)$, it will be expanded.
- If $N(s_t, a)$ is larger than a threshold:
  - Add the successor $s'_t = \mathrm{NEXT}(s_t, a)$ into the tree as a leaf.
  - The new node is initialized with $N(s'_t, a) = 0$ and $P(s'_t, a) = p_\sigma$.

MCTS: Evaluation

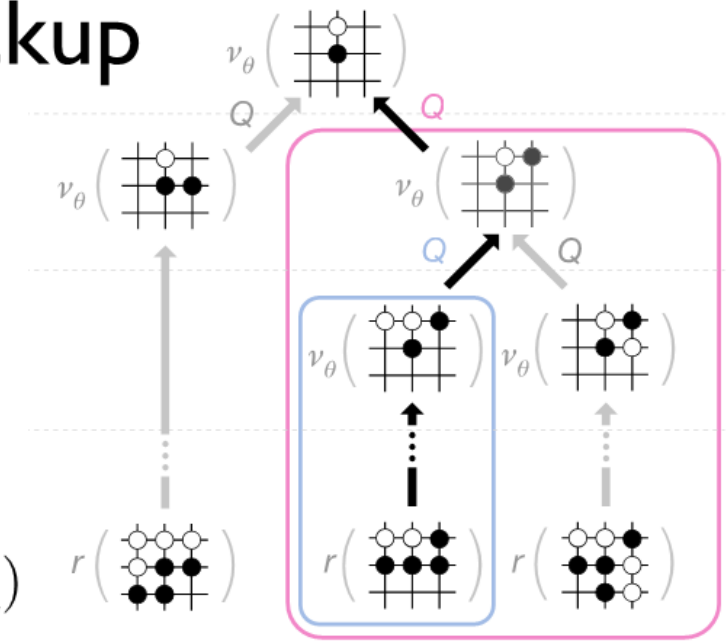


- How to compute $Q(s_t, a)$?
  - We need to compute reward (value) first.
- In the previous MCTS, random phase is used.
- In AlphaGo, the value network is used.
  - Value network predicts the probability of winning.
- And the smaller policy network p_π is used.
  - Plays a game fast and predicts the winner directly.
- The two results are combined to predict value:

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

- Using these two methods together will bring significant improvement.

MCTS: Backup



- After computing the values
  - The $Q(s_t, a)$ is computed as:

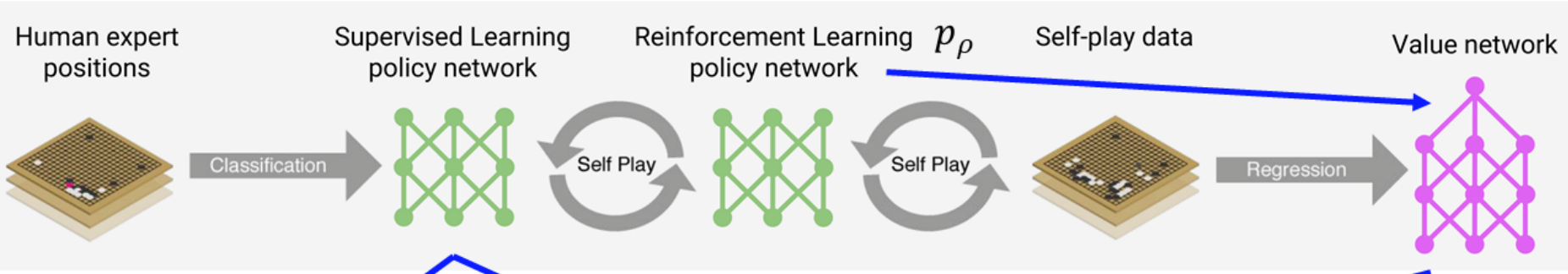
$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i)$$

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

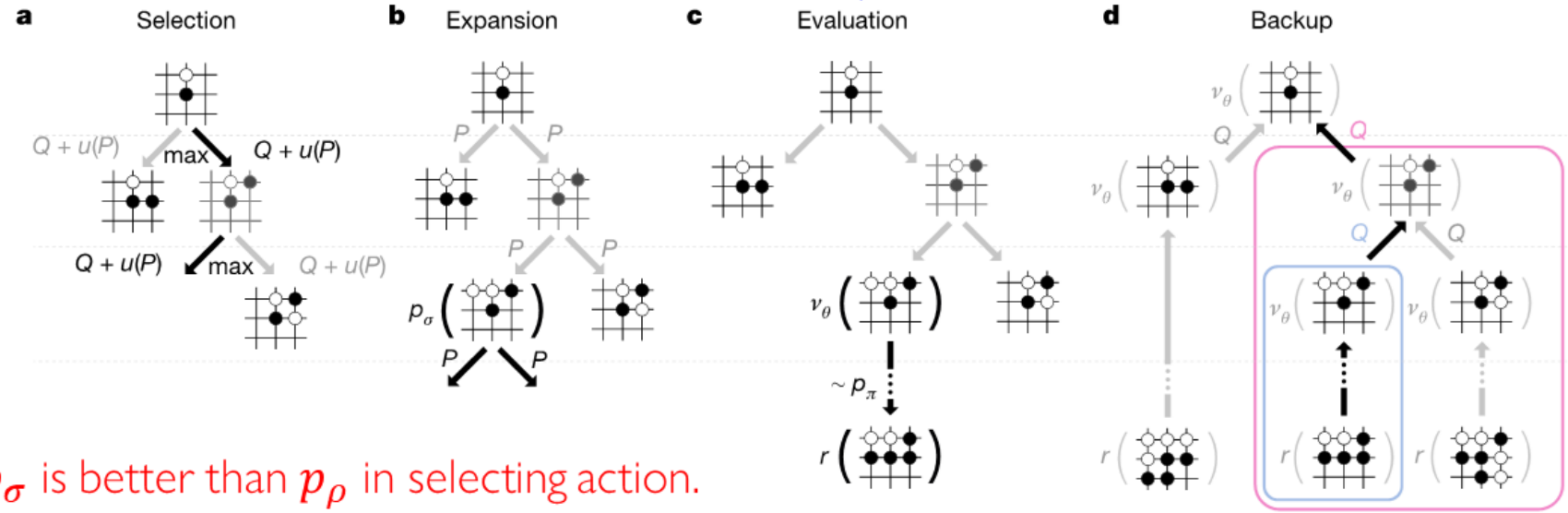
  - Where i denotes the i^{th} search in the MCTS procedure.
  - Searches could be done parallelly on a GPU cluster.
- Recall that $N(s,a)$ is the criterion for action.
  - AlphaGo will take the action with largest $N(s,a)$ as next step.
  - And beat Lee Se-dol.

AlphaGo: Summary

p_ρ is only used in training value network



Slow p_σ Fast p_π Value v_θ



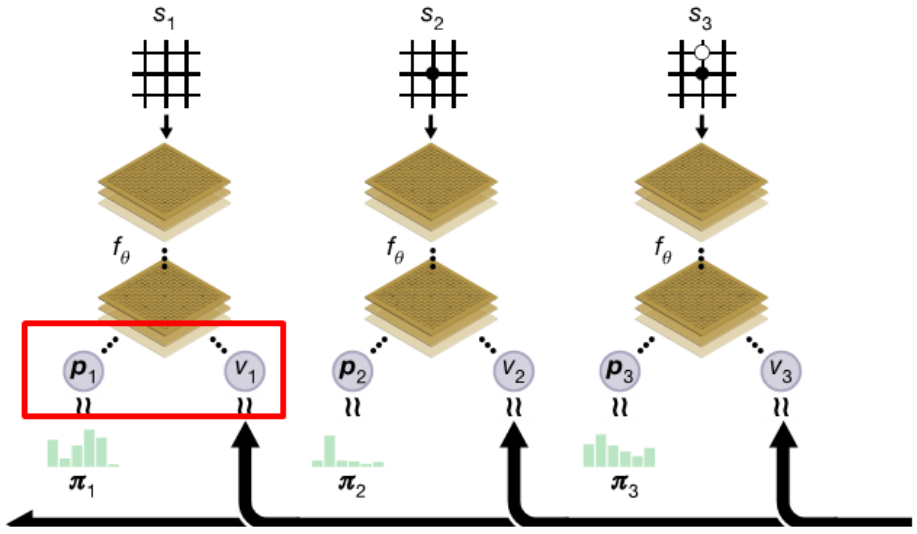
p_σ is better than p_ρ in selecting action.

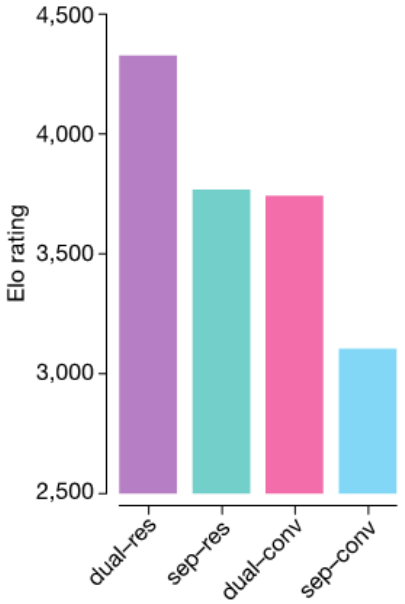
AlphaGo Zero

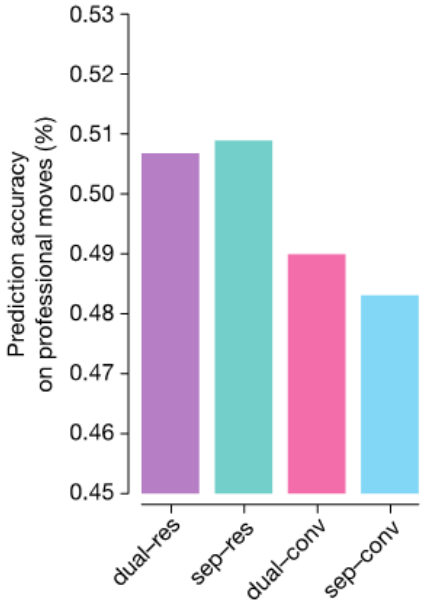
- No human data:
  - Learns solely by self-play RL, staring from random.
- No human features:
  - Only takes raw board as an input.
- Single neural network:
  - Policy and value networks are combined into one ResNet.
  - One total objective function.
    - »Multi-task learning! (lec10)
- Simpler search:
  - No randomized Monte-Carlo rollouts, only uses NN to evaluate.

AlphaGo Zero vs. AlphaGo

- First, deeper networks bring better results.
  - Residual Network with 19 or 39 residual blocks (with BN).
- Combine value network and policy network in a single network.
  - Multi-task learning with knowledge transfer brings better results.

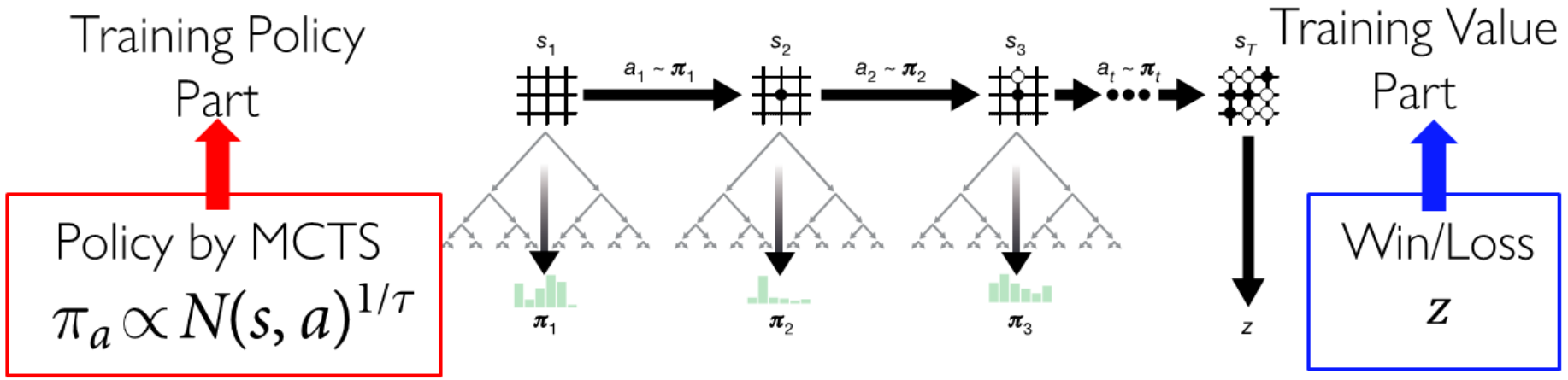






AlphaGo Zero

- AlphaGo:
  - Train-Test Mismatch: SL in Training but MCTS in Testing.
  - The policy network p_σ could only imitate human action.
    - »The true value of the action is computed as $N(s,a)$ in MCTS.
- So, why not let the policy network directly learn to predict $N(s,a)$?
  - Using MCTS in Training.

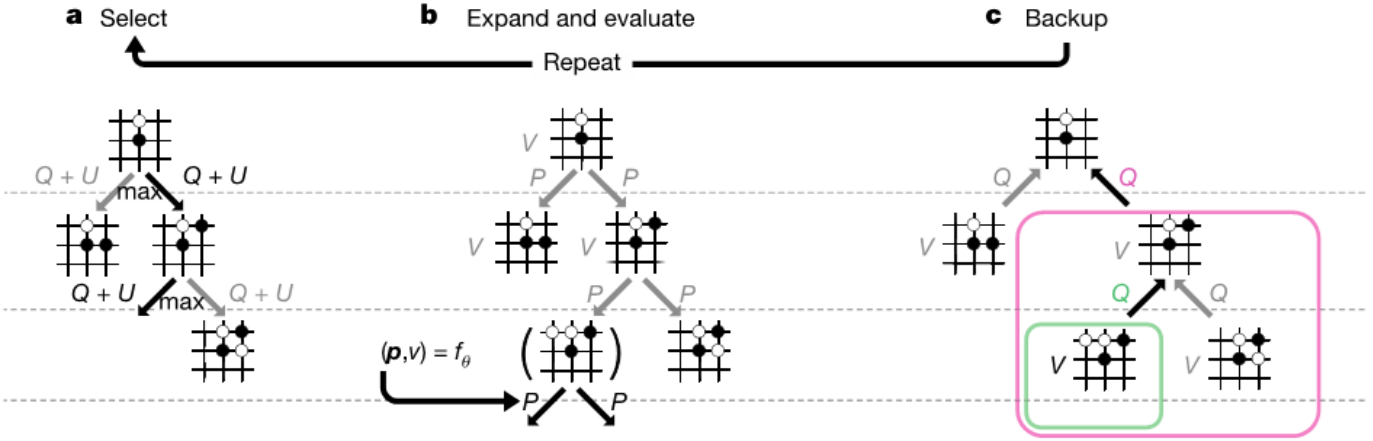


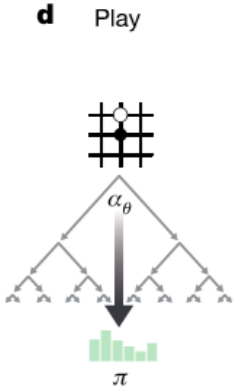
AlphaGo Zero

- The objective function combines all loss terms:

$$(\boldsymbol{p}, v) = f_\theta(s) \text{ and } l = (z - v)^2 - \boldsymbol{\pi}^{\mathrm{T}} \log \boldsymbol{p} + c\|\theta\|^2$$

Network Value Policy Regularization





MCTS similar with AlphaGo in self-Training

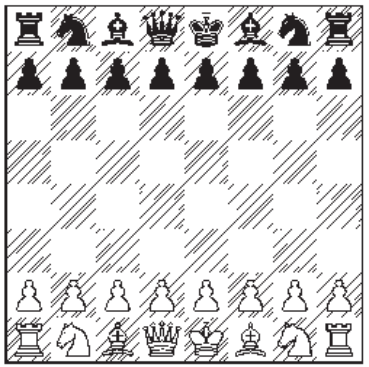
During test: using NN outputs

AlphaZero

- Made some minor modifications to accommodate a wider range of board games
  - Chess, Shogi, as well as Go

Chess

AlphaZero vs. Stockfish



W: 29.0% D: 70.6% L: 0.4%

W: 2.0% D: 97.2% L: 0.8%

Shogi

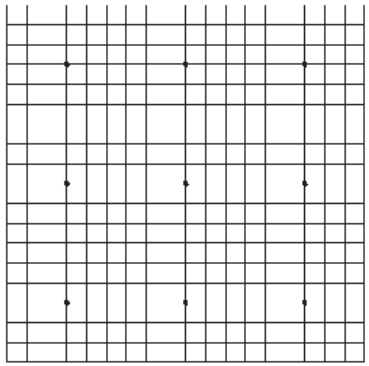
AlphaZero vs. Elmo

W: 84.2% D: 2.2% L: 13.6%

W: 98.2% D: 0.0% L: 1.8%

Go

AlphaZero vs. AG0



W: 68.9% L: 31.1%

W: 53.7% L: 46.3%

AlphaZero vs. AlphaGo Zero

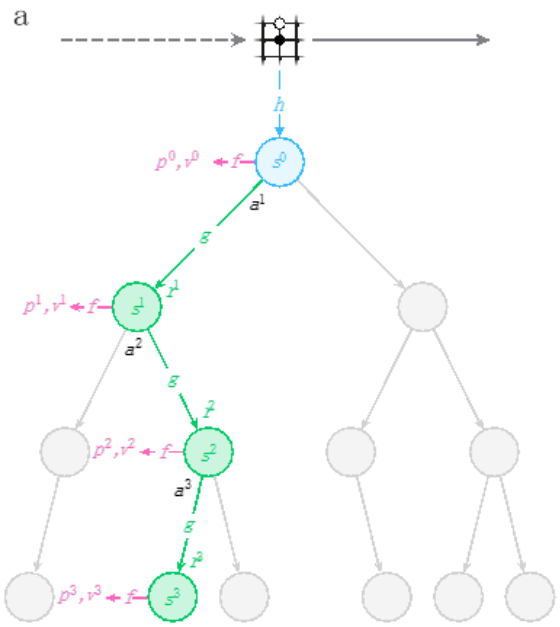
- Both chess and shogi may end in drawn outcome.
  - AlphaGo Zero estimates and optimizes the probability of winning
  - AlphaZero estimates and optimizes the expected outcome
- The rules of chess and shogi are asymmetric
  - AlphaGo Zero augments board positions via rotation and reflection in MCTS and training data
  - AlphaZero does not assume symmetry
- Alpha Zero self-plays with the latest player.
  - AlphaGo Zero maintains the best player from all previous players.

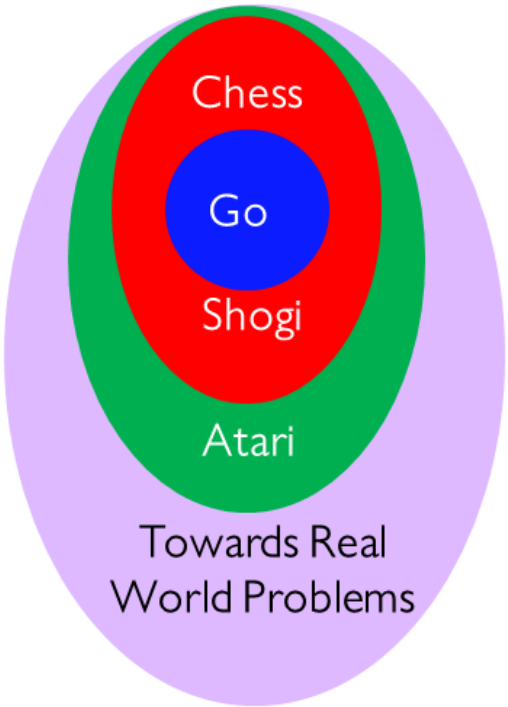
MuZero

- Bring the power of planning with MCTS and deep networks to an increasingly larger set of domains.
- Take advantage of planning with a learned model to learn more efficiently.

model-based

In AlphaZero,
A perfect simulator
based on rules is used

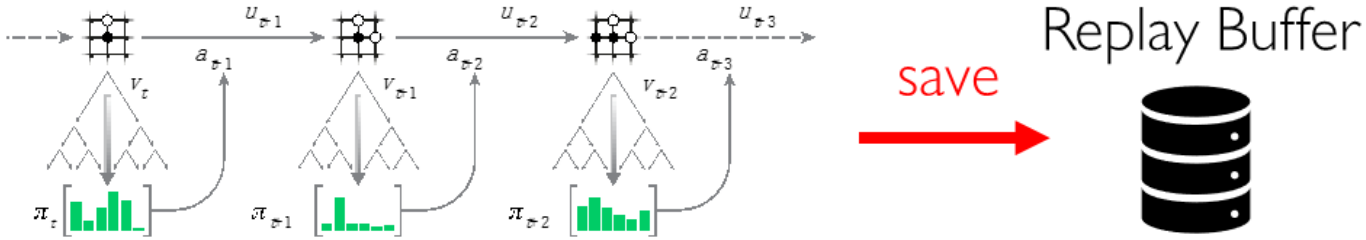




No rules, only a learned model

MuZero

- MuZero acts in the environment and stores the trajectory data into a replay buffer

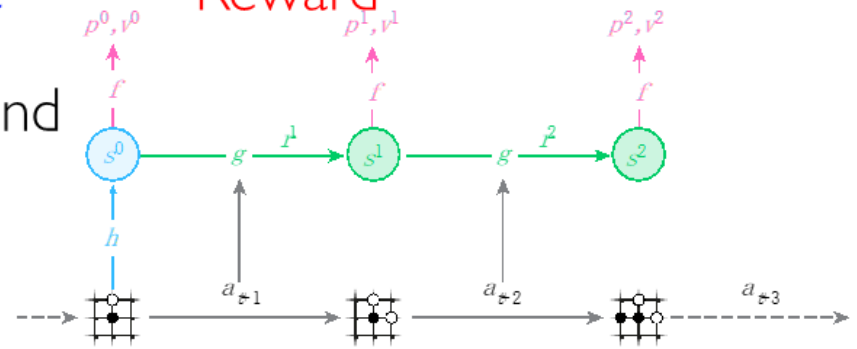


- Sample data from replay buffer, optimize

$$l_t(\theta) = \sum_{k=0}^{K} l^p(\pi_{t+k}, p_t^k) + \sum_{k=0}^{K} l^v(z_{t+k}, v_t^k) + \sum_{k=1}^{K} l^r\left(u_{t+k}, r_t^k\right) + c\|\theta\|^2$$

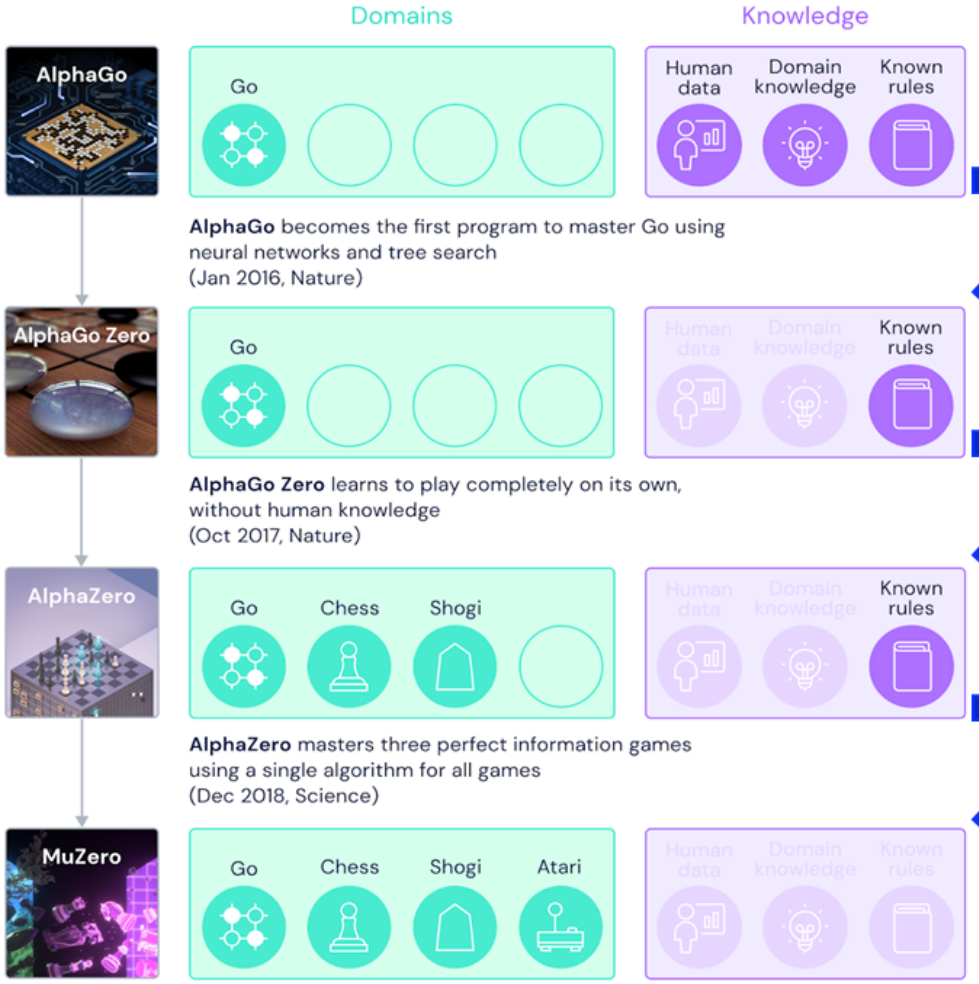
Policy Value Reward

- Jointly train all components, end to end



MuZero: Mastering Go, chess, shogi and Atari without rules. **Nature**. 2020.

Summary



Without expert moves;
Use ResNet as Base Model.

Restructure the input and output representation.

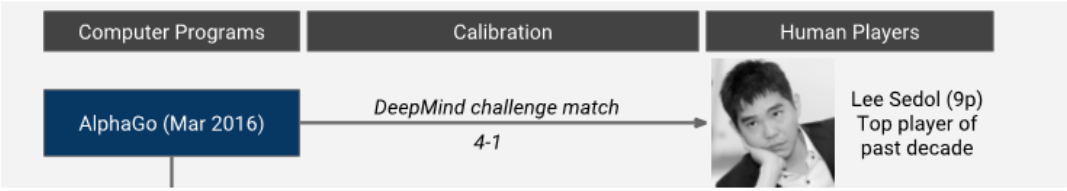
Without simulator;
Use a dynamic model to predict the future state.

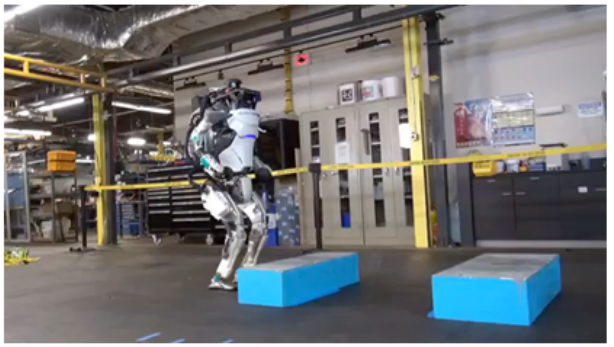
MuZero: Mastering Go, chess, shogi and Atari without rules. **Nature**. 2020.

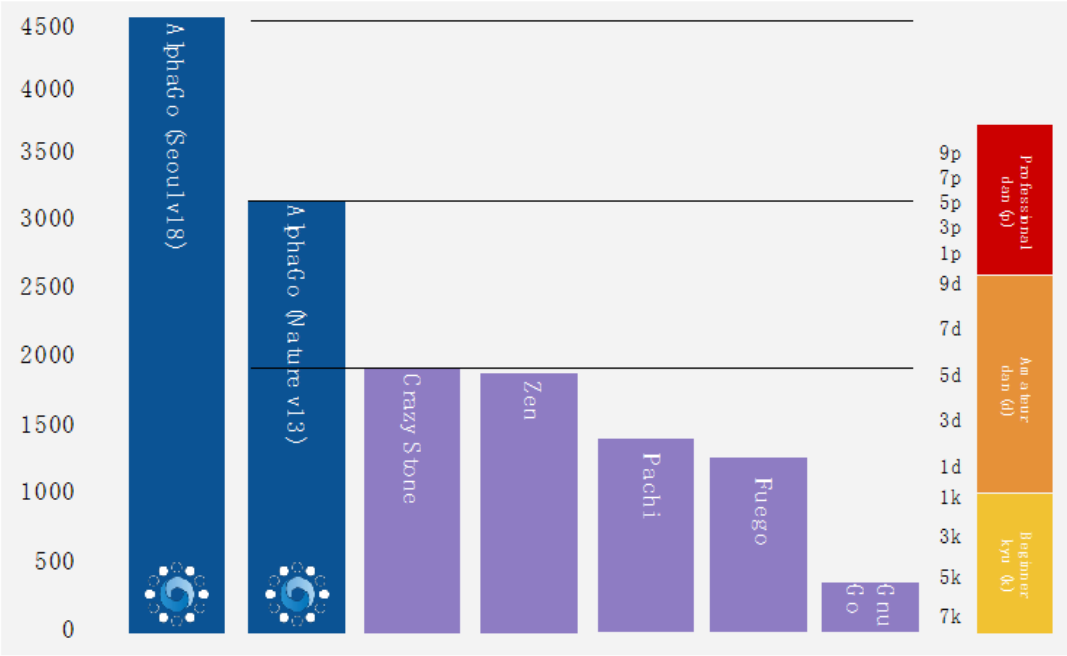
Outline

- Reinforcement Learning
  - Markov Decision Process (MDP)
- Value-based RL
  - Q-Learning (DQN)
- Policy-based RL
  - Policy Gradient (REINFORCE)
- Games
  - AlphaGo, AlphaGo Zero, AlphaZero, MuZero
- **Real World**

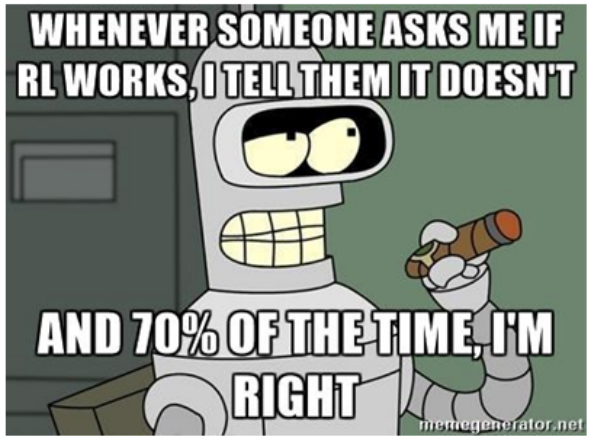
Deep RL Doesn't Work Yet

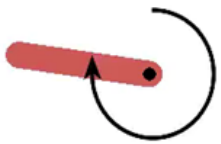


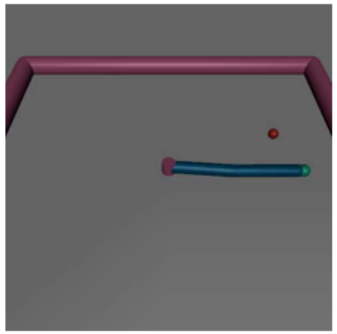


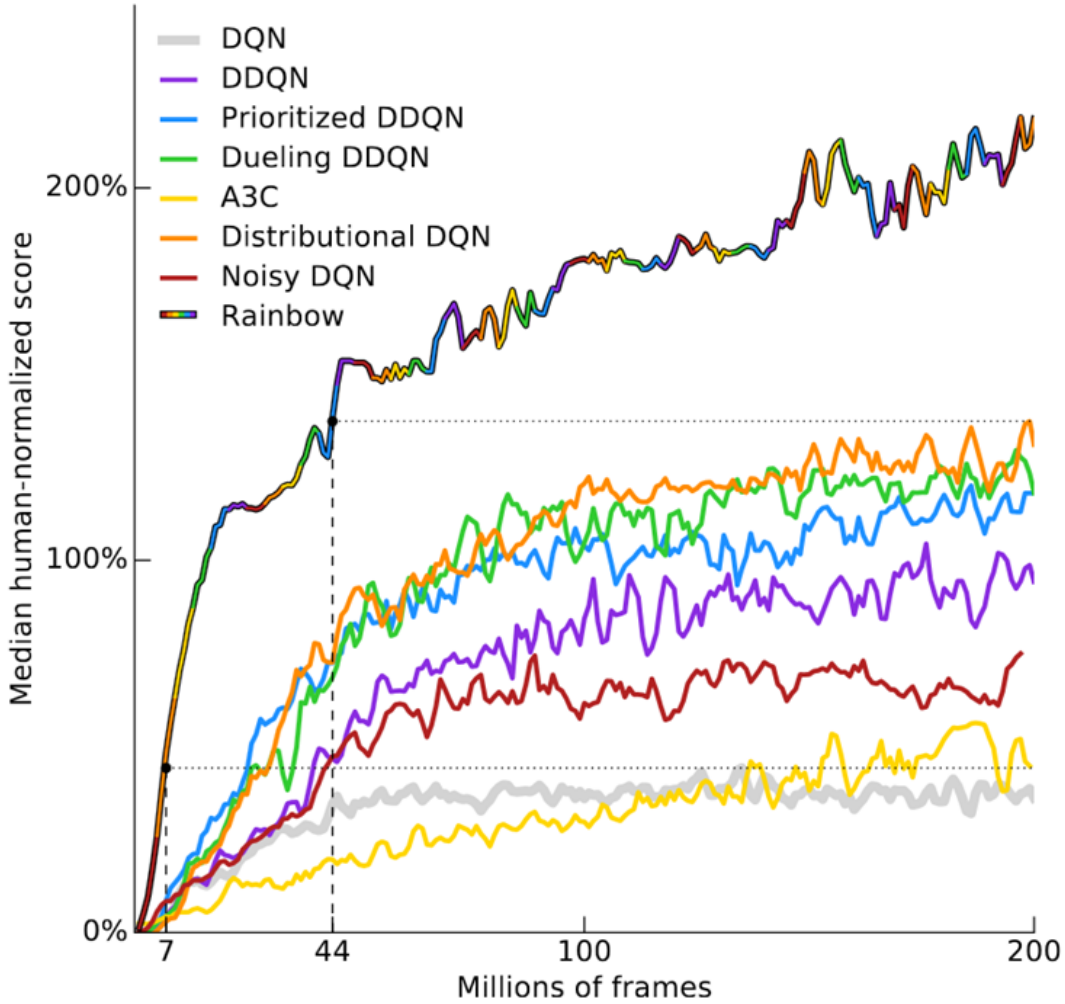


RL is Hard



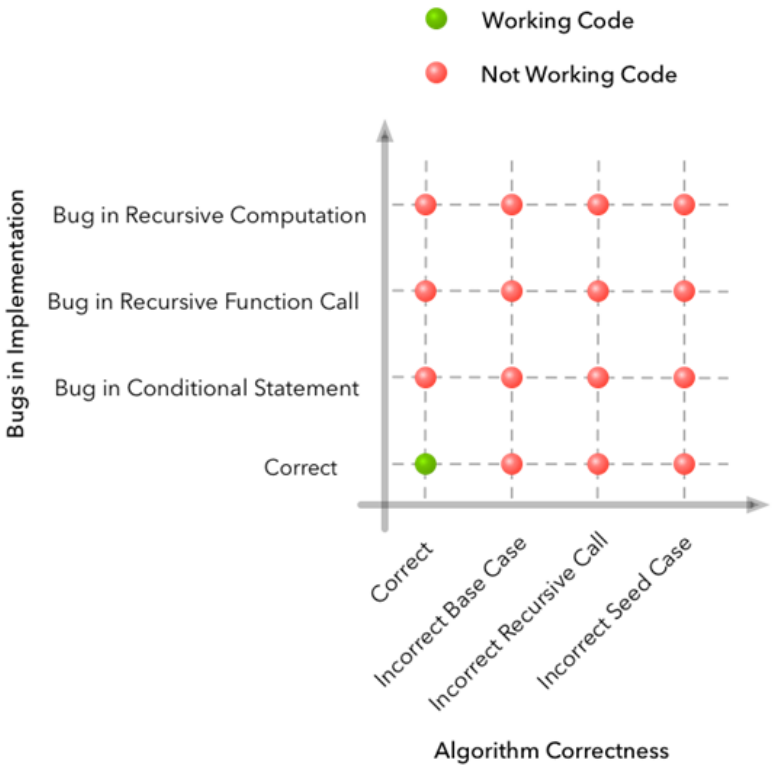




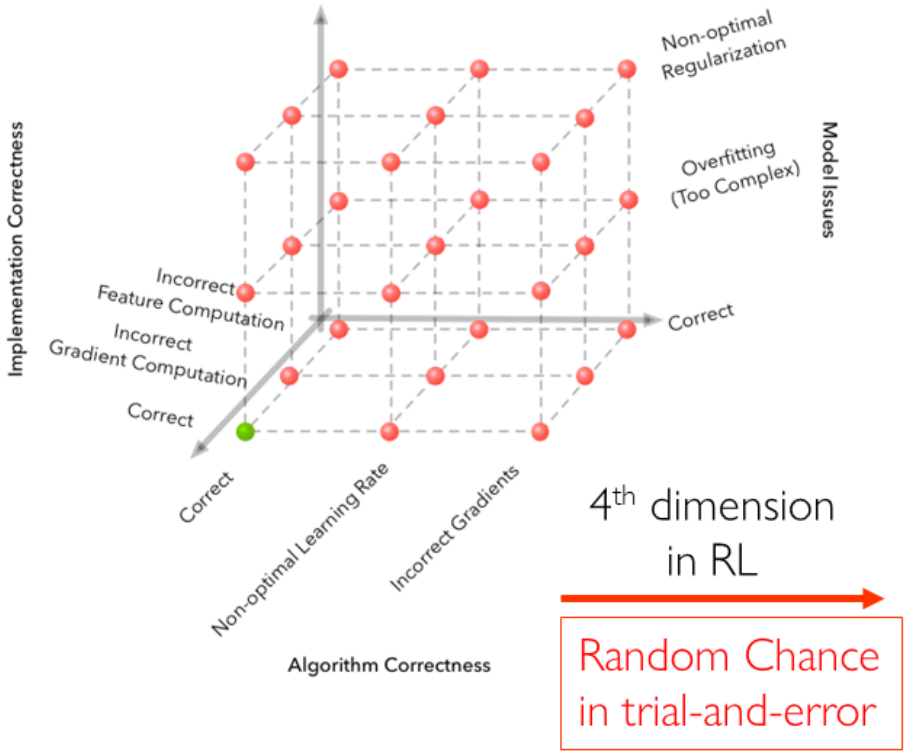


https://www.alexirpan.com/2018/02/14/rl-hard.html

Why RL is Hard?



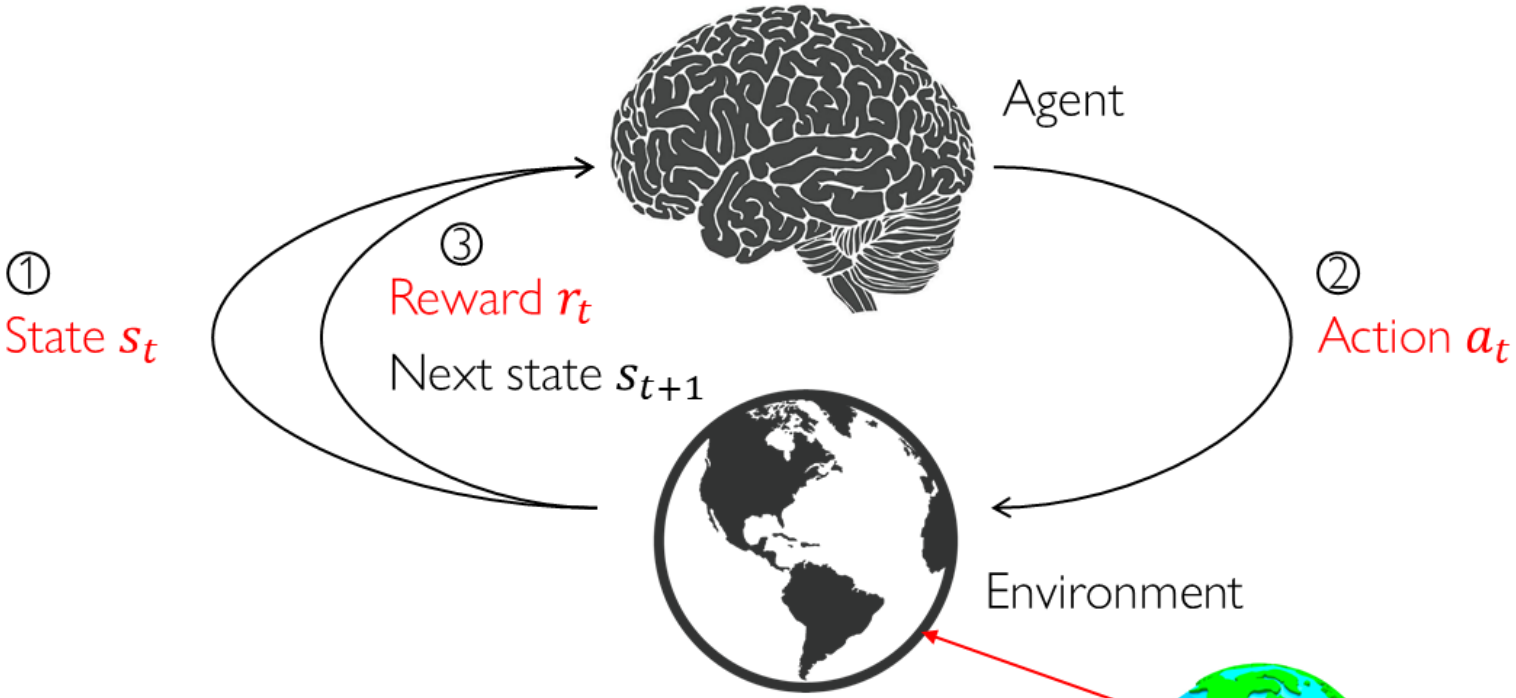
Debug in Software Engineering



Debug in Reinforcement Learning

Simulator Is All You Need

Challenging to plan due to compounding errors☹



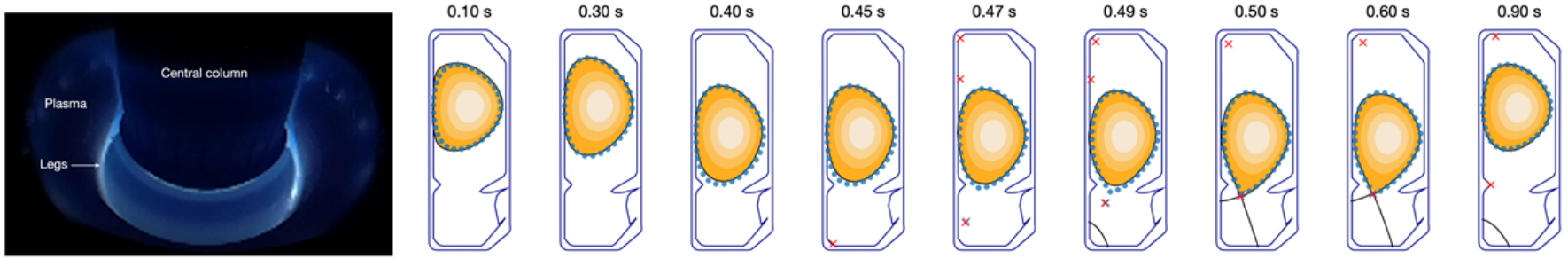
What if we have a perfect model? e.g. game rules are known☺
The reason for the success of AlphaGo and AlphaGo Zero!

Simulator!

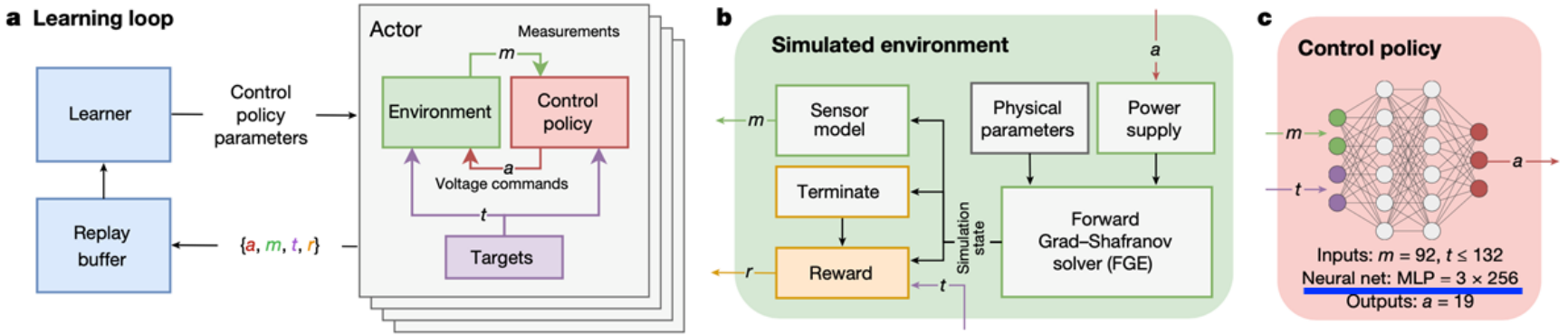
AlphaGo paper: www.nature.com/articles/nature16961

Magnetic Control of Tokamak Plasmas

- Task: control the coils in Tokamak to achieve the best voltage, making the plasmas stabilized to the target shape.



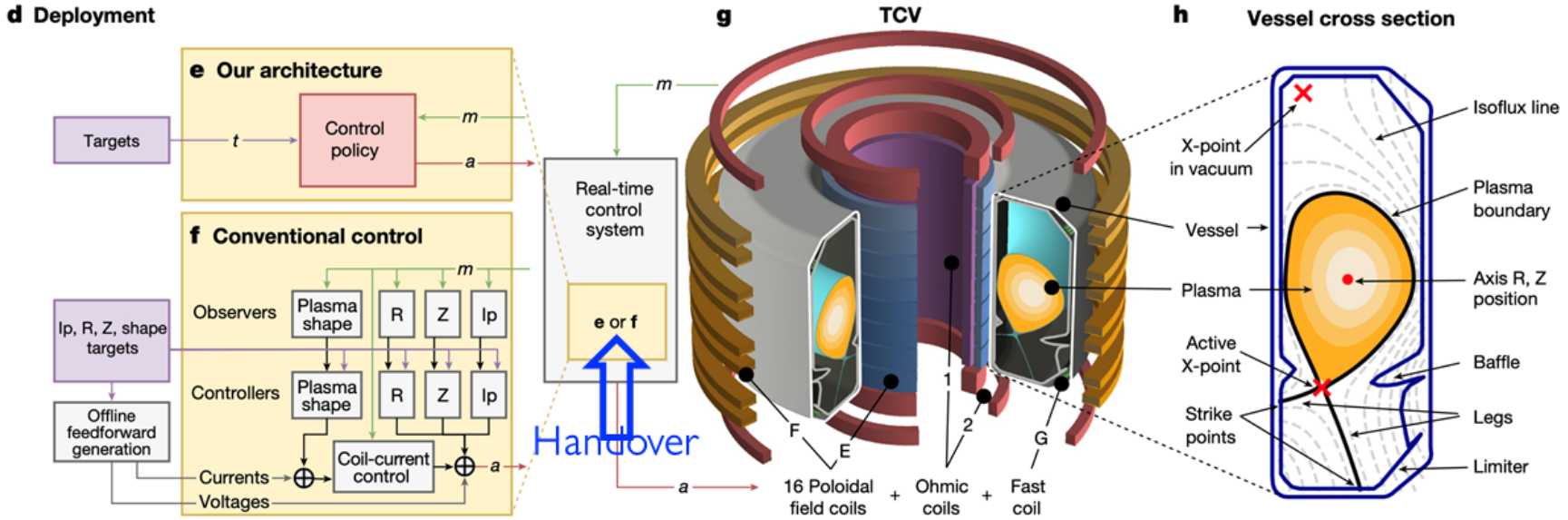
- Method: great simulator and simple control policy



Degrave, Jonas et al. Magnetic control of tokamak plasmas through deep reinforcement learning. **Nature** 2022

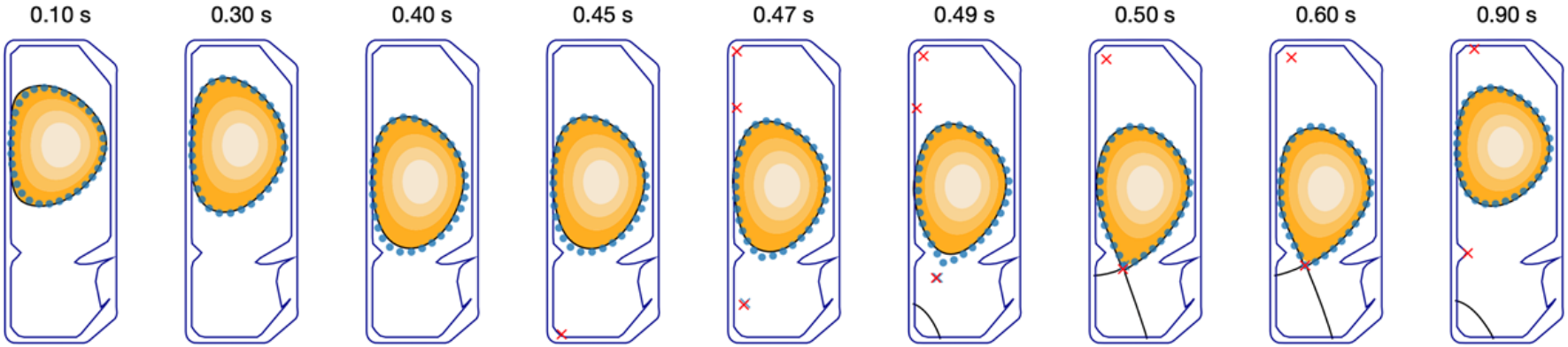
Magnetic Control of Tokamak Plasmas

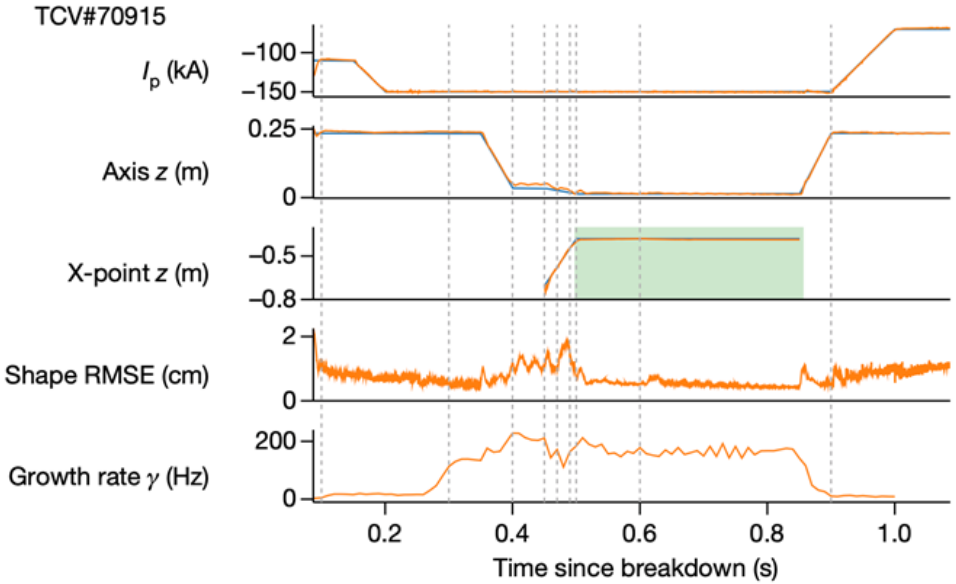
- Deployment: high-dimensional, high-frequency, closed-loop control
- Experiments in this paper:
  - Start from standard initialization
  - Adopt the conventional control for a certain time interval
  - Handover to deep model (trained from simulator) to achieve pre-defined target

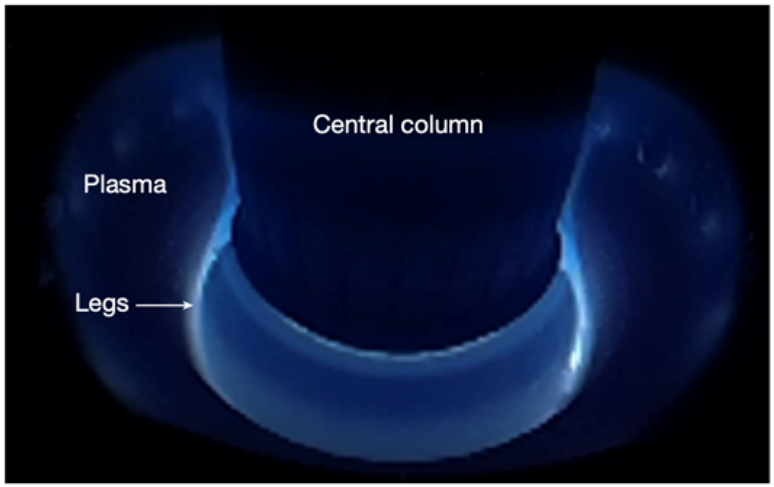


Degrave, Jonas et al. Magnetic control of tokamak plasmas through deep reinforcement learning. **Nature** 2022

Magnetic Control of Tokamak Plasmas







Successfully achieve stabilization

Degrave, Jonas et al. Magnetic control of tokamak plasmas through deep reinforcement learning. **Nature** 2022

Real-World RL

- Tasks are continual as opposed to episodic, and there is no starting from scratch
- Environment is not stationary but evolving
- Prior knowledge must be extracted from past data instead of simulators
- It is an open world, subject to exogenous interferences
- The goal is to achieve decent success, or survive, instead of blowing up the score charts

Thank You

Questions?

Mingsheng Long
mingsheng@tsinghua.edu.cn
http://ise.thss.tsinghua.edu.cn/~mlong

答疑：东主楼11区413室