

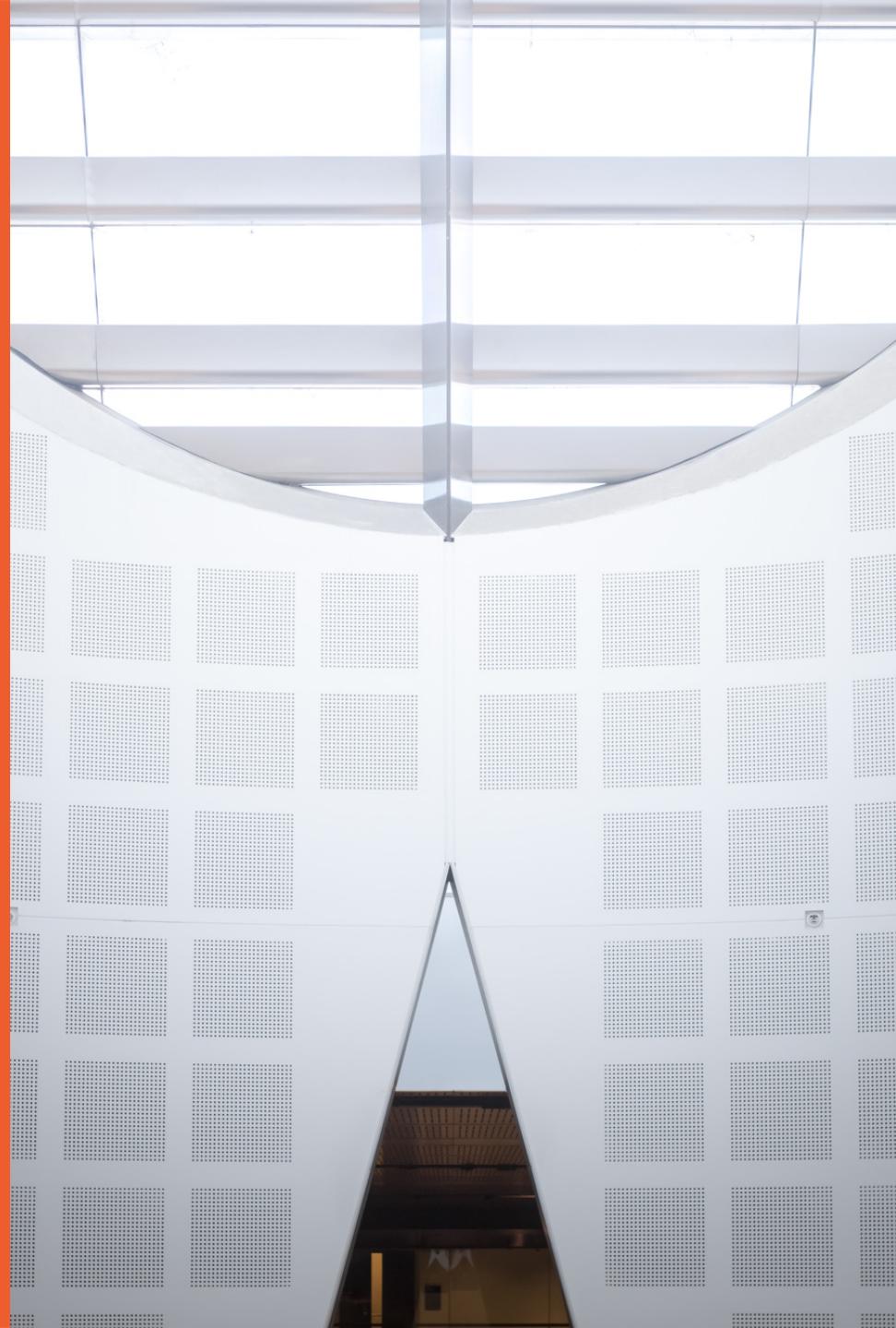
# Deep Reinforcement Learning

Dr Chang Xu  
UBTECH Sydney AI Centre

Acknowledgements:  
Shan You and Jiayan Qiu



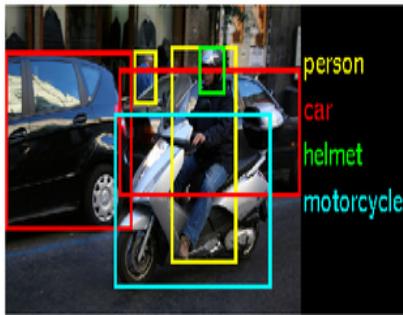
THE UNIVERSITY OF  
SYDNEY



# **Quick Review**

# Background

- Deep learning methods are making major advances in solving many low-level perceptual tasks.



See (visual object  
recognition)



# Read (text understanding)



# Hear (speech recognition)

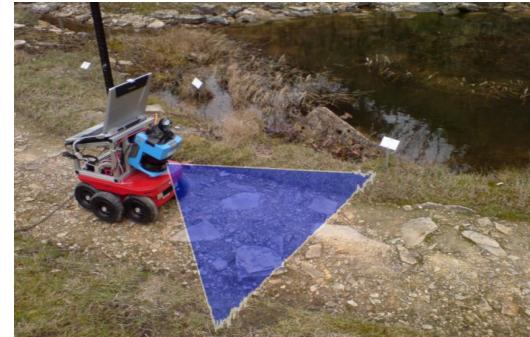
From the talk *Introduction to Deep Reinforcement Learning From Theory to Applications* by Siyi Li (HKUST)

# Background

- More sophisticated tasks that involve decision and planning require a higher level of intelligence.
- Real Artificial Intelligence system also requires the ability of reasoning, thinking and planning.



Playing Atari game

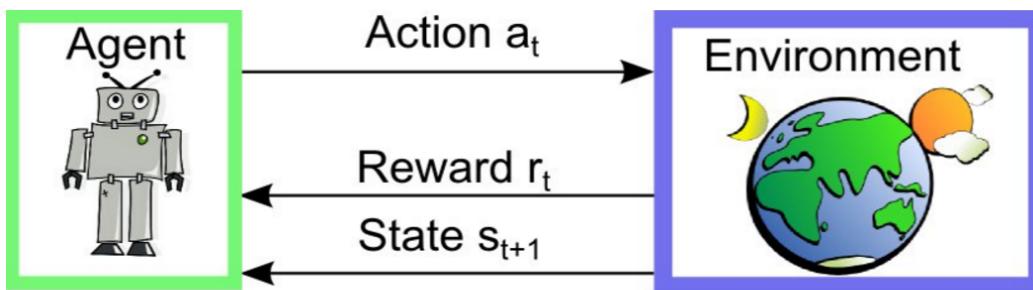


Robotic navigation

From the talk *Introduction to Deep Reinforcement Learning From Theory to Applications* by Siyi Li (HKUST)

# Reinforcement Learning (RL) in a nutshell

- RL is a general-purpose framework for decision making
  - RL is for an **agent** to **act** with an **environment**
  - Each **action** influences the agent's future **state**
  - Feedback is given by a scalar **reward** signal
  - Goal: **select actions to maximize future reward**

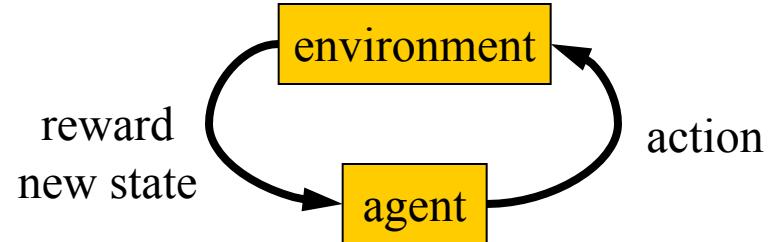


Actions: muscle contractions  
Observations: sight, smell  
Rewards: food

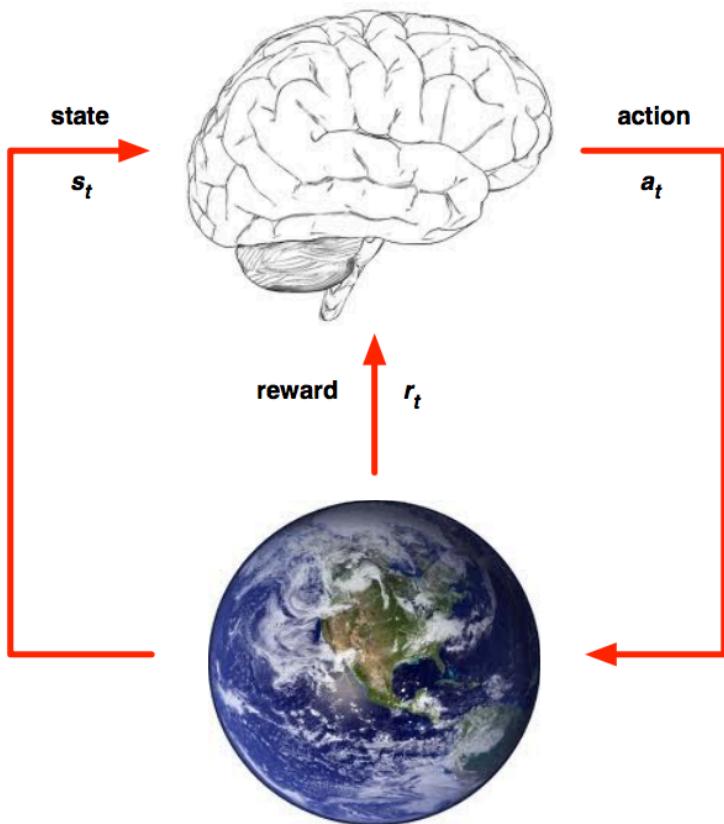
From the talk *Introduction to Deep Reinforcement Learning From Theory to Applications* by Siyi Li (HKUST)

# Markov Decision Process (MDP)

- set of states  $S$ , set of actions  $A$ , initial state  $S_0$
- transition model  $P(s,a,s')$ 
  - $P(\text{frame}_{(t)}, \text{right}, \text{frame}_{(t')}) = 0.8$
- reward function  $r(s)$ 
  - $r(\text{frame}_{(t)}) = +1$
- goal: maximize cumulative reward in the long run
- policy: mapping from  $S$  to  $A$ 
  - $a=\pi(s)$  or  $\pi(s, a)$  (deterministic vs. stochastic)
- reinforcement learning
  - transitions and rewards usually not available
  - how to change the policy based on experience
  - how to explore the environment



# Markov Decision Processes (MDPs)



- ▶ At each step  $t$  the agent:
  - ▶ Receives state  $s_t$
  - ▶ Receives scalar reward  $r_t$
  - ▶ Executes action  $a_t$
- ▶ The environment:
  - ▶ Receives action  $a_t$
  - ▶ Emits state  $s_t$
  - ▶ Emits scalar reward  $r_t$

From the *Tutorial: Deep Reinforcement Learning* by David Silver, Google DeepMind

# Computing return from rewards

- episodic (vs. continuing) tasks
  - “game over” after N steps
  - optimal policy depends on N; harder to analyze
- additive rewards
  - $V(s_0, s_1, \dots) = r(s_0) + r(s_1) + r(s_2) + \dots$
  - infinite value for continuing tasks
- discounted rewards
  - $V(s_0, s_1, \dots) = r(s_0) + \gamma * r(s_1) + \gamma^2 * r(s_2) + \dots$
  - value bounded if rewards bounded

The goal of RL is to find the policy which maximizes the expected return.

# Value Function

- A value function is the prediction of the future return
- Two definitions exist for the value function
  - State value function
  - “How much reward will I get from state s?”
  - expected return when starting in s and following  $\pi$

$$V^\pi(s) = \mathbb{E} \left\{ \sum_{k=0}^{\infty} \gamma^k r_k \middle| s_0 = s, \pi \right\}$$



$$V^\pi(s) = \mathbb{E} \{ Q^\pi(s, a) | a \sim \pi(s, \cdot) \}$$

- State-action value function
- “How much reward will I get from action a in state s?”
- expected return when starting in s, performing a, and following  $\pi$

$$Q^\pi(s, a) = \mathbb{E} \left\{ \sum_{k=0}^{\infty} \gamma^k r_k \middle| s_0 = s, a_0 = a, \pi \right\}$$

# Bellman Equation and Optimality

- Value functions decompose into **Bellman equations**, i.e. the value functions can be decomposed into *immediate reward plus discounted value of successor state*

$$V^\pi(s) = \mathbb{E} \left\{ \sum_{k=0}^{\infty} \gamma^k r_k \middle| s_0 = s, \pi \right\} \quad \longrightarrow \quad V^\pi(s) = \mathbb{E} \{ R(s, a, s') + \gamma V^\pi(s') \}$$

$$Q^\pi(s, a) = \mathbb{E} \left\{ \sum_{k=0}^{\infty} \gamma^k r_k \middle| s_0 = s, a_0 = a, \pi \right\}$$

↓

$$Q^\pi(s, a) = \mathbb{E} \{ R(s, a, s') + \gamma Q^\pi(s', a') \}$$

- An **optimal** value function is the maximum achievable value.

$$V^*(s) = \max_{\pi} V^\pi(s) \quad Q^*(s, a) = \max_{\pi} Q^\pi(s, a)$$

# Bellman Optimality Equation

- Optimality for value functions is governed by the **Bellman optimality equations**.
- Two equations:

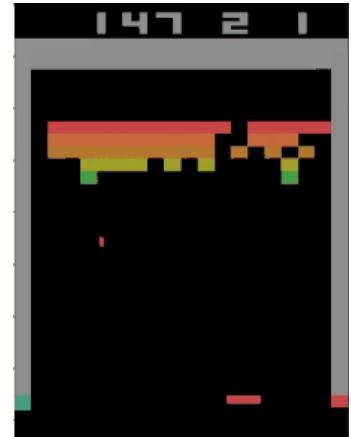
$$V^*(s) = \max_a \mathbb{E} \left\{ R(s, a, s') + \gamma V^*(s') \right\}$$

$$Q^*(s, a) = \mathbb{E} \left\{ R(s, a, s') + \max_{a'} \gamma Q^*(s', a') \right\}$$

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

# **Q-Learning**

# Initializing the $Q(s, a)$ function



states

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Noop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Fire	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Right	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Left	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

# Q-Learning

$$Q^*(s, a) = \mathbb{E} \left\{ R(s, a, s') + \max_{a'} \gamma Q^*(s', a') \right\}$$

Initialize  $Q(s, a)$  arbitrarily.

Start with  $s$ .

Before taking action  $a$ , we calculate the current expected return as

$$Q(s, a)$$

After taking action  $a$ , and observing  $r$  and  $s'$ , we calculate the target expected return as

$$\overbrace{R(s, a, s')} + \max_{a'} \gamma Q(s', a')$$

$$\Delta Q(s, a) = R(s, a, s') + \max_{a'} \gamma Q(s', a') - Q(s, a)$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha \Delta Q(s, a)$$

# Q-Learning

$$Q^*(s, a) = \mathbb{E} \left\{ R(s, a, s') + \max_{a'} \gamma Q^*(s', a') \right\}$$

Initialize  $Q(s, a)$  arbitrarily.

Repeat (for each episode)

    Initialize  $s$

    Repeat (for each step of the episode)

        Choose  $a$  from  $s$  using a policy

        Take action  $a$ , observe  $r, s'$

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

$s \leftarrow s'$

# Exploration and Exploitation

Initialize  $Q(s, a)$  arbitrarily.

Repeat (for each episode)

    Initialize  $s$

    Repeat (for each step of the episode)

        Choose  $a$  from  $s$  using a policy

        Take action  $a$ , observe  $r, s'$

        Update  $Q$  and  $s$ ....

Random policy; Exploration

Greedy policy; Exploitation

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

$\epsilon$ -greedy policy: With probability  $\epsilon$  select a random action

# Deep RL: Deep Learning + RL

- Traditional RL
- low-dimensional state spaces
- handcrafted features
- DL's representation power + RL's generalization ability
  - RL defines the objective
  - Deep Learning learns the representation

# **Deep Q-Learning**

# Deep Q-Learning



210

actions

states

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Noop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Fire	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Right	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Left	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$$\# \text{states} = 256^{210 \times 160}$$

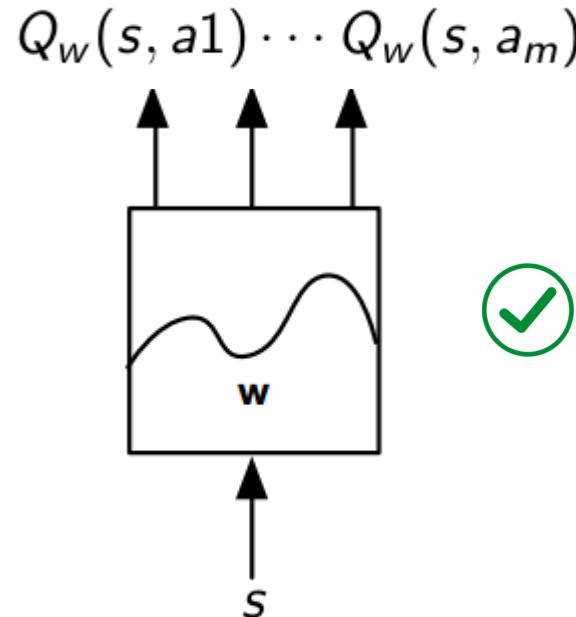
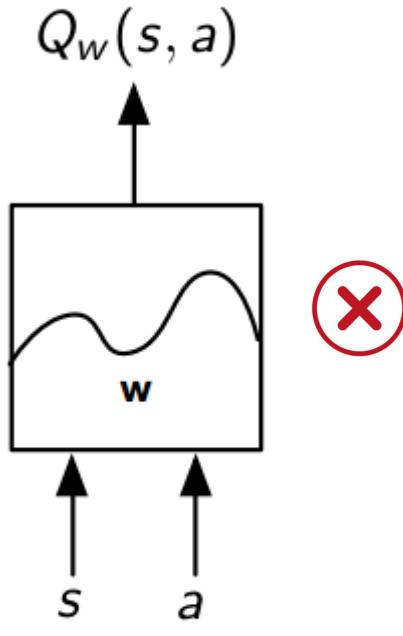
Value Function Approximation

$$Q(s, a) = f(s, a, w)$$

# Q-Networks

- Represent the state-action value function (discrete actions) by Q-network with weights  $w$

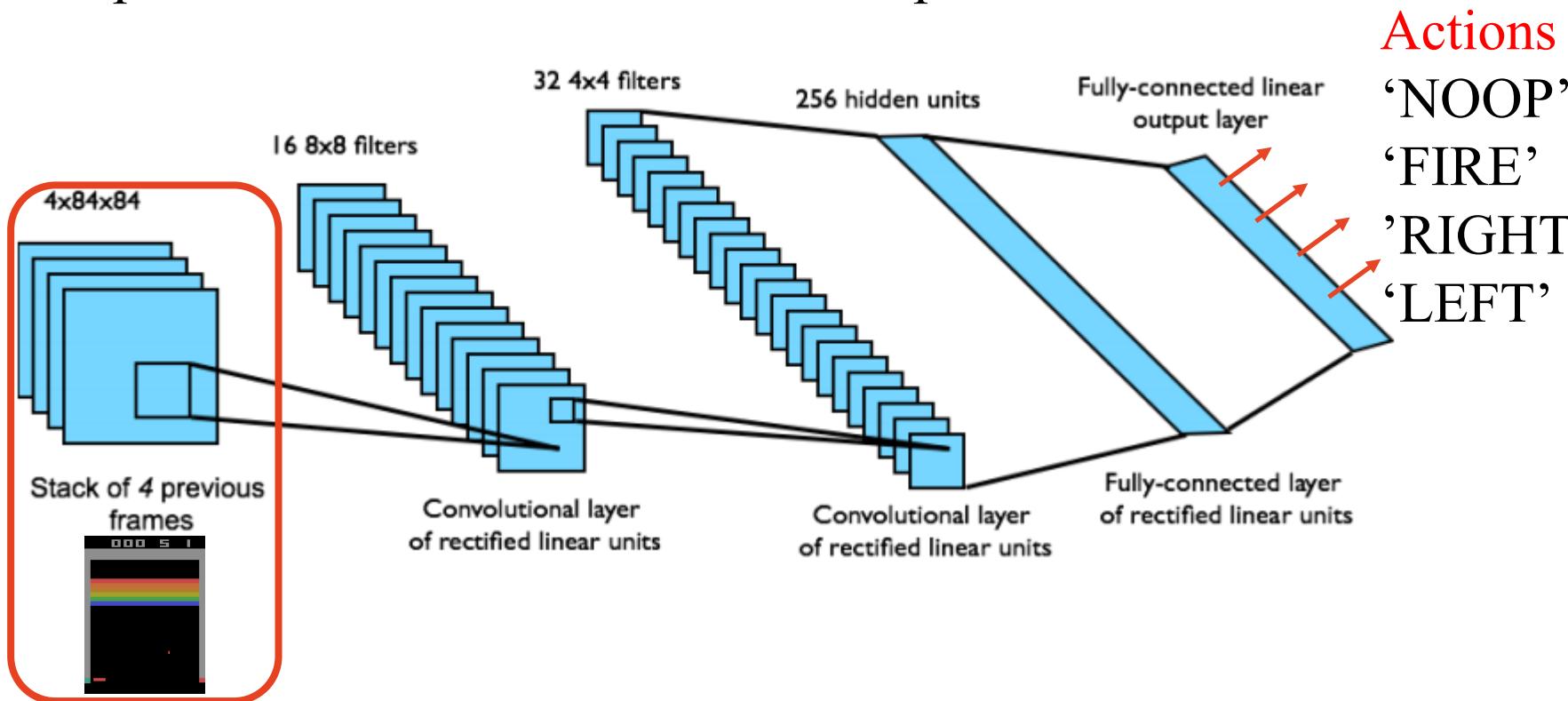
$$Q_w(s, a) \approx Q^*(s, a)$$



From the Tutorial: Deep Reinforcement Learning by David Silver, Google DeepMind

# Deep Q-Learning

- End-to-end learning of state-action values from raw pixels
- Input state is stack of raw pixels from last 4 frames
- Output are state-action values from all possible actions



From the Tutorial: Deep Reinforcement Learning by David Silver, Google DeepMind

# Deep Q-Networks (DQN)

- Optimal Q-values obey Bellman equation

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

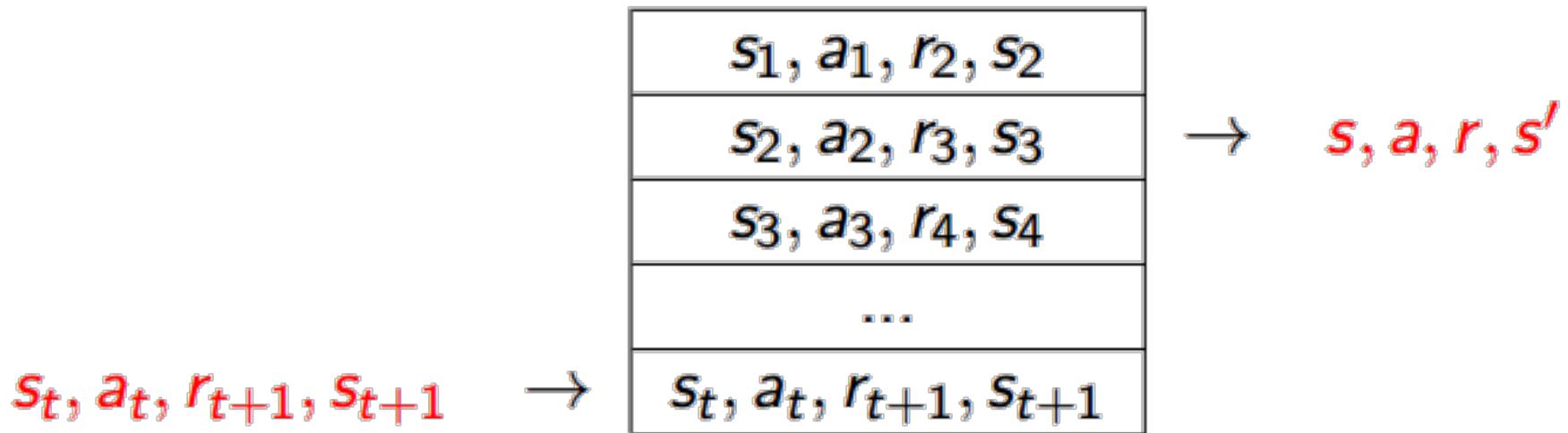
- Treat right-hand size as a target and minimize MSE loss by SGD

$$l = \frac{\left( r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a) \right)^2}{\text{Target}}$$

- Divergence issues using neural networks due to
  - Correlations between samples
  - Non-stationary targets

## Experience replay

- Build data set from agent's own experience
- Sample experiences uniformly from data set to remove correlations



# Algorithm

## Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$        **$\epsilon$ -greedy**  
        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        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  $\mathcal{D}$

    Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$

    Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

    Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to

**end for**

**end for**

# Improvements: Target Network

- To deal with non-stationarity, target parameters  $\hat{w}$  are held fixed

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

$$l = \mathbb{E}_{(s, a, r, s') \sim U(D)} \left\{ \left( r + \gamma \max_{a'} Q_{\hat{w}}(s', a') - Q_w(s, a) \right)^2 \right\}$$

# Improvements: Double DQN

- Q-learning is known to **overestimate** state-action values
  - The max operator uses the same values to select and evaluate an action

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

- The upward bias can be removed by decoupling the **selection** from the **evaluation**
  - Current Q-network is used to **select** actions
  - Older Q-network is used to **evaluate** actions

$$l = \mathbb{E}_{(s, a, r, s') \sim U(D)} \left\{ \left( r + \gamma Q_{\hat{W}_i}(s', \arg \max_{a'} Q_{W_i}(s', a')) - Q_{W_i}(s, a) \right)^2 \right\}$$

From the talk *Introduction to Deep Reinforcement Learning From Theory to Applications* by Siyi Li (HKUST)

# Improvements: Prioritized Replay

- Uniform experience replay samples transitions regardless of their significance
- Can weight experiences according to their significance
- Prioritized replay stores experiences in a priority queue according to the **TD error**
- Use **stochastic sampling** to increase sample diversity

$$|r + \gamma \max_{a'} Q_{\hat{w}}(s', a') - Q_w(s, a)|$$

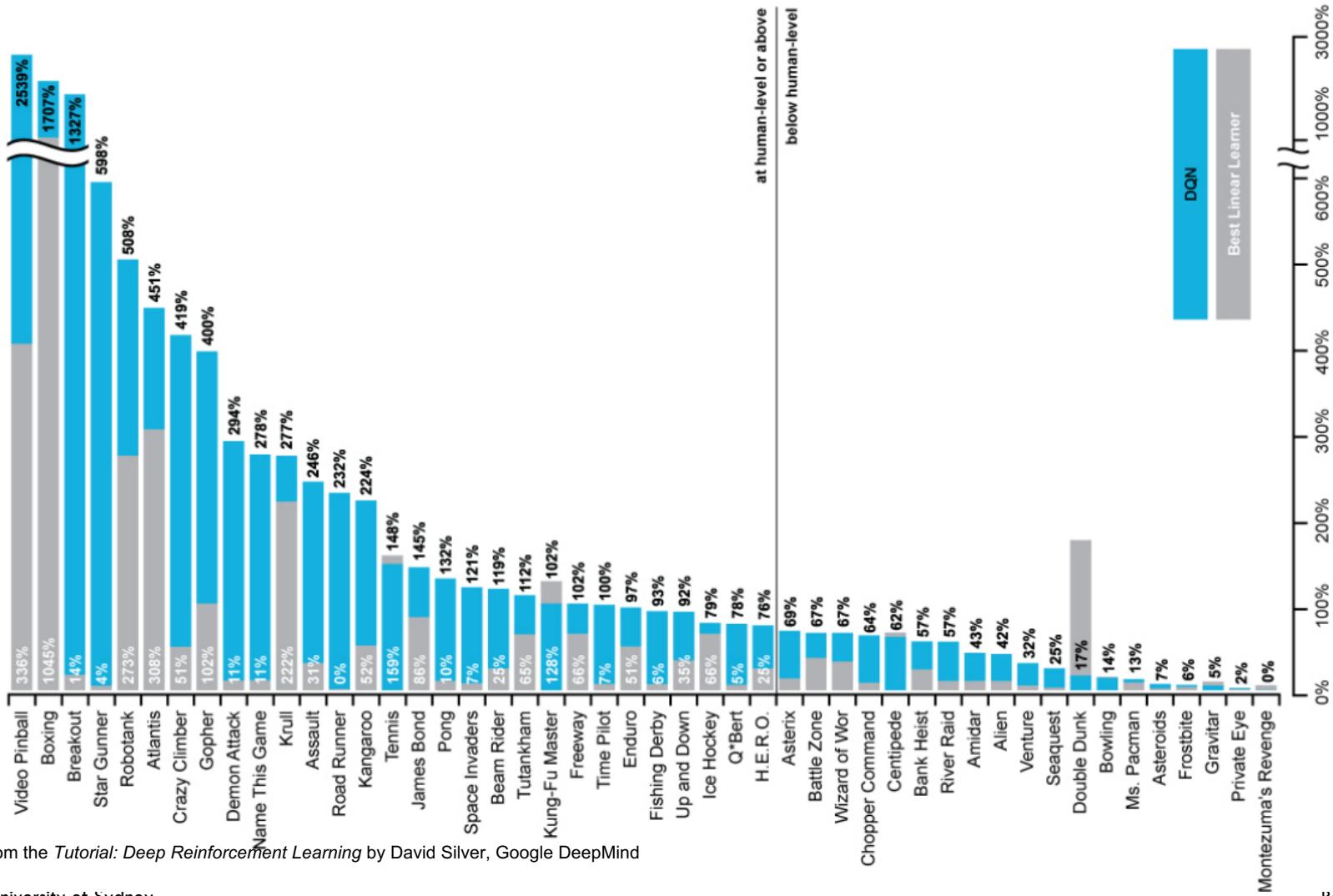
$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

$$p_i = |\delta_i| + \epsilon$$

From the talk *Introduction to Deep Reinforcement Learning From Theory to Applications* by Siyi Li (HKUST)

# **Deep RL Applications**

# DQN Results in Atari



From the *Tutorial: Deep Reinforcement Learning* by David Silver, Google DeepMind

# Demos

Breakout



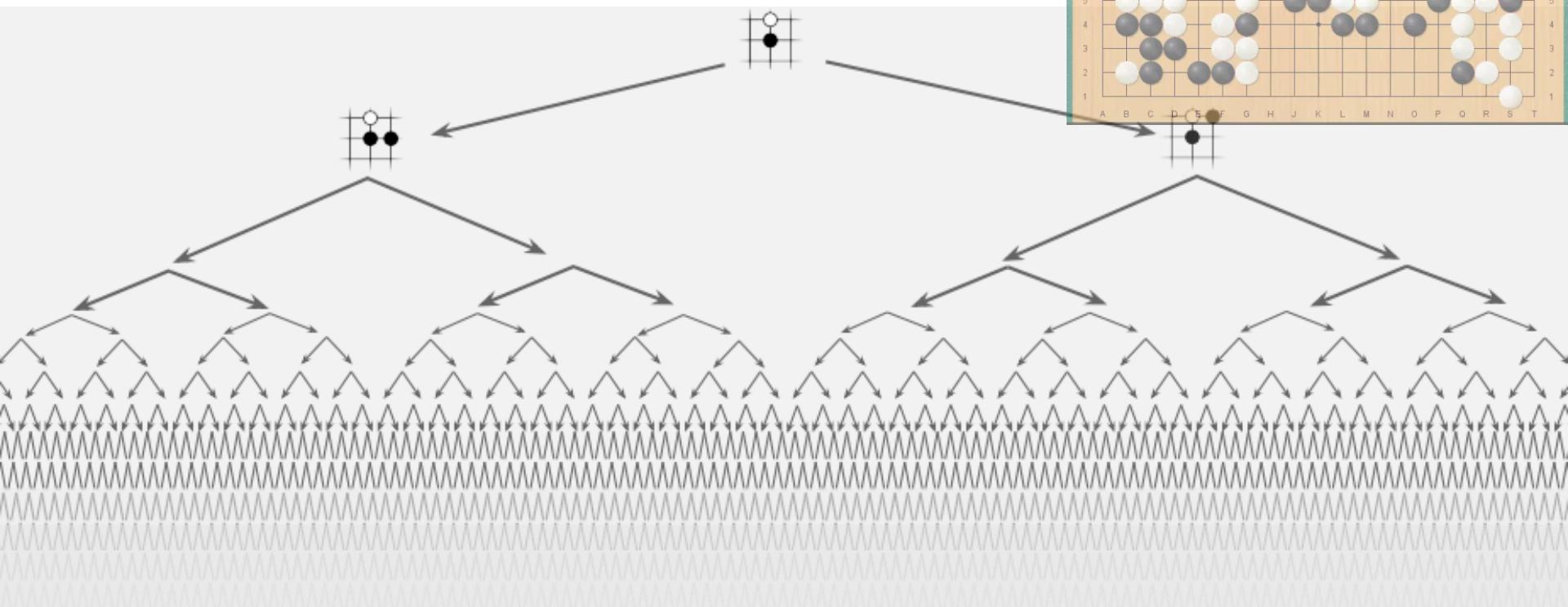
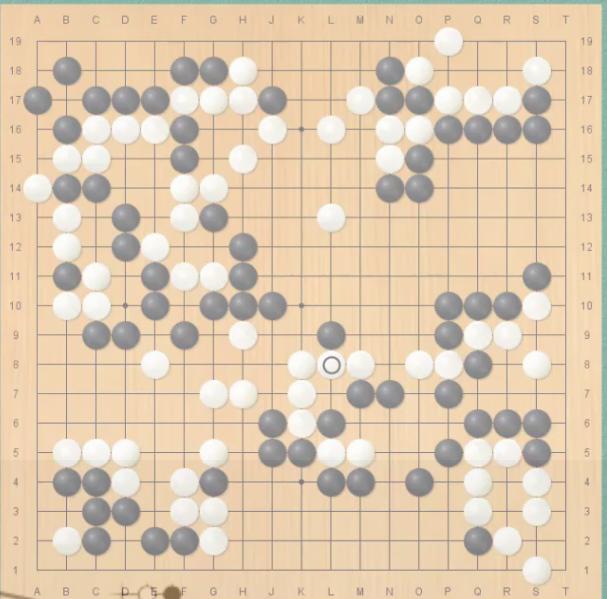
Super Mario Games



# Deep RL in Alpha Go

- To win a game, exhaustive search?

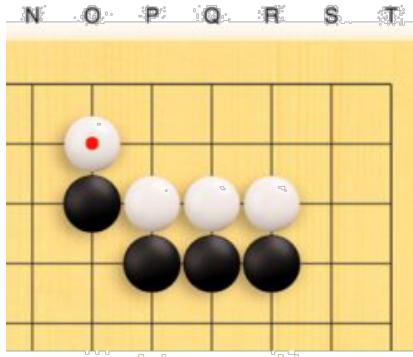
Monte Carlo search tree



From <https://icml.cc/2016/tutorials/AlphaGo-tutorial-slides.pdf>

# AlphaGo

## Feature Planes:



White

0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

#liberty = 1

Black

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0

- Black/white/empty
- Liberties
- Liberties after move
- Legality
- Turns since
- Capture size
- Ladder move
- KGS rank

Empty

1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1
1	0	0	0	0	0	1	1
1	1	0	0	0	0	1	1
1	1	1	1	1	1	1	1

#liberty = 2

#liberty = 3

#liberty  $\geq 4$

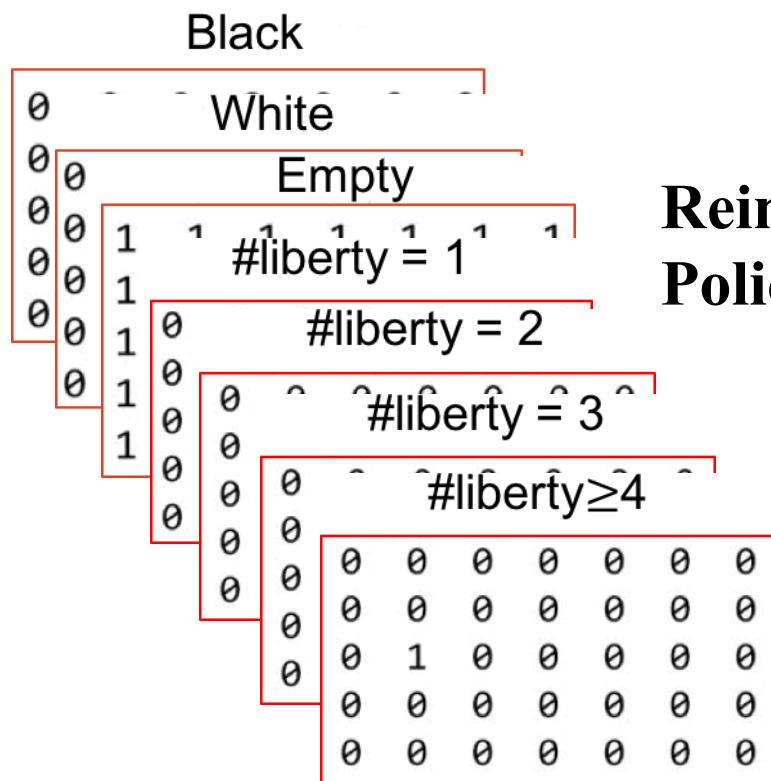
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

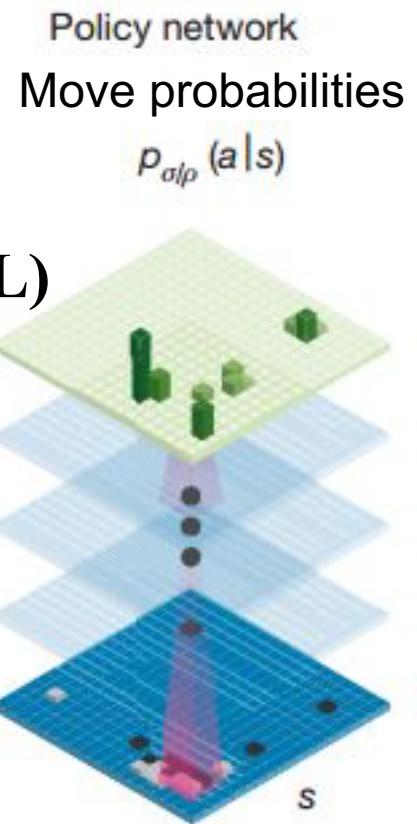
# Policy Network



## Supervised Learning (SL) Policy Network



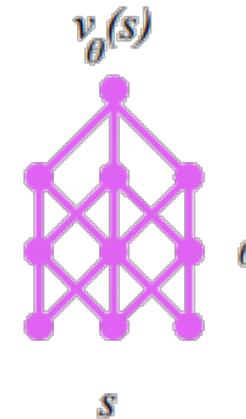
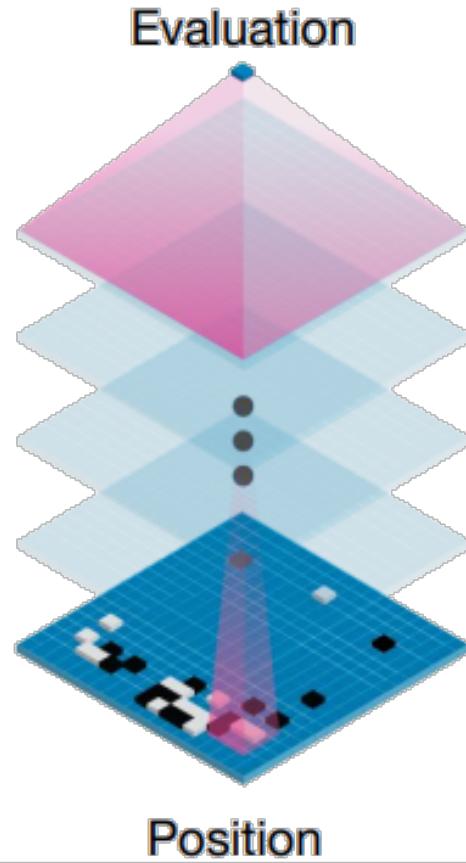
## Reinforcement Learning (RL) Policy Network



Convolutional Neural Network

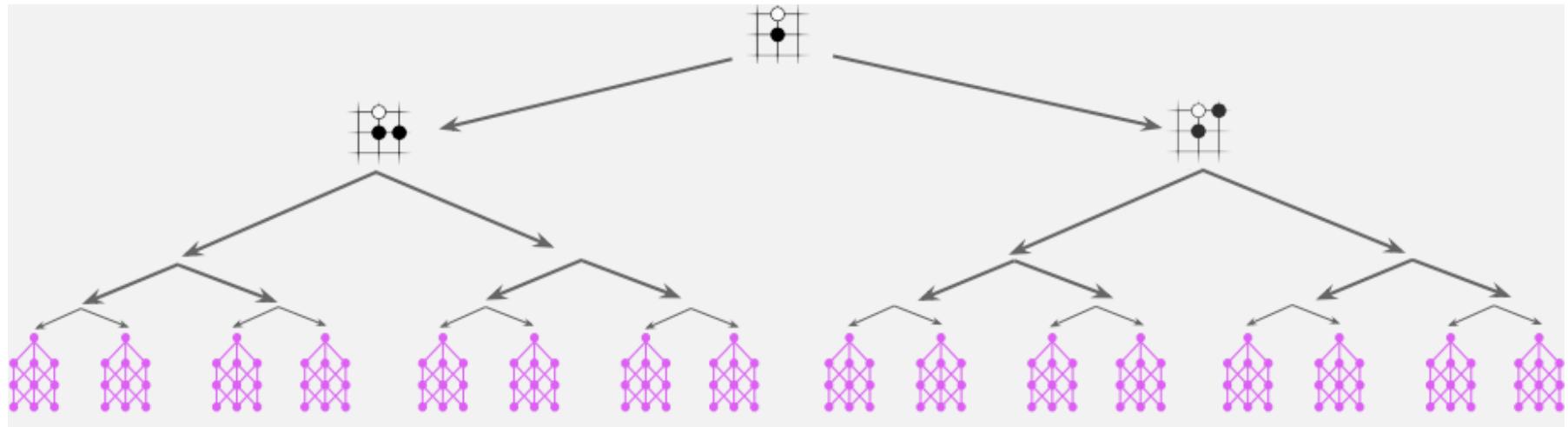
# Value Network

- To estimating probability that current move leads to win



# Deep RL in Alpha Go

- Reducing depth with value network



# Deep RL in Alpha Go

- Reducing breadth with policy network

