Reinforcement Learning

Week 12

Nguyen Hoang Tran

Based on slides prepared by Chang Xu and Caren Han

Reference can be found in the end of this lecture slide

THE UNIVERSITY OF SYDNEY

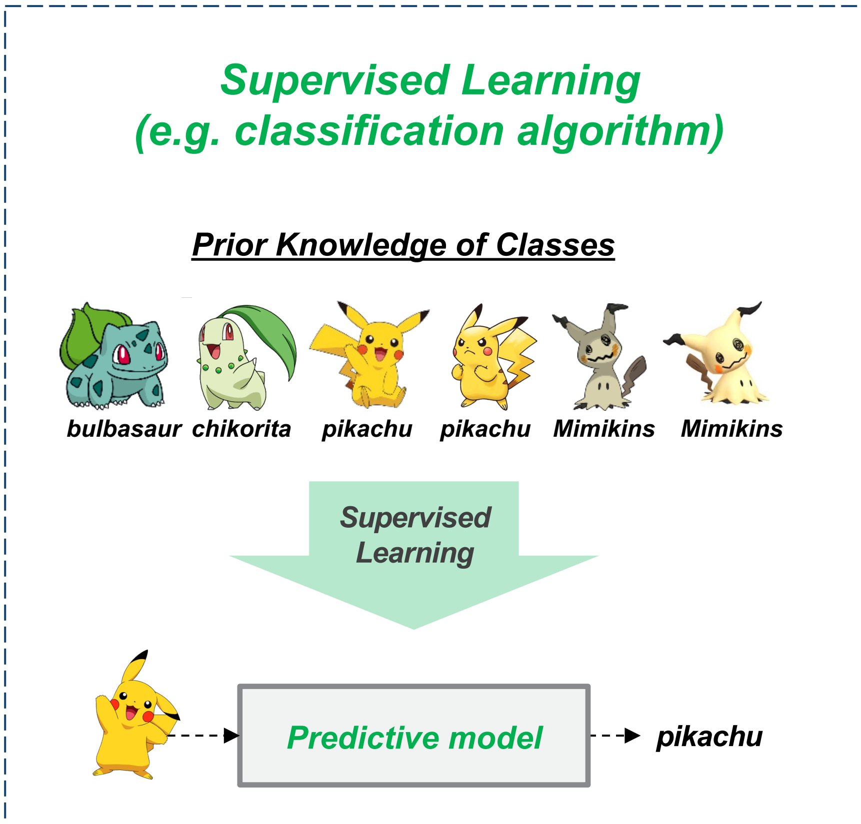
Today's Lecture

Lecture 12: Reinforcement Learning

1. Markov Decision Processes (MDP)
2. Q-Learning
3. Deep Q Learning
4. Applications

Supervised Learning

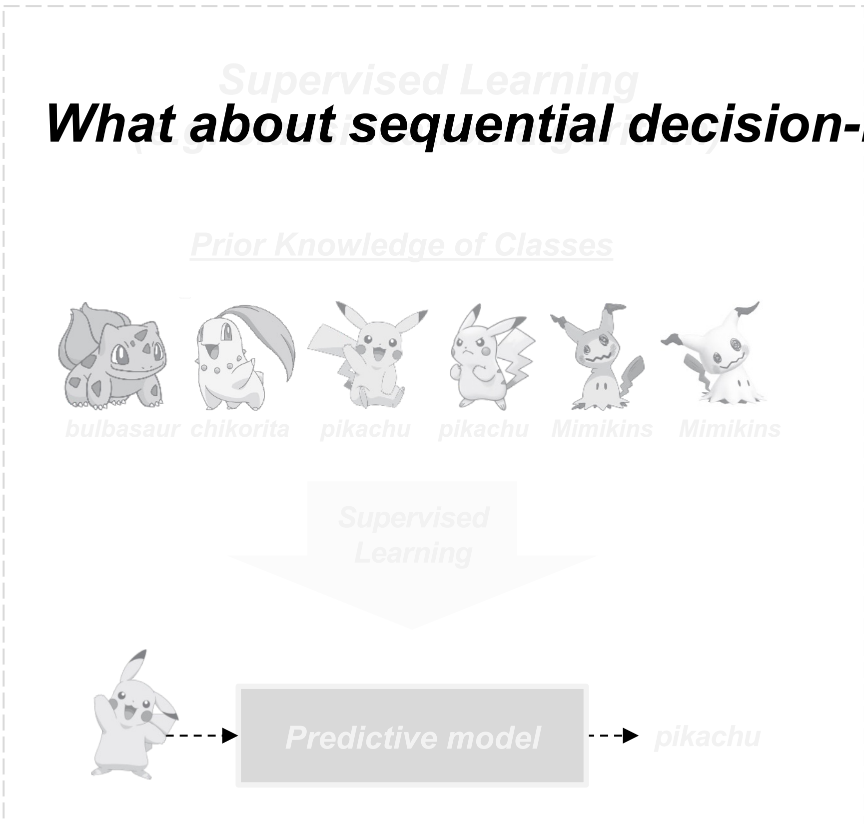
Supervised Learning algorithms tried to make their outputs mimic the labels y given in the training set.



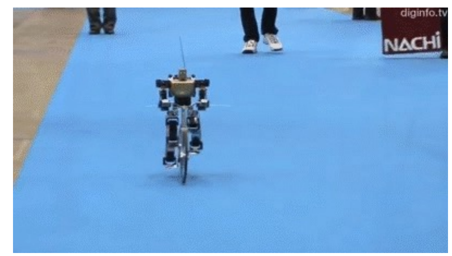
The labels gave a clear "right answer" for each of the inputs x

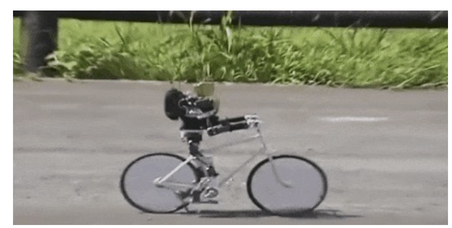
Supervised Learning

Supervised Learning algorithms tried to make their outputs mimic the labels y given in the training set.



What about sequential decision-making problems?



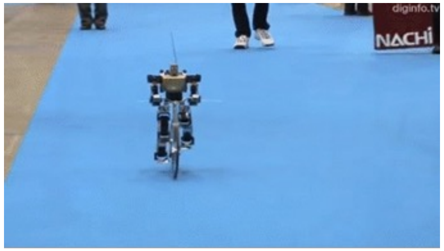


Supervised Learning

Supervised Learning algorithms tried to make their outputs mimic the labels y given in the training set.

What about sequential decision-making problems?

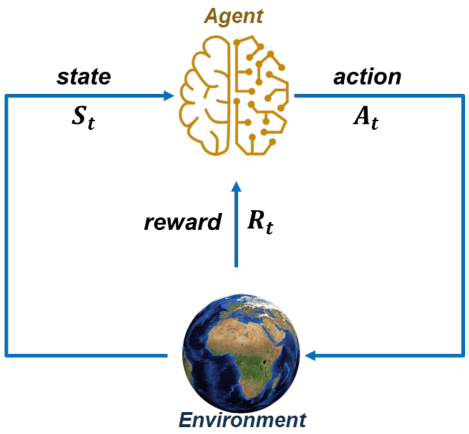
- *Difficult to define what the **correct** actions are to make it ride a bike*
- *Difficult to give explicit supervision for a learning algorithm to try to mimic. (like supervised learning)*



➔ **“Reinforcement Learning”**

Today's Lecture

Reinforcement Learning



THE UNIVERSITY OF SYDNEY

Reinforcement Learning

- ***We Learn from past experiences.***
  - *When an infant plays, waves its arms, or looks about, it has no explicit teacher*
  - *But it does have direct interaction to its environment*
  - *Years of positive compliments as well as negative experience (e.g. criticism) have all helped shape who we are today.*
- ***Reinforcement learning is a computational approach to learning from interaction.***

What is a reinforcement learning?

Learn to make good sequence of decisions

Provide our algorithms only a ***reward function***

indicates to the learning agent

- when it is **doing well**
- when it is **doing poorly**

What is a reinforcement learning?

Learn to make good sequence of decisions

Provide our algorithms only a ***reward function***

- *A reward $\boldsymbol{R_t}$ is a scalar feedback signal*
- *Indicates how well agent is doing at step **t***
- *The agent's job is to maximise cumulative reward*

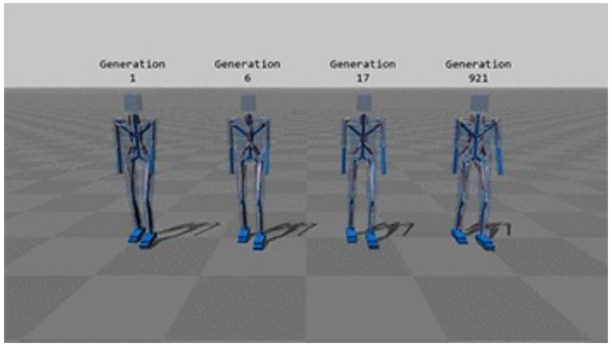
Reward hypothesis

"All goals can be described by the maximization of expected cumulative reward"

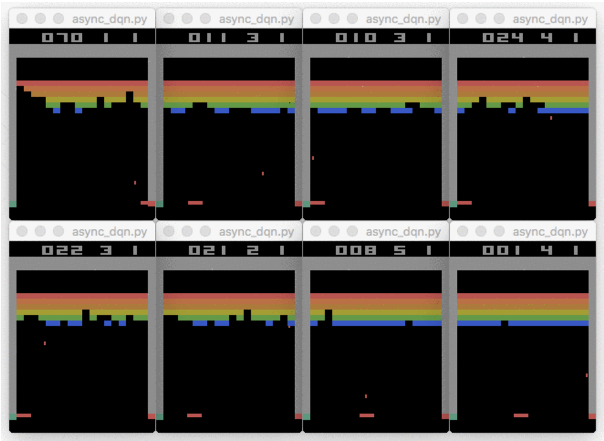
Reinforcement Learning

Example of Rewards

- *Make a humanoid robot walk/robot bicycle riding*
  - ***(positive reward) for forward motion***
  - ***(negative reward) for falling over***

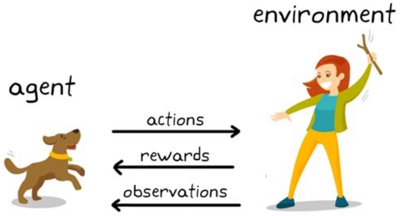


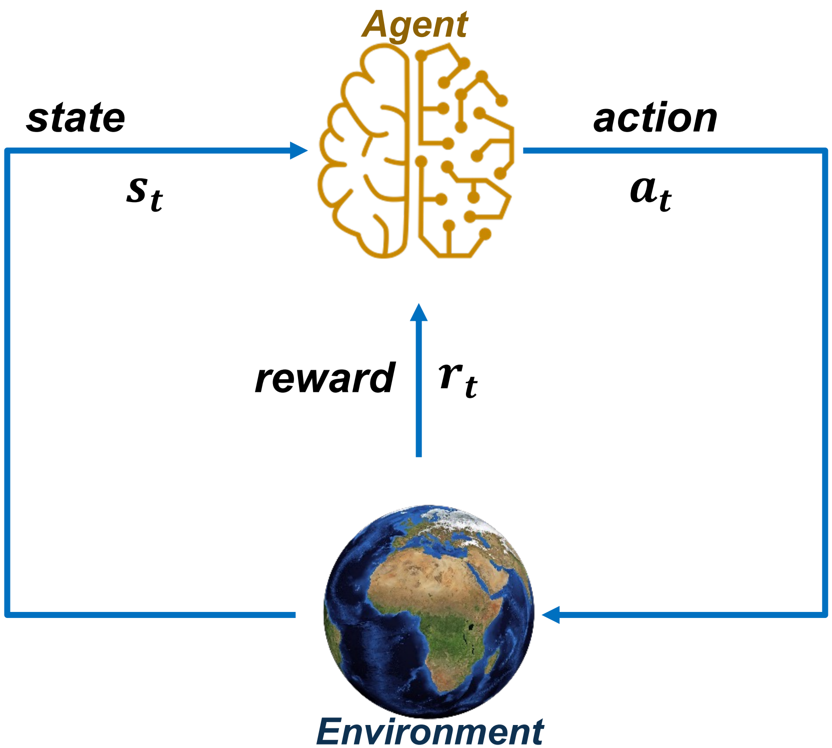
- *Play many different Atari games better than human*
  - ***(positive reward) for increasing score***
  - ***(negative reward) for decreasing score***



Reinforcement Learning

What is a reinforcement learning?





At each step t the agent:
- *Executes action* $\boldsymbol{a_t}$
- *Receives state* $\boldsymbol{s_t}$
- *Receives scalar reward* $\boldsymbol{r_t}$

The environment:
- *Receives action* $\boldsymbol{a_t}$
- *Emits state* $\boldsymbol{s_{t+1}}$
- *Emits scalar reward* $\boldsymbol{r_{t+1}}$

t increments at env. step

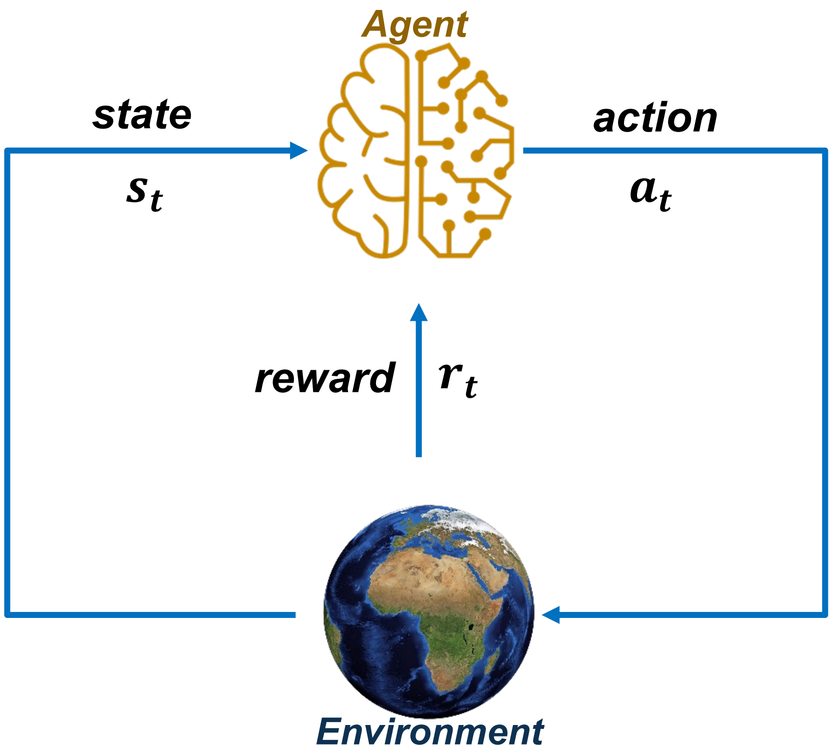
Reinforcement Learning

› ***The history H_t is the sequence of observations, actions, rewards***

$H_t = s_1, r_1, a_1, \ldots, a_{t-1}, s_t, r_t$ ** What happens next depends on the history*

› ***State S_t is the information used to determine what happens next***

$S_t = \mathrm{f}(H_t)$

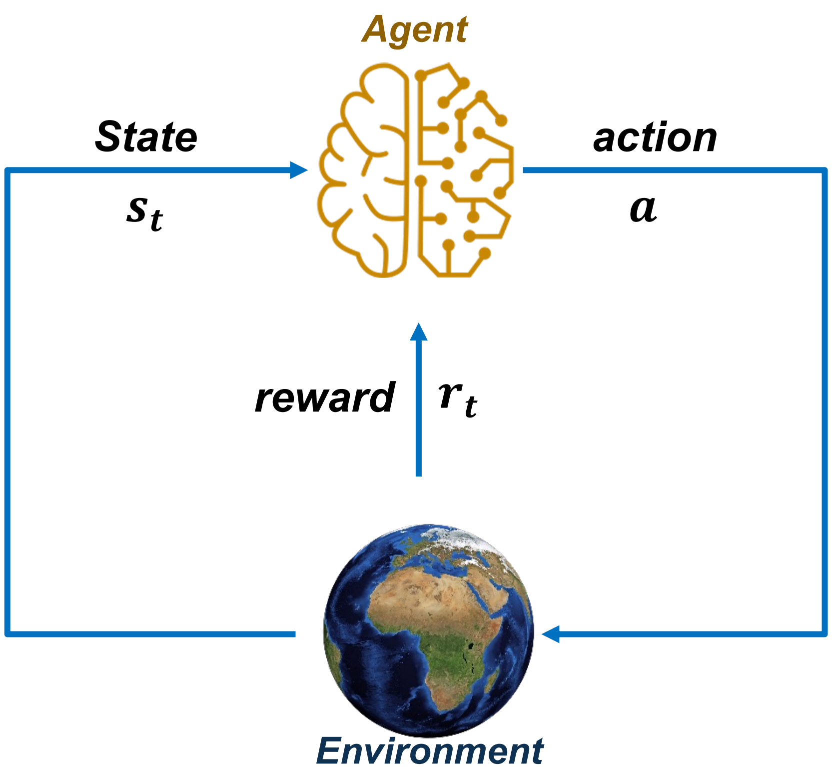


At each step t the agent:
- *Executes action a_t*
- *Receives state s_t*
- *Receives scalar reward r_t*

The environment:
- *Receives action a_t*
- *Emits state s_{t+1}*
- *Emits scalar reward r_{t+1}*

t increments at env. step

Markov Decision Process



$(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: **transition probability**
i.e. distribution over next state given (state, action) pair

γ : discount factor

History

$$H_t = s_1, r_1, a_1, \ldots, a_{t-1}, s_t, r_t$$

Markov Decision Process

Major Components of an RL Agent

An RL agent may include one or more of these components:

- ***Policy:*** *agent's behaviour function*
- ***Value Function:*** *how good is each state and/or action*
- ***Model:*** *agent's representation of the environment*

Markov Decision Process

Let's try to work on a simple Markov Decision Process (MDP) example

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

actions = {

right →

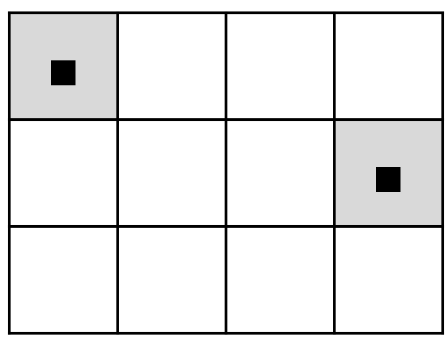
left ←

up ↑

down ↓

}

states



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

THE UNIVERSITY OF SYDNEY

Markov Decision Process

Let's try to work on a simple Markov Decision Process (MDP) example

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

actions = {

right

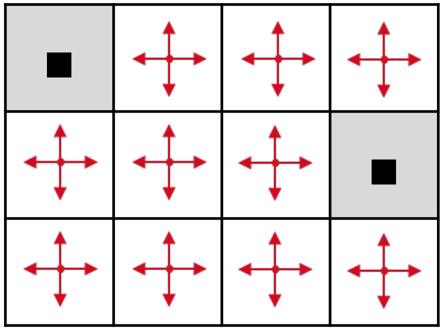
left

up

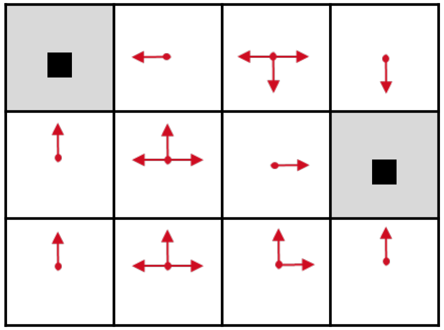
down

}

Random Policy



Optimal Policy



****Policy:** agent's behaviour function*

Markov Decision Process

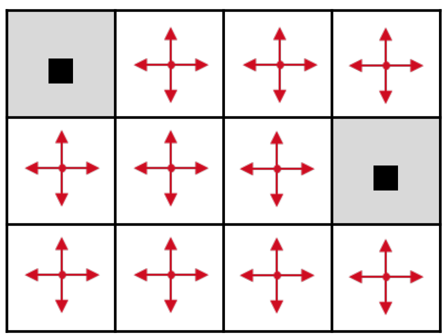
Let’s try to work on a simple Markov Decision Process (MDP) example

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

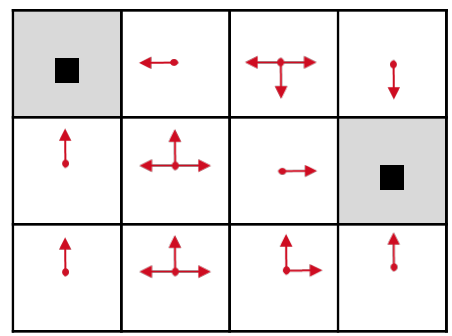
actions = {
- right →
- left ←
- up ↑
- down ↓

}

Random Policy



Optimal Policy



***Policy:** agent’s behaviour function*

Markov Decision Process

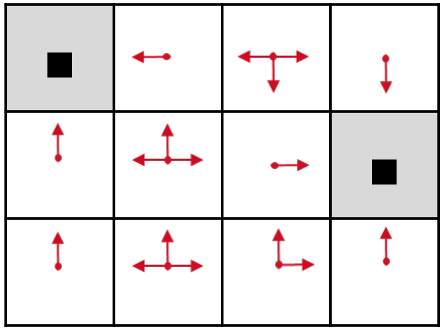
Definition: Policy and the optimal policy π*

› *We want to find optimal policy π* that maximizes the sum of rewards.*

› *How do we handle the randomness (initial state, transition probability…)?*

- ***Maximize the expected sum of rewards!***

Optimal Policy



Formally: $\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right]$ with $s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$

Markov Decision Process

Definitions: Value function and Q-value function

› *Following a policy produces sample trajectories (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$*

› ***State-Value Function*** *(How good is a state?)*

- *The value function at state s, is the expected cumulative reward from following the policy from state s:*

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

› ***Action-Value Function*** *(How good is a state-action pair?)*

- *The Q-value function at state s and action a, is the expected cumulative reward from taking action a in state s and then following the policy:*

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

Markov Decision Process

Bellman equation

- *The optimal Q-value function Q* is the maximum expected cumulative reward achievable from a given (state, action) pair:*

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

- *Q* satisfies the following* ***Bellman equation***, i.e., immediate reward plus discounted value of successor state

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

- ***Intuition: if the optimal state-action values for the next time-step Q*(s',a') are known, then the optimal strategy is to take the action that maximizes the expected value of***

$$r + \gamma Q^*(s', a')$$

- *The optimal policy* ***π*** ** corresponds to taking the best action in any state as specified by Q**

Initializing the Q(s, a) function

states

actions

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
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
Up	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Down	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

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

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

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

Reinforcement Learning

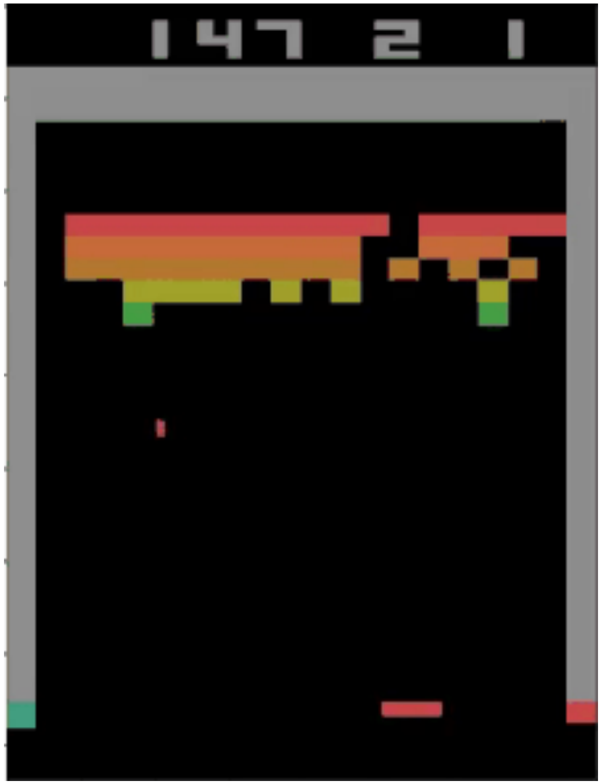
Reinforcement learning approaches

- Monte-Carlo Learning (MC)
- Temporal-difference learning (TD)
- **Q-Learning**
- Policy Gradient (PG)
- Actor-Critic (AC)
- Advantage Actor-Critic (A2C)
- Asynchronous advantage actor-critic (A3C)

Deep Q Learning

Lecture 12: Reinforcement Learning

Deep Q-Learning



160

210

states

actions

	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

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

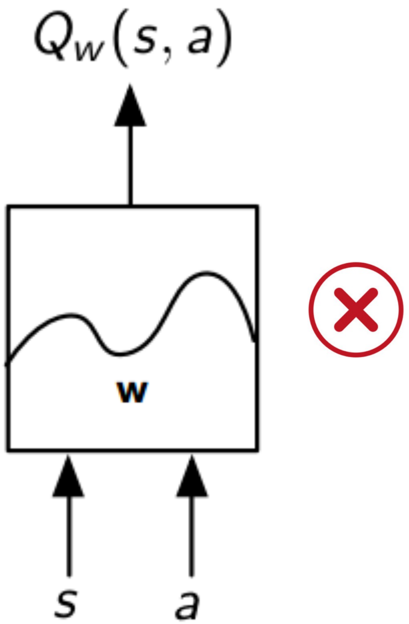
Value Function Approximation

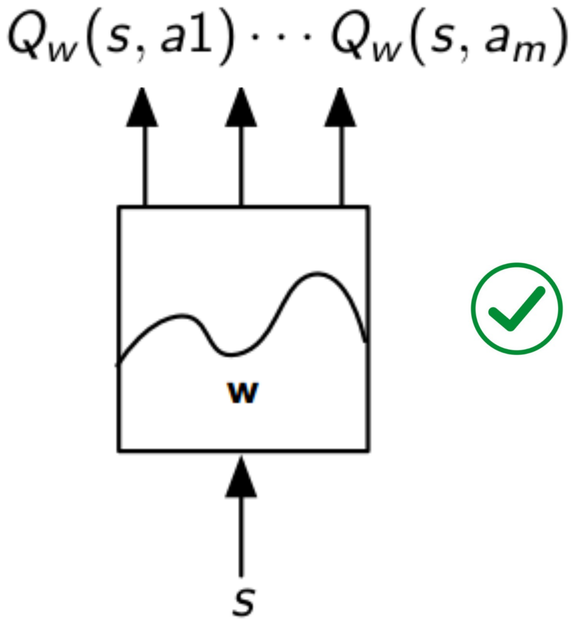
$$Q(s,a) = f(s,a,w)$$

Deep Q-Learning

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

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

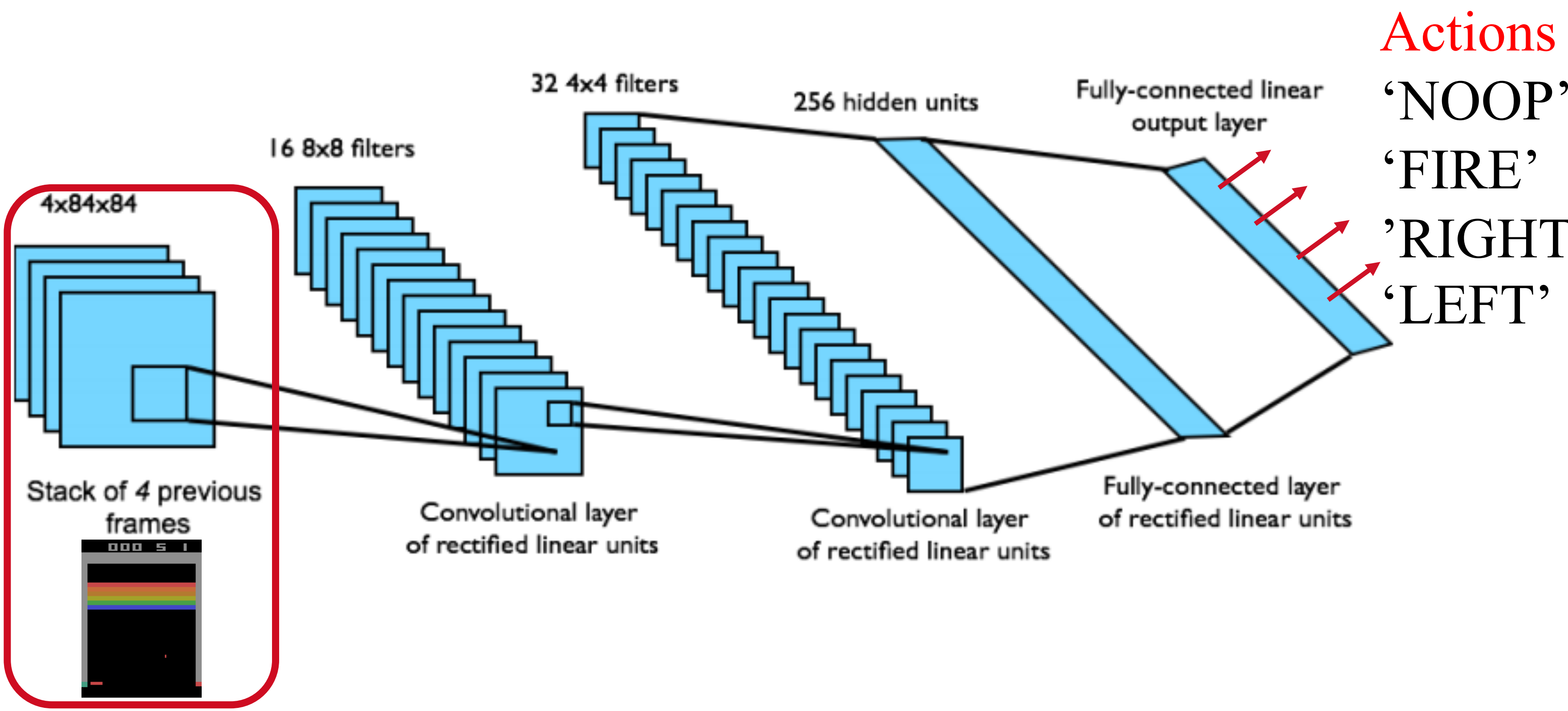




From the *Tutorial: Deep Reinforcement Learning* by David Silver, Google DeepMind

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-Learning

- Optimal Q-values obey Bellman equation

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

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

$$l = \Big(\underbrace{r + \gamma \max_{a'} Q_w(s',a')}_{\text{Target}} - Q_w(s,a) \Big)^2$$

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

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

Experience replay

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

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

$$\rightarrow \quad s, a, r, s'$$

$$s_t, a_t, r_{t+1}, s_{t+1} \quad \rightarrow$$

Deep Q Learning

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 ϵ select a random action a_t
    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

ϵ-greedy

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

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

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

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

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

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

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

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

Deep Q Learning

Lecture 12: Reinforcement Learning

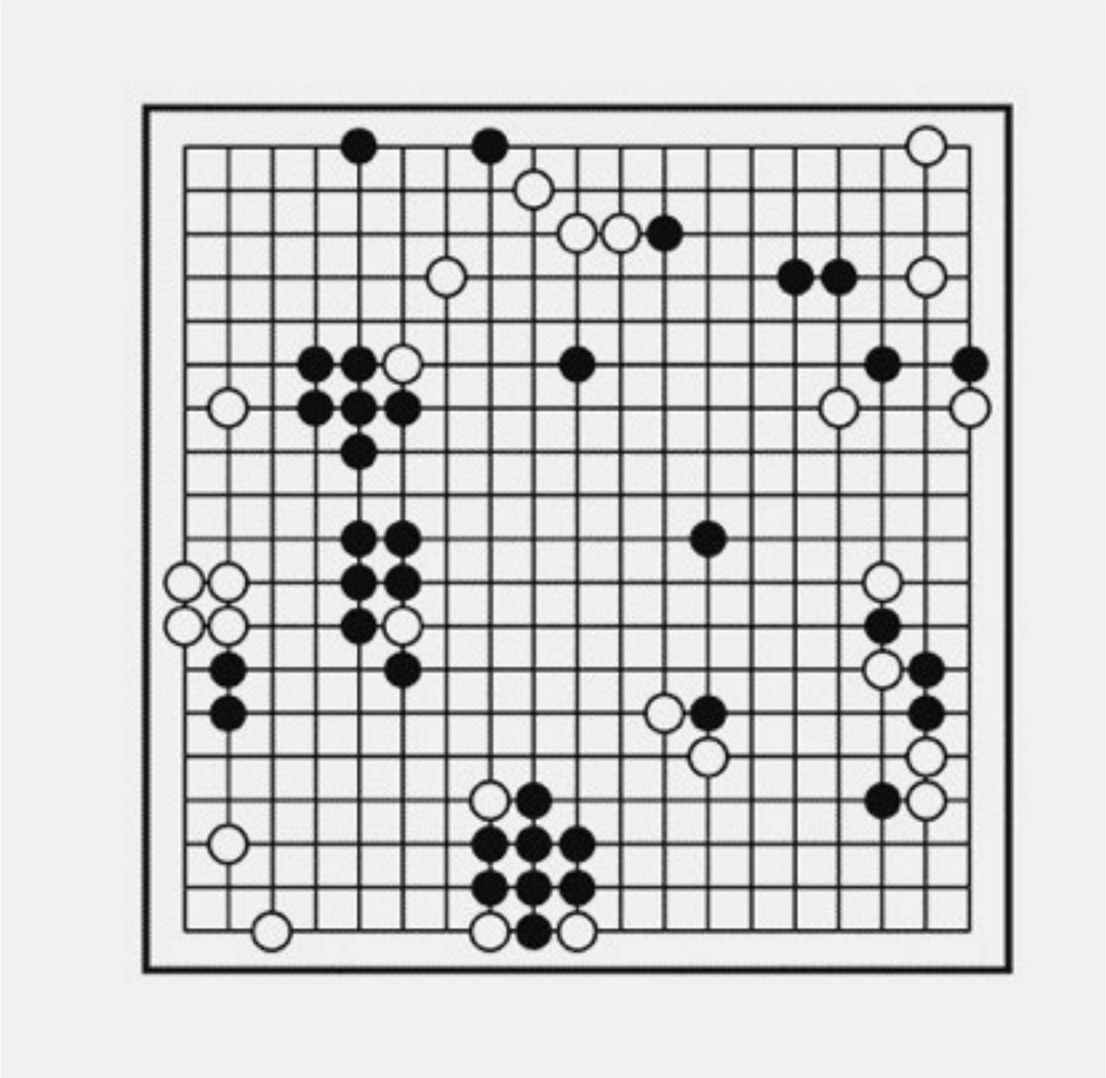
AlphaGo and AlphaZero

A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play learning

(Science, 2018)

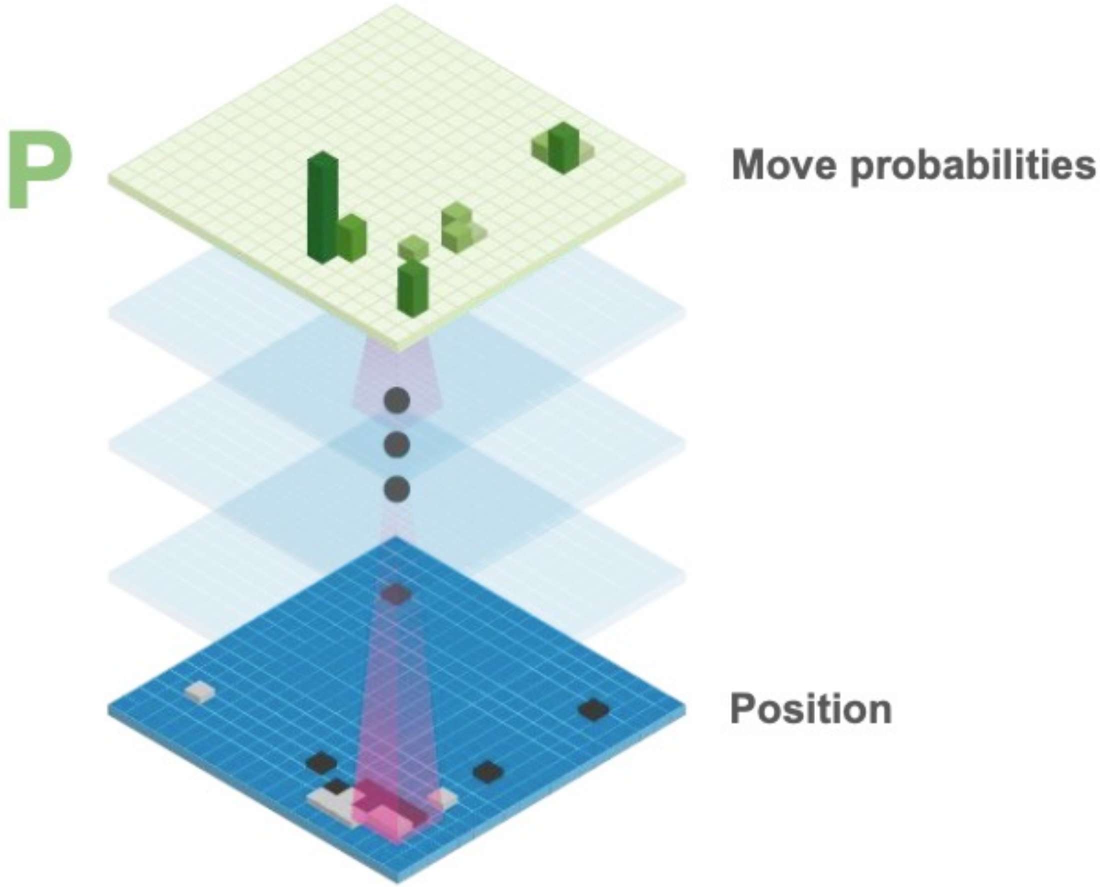
David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis

AlphaGo and AlphaZero



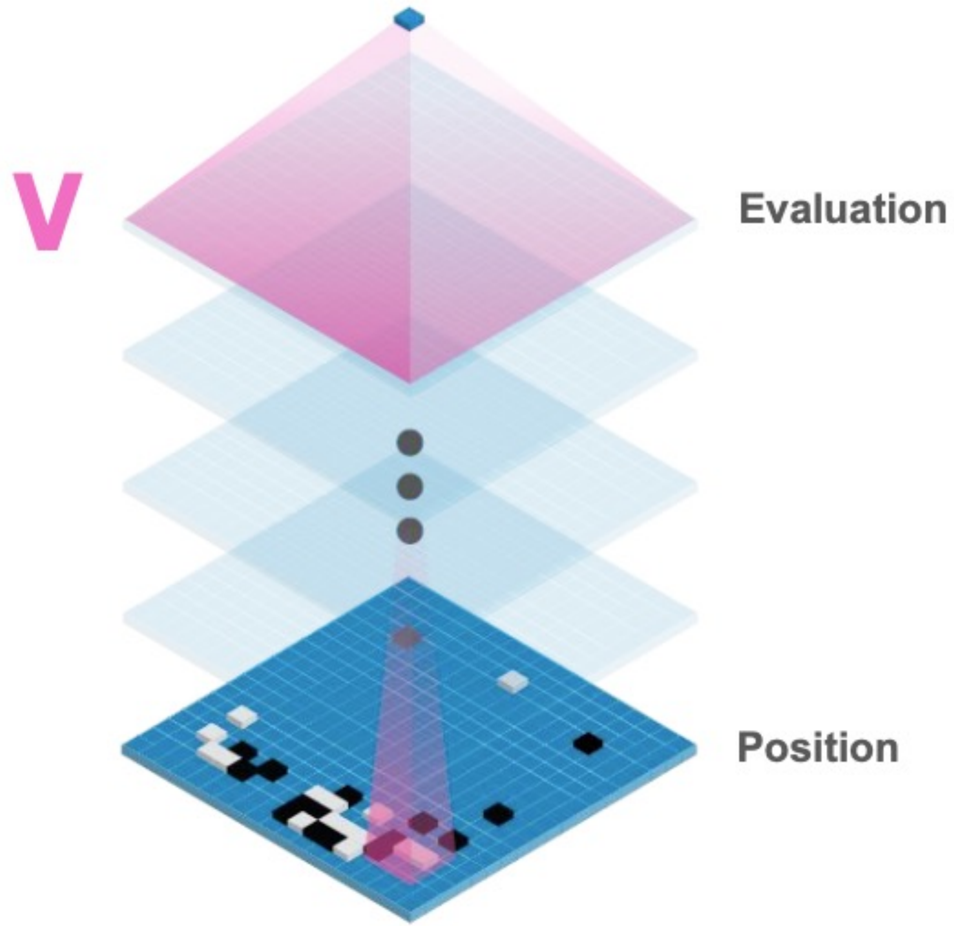
AlphaGo and AlphaZero

Deep Learning in AlphaGo : Policy Network



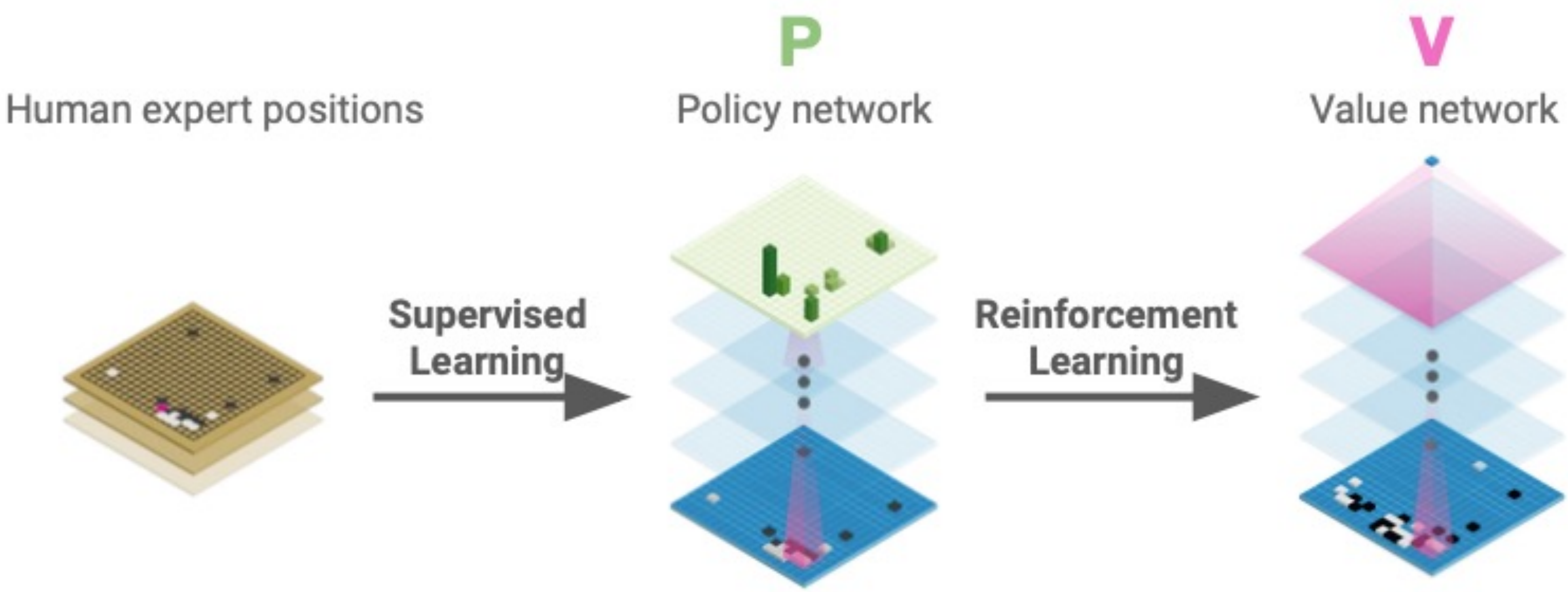
AlphaGo and AlphaZero

Deep Learning in AlphaGo: Value Network

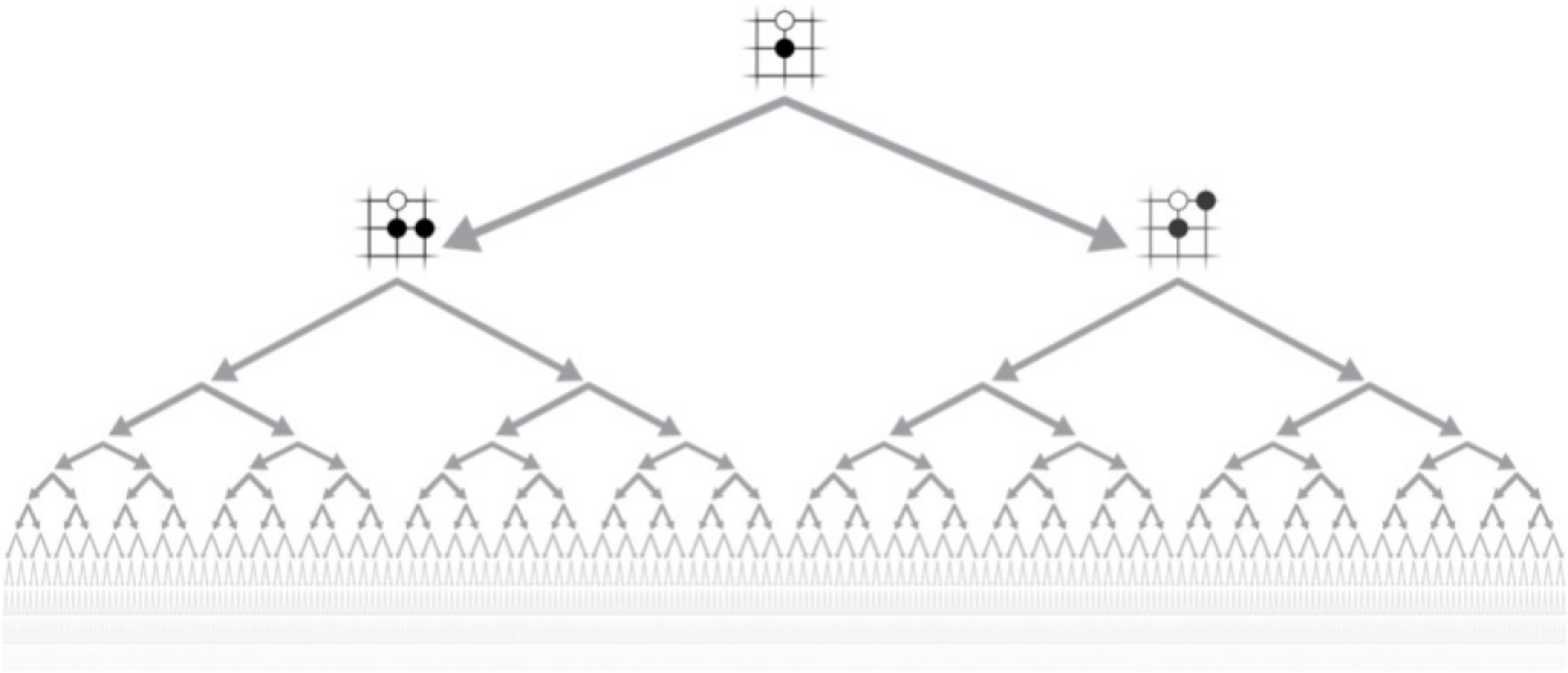


AlphaGo and AlphaZero

Training AlphaGo

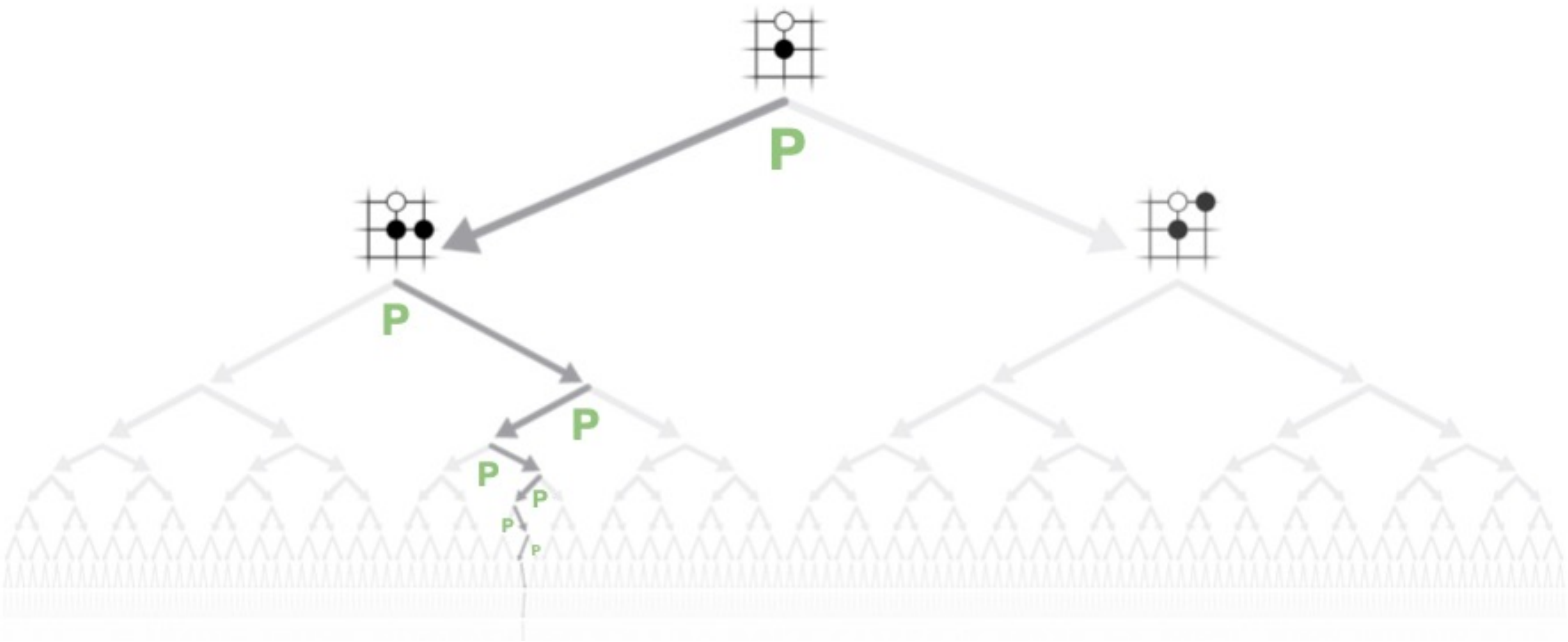


Exhaustive Search



AlphaGo and AlphaZero

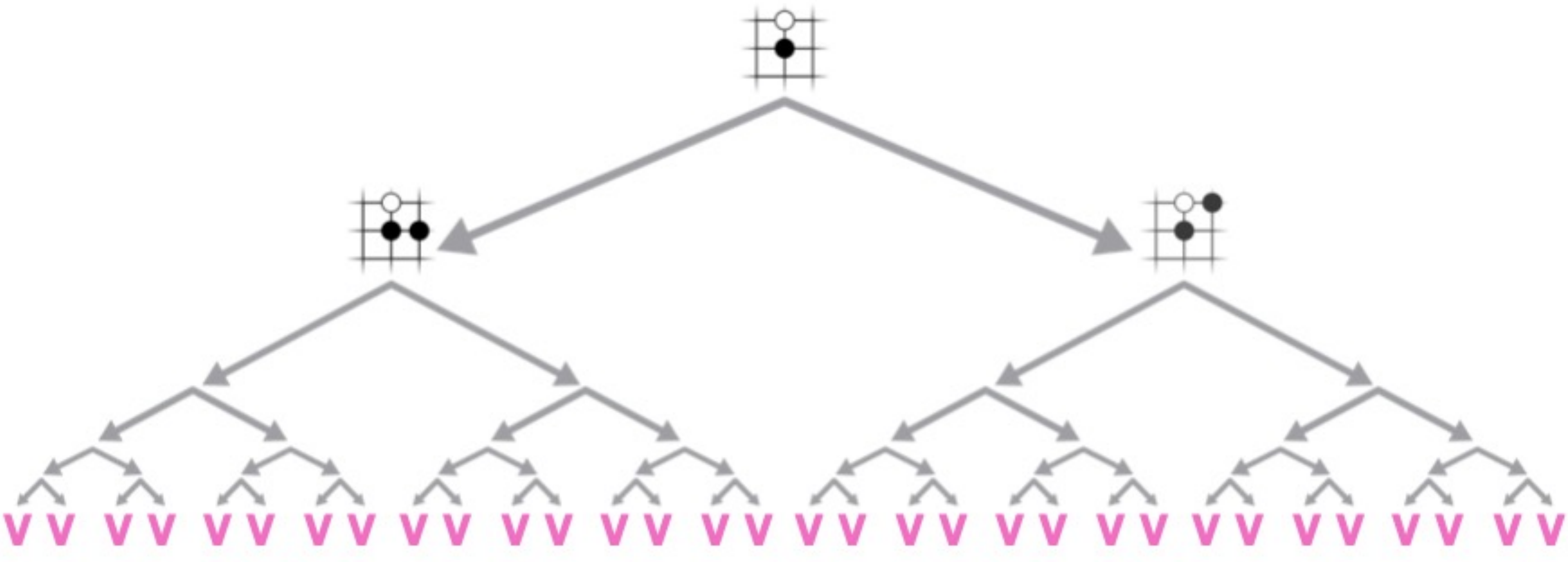
Reducing breadth with the policy network



THE UNIVERSITY OF SYDNEY

AlphaGo and AlphaZero

Reducing depth with the value network



aZero

AlphaGo won the match 4-1

THE UNIVERSITY OF SYDNEY

AlphaGo and AlphaZero

AlphaZero: One Algorithm, Three Games

Chess

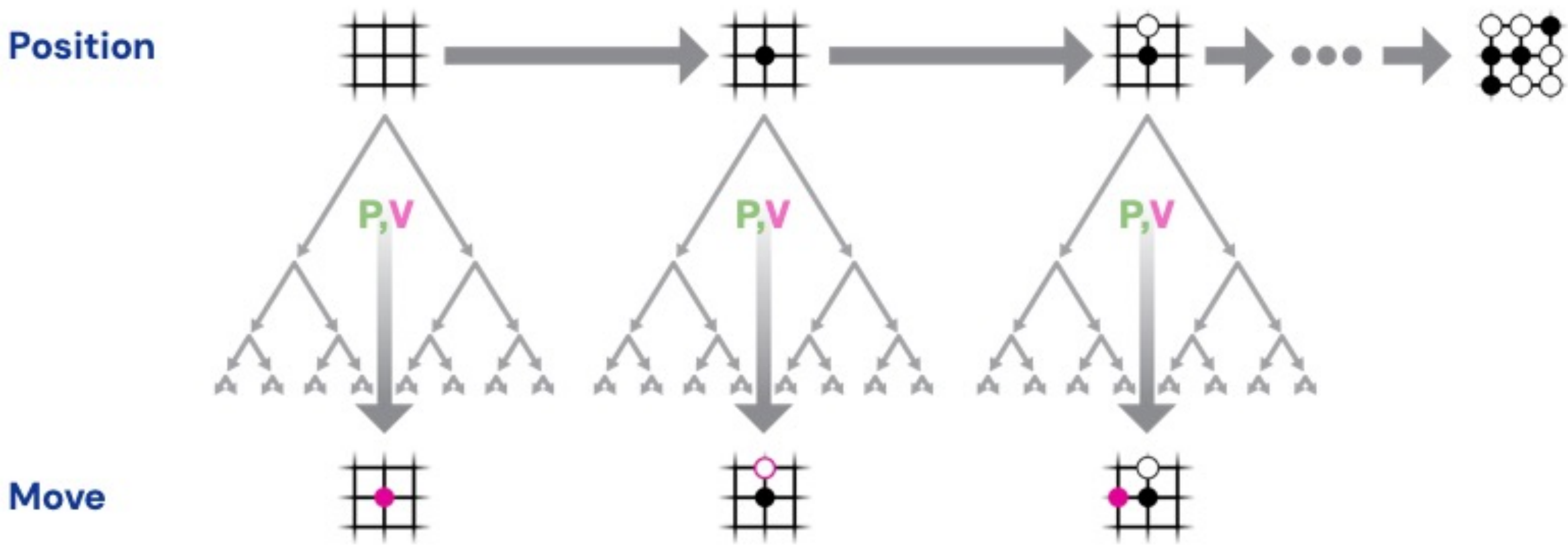
Shogi

Go

A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play, Science 2018, Joint work with: **David Silver**, **Thomas Hubert**, **Julian Schrittwieser**, Arthur Guez, Ioannis Antonoglou, Matthew Lai, Karen Simonyan, Marc Lanctot, Timothy Lillicrap, Laurent Siffre, Dharshan Kumaran, and Demis Hassabis

AlphaGo and AlphaZero

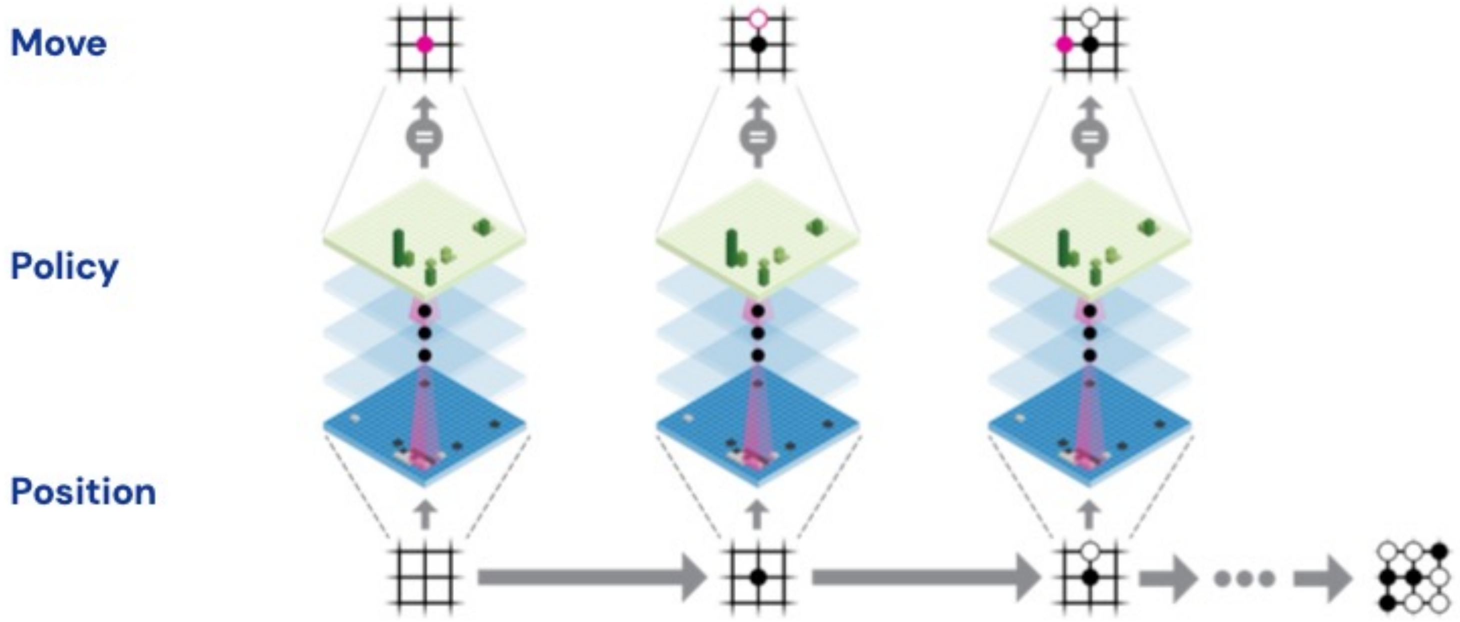
Reinforcement Learning in AlphaZero



AlphaZero plays games against itself

AlphaGo and AlphaZero

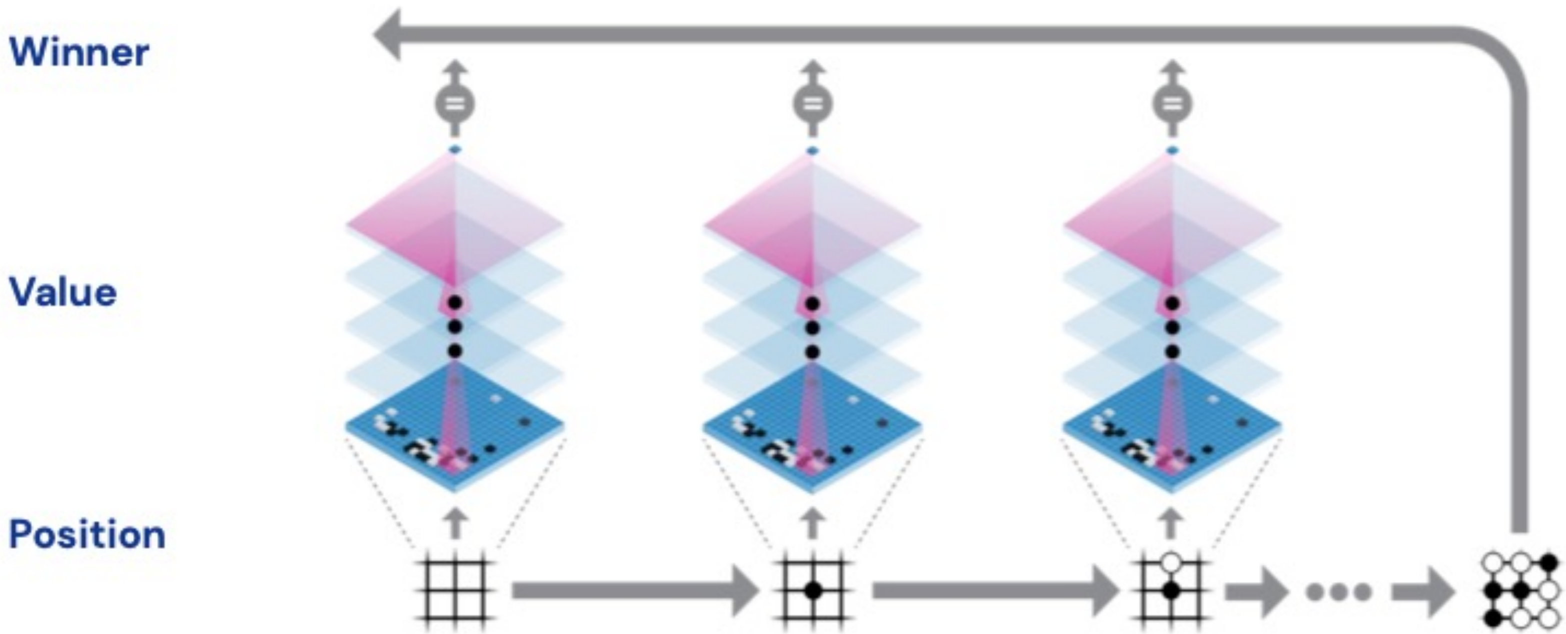
Reinforcement Learning in AlphaZero



New policy network P' is trained to predict AlphaGo's moves

AlphaGo and AlphaZero

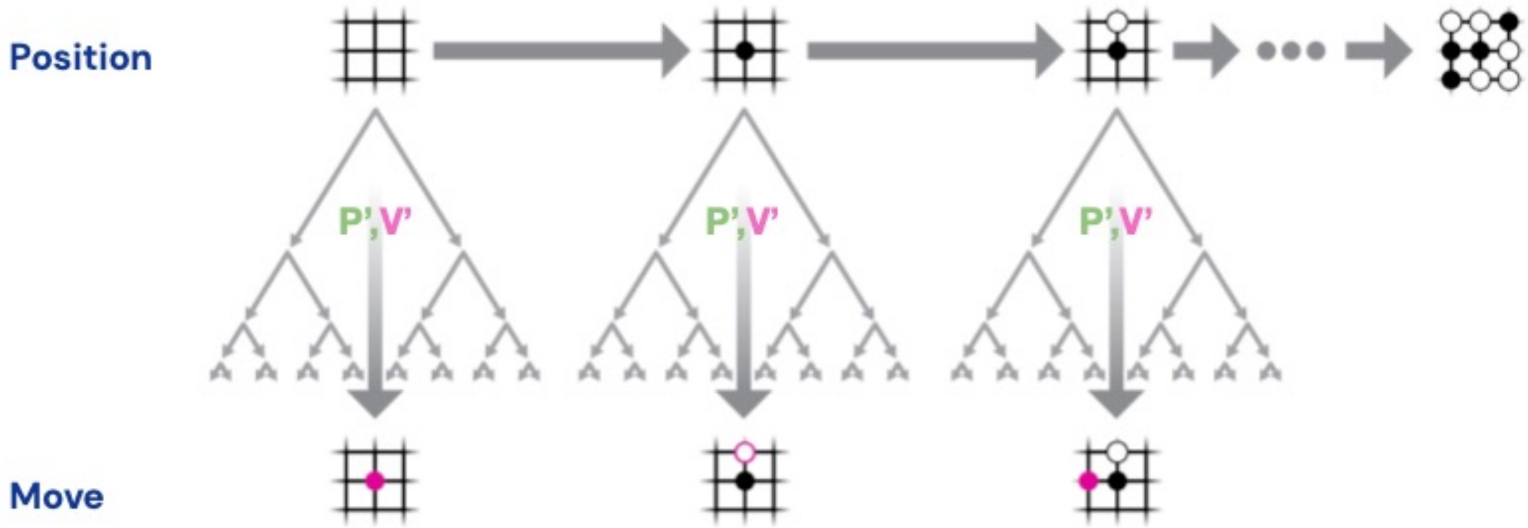
Reinforcement Learning in AlphaZero



New value network V’ is trained to predict winner

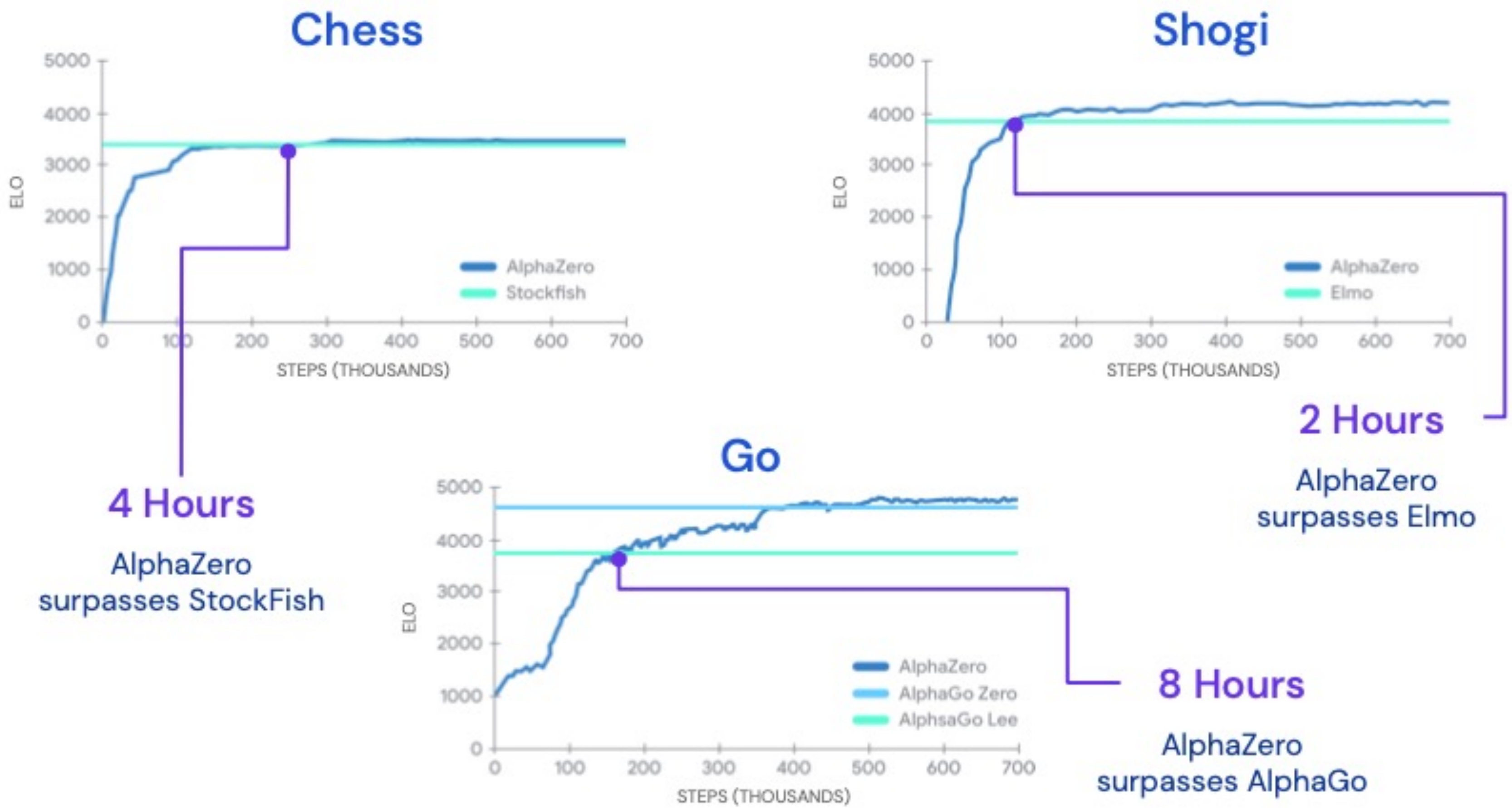
AlphaGo and AlphaZero

Reinforcement Learning in AlphaZero



New policy/value network is used in next iteration of AlphaGo Zero

AlphaGo and AlphaZero



AlphaGo and AlphaZero

Deep Learning enables us to search the huge search space of complex board games

Self-play produces large amounts of data necessary for training the deep neural networks

Self-play provides an automatic curriculum, starting from simple opponents to stronger and stronger opponents.
System discovers new knowledge

New directions: Learn rules of the game, more than two players, imperfect information, larger action spaces etc.

Reinforcement learning (RL)

RL is a general-purpose framework for decision-making

- RL is for an agent with the capacity to act
- Each action influences the agent's future state
- Success is measured by a scalar reward signal
- Goal: select actions to maximise future reward

Deep learning (DL)

DL is a general-purpose framework for representation learning

- Given an objective
- Learn representation that is required to achieve objective
- Directly from raw inputs
- Using minimal domain knowledge

Deep reinforcement learning: AI = RL + DL

We seek a single agent which can solve any human-level task

- RL defines the objective
- DL gives the mechanism
- RL + DL = general intelligence

Reference

- Ng, A. (2016). Reinforcement learning and control. CS229, Machine Learning, Lecture Notes. Stanford University, Stanford, Calif., nd http://cs229. stanford. edu/notes/cs229-notes12. pdf. Accessed July.
- Mitchell, T. M. (1997). Machine learning: Chapter 13 Reinforcement Learning
- Silver. D. (2015). UCL Course on Reinforcement Learning: Lecture 1: Introduction to Reinforcement Learning.
- Silver. D. (2015). UCL Course on Reinforcement Learning: Lecture 2: Markov Decision Process.