University of Nottingham
UK | CHINA | MALAYSIA

COMP3055 Machine Learning

Topic 17 – Deep Reinforcement Learning Intro

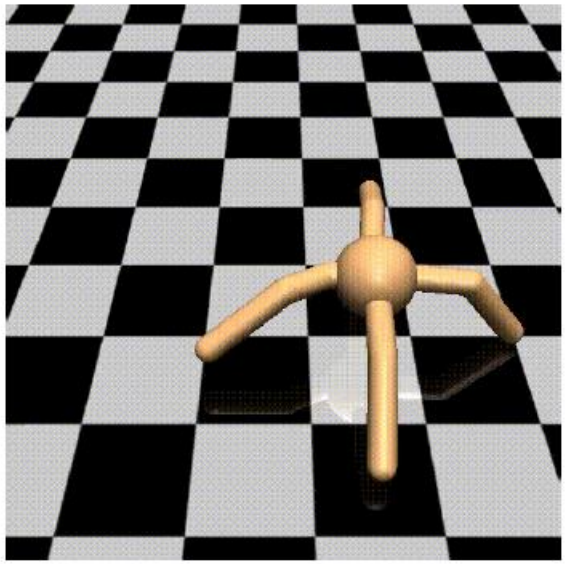
Zheng Lu

2024 Autumn

Deep Reinforcement Learning

Deep Learning

Deep RL

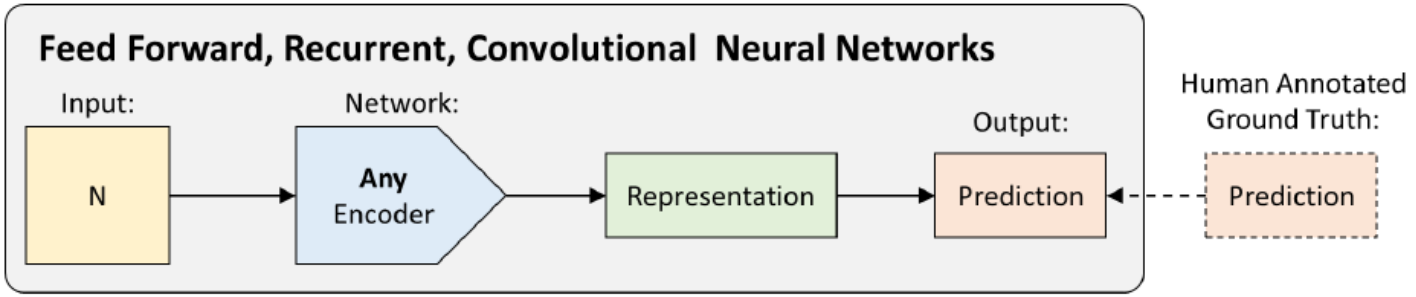


- **What is it?** Framework for learning to solve sequential decision making problems.
- **How?** Trial and error in a world that provides occasional rewards

Deep Reinforcement Learning

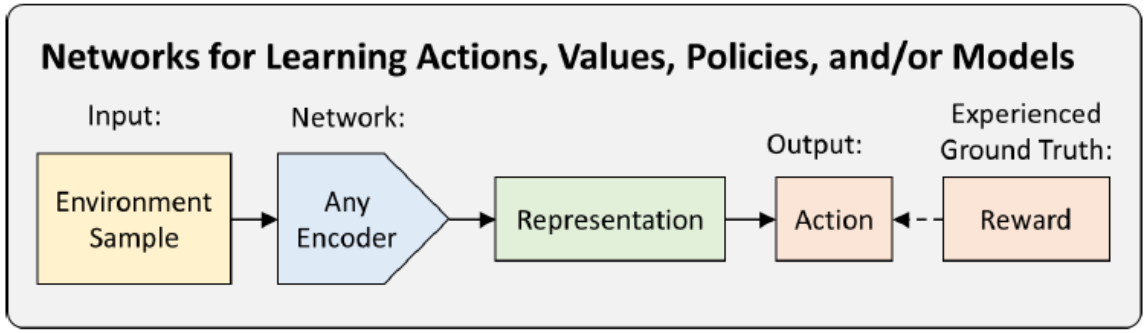
Deep Reinforcement Learning: AI = RL + DL

Deep Reinforcement Learning

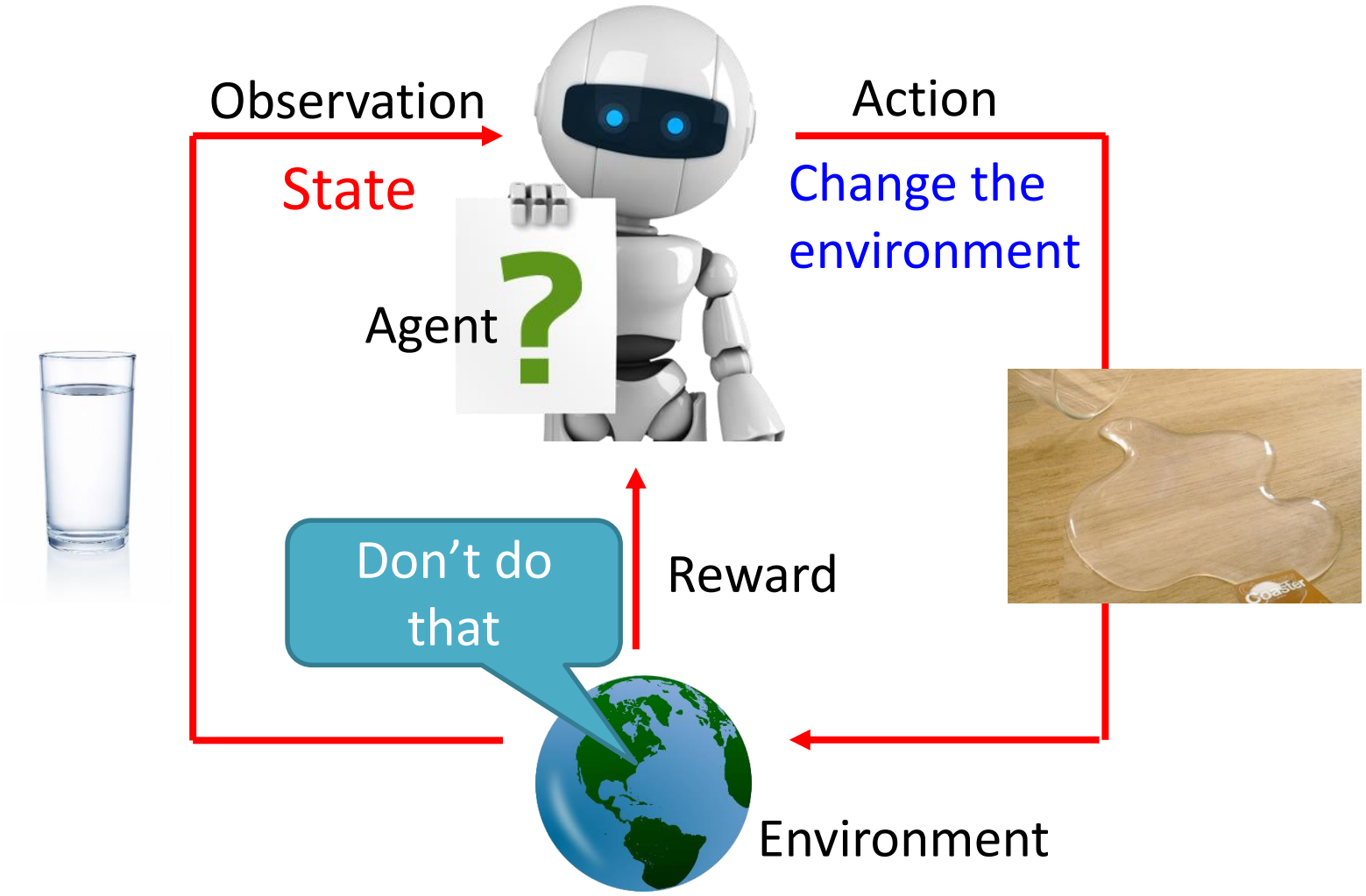


Supervised learning is "teach by **example**":
Here's some examples, now learn patterns in these example.

Reinforcement learning is "teach by **experience**":
Here's a world, now learn patterns by exploring it.



Scenario of Reinforcement Learning

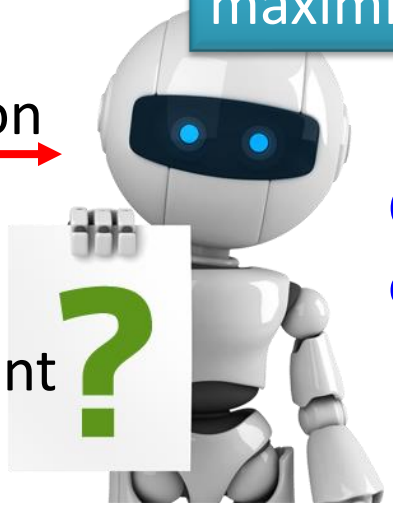


Scenario of Reinforcement Learning

Agent learns to take actions to maximize expected reward.

Observation

State



Agent

Action

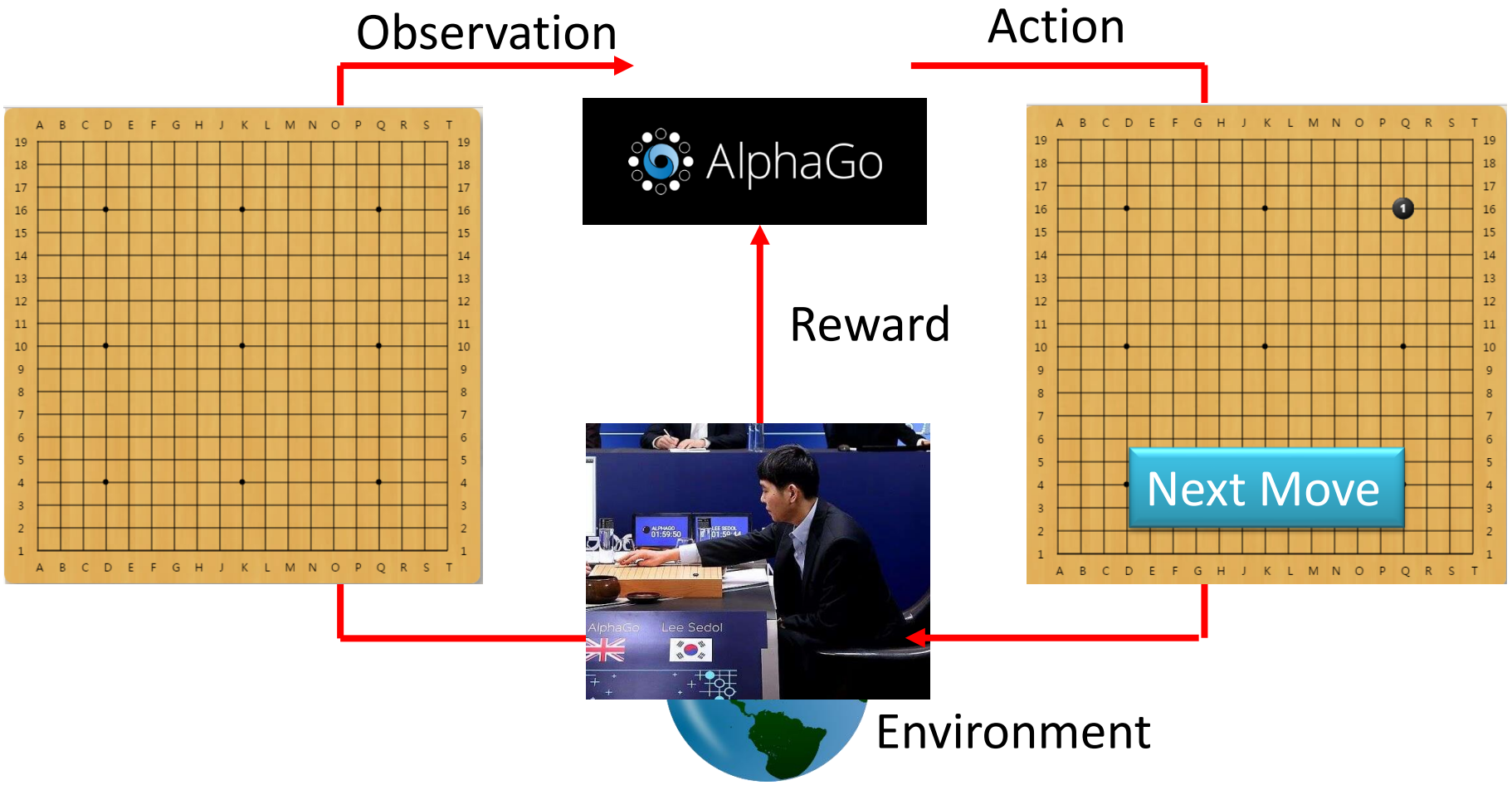
Change the environment

Reward

Thank you.

Environment

Learning to play Go

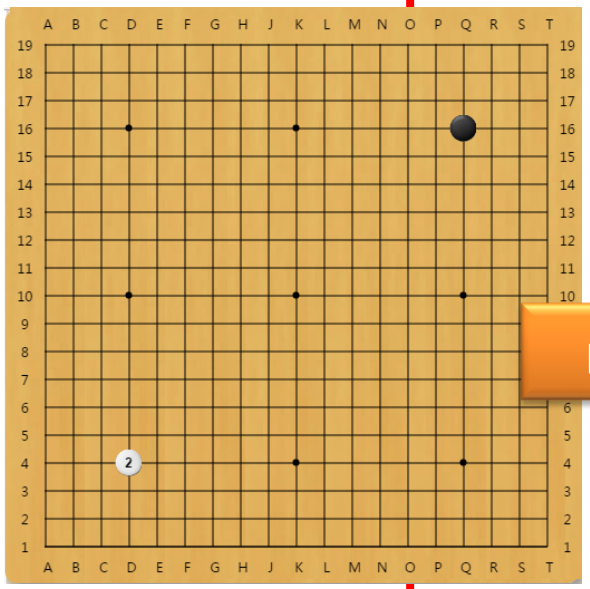


Learning to play Go

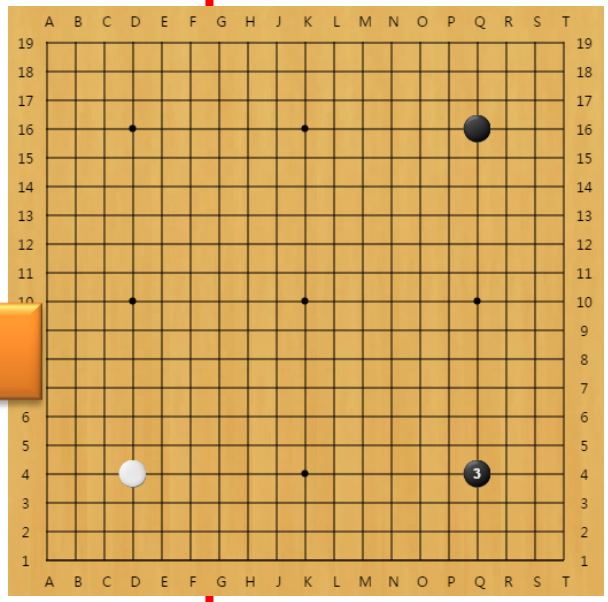
Agent learns to take actions to maximize expected reward.

Observation

Action



AlphaGo

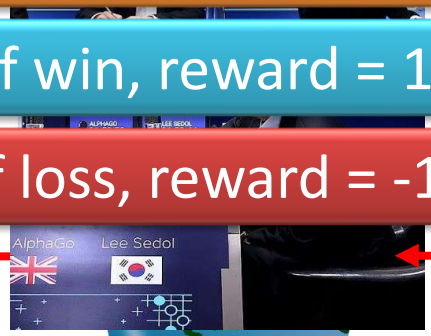


Reward

reward = 0 in most cases

If win, reward = 1

If loss, reward = -1

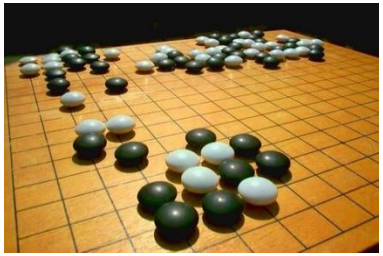


Environment

Learning to play Go
- Supervised v.s. Reinforcement

- Supervised: Learning from teacher

Next move:
"5-5"



Next move:
"3-3"

Learning from experience

- Reinforcement Learning

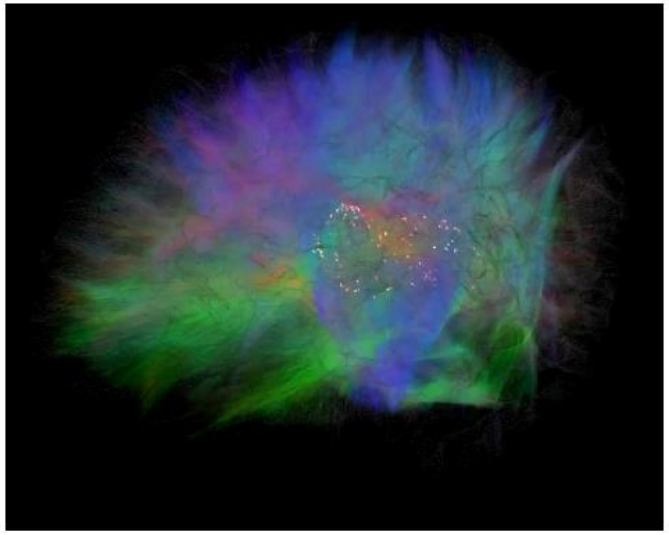
First move → …… many moves …… → Win!

(Two agents play with each other.)

Alpha Go is supervised learning + reinforcement learning.

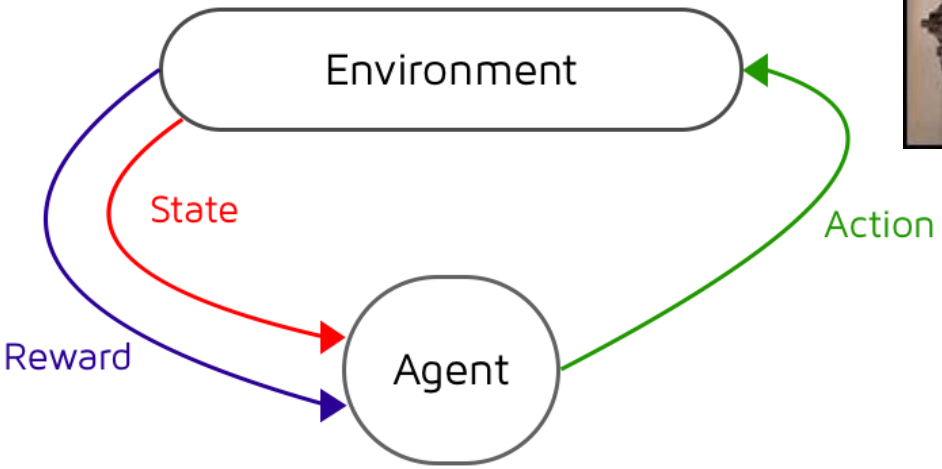
RL in Humans

- Human appear to learn to walk through “very few examples” of trial and error. How is an open question…
- Possible answers:
  - **Hardware**: 3 million years of evolutionary data
  - **Imitation Learning**: observation of other humans walking
  - **Algorithms**: better than backpropagation and stochastic gradient descent



RL Framework

- At each step, the agent should:
  - Executes **action**
  - Observes new **state**
  - Receives **reward**



Environment and Actions

- Fully Observable (Chess) vs Partially Observable (Poker, SC2)
- Single Agent (Atari) vs Multi Agent (SC2)
- Deterministic (Cart Pole) vs Stochastic (SC2)
- Static (Chess) vs Dynamic (SC2)
- Discrete (Chess) vs Continuous (Cart Pole)
- ***Note:*** ***Real-world environment might not technically be stochastic or partially-observable but might as well be treated as such due to their complexity***

The Challenge for RL in Real-World Applications

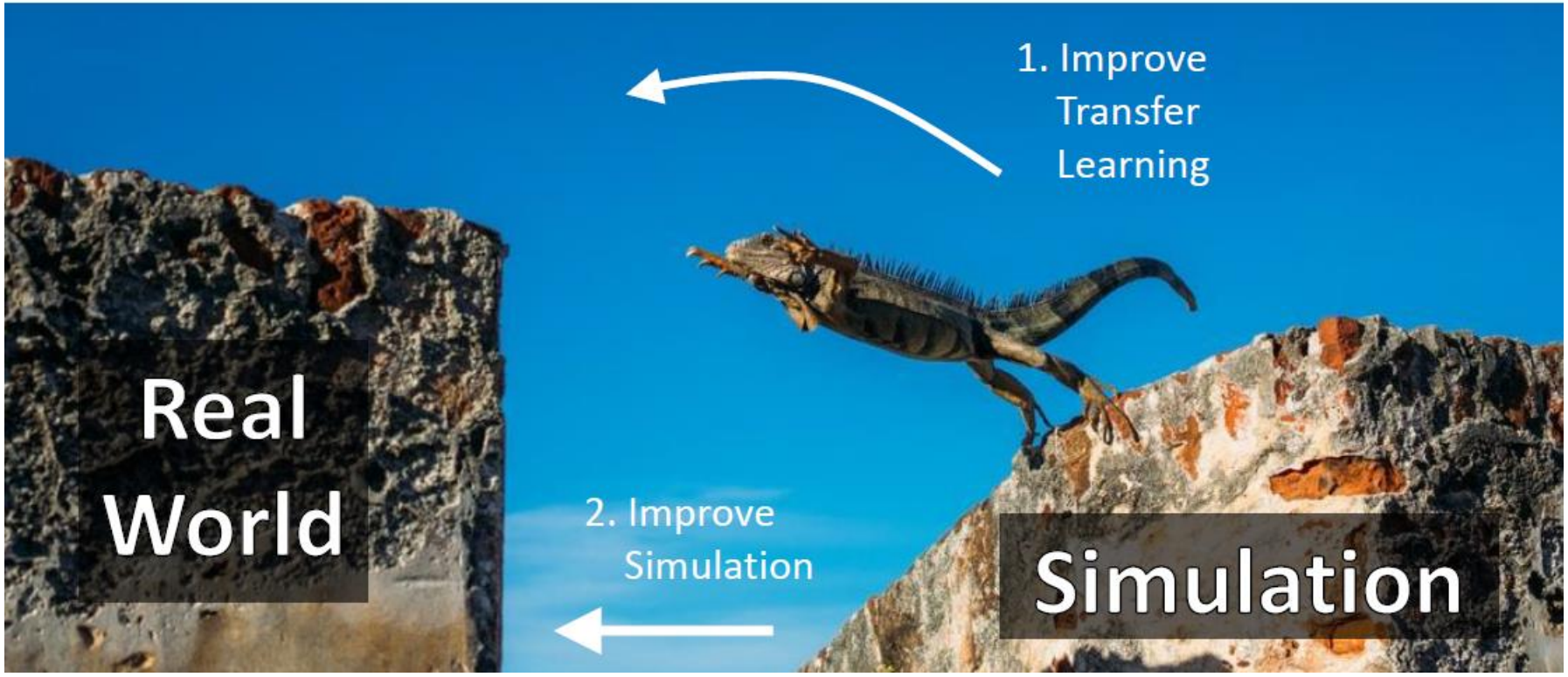
Reminder:

Supervised learning:
teach by example

Reinforcement learning:
teach by experience

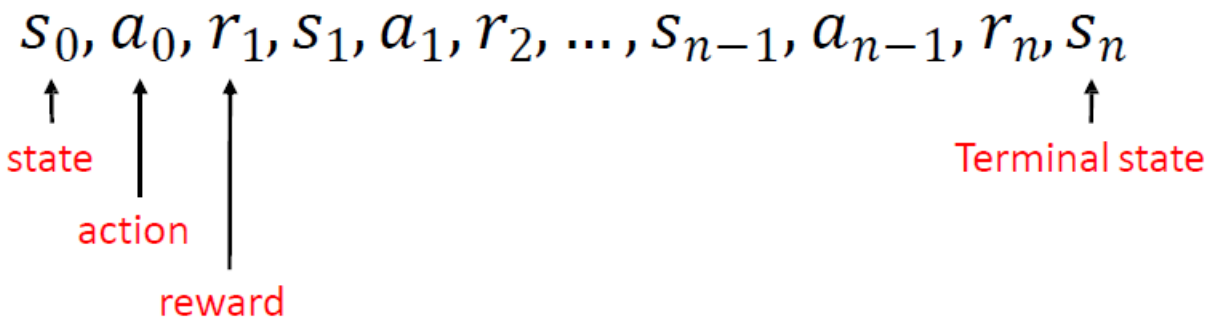
Open **Challenges**. Two Options:

1. Real world observation + one-shot trial & error
2. Realistic simulation + transfer learning



Major Components of an RL Agent

- An RL agent may be directly or indirectly trying to learn a:
  - **Policy**: agent's behavior function
  - **Value function**: how good is each state and/or action
  - **Model**: agent's representation of the environment



RL Agent: Maximize Reward

- Future reward: $R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_n$
- Discounted future reward:

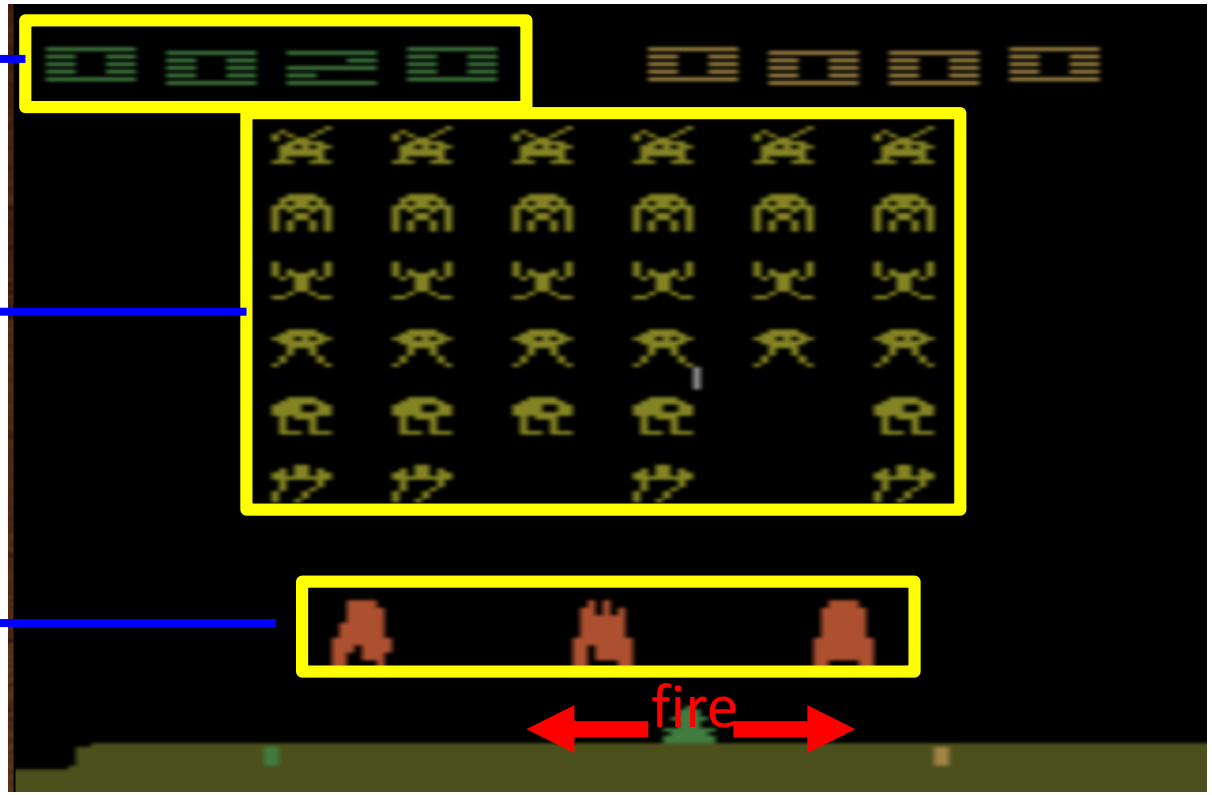
$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n$$

- A good strategy for an agent would be to always choose an action that maximizes the (discounted) future reward
- Why "discounted"?
  - Math trick to help analyze convergence
  - Uncertainty due to environment stochasticity, partial observability, or that life can end at any moment:

Example: Playing Video Game

- Space invader

Termination: all the aliens are killed, or your spaceship is destroyed.

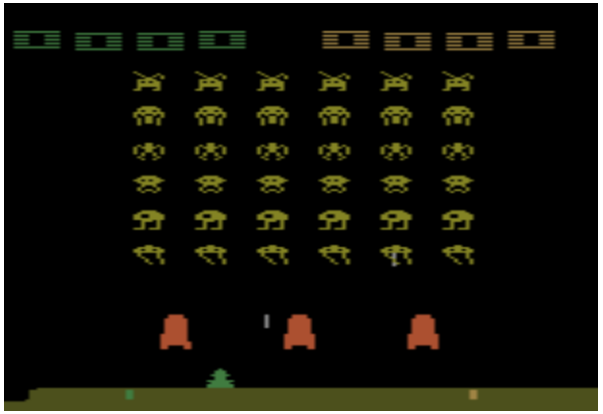


Example: Playing Video Game

Start with observation s_1

Observation s_2

Observation s_3



Obtain reward $r_1 = 0$

Action a_1: "right"

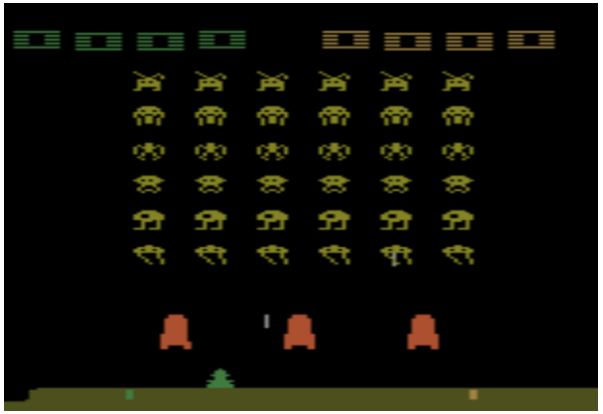
Obtain reward $r_2 = 5$

Action a_2 : "fire"
(kill an alien)

Example: Playing Video Game

Start with observation s_1

Observation s_2



Observation s_3

After many turns

Game Over (spaceship destroyed)

Obtain reward r_T

Action a_T

This is an ***episode***.

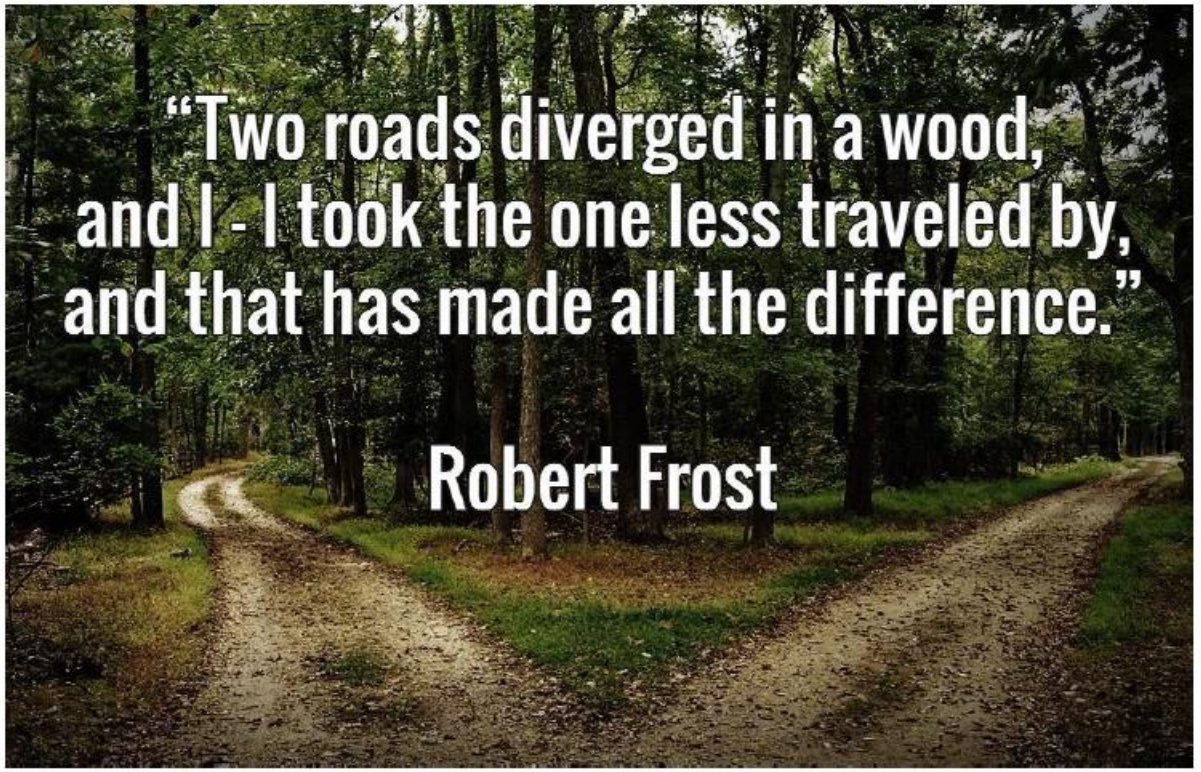
Learn to maximize the expected cumulative reward per episode

Difficulties of Reinforcement Learning

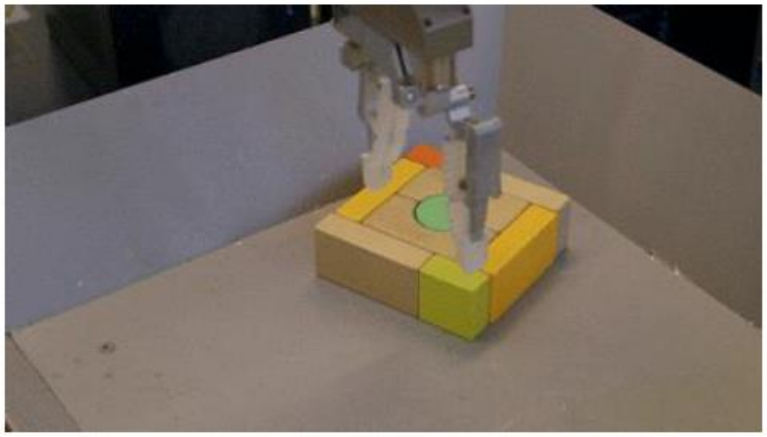
- Reward delay
  - In space invader, only “fire” obtains reward
    - Although the moving before “fire” is important
  - In Go playing, it may be better to sacrifice immediate reward to gain more long-term reward
- Agent’s actions affect the subsequent data it receives
  - E.g. Exploration

Lessons Take Away

- Environment model has big impact on optimal policy
- Reward structure has big impact on optimal policy



Real World Example



Grasping Objects with Robotic Arm

- **Goal** - Pick an object of different shapes
- **State** - Raw pixels from camera
- **Actions** – Move arm. Grasp.
- **Reward** - Positive when pickup is successful

RL in Humans

- Human – Life is a big training set
- **Goal**: Survival?
- **State**: Sight. Hearing. Taste. Smell. Touch.
- **Actions**: Think. Move.
- **Reward**: Happiness?

Key Takeaways

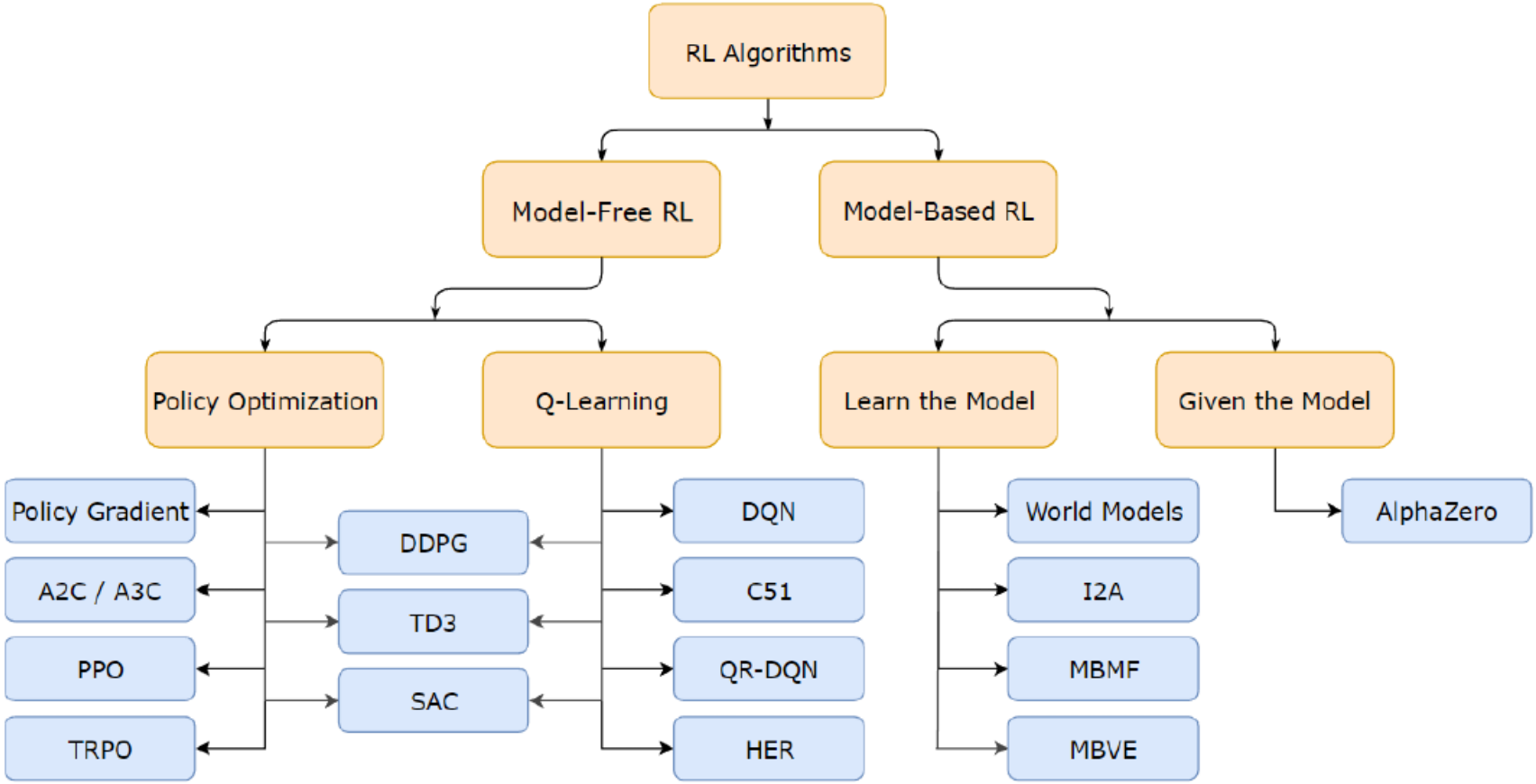
- **Deep Learning**:
  - Fun part: Good algorithms that learn from data
  - Hard part: Good questions, huge amounts of representative data

- **Deep Reinforcement Learning**:
  - Fun part: Good algorithms that learn from data
  - Hard part: Defining a useful state space, action space, and reward
  - Hardest part: Getting meaningful data for the above formalization

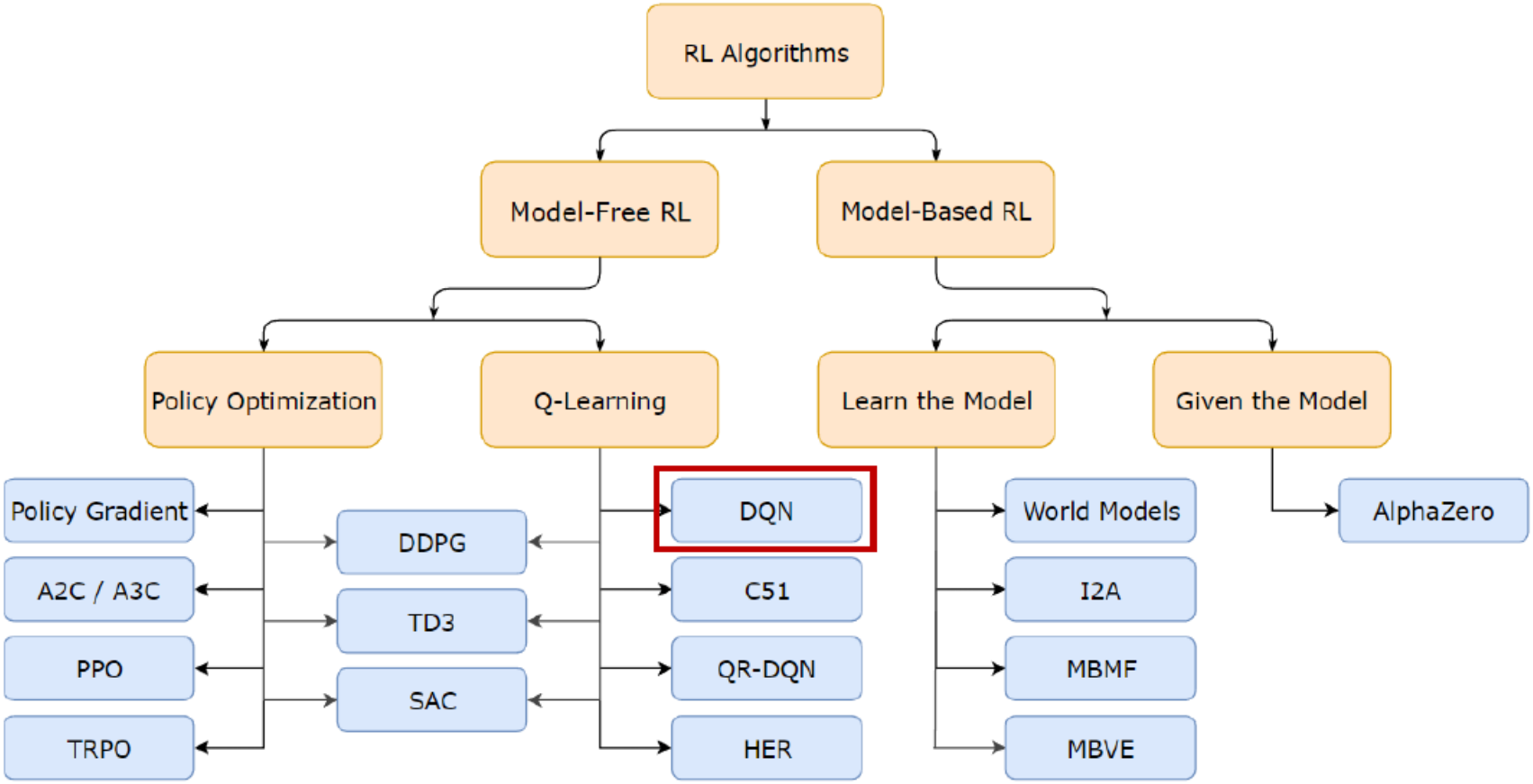
RL Methods

- **Model-based**
  - Learn the model of the world, then plan using the model
  - Update model often
  - Re plan often
- **Value-based**
  - Learn the state or state action value
  - Act by choosing best action in state
  - Exploration is a necessary add on
- **Policy-based**
  - Learn the stochastic policy function that maps state to action
  - Act by sampling policy
  - Exploration is baked in

RL Methods

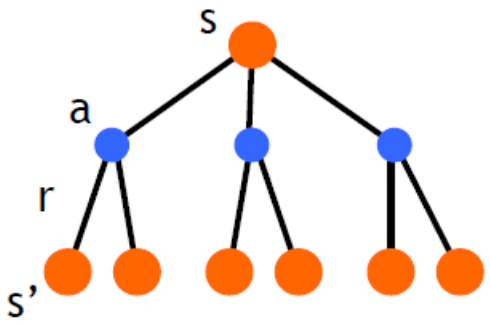

RL Algorithms
Model-Free RL
Model-Based RL
Policy Optimization
Q-Learning
Learn the Model
Given the Model
Policy Gradient
A2C / A3C
PPO
TRPO
DDPG
TD3
SAC
DQN
C51
QR-DQN
HER
World Models
I2A
MBMF
MBVE
AlphaZero

RL Methods


RL Algorithms
Model-Free RL
Model-Based RL
Policy Optimization
Q-Learning
Learn the Model
Given the Model
Policy Gradient
A2C / A3C
PPO
TRPO
DDPG
TD3
SAC
DQN
C51
QR-DQN
HER
World Models
I2A
MBMF
MBVE
AlphaZero

Q-Learning

- State-action value function: $Q^{\pi}(s,a)$
  - Expected return when starting in *s*, performing *a*, and following π



- Q-Learning: Use **any policy** to estimate Q that maximizes future reward:
  - Q directly approximates Q* (Bellman optimality equation)
  - Independent of the policy being followed
  - Only requirement: keep updating each (s,a) pair

Learning Rate

Discount Factor

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

New State

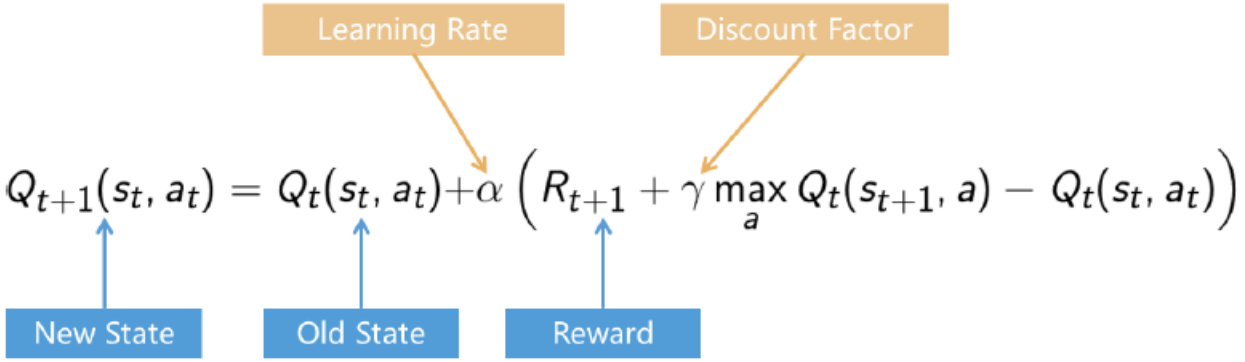
Old State

Reward

Exploration vs Exploitation

- Deterministic/greedy policy won't explore all actions
  - Don't know anything about the environment at the beginning
  - Need to try all actions to find the optimal one
- ε-greedy policy
  - With probability 1-ε perform the optimal/greedy action, otherwise random action
  - Slowly move it towards greedy policy: ε -> 0

Q-Learning: Value Iteration

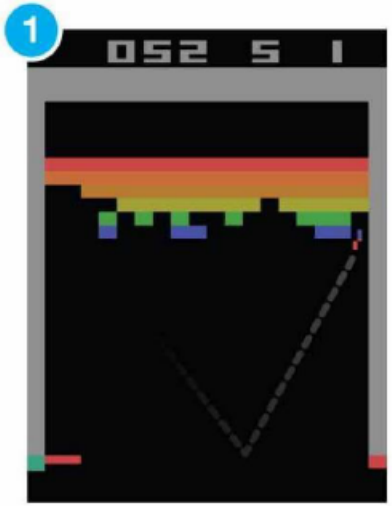


	A1	A2	A3	A4
S1	+1	+2	-1	0
S2	+2	0	+1	-2
S3	-1	+1	0	-2
S4	-2	0	+1	+1

```
initialize Q[num_states,num_actions] arbitrarily
observe initial state s
repeat
      select and carry out an action a
      observe reward r and new state s'
      Q[s,a] = Q[s,a] + α(r + γ max_a' Q[s',a'] - Q[s,a])
      s = s'
until terminated
```

Q-Learning: Representation

- In practice, Value Iteration is impractical
  - Very limited states/actions
  - Cannot generalize to unobserved states



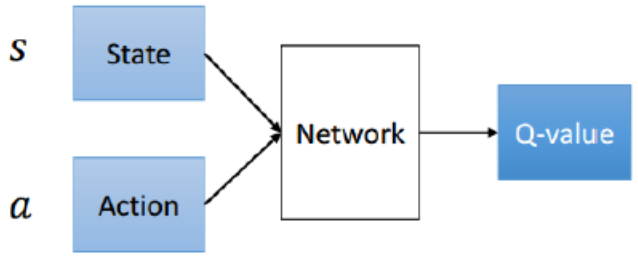
- Think about the **Breakout** game
  - State: screen pixels
    - Image size: 84×84 (resized)
    - Consecutive 4 images
    - Grayscale with 256 gray levels

$256^{84\times84\times4}$ rows in the Q-table!

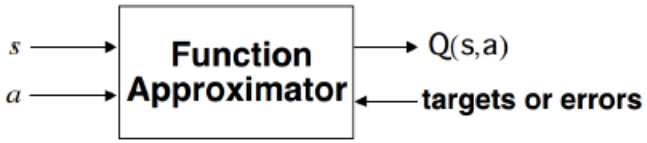
$= 10^{69,970} >> 10^{82}$ atoms in the universe

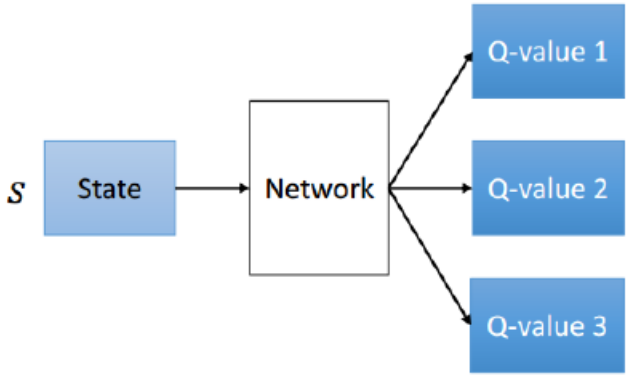
Deep Q-Learning

Use a neural network to approximate the Q-function:

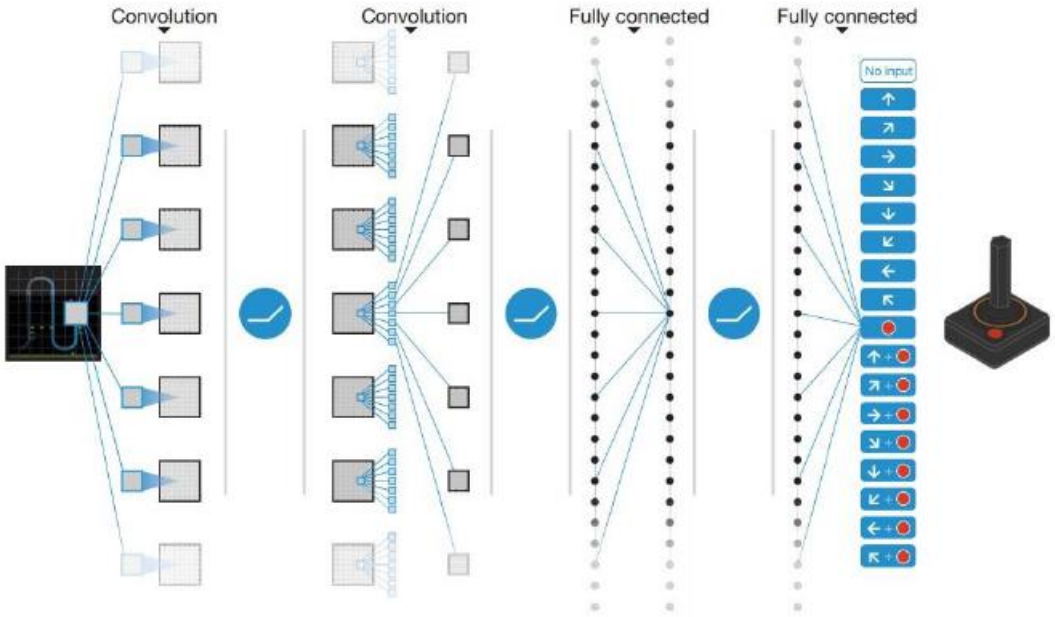


$$Q(s, a; \boldsymbol{\theta}) \approx Q^*(s, a)$$





DQN: Atari



Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8x8	4	32	ReLU	20x20x32
conv2	20x20x32	4x4	2	64	ReLU	9x9x64
conv3	9x9x64	3x3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

Mnih et al. "Playing atari with deep reinforcement learning." 2013.

Double DQN

- Loss function (squared error):

$$L = \mathbb{E}[(\underbrace{\boldsymbol{r + \gamma max_{a'} Q(s', a')}}_{\textbf{target}} - \underbrace{Q(s, a)}_{\textbf{prediction}})^2]$$

- DQN: same network for both Q
- Double DQN: separate network for each Q
  - Helps reduce bias introduced by the inaccuracies of Q network at the beginning of training

DQN Tricks

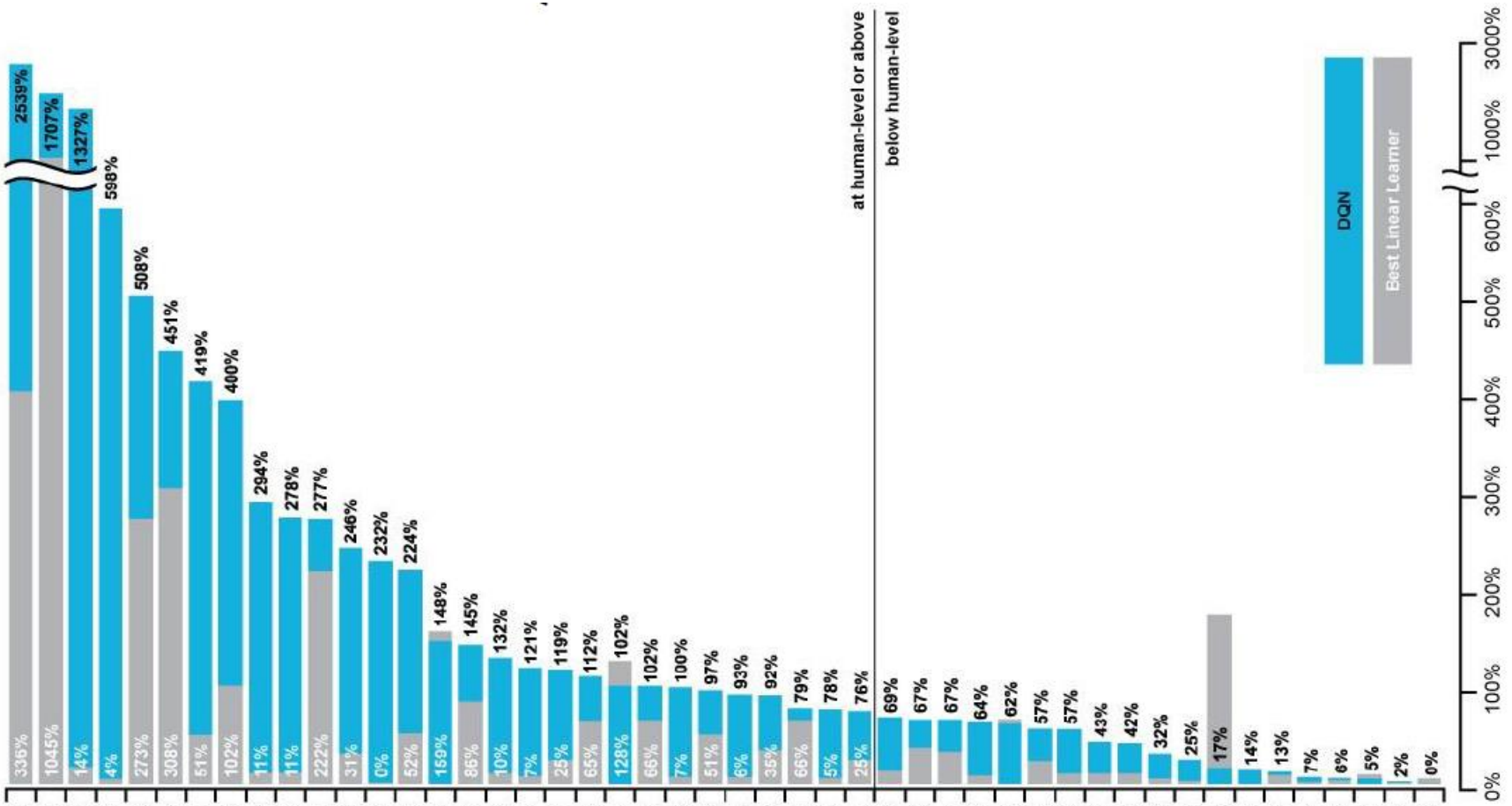
- Experience Replay
  - Stores experiences (actions, state transitions, and rewards) and creates mini-batches from them for the training process
- Fixed Target Network
  - Error calculation includes the target function depends on network parameters and thus changes quickly. Updating it only every 1,000 steps increases stability of training process.

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left[r_{t+1} + \gamma \max_p Q(s_{t+1}, p) - Q(s_t, a) \right]$$

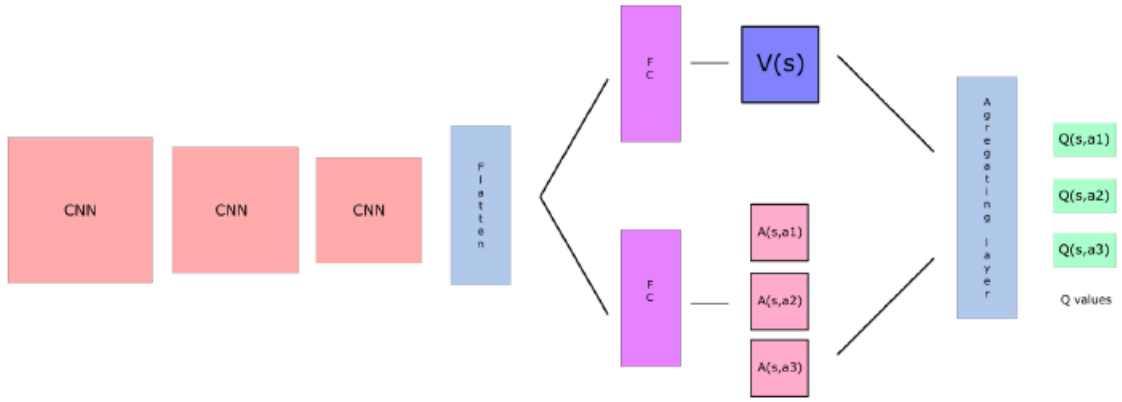
target Q function in the red rectangular is fixed

Replay	○	○	×	×
Target	○	×	○	×
Breakout	**316.8**	240.7	10.2	3.2
River Raid	**7446.6**	4102.8	2867.7	1453.0
Seaquest	**2894.4**	822.6	1003.0	275.8
Space Invaders	**1088.9**	826.3	373.2	302.0

DQN in Atari


Video Pinball 336% 2539%
Boxing 1045% 1707%
Breakout 14% 1327%
Star Gunner 4% 598%
Robotank 273% 508%
Atlantis 308% 451%
Crazy Climber 51% 419%
Gopher 102% 400%
Demon Attack 11% 294%
Name This Game 11% 278%
Krull 222% 277%
Assault 31% 246%
Road Runner 0% 232%
Kangaroo 52% 224%
Tennis 159% 148%
James Bond 86% 145%
Pong 10% 132%
Space Invaders 7% 121%
Beam Rider 25% 119%
Tutankham 65% 112%
Kung-Fu Master 128% 102%
Freeway 66% 102%
Time Pilot 7% 100%
Enduro 51% 97%
Fishing Derby 6% 93%
Up and Down 35% 92%
Ice Hockey 66% 79%
Q*Bert 5% 78%
H.E.R.O. 25% 76%
at human-level or above
below human-level
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pacman 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
DQN
Best Linear Learner
0%
100%
200%
300%
400%
500%
600%
1000%
3000%

Dueling DQN (DDQN)



- Decompose **Q(s,a)**

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

  - **V(s)**: the value of being at that state
  - **A(s,a)**: the advantage of taking action ***a*** in state ***s*** versus all other possible actions at that state
- Use two streams:
  - one that estimates the state value V(s)
  - one that estimates the advantage for each action A(s,a)
- Useful for states where action choice does not affect Q(s,a)

Prioritized Experience Replay

$$p_t = |\delta_t| + e$$

Magnitude of our TD error

Constant assures that no experience has 0 probability to be taken.

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

Priority value

Normalized by all priority values in Replay Buffer

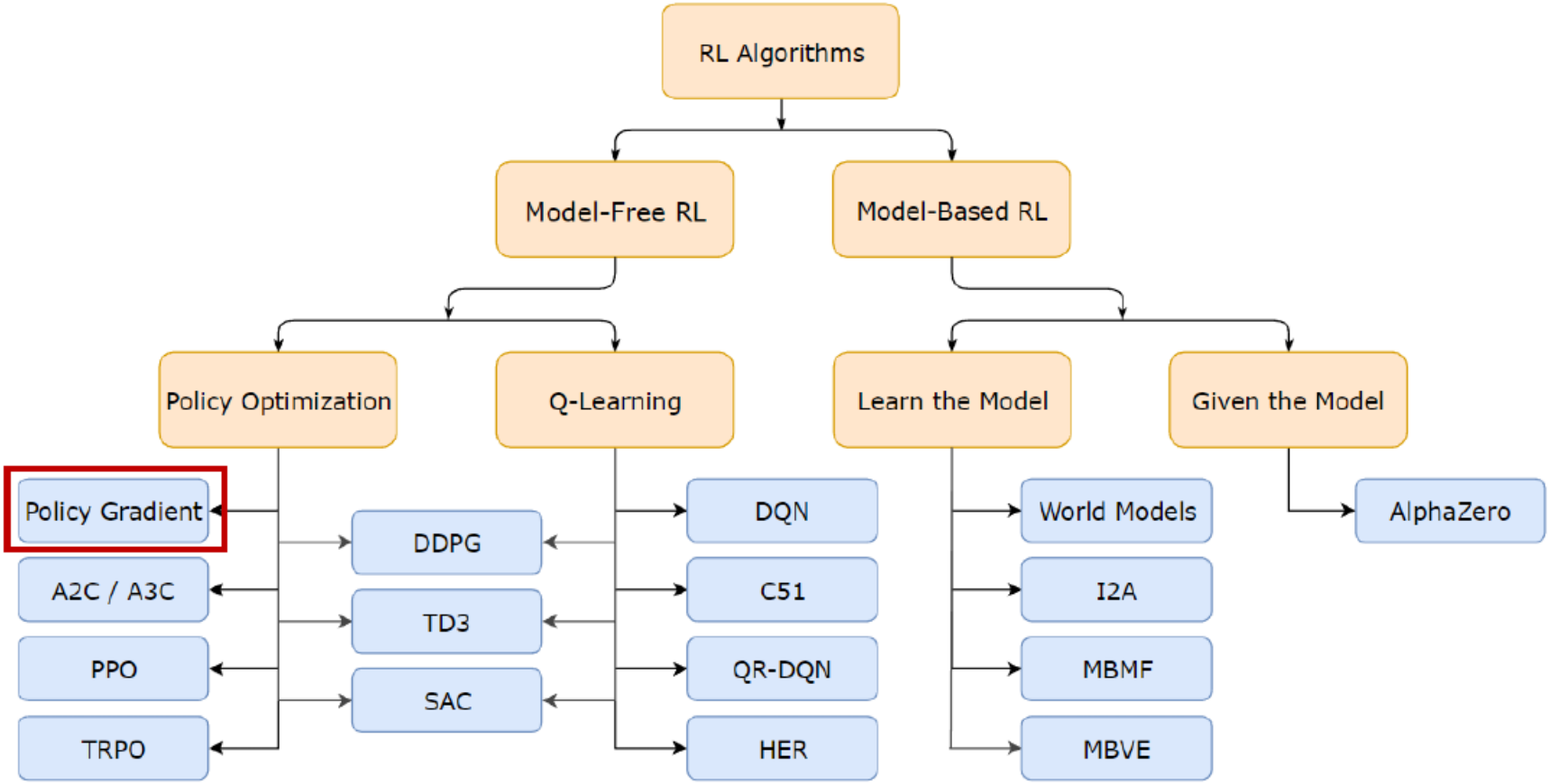
Hyperparameter used to **reintroduce some randomness in the experience selection for the replay buffer**

If **a = 0** pure uniform randomness

If **a = 1** only select the experiences with the highest priorities

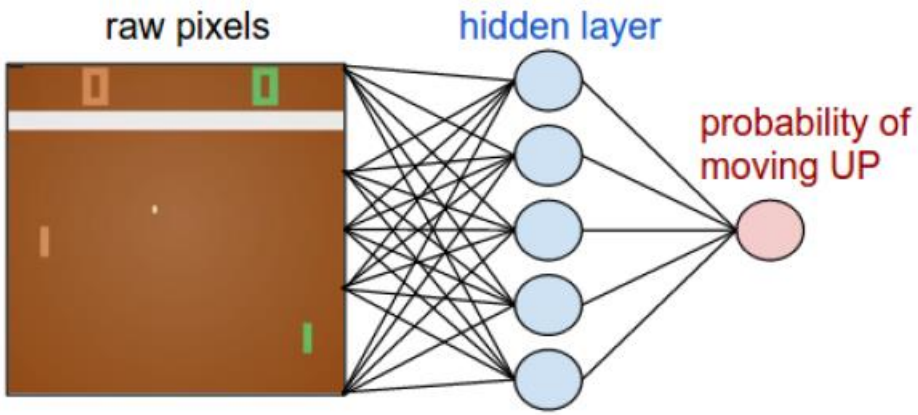
- Sample experiences based on impact not frequency of occurrence

RL Methods


RL Algorithms
Model-Free RL
Model-Based RL
Policy Optimization
Q-Learning
Learn the Model
Given the Model
Policy Gradient
A2C / A3C
PPO
TRPO
DDPG
TD3
SAC
DQN
C51
QR-DQN
HER
World Models
I2A
MBMF
MBVE
AlphaZero

Policy Gradient (PG)

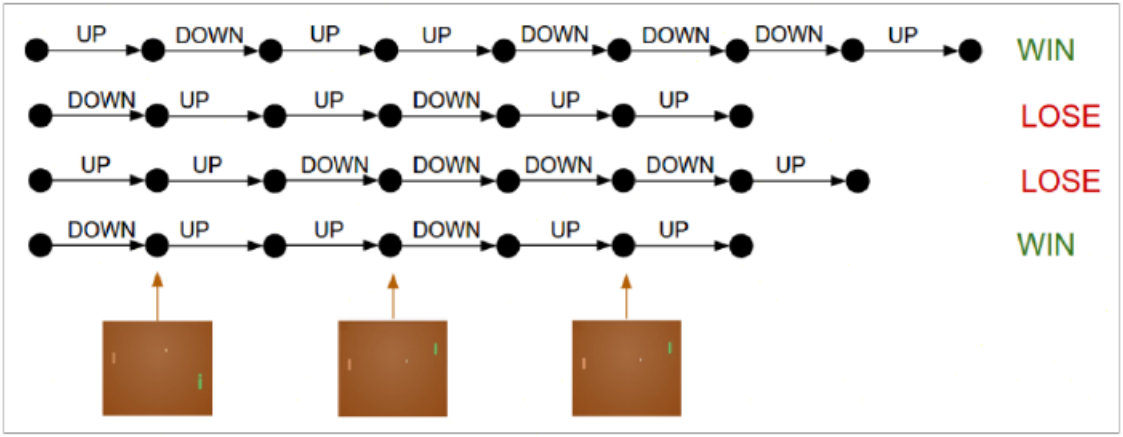
- **DQN (off-policy):** Approximate Q and infer optimal policy
- **PG (on-policy):** Directly optimize policy space



Policy Network

Policy Gradient: REINFORCE

Policy Gradients: Run a policy for a while. See what actions led to high rewards. Increase their probability.



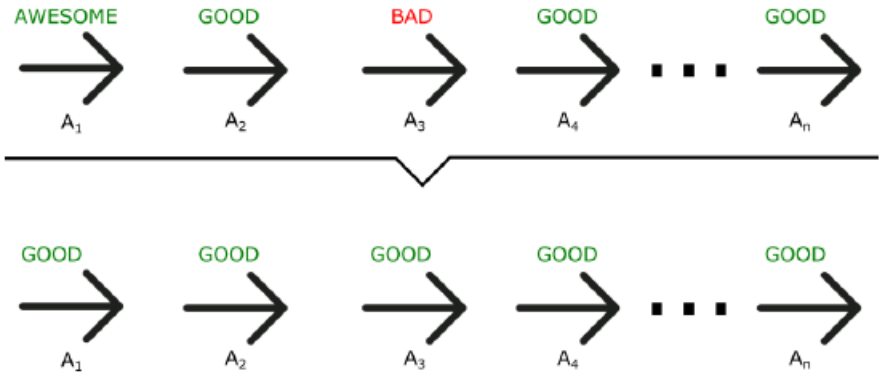
- **REINFORCE:** Policy gradient that increases probability of good actions and decreases probability of bad action:

$$\nabla_\theta E[R_t] = E[\nabla_\theta log P(a) R_t]$$

Policy Gradient

- Pros vs DQN:
  - **Messy World:** If Q function is too complex to be learned, DQN may fail miserably, while PG will still learn a good policy.
  - **Speed:** Faster convergence
  - **Stochastic Policies:** Capable of learning stochastic policies - DQN can't
  - **Continuous actions:** Much easier to model continuous action space
- Cons vs DQN:
  - **Data:** Sample inefficient (needs more data)
  - **Stability:** Less stable during training process.
  - Poor **credit assignment** to (state, action) pairs for delayed rewards

Problem with REINFORCE:
Calculating the reward at the end, means all the actions will be averaged as good because the total reward was high.



Actor-Critic

- Combine DQN (value-based) and REINFORCE (policy-based)
- Two neural networks (Actor and Critic):
  - **Actor** is policy-based: Samples the action from a policy
  - **Critic** is value-based: Measures how good the chosen action is

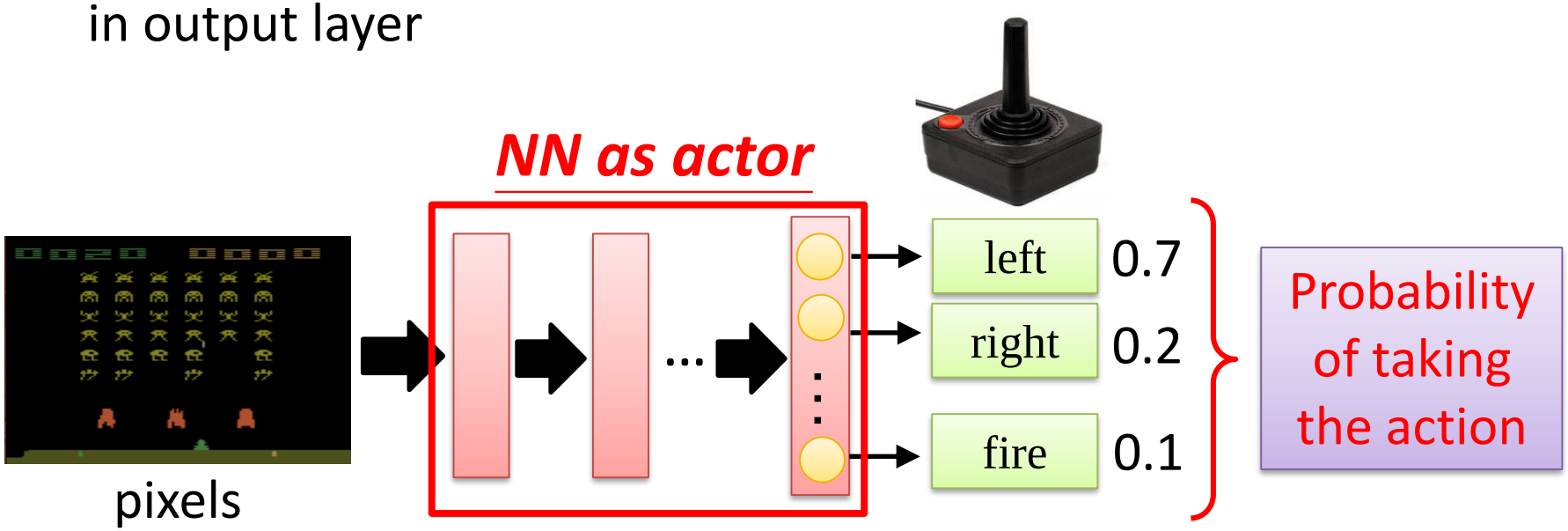
Policy Update: $$\Delta\theta = \alpha * \nabla_\theta * (log\ \pi(S_t, A_t, \theta)) * R(t)$$

New update: $$\Delta\theta = \alpha * \nabla_\theta * (log\ \pi(S_t, A_t, \theta)) * Q(S_t, A_t)$$

- Update at each time step - temporal difference (TD) learning

Neural network as Actor

- Input of neural network: the observation of machine represented as a vector or a matrix
- Output neural network : each action corresponds to a neuron in output layer

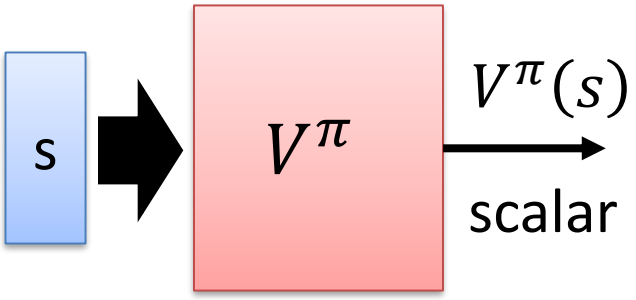


Critic

- A critic does not determine the action.
- Given an actor, it evaluates the how good the actor is

Critics

- A critic is a function depending on the actor π it is evaluated
  - The function is represented by a neural network
- State value function $V^{\pi}(s)$
  - When using actor π, the *cumulated* reward expects to be obtained after seeing observation (state) s

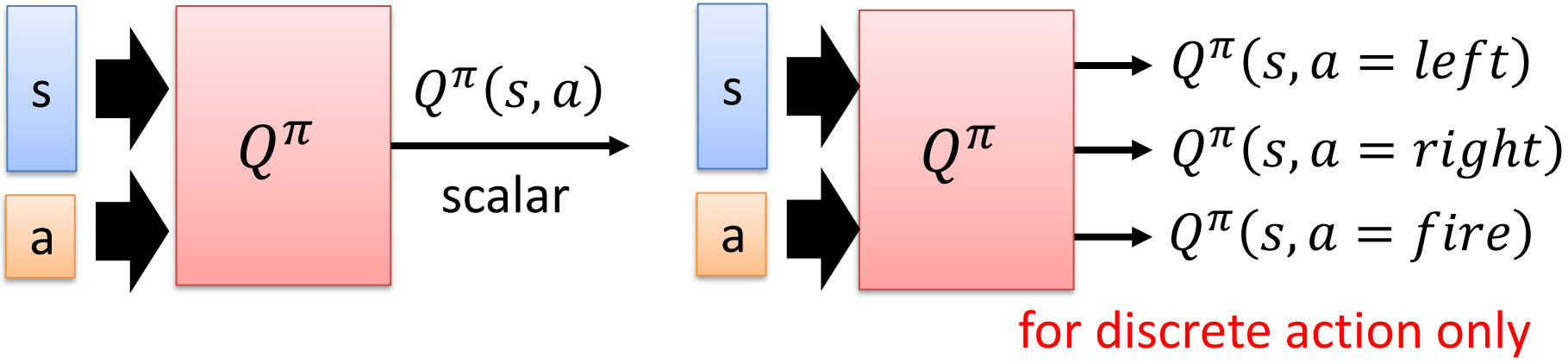


$V^{\pi}(s)$ is large

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

Critics

- State-action value function $Q^{\pi}(s, a)$
  - When using actor π, the *cumulated* reward expects to be obtained after seeing observation s and taking a

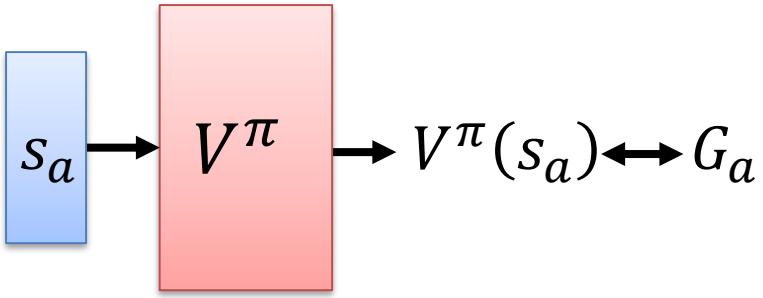


How to estimate $V^{\pi}(s)$

- Monte-Carlo based approach
  - The critic watches π playing the game

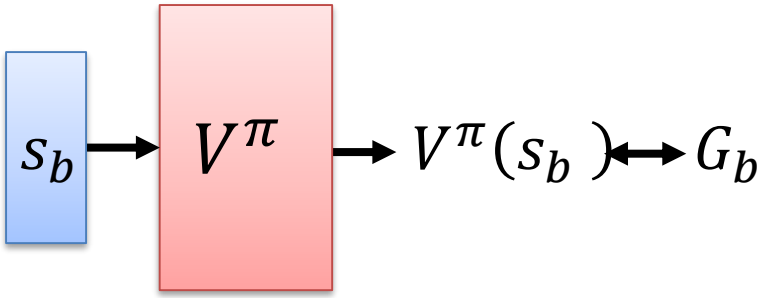
After seeing s_a,

Until the end of the episode, the cumulated reward is G_a



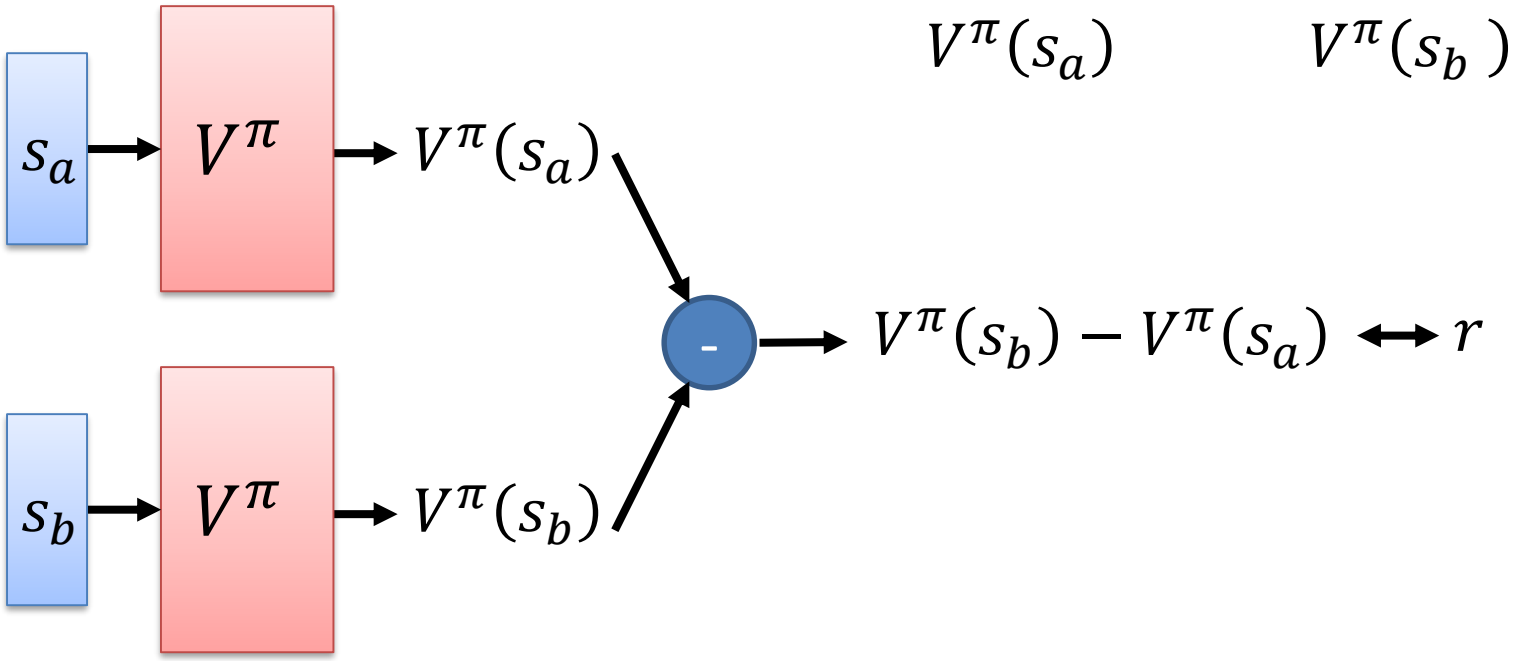
After seeing s_b,

Until the end of the episode, the cumulated reward is G_b



How to estimate $V^{\pi}(s)$

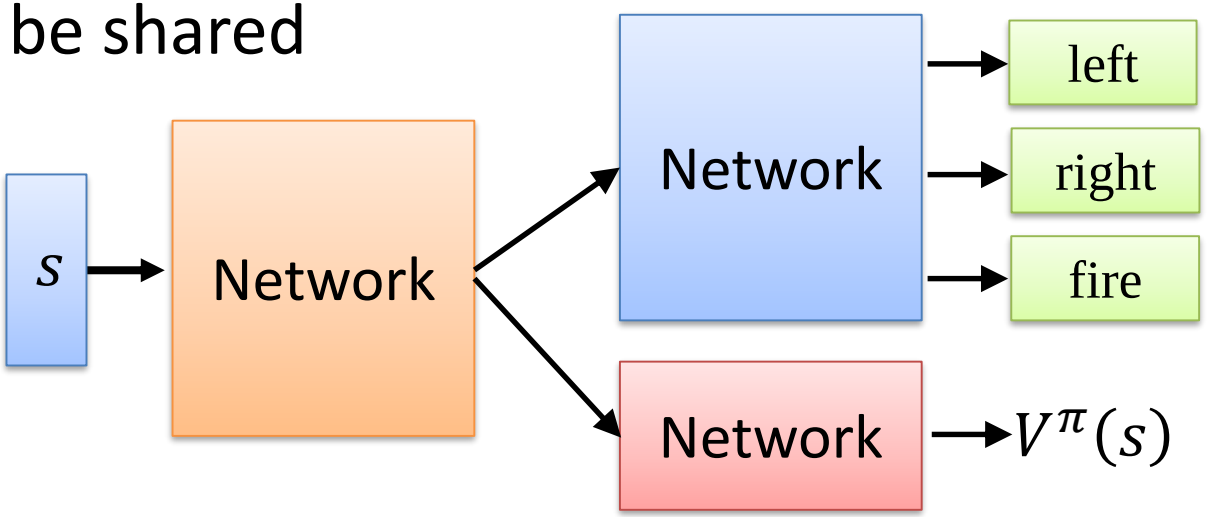
- Temporal-difference approach $\ldots s_a, a, r, s_b \ldots$



Some applications have very long episodes, so that delaying all learning until an episode's end is too slow.

Actor-Critic

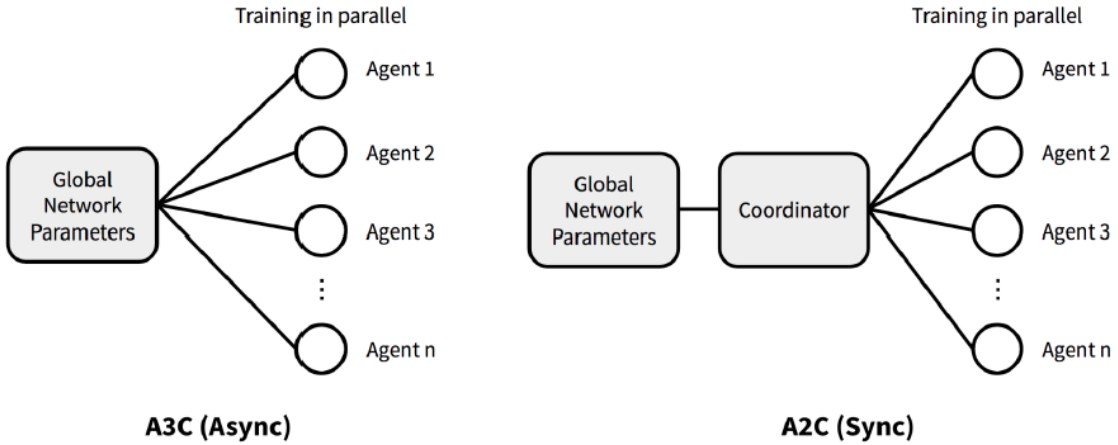
- Tips
  - The parameters of actor $\pi(s)$ and critic $V^{\pi}(s)$ can be shared



  - Use output entropy as regularization for $\pi(s)$
    - Larger entropy is preferred → exploration

A2C, A3C

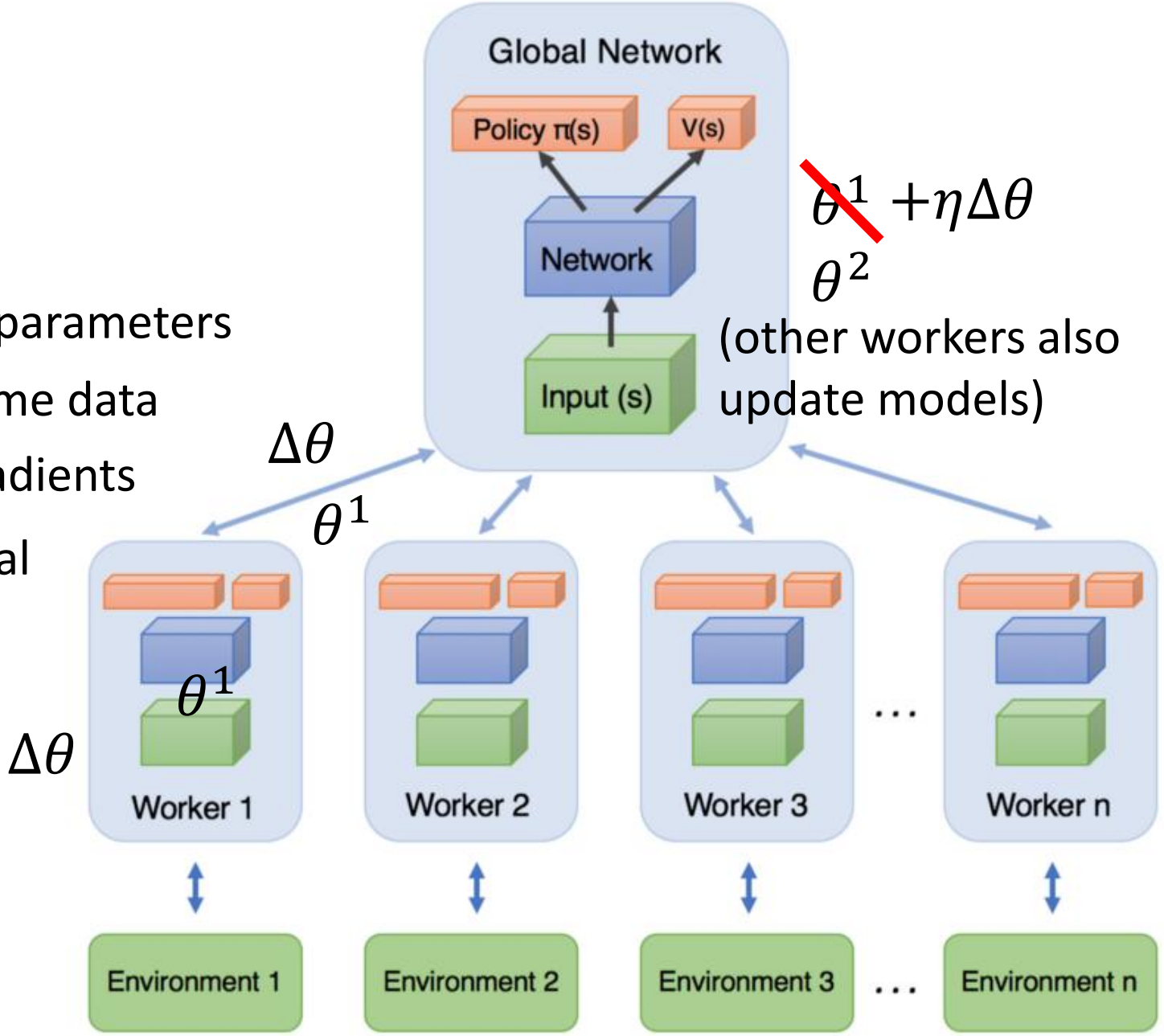
- Advantage Actor-Critic (A2C)
- Asynchronous Advantage Actor-Critic (A3C)



- Both use parallelism in training
- A2C syncs up for global parameter update and then start each iteration with the same policy

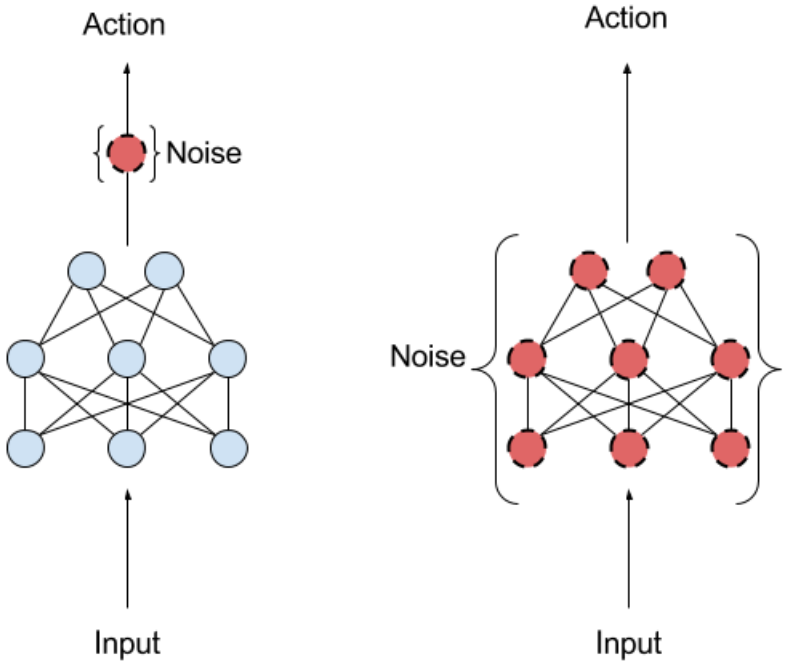
A3C

1. Copy global parameters
2. Sampling some data
3. Compute gradients
4. Update global models

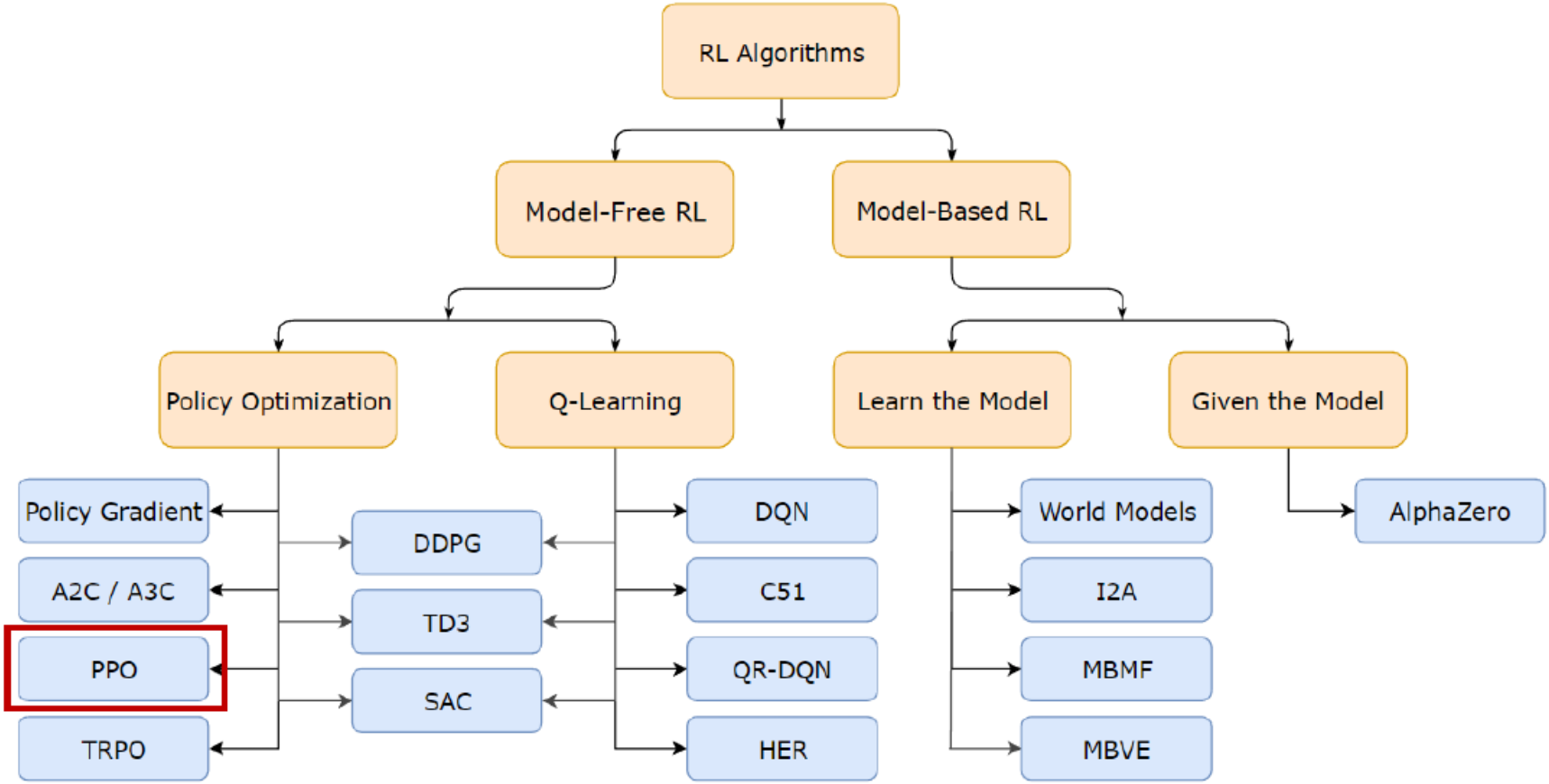


Deep Deterministic Policy Gradient (DDPG)

- Actor-Critic framework for learning a deterministic policy
- Can be thought of as: DQN for continuous action spaces
- As with all DQN, following tricks are required:
  - Experience Replay
  - Target network
- Exploration: add noise to actions, reducing scale of the noise as training progresses



RL Methods


RL Algorithms
Model-Free RL
Model-Based RL
Policy Optimization
Q-Learning
Learn the Model
Given the Model
Policy Gradient
A2C / A3C
PPO
TRPO
DDPG
TD3
SAC
DQN
C51
QR-DQN
HER
World Models
I2A
MBMF
MBVE
AlphaZero

Policy Optimization

- **Progress** beyond Vanilla Policy Gradient:
  - Natural Policy Gradient
  - TRPO
  - PPO
- Basic idea in on-policy optimization:
  Avoid taking bad actions that collapse the training performance.

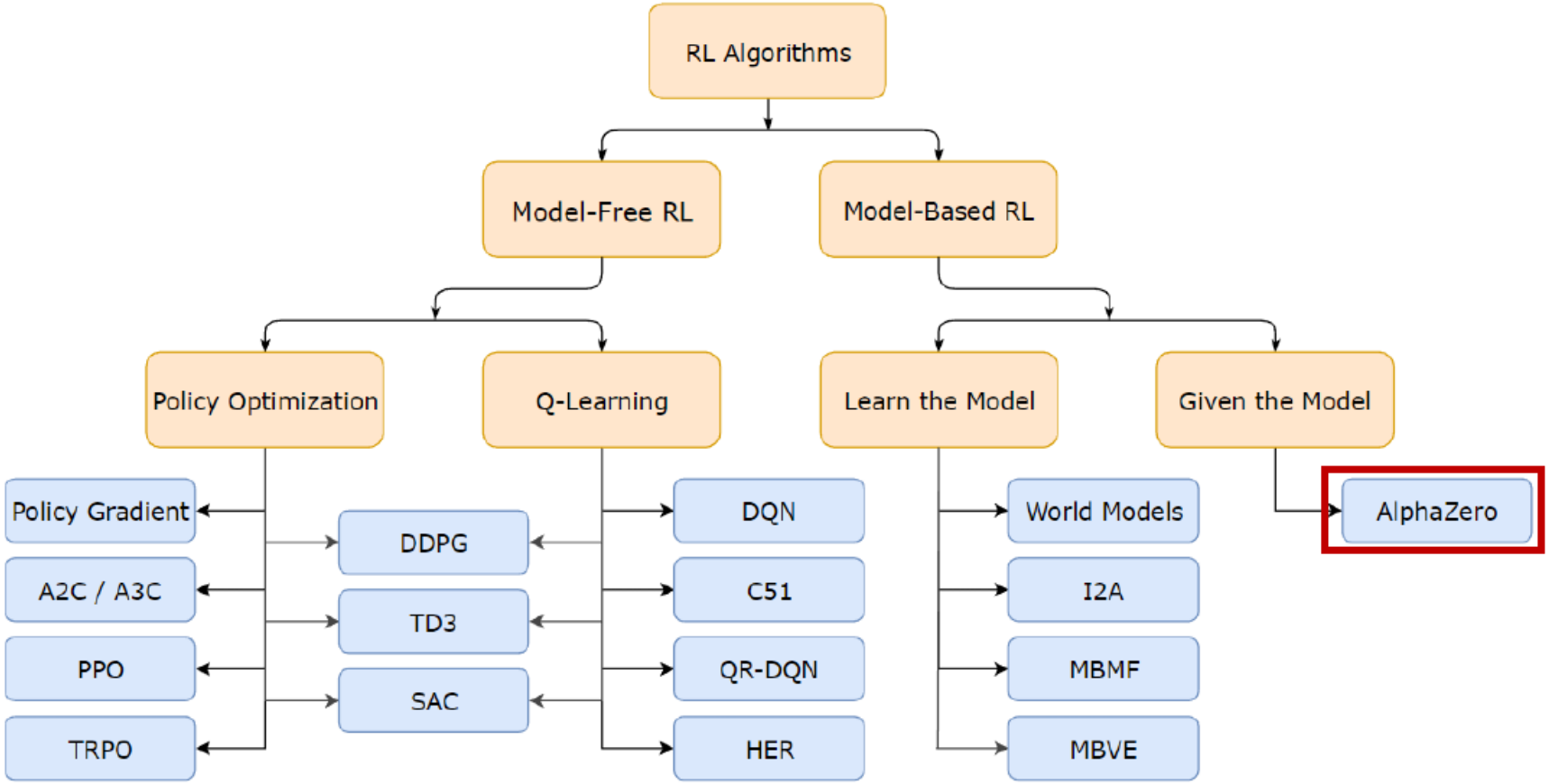
Line Search:

First pick direction, then step size

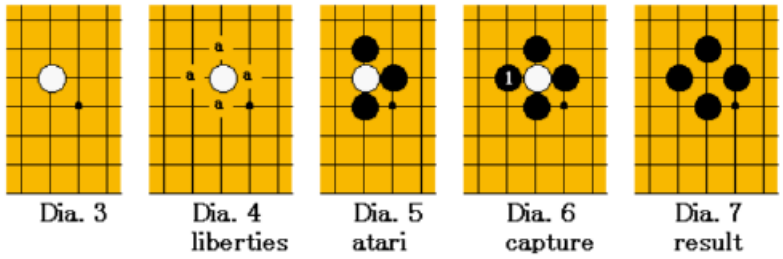
Trust Region:

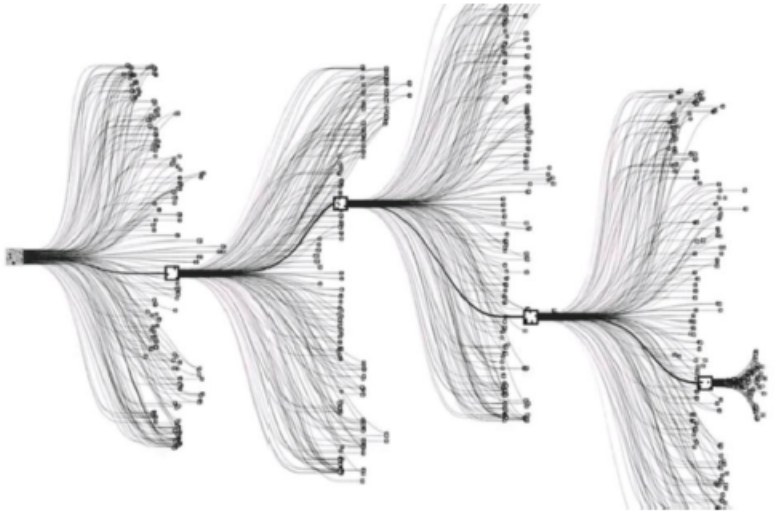
First pick step size, then direction

RL Methods


RL Algorithms
Model-Free RL
Model-Based RL
Policy Optimization
Q-Learning
Learn the Model
Given the Model
Policy Gradient
A2C / A3C
PPO
TRPO
DDPG
TD3
SAC
DQN
C51
QR-DQN
HER
World Models
I2A
MBMF
MBVE
AlphaZero

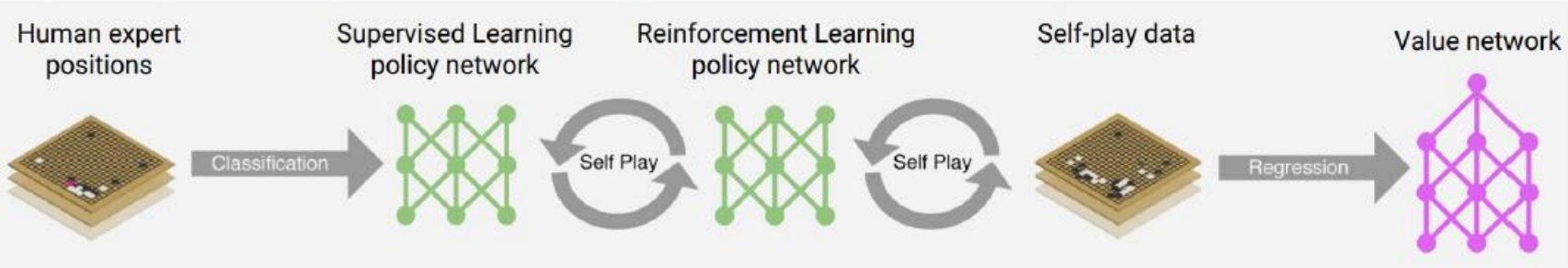
Game of Go

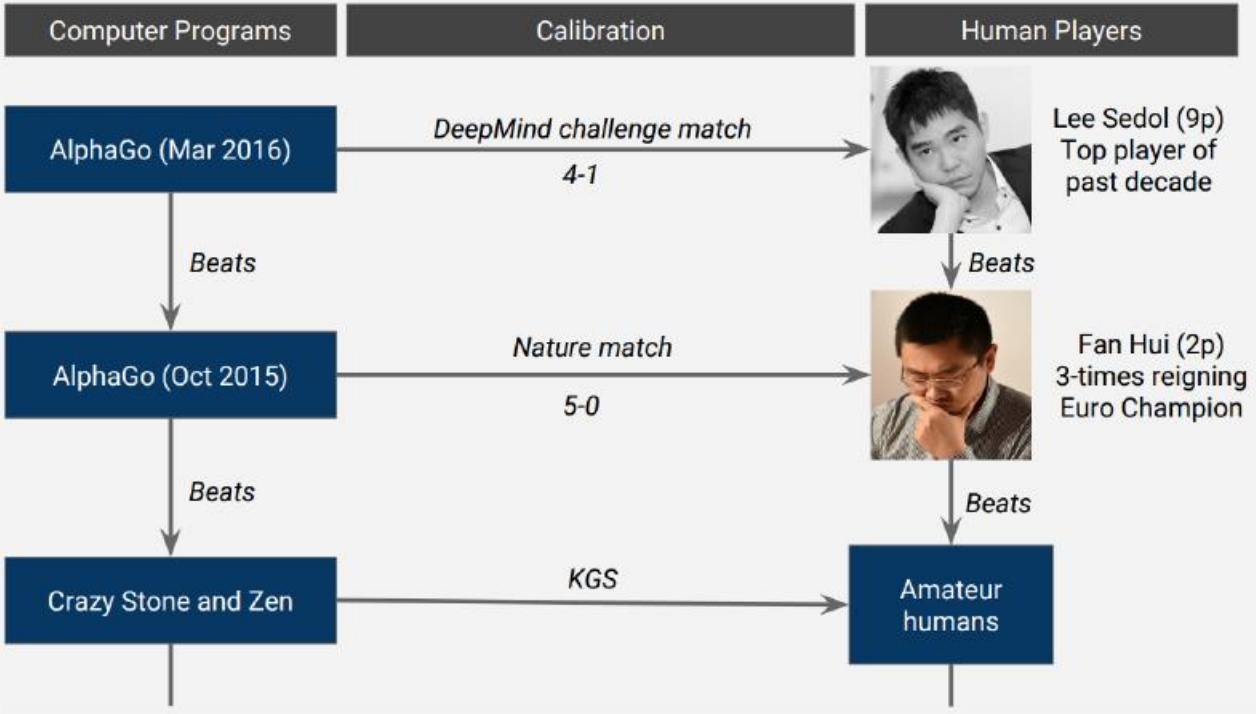




Game size	Board size N	3^N	Percent legal	legal game positions (A094777)[11]
1×1	1	3	33%	1
2×2	4	81	70%	57
3×3	9	19,683	64%	12,675
4×4	16	43,046,721	56%	24,318,165
5×5	25	8.47×10^{11}	49%	4.1×10^{11}
9×9	81	4.4×10^{38}	23.4%	1.039×10^{38}
13×13	169	4.3×10^{80}	8.66%	3.72497923×10^{79}
19×19	361	1.74×10^{172}	1.196%	$2.08168199382\times10^{170}$

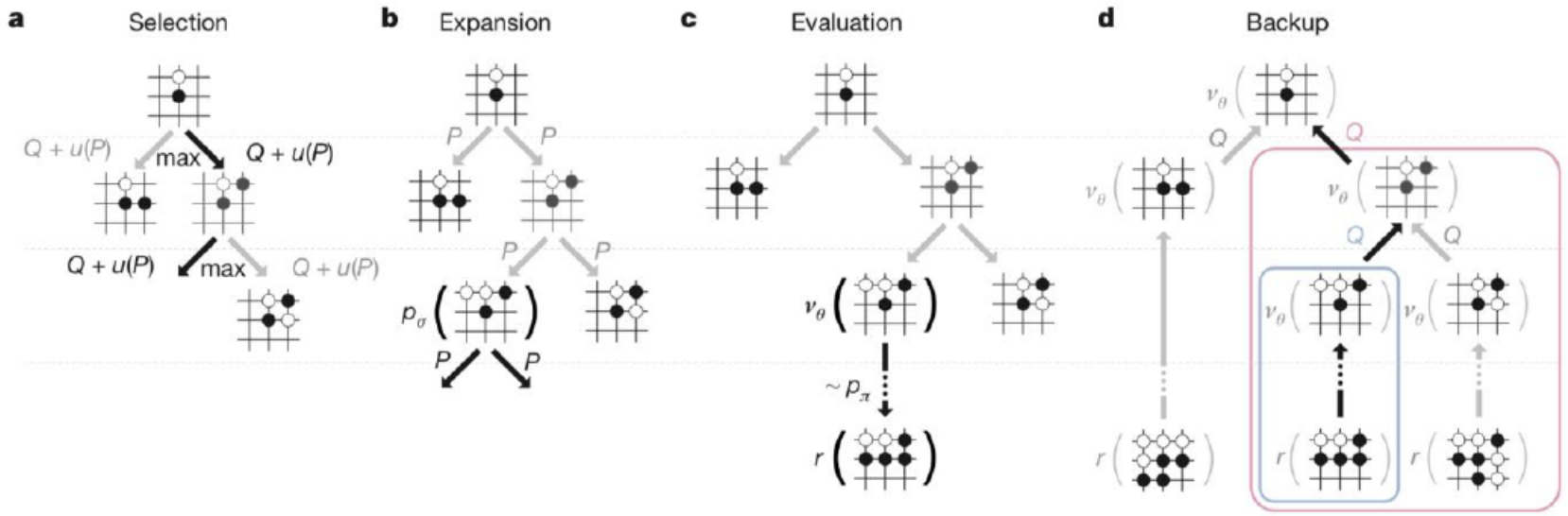
AlphaGo (2016)





AlphaZero

- Same as the best before: Monte Carlo Tree Search (MCTS)
  - Balance exploitation/exploration (going deep on promising positions or exploring new underplayed positions)
- Use a neural network as "intuition" for which positions to expand as part of MCTS (same as AlphaGo)



AlphaZero

- Same as the best before: Monte Carlo Tree Search (MCTS)
  - Balance exploitation/exploration (going deep on promising positions or exploring new underplayed positions)
- Use a neural network as "intuition" for which positions to expand as part of MCTS (same as AlphaGo)
- "Tricks"
  - Use MCTS intelligent look-ahead (instead of human games) to improve value estimates of play options
  - Multi-task learning: "two-headed" network that outputs (1) move probability and (2) probability of winning.
  - Updated architecture: use residual networks

AlphaZero vs Chess, Shogi, Go