Deep Reinforcement Learning in the Real Life

©Dr. Min Chi (& Dr. Markel Sanz Ausin)
mchi@ncsu.edu

AI Academy

Applied Reinforcement Learning

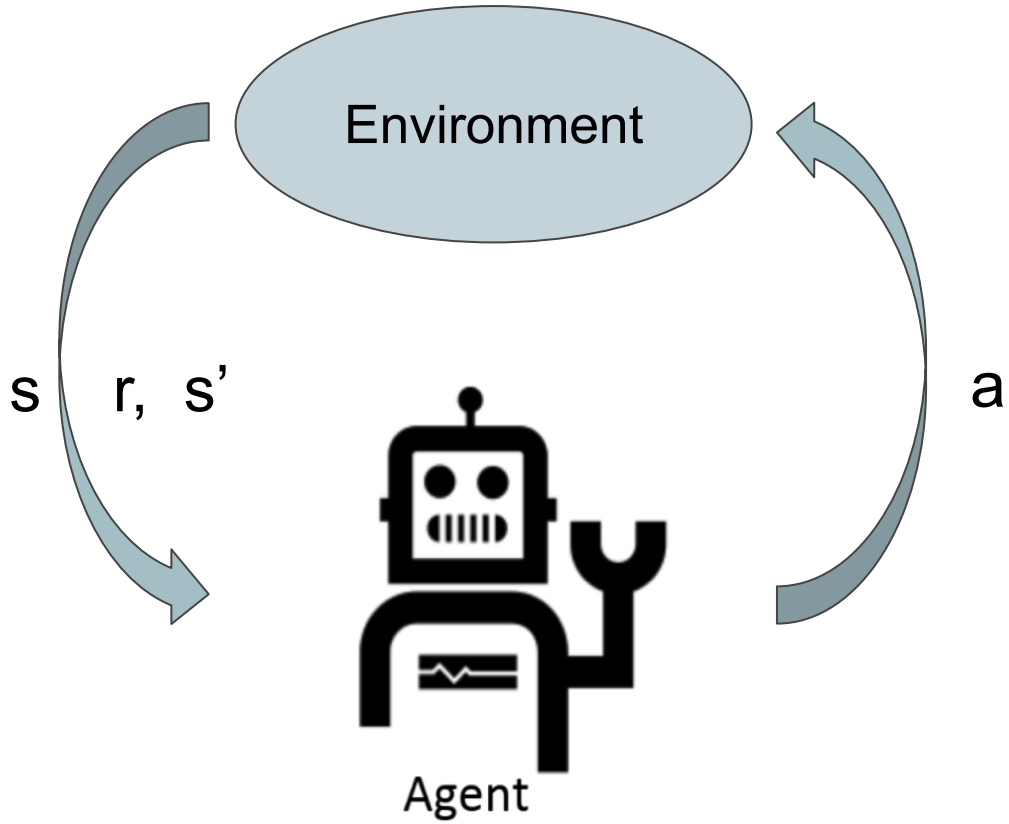
- Build an AI agent that learns only by interacting with the environment.
- The agent must learn to take the optimal action from each state it encounters.
- Examples:
  - Select the best movement in chess.
  - Move a robotic hand in order to grab an object.
  - Select the best medicine for a sick patient.
  - Decide how to teach some content to maximize the student learning.

Applied Reinforcement Learning

- Key concepts in RL:
  - **State (s):** it represent the situation the environment is in.
  - **Action (a):** the decision the agent takes in each state.
  - **Reward (r):** the scalar signal the agent receives, which determines whether the action was good or bad.
  - Transition probability

- Goal: Learn a **policy** $\pi(a \mid s)$ that takes the action ***a*** that maximizes the long term ***reward (or Return)*** from any state ***s***.
- Return $G_t = r_t + r_{t+1} + \ldots + r_T$

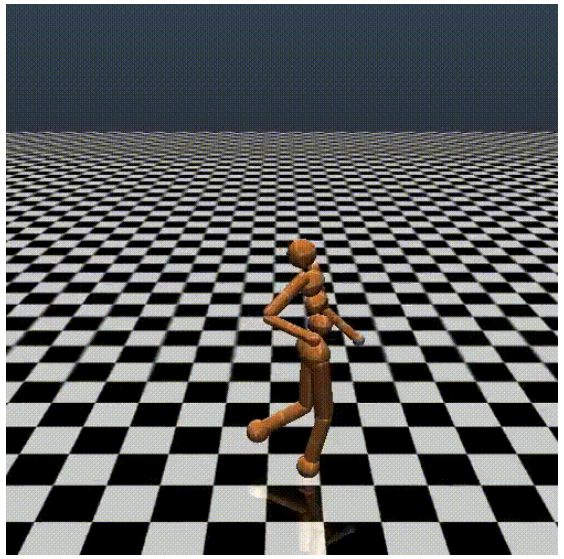
Reinforcement Learning



Examples of Different Environments

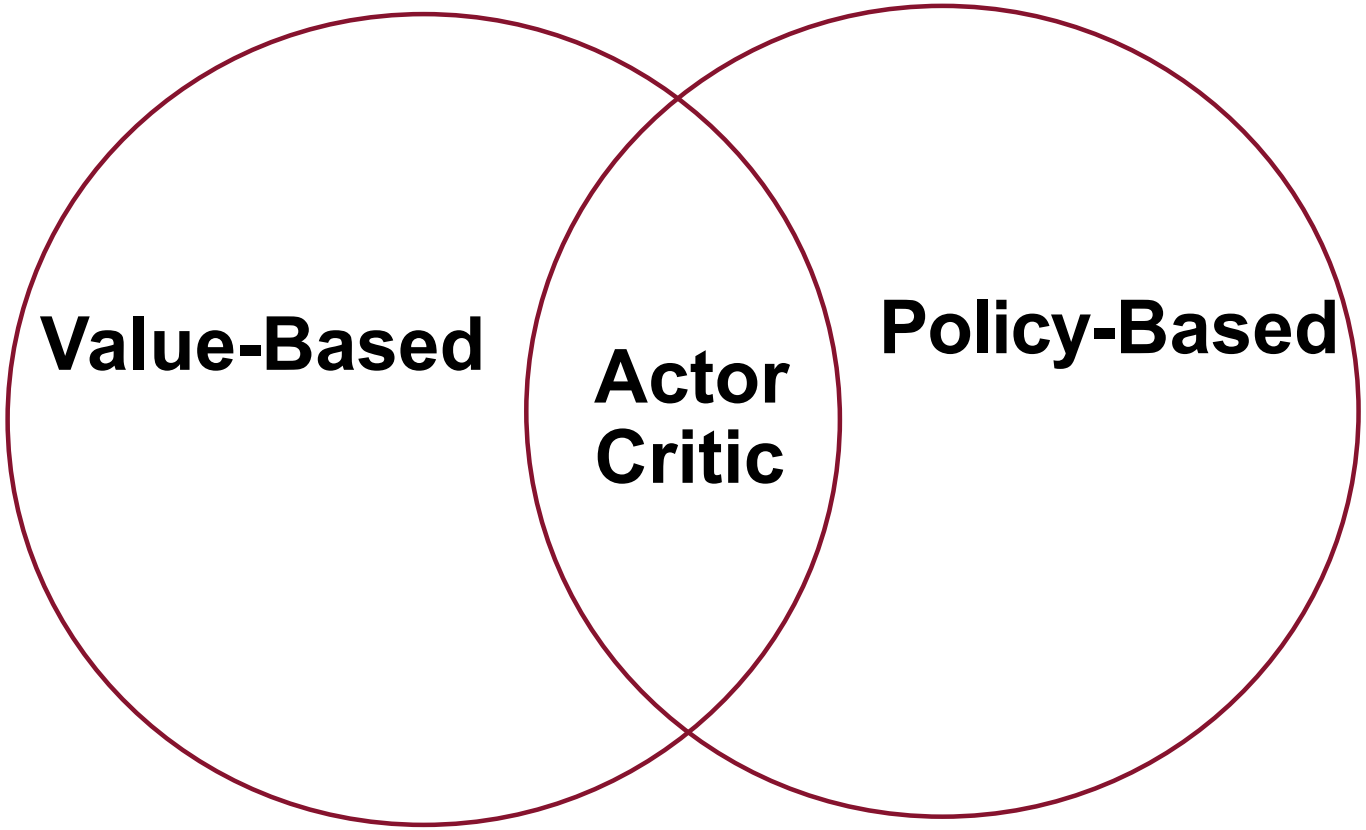
Games

Physics
Simulators



Robotics

Types of RL Algorithms



Types of RL Algorithms

- Value-Based Methods (aka Q-learning Based Methods):
  - They learn a Value Function.
    - State Value Function: V(s)
    - Action Value Function: Q(s,a)
  - The policy is implicit.
    - Greedy policy: always take the action with the highest value.
    - ε-greedy policy: take best action with probability (1-ε) and a random action with probability ε.

Types of RL Algorithms

- Policy-Based Methods (aka Policy Gradient Methods):
  - They don't learn a Value Function.
  - The policy is explicitly defined by some function $\pi(a \mid s)$.
  - They don't learn how much reward we will get from each state. They just learn which action to take in each situation.

Types of RL Algorithms

- Actor-Critic Methods:
  - They learn a Value Function.
  - The policy is explicitly defined by some function $\pi(a \mid s)$.
  - They use both a value function and a function that explicitly learns the policy.
    - Usually, they use the value function in order to better learn the policy function.

Model of the Environment

- Model-Based Algorithms:
  - Build a model of the environment, similar to learning the rules of a game.
    - Deterministic Model: $s_{t+1} = f(s_t, a_t)$
    - Stochastic Model: $S_{t+1} \sim P(\bullet \mid s_t, a_t)$
  - It also learns a reward function: r(s)
  - The model can either be provided or learnt.
  - It allows the agent to plan by thinking ahead, seeing where different actions would lead.
  - Examples: Chess, Go, Checkers, …

Model of the Environment

- Model-Free Algorithms:
  - No access to the environment dynamics.
  - It learns a value-function or a policy function without the need to learn the model dynamics directly.
  - The agent cannot look into the future states and rewards to decide which action to take in each moment.
  - Examples: Robotics, healthcare (difficult to learn the true model)

ON-POLICY VS. OFF-POLICY

- On-Policy Algorithms:
  - Update the policy using the data collected using the same policy.
  - The **behavior policy** and the **target policy** have to be the same.
  - **Data cannot be reused**. We first collect some data, use it to train our policy, and as our policy has changed, we cannot use that data anymore.
  - Policy-based methods are usually on-policy.

ON-POLICY VS. OFF-POLICY

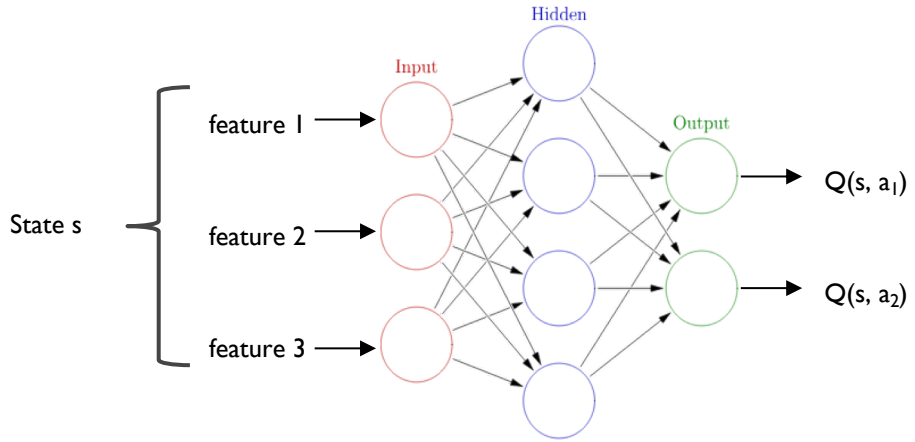
- Off-Policy Algorithms:
  - Update the value-function using any data.
  - The **behavior policy** and the **target policy** don't have to be the same.
  - **Data can be reused**. We can collect some data and use it to learn the value-function of those states.
  - Value-based methods are usually off-policy.

Value Functions

- State Value Function: $V(s)$
- Action Value Function: $Q(s, a)$
- Bellman Equations (without transition probabilities):
  - $V(s_t) = E\ [\ r(s_t, a_t) + \gamma * V(s_{t+1})\ ]$
  - $Q(s_t, a_t) = E\ [\ r(s_t, a_t) + \gamma * E[Q(s_{t+a}, a_{t+1})]\ ]$
- Bellman Optimality Equations:
  - $V^*(s_t) = \max_a E\ [\ r(s_t, a_t) + \gamma V^*(s_{t+1})\ ]$
  - $Q^*(s_t, a_t) = E\ [\ r(s_t, a_t) + \gamma \max_{at+1} Q^*(s_{t+1}, a_{t+1})\ ]$

Function Parametrization

- When the state and/or action spaces are too large, tabular methods are not enough, and we need to use function approximations. For example, a linear function, a gaussian function, or a neural network.



Deep Reinforcement Learning Algorithms

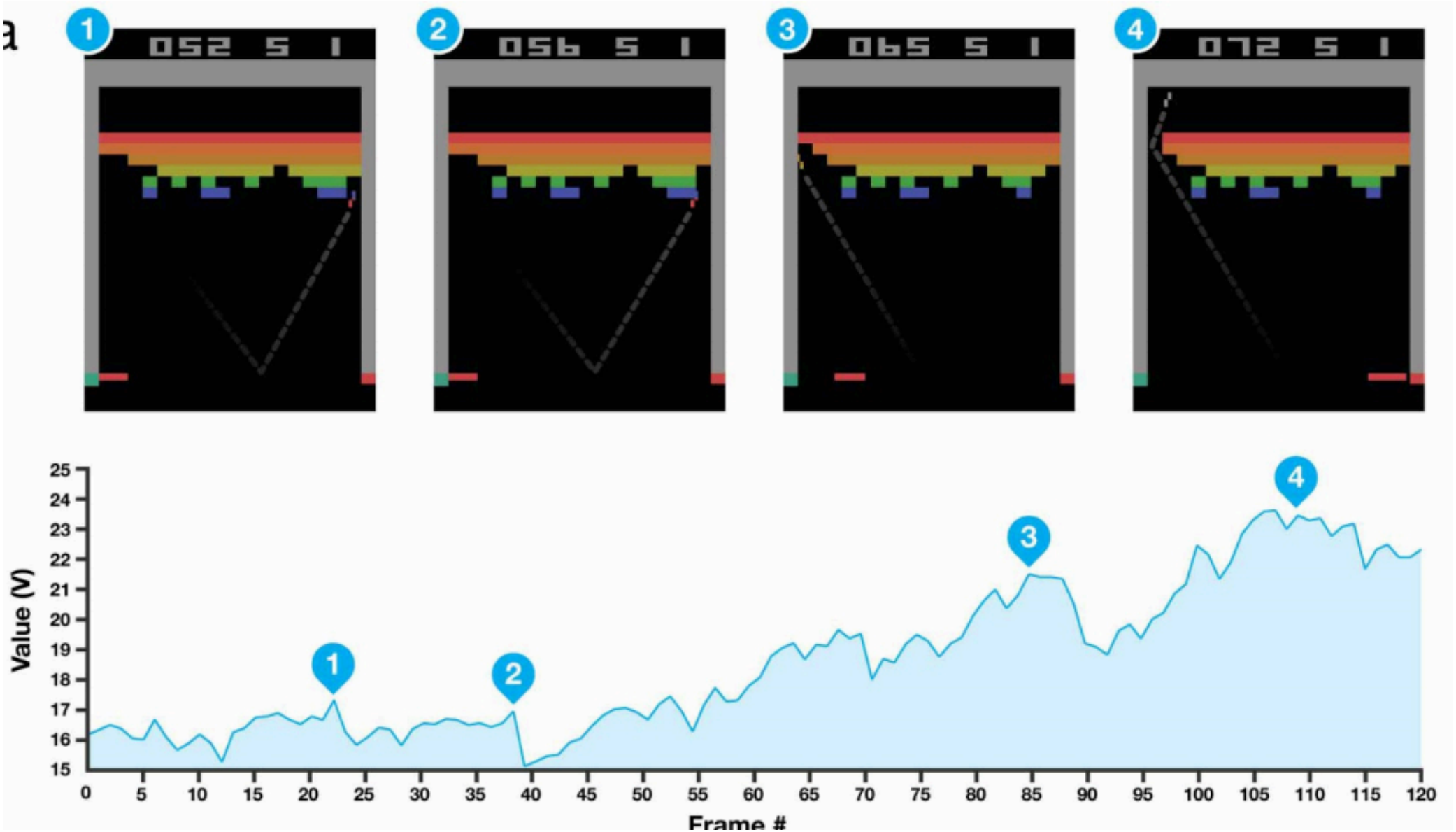
Deep Reinforcement Learning Algorithms

- Q-Learning Based Algorithms:
  - DQN
  - Dueling-DQN
  - HER
- Policy Optimization Algorithms:
  - Vanilla Policy Gradient
  - A2C/A3C
  - TRPO
  - PPO

DEEP Q-NETWORK (DQN)

- Bellman Equation for DQN: $Q(s, a; \theta) = r + \gamma \max_{a'} Q(s', a', \theta')$
- Main NN: θ and Target NN: θ'
- The main NN gets trained in each iteration, but the target network is frozen for ~10000 iterations and then the weights are copied from the main to the target network.
- Experience Replay Buffer: It is used to store the last 1M (s, a, r, s') steps. In each training iteration, 32 steps are samples uniformly and a gradient descent step is performed to train the main NN.

Mnih et al. 2015. Human-level control through deep reinforcement learning.


a
1
052 5 1
2
056 5 1
3
065 5 1
4
072 5 1
Value (V)
25
24
23
22
21
20
19
18
17
16
15
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
105
110
115
120
Frame #

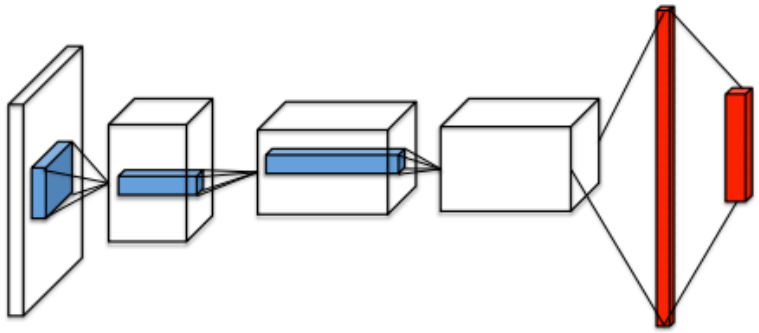
DOUBLE-DQN

- Combines Double Q-Learning with DQN.
- Decouples action selection from action evaluation.
- This trick reduces Q-value overestimations.
- Modified Bellman Equation: $Q(s, a; \theta) = r + \gamma\, Q(s', \text{argmax}_{a'} Q(s', a'; \theta), \theta')$
- It generally results in better final policies.

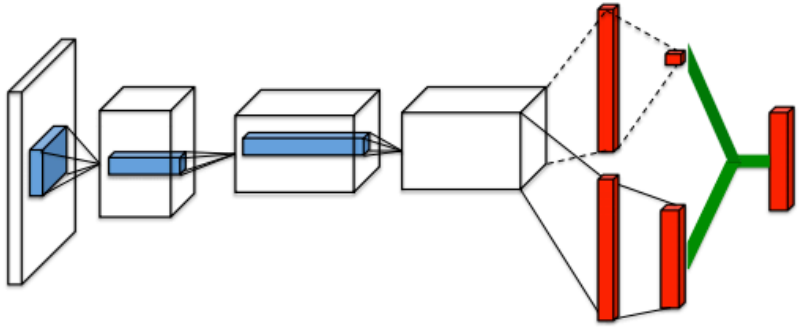
Van Hasselt et al. 2015. Deep Reinforcement Learning with Double Q-learning

DUELING-DQN

- Splits Q-values into a Value function and an Advantage Function:
  - $Q(s, a) = V(s) + A(s, a)$
- This is done in the last layer of the NN.

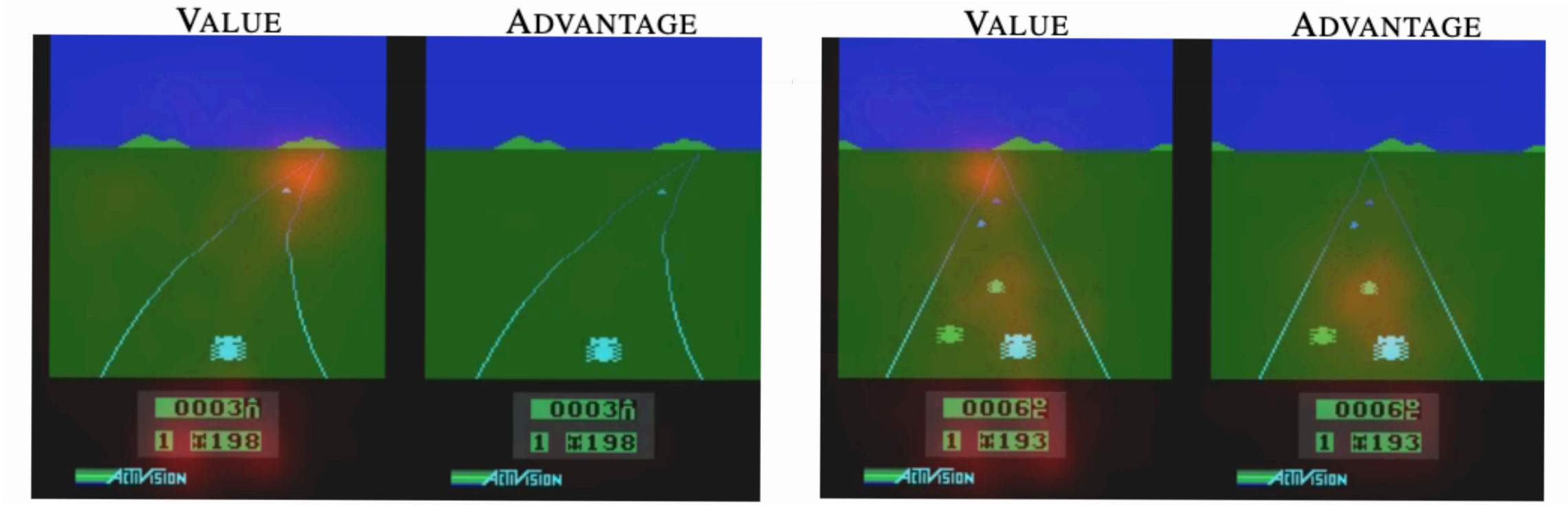


DQN



Dueling-DQN

Wang et al. 2015. Dueling network architectures for deep reinforcement learning


VALUE
ADVANTAGE
VALUE
ADVANTAGE
0003
1 198
0003
1 198
0006
1 193
0006
1 193

HINDSIGHT EXPERIENCE REPLAY (HER)

- Useful for certain environments where the goal state is known.
- It concatenates the current state s with the goal state g and pass it as input to the NN.
- Not only stores the real goal, but also other fictitious goals.
- Example: Bit Flipping Environment.
  - State: (0, 0, 1). Goal: (1, 1, 1). Action: Flip the middle bit. Reward: -1 (goal not reached).
  - State: (0, 0, 1). Create a new fake goal: (0, 1, 1). If this was the goal, reward would be +1.
- https://www.youtube.com/watch?v=Dz_HuzgMxzo

Andrychowicz et al. 2017. Hindsight Experience Replay.

Policy Optimization Algorithms

- Advantages of Policy-Based algorithms:
  - They can represent continuous actions. Better for stochastic and/or high-dimensional action-spaces.
  - They converge faster.
  - They optimize the policy function directly.
- Disadvantages of Policy-Based algorithms:
  - They often converge to a local optima, not a global optima.
  - Sample inefficiency. As they learn on-policy, data can't be reused.
  - Higher variance than value-based algorithms.

Policy Gradient Theorem (For Stochastic Policies)

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta}[R(\tau)] = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta}[\sum_{t=0}^{T} r(s_t, a_t)] = \int \pi_\theta(\tau) R(\tau) d\tau$$

Gradient of the return:

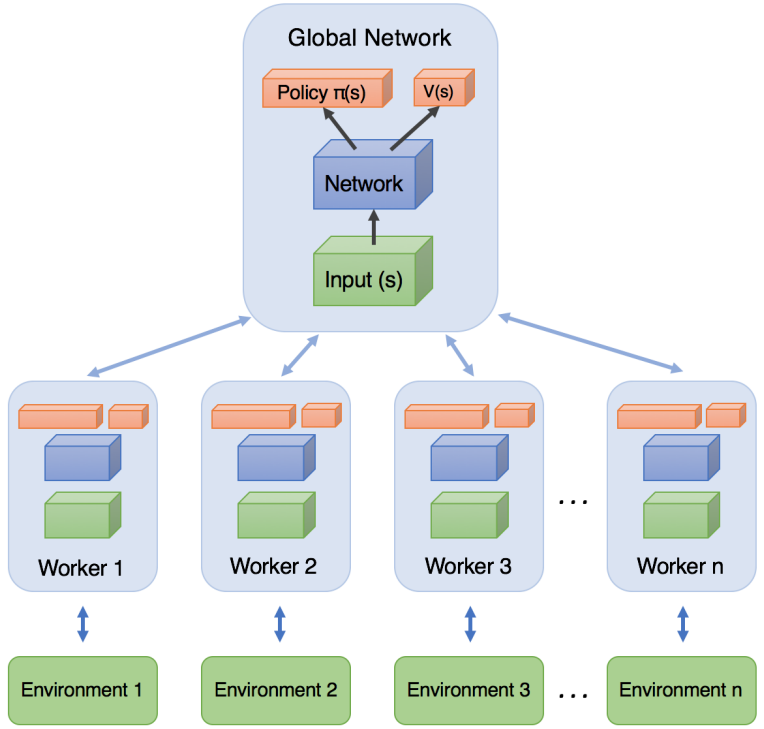
$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \int \pi_\theta(\tau) R(\tau) d\tau = \int \nabla_\theta \pi_\theta(\tau) R(\tau) d\tau =$$

$$\int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) R(\tau) d\tau = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(\tau) R(\tau)] =$$

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)]$$

Asynchronous Advantage Actor-Critic (A3C)

- On-policy algorithm.
- Multiple workers interact with their own copy of the environment.
- They compute the gradients and send them to the global network.
- The global network weights are copied into the worker networks.



Mnih et al. 2016. Asynchronous Methods for Deep Reinforcement Learning.

Trust Region Policy Optimization (TRPO)

- Optimizes the policy ensuring that the gradient ascent step is not going to change the policy function too much.
- Makes it more robust to different learning rates and it helps ensure convergence.

Line search
(like gradient ascent)

Trust region

Schulman et al. 2017. Trust Region Policy Optimization.

Deep Deterministic Policy Gradient (DDPG)

- We can also run gradient ascent when using a deterministic policy: a = μ(s)
- Off-Policy algorithm!
- Critic Update Q(s, a):

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))\right]$$

- Actor Update μ(s):

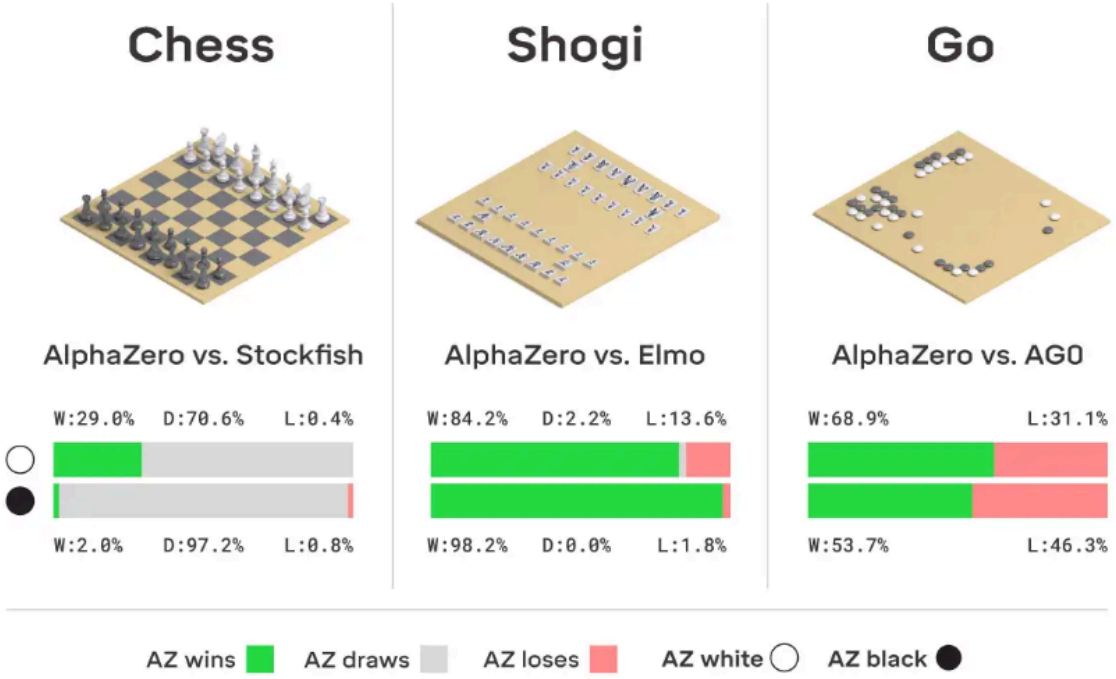
$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_{i} \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i}$$

Lillicrap et al. 2015. Continuous Control With Deep Reinforcement Learning.

Deep Reinforcement Learning Applications

Deep Reinforcement Learning Applications

- Games of Go/Chess/Shogi: AlphaZero
- Monte-Carlo Tree Search (MCTS)
- Uses a NN to estimate the value of each board position.
- This reduces the search space in the tree search algorithm.



Silver et al. 2018. A general reinforcement learning algorithm that masters chess, shogi and Go through self-play

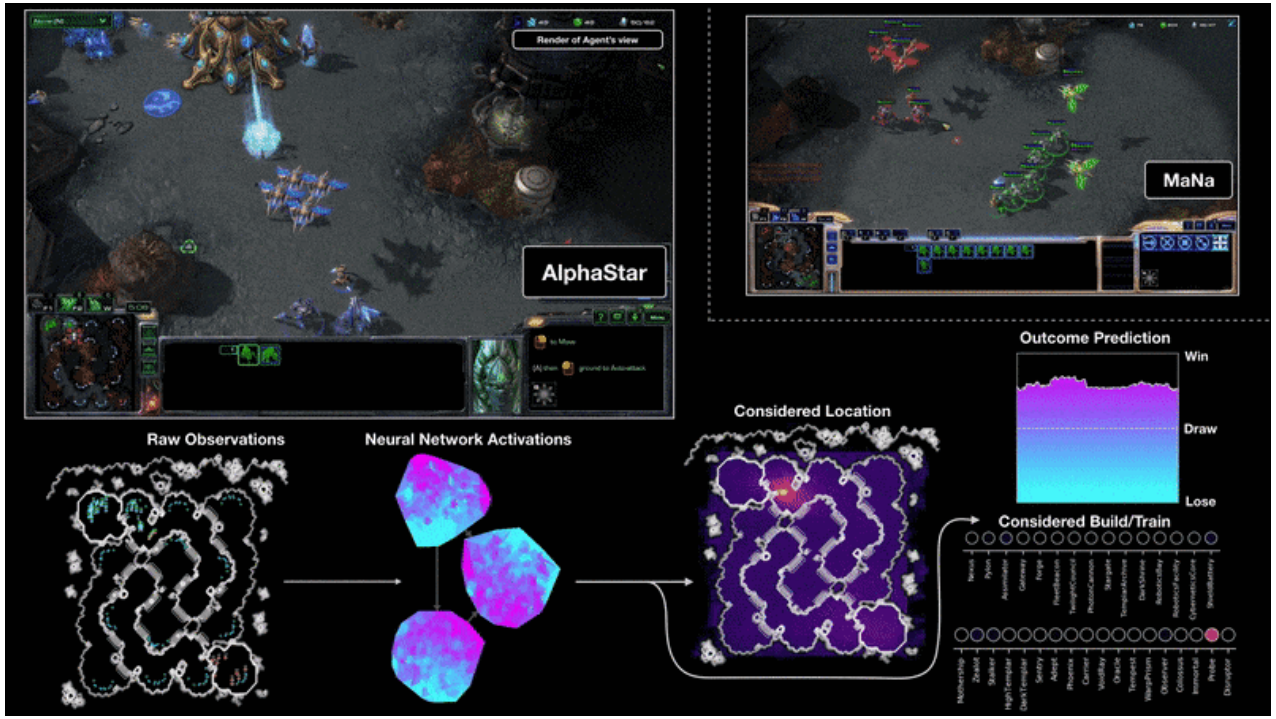
Deep Reinforcement Learning Applications

- Robotic Hand Dexterity
- Combines a traditional algorithm to solve the Rubik's cube with Deep RL to control the robotic hand (PPO).
- Overcomes the **Reality Gap** with domain randomization.
- https://www.youtube.com/watch?v=kVmp0uGtShk&feature=youtu.be

Akkaya et al. 2019. Solving Rubik's Cube with a Robot Hand

Deep Reinforcement Learning Applications

- StartCraft II: AlphaStar
- NN receives raw game observations as input to the policy network.
- LSTM, auto-regressive policy, recurrent pointer network.
- Initially trained using supervised learning using human player data.
- Off-policy Actor-Critic algorithm.



Vinyals et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning

Challenges In Deep Reinforcement Learning

- Sample Inefficiency
- Sparse Rewards
- Long-Term Credit Assignment Problem
- Batch (Offline) Reinforcement Learning
- Reality Gap

Other Ideas In RL

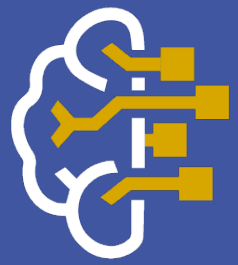
- Learning from human demonstrations
- Learning from a video
- Neural Architecture Search / Meta Learning
- Safe Reinforcement Learning
- Hierarchical Reinforcement Learning
- Transfer Learning

Stay Connected

Dr. Min Chi

Associate Professor
mchi@ncsu.edu
(919) 515-7825

AI Academy



AI Academy

go.ncsu.edu/aiacademy

NC STATE UNIVERSITY