Introduction to Reinforcement Learning

Policy networks and Actor-Critic Algorithms

Trimbach Ekaterina

Agenda and Expectations

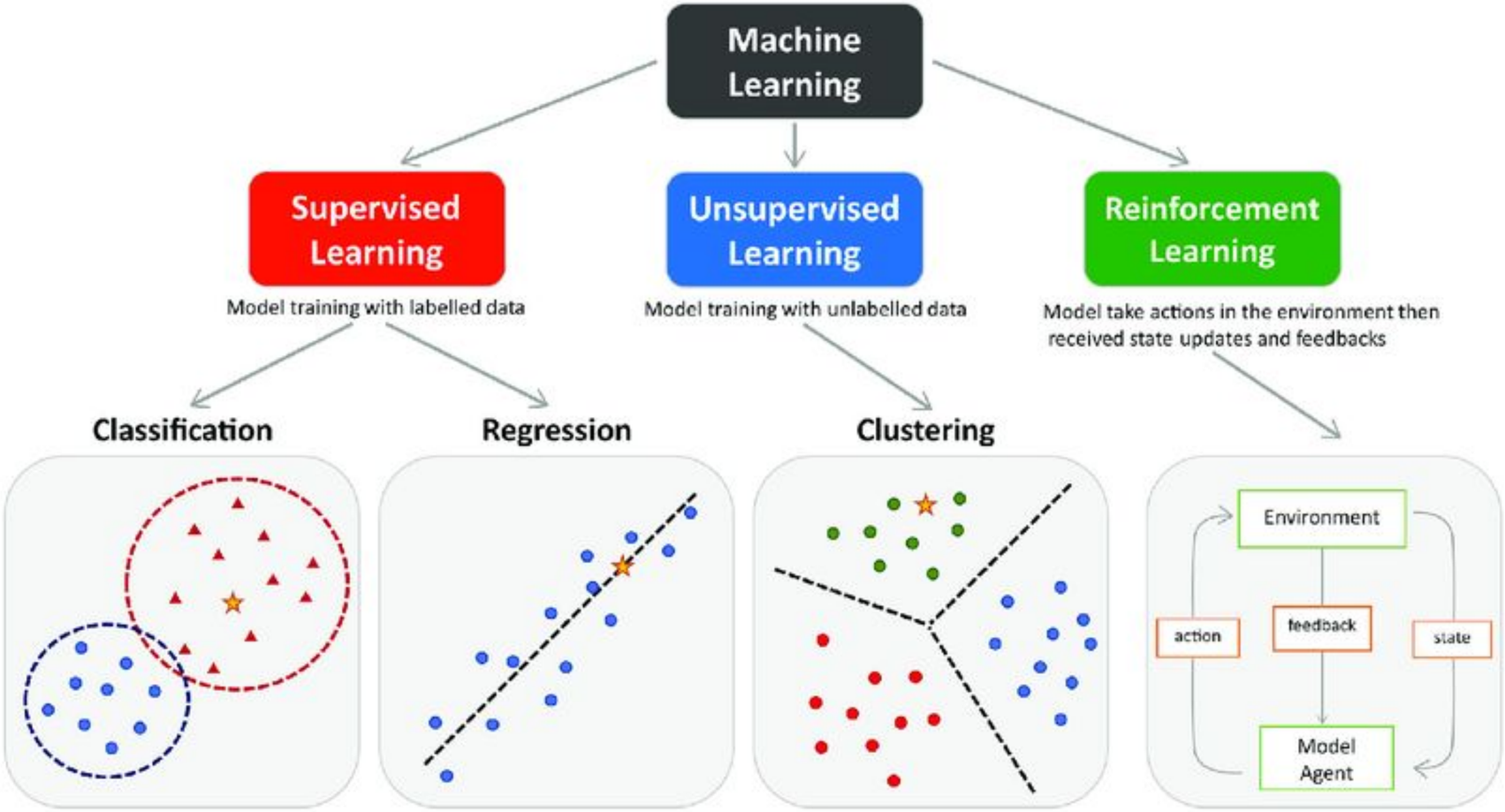
Agenda :

- What is Reinforcement Learning?
- What do we need for it?
- Theory :
  - Markov Decision process
  - Value-based RL
  - Policy based RL
  - Actor-Critic algorithms
- Practice

Expectations:

- Understanding where RL can be used
- Understanding basic ideas of RL
- Become curious and willing to continue explore this topic.

Types of Machine Learning tasks

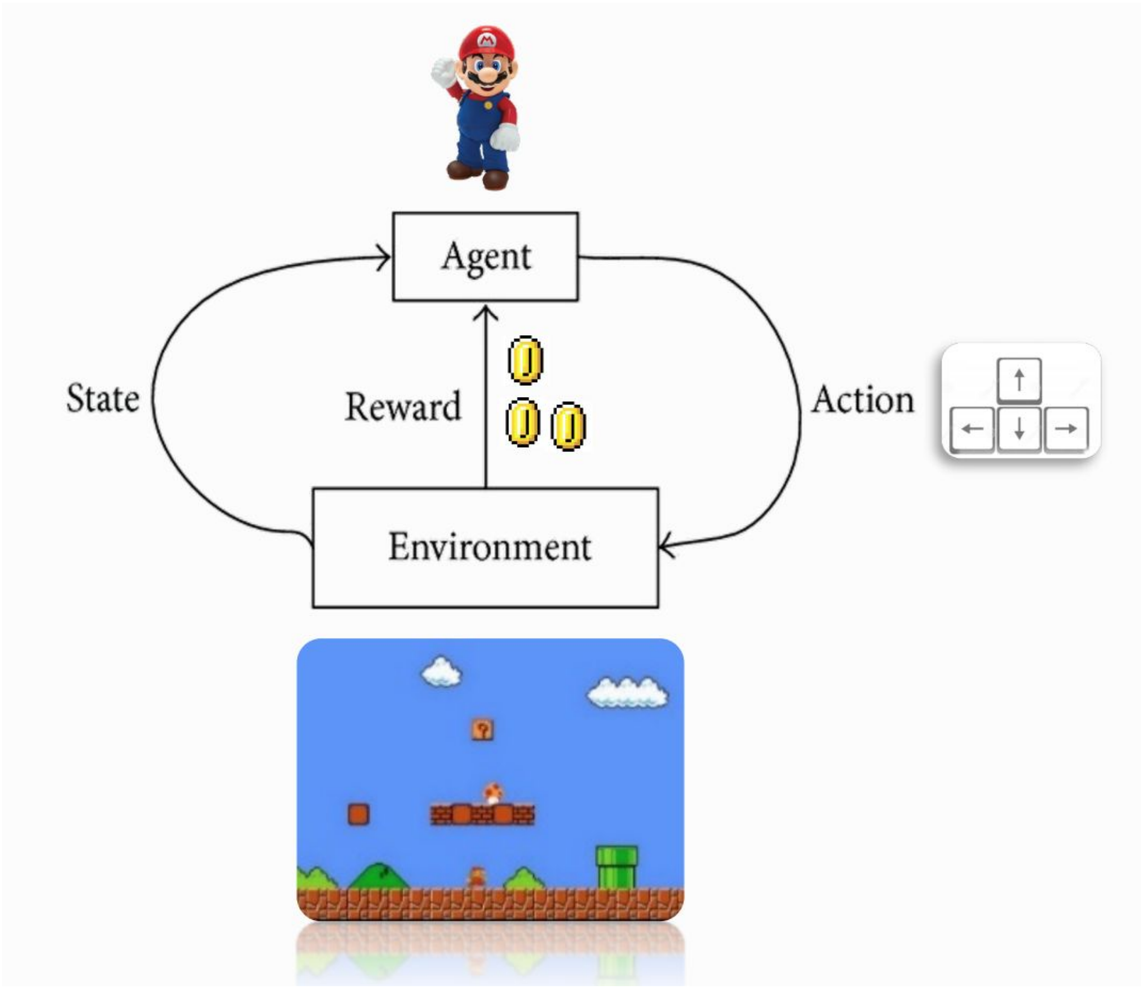


source :
https://www.researchgate.net/figure/The-main-types-of-machine-learning-Main-approaches-include-classification-and_fig1_354960266

What is reinforcement learning?

To work with Reinforcement Learning
we need to have :

1) Agent
2) Environment



An agent learns to act optimally by
interacting with the uncertain environment

source : https://odi.inf.ethz.ch/files/zinal/Lecture-1-RL-introduction.pdf

History of Reinforcement Learning

2015

Atari Game from DeepMind:
- high-dimensional input space

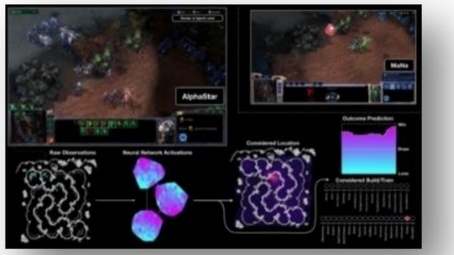
2017

AlphaGo/ AlphaGo Zero/ Alpha Zero:
- self learning & large-scale training

2018

Dota 2 w/ OpenAI Five

2019



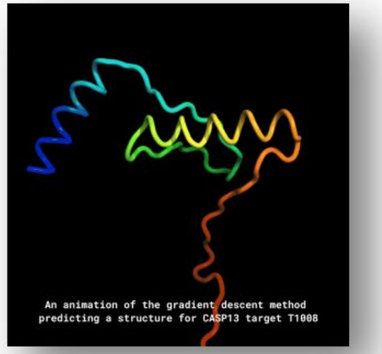
StarCraft II w/Alpha Star

- real-time strategy

2019

Rubik's Cube from OpenAI:
- simulation to real robotic arms

2020



AlphaFold:
- 3D Protein Structures

source : https://odi.inf.ethz.ch/files/zinal/Lecture-1-RL-introduction.pdf

Where can it be used?

source: unite.ai

source: therobotreport.com

source: ieor8100.github.io

source: tradingmarket

source : https://odi.inf.ethz.ch/files/zinal/Lecture-1-RL-introduction.pdf

Quizz

https://red-field-0a6ddfd03.1.azurestaticapps.net/quiz/122

To train RL model, we need

- Simulation environment
- Labeled dataset
- Unlabeled dataset

What is a good example of Reinforcement learning:

- Zero-shot image classification
- Zero-shot text classification
- Learning to play chess

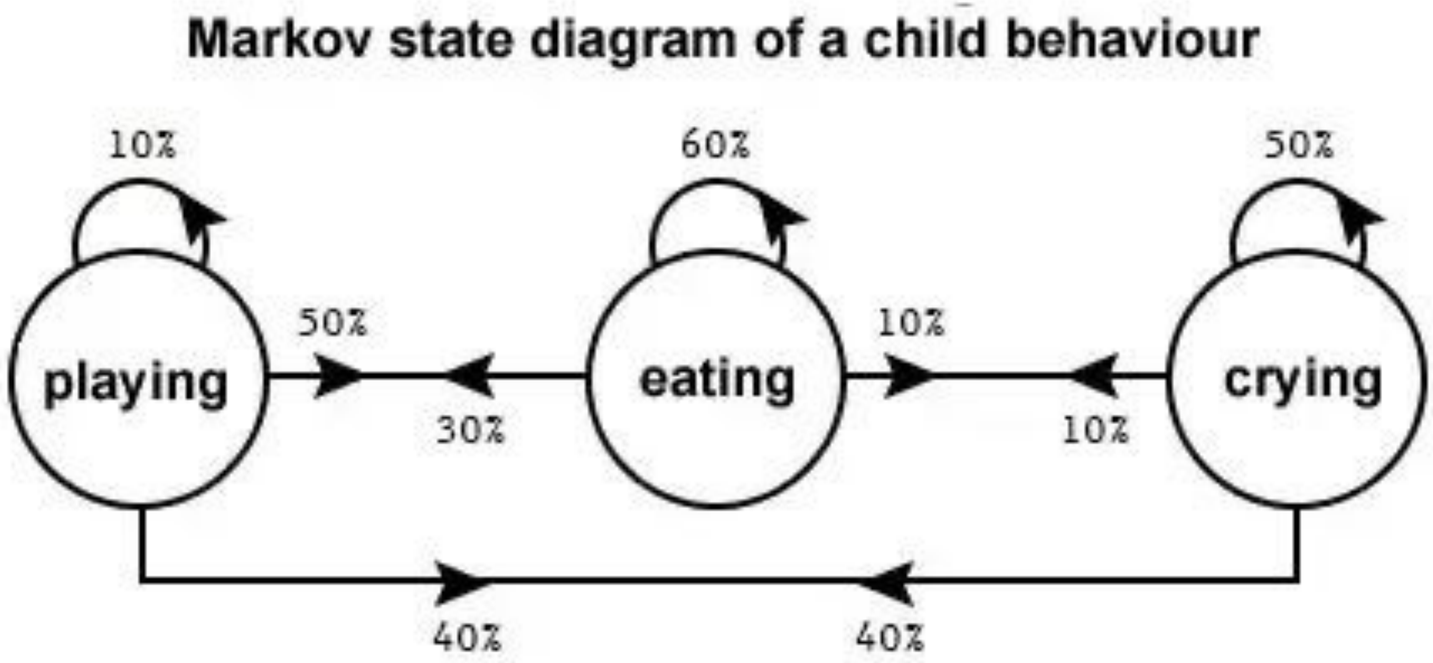
When creating RL-based chess engine, we need to

- use all existing chess matches as a dataset
- let computer play against itself many times
- program exhaustive search algorithm

Theory time

Markov chain

A Markov chain is a model that describes a system's progression through different states, where the likelihood of moving to the next state depends *only on the current state*. It follows the *memoryless principle*, meaning future states are determined solely by the present state.

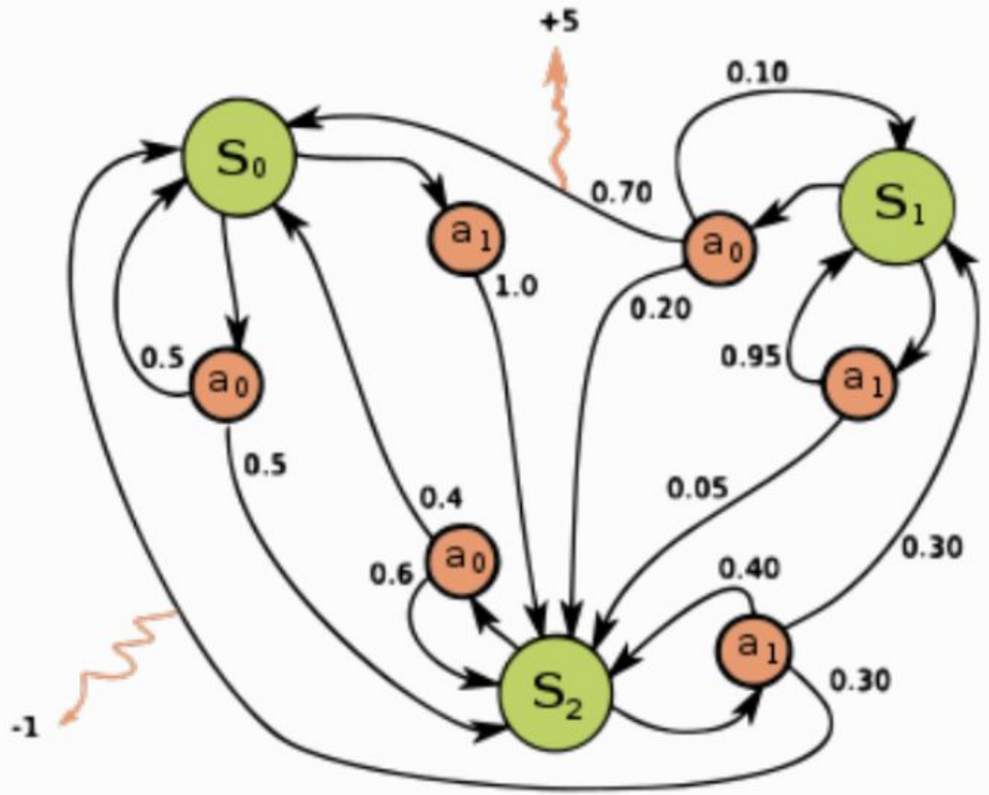


picture : https://www.quora.com/What-is-a-state-of-Markov-chain

Markov Decision process

Example : **Blackjack**.

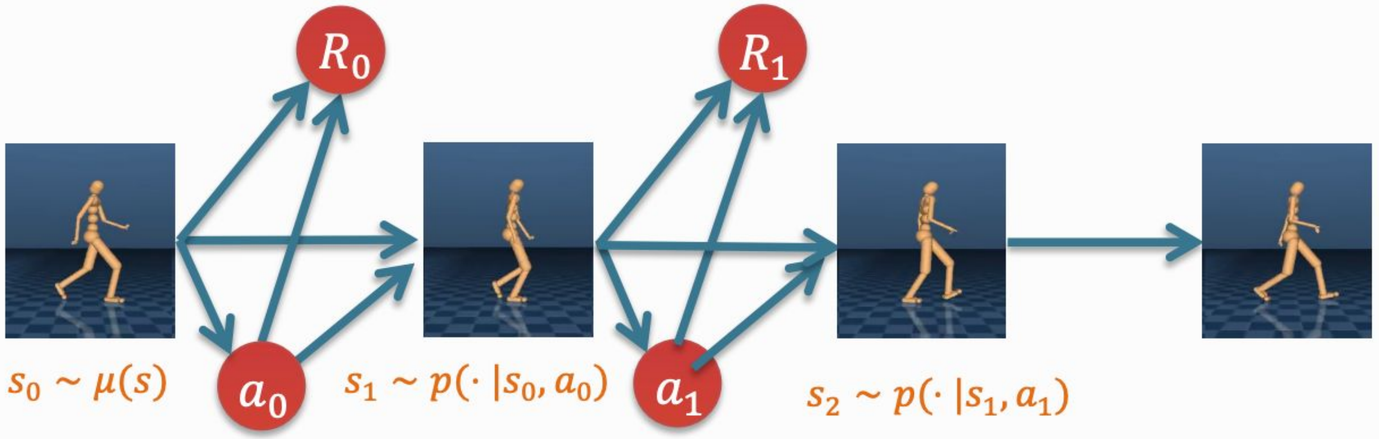
In Blackjack, players aim to achieve a hand value as close to 21 as possible without exceeding it. The **states** represent different combinations of cards the player and the dealer hold, th **actions** are decisions such as "hit" (tak another card) or "stand" (keep the current hand), and the rewards are associated with winning, losing, or pushing (a tie).



source : https://odi.inf.ethz.ch/files/zinal/Lecture-1-RL-introduction.pdf

Markov Decision Processes

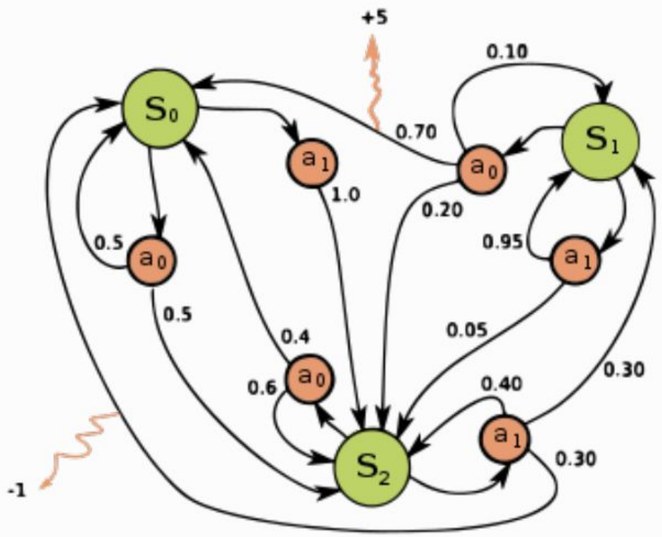
- **MDP** $(S, \mathcal{A}, P, R, \mu, \gamma)$
  - State (S): a (finite) set of states
  - Action $(\mathcal{A})$: a (finite) set of actions
  - Probability transition matrix $(P_{ss'}^{a})$: $P(s_{t+1} = s' | s_t = s, a_t = a)$
  - Reward function (R): the immediate reward $R(s, a)$
  - Initial distribution (μ): initial state $s_0 \sim \mu(\cdot)$
  - Discount factor (γ): $\gamma \in [0,1]$



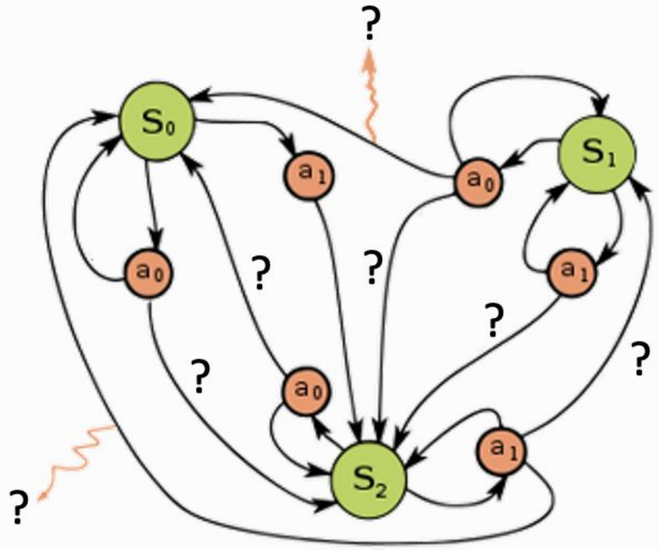
source : https://odi.inf.ethz.ch/files/zinal/Lecture-1-RL-introduction.pdf

Planning vs. Reinforcement Learning

- **Planning**
  - Fully observable state and reward
  - Known reward distribution and transition probabilities

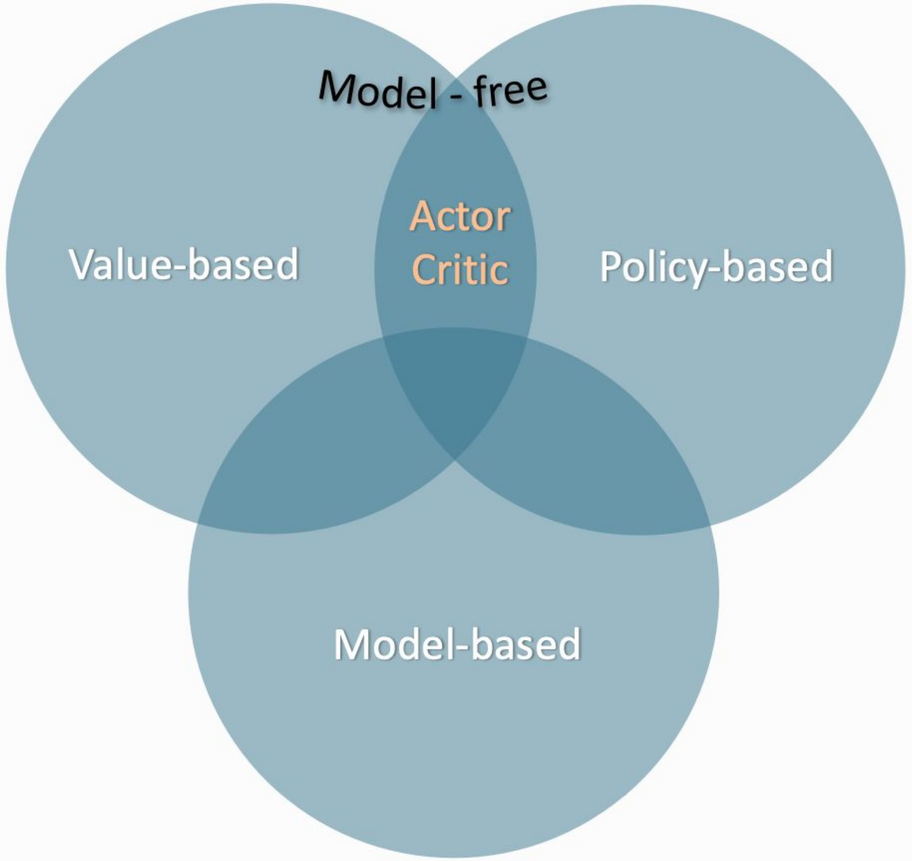


- **Reinforcement Learning**
  - Observable trajectories
  - Unknown reward distribution
  - Unknown transition probabilities



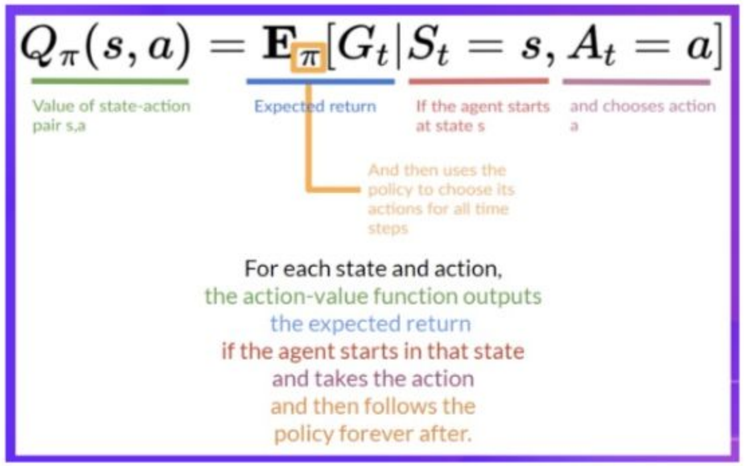
source : https://odi.inf.ethz.ch/files/zinal/Lecture-1-RL-introduction.pdf

Overview of Reinforcement Learning Approaches



- **Value-based RL**
  - Estimate the optimal value function $Q^*(s, a)$
  - Example: Q-learning
- **Policy-based RL**
  - Search directly the optimal policy $\pi^*(\cdot\,|s)$
  - Example: Policy Gradient Method
- **Model-based RL**
  - First estimate the model P, R and then do planning
  - Example: Dyna-Q algorithm

Action-Value function



- The value function of a state s under a policy π is the **expected return** when **starting in s and following π thereafter**.

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

- Idem for the value function of a state-action pair

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

Key components

- **Policy (π):** the agent's behavior strategy
  - Deterministic policy: π: $S \to \mathcal{A}$, $a = \pi(\cdot|s)$
  - Stochastic policy: π: $S \to \Delta(\mathcal{A})$, $a \sim \pi(\cdot|s)$

- **Value function**: how good is each state and/or action
  - Infinite horizon

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_t \sim \pi(s_t), s_{t+1} \sim P(\cdot|s_t, a_t)\right]$$

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | (s_0, a_0) = (s, a), a_t \sim \pi(s_t), s_{t+1} \sim P(\cdot|s_t, a_t)\right]$$

  - Finite horizon / average reward

- **Model:** representation of the environment
  - Transition dynamics $P_{ss'}^{a} = P(s_{t+1} = s' | s_t = s, a_t = a)$ and reward $R(s, a)$

source : https://odi.inf.ethz.ch/files/zinal/Lecture-1-RL-introduction.pdf

Value-based RL

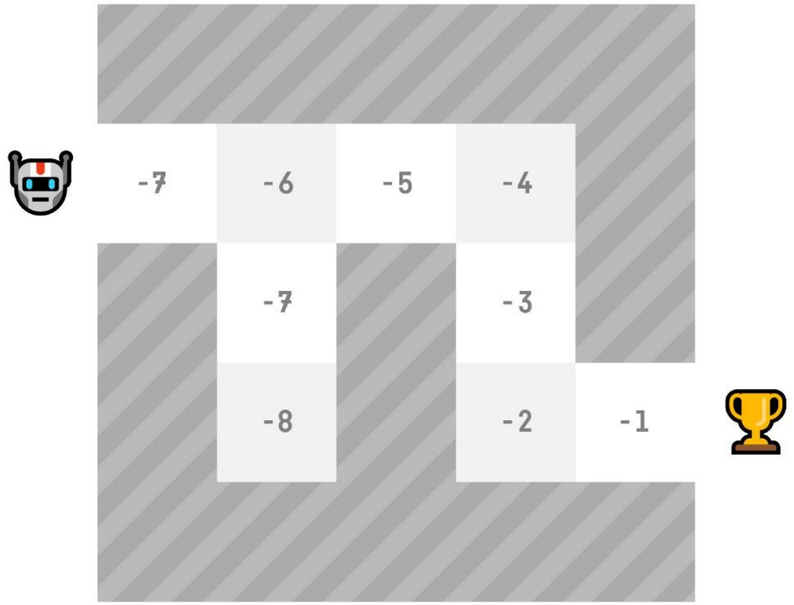
$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right]$$

Value function

Expected discounted return

Starting at state s

Here we see that our value function defined value for each possible state.



source : https://www.freecodecamp.org/news/an-introduction-to-reinforcement-learning-4339519de419/

Discounted return

$$R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots$$

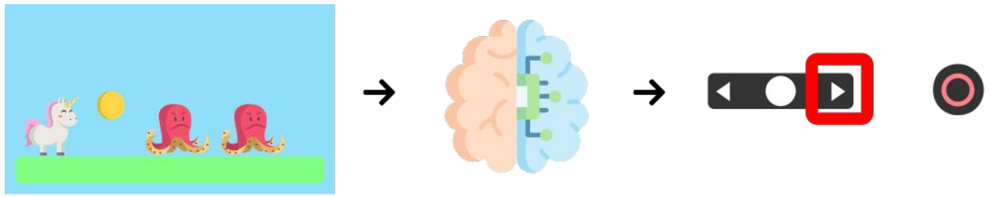
Return: cumulative reward

Gamma: discount rate

Trajectory (read Tau)
Sequence of states and actions

$$R(\tau) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

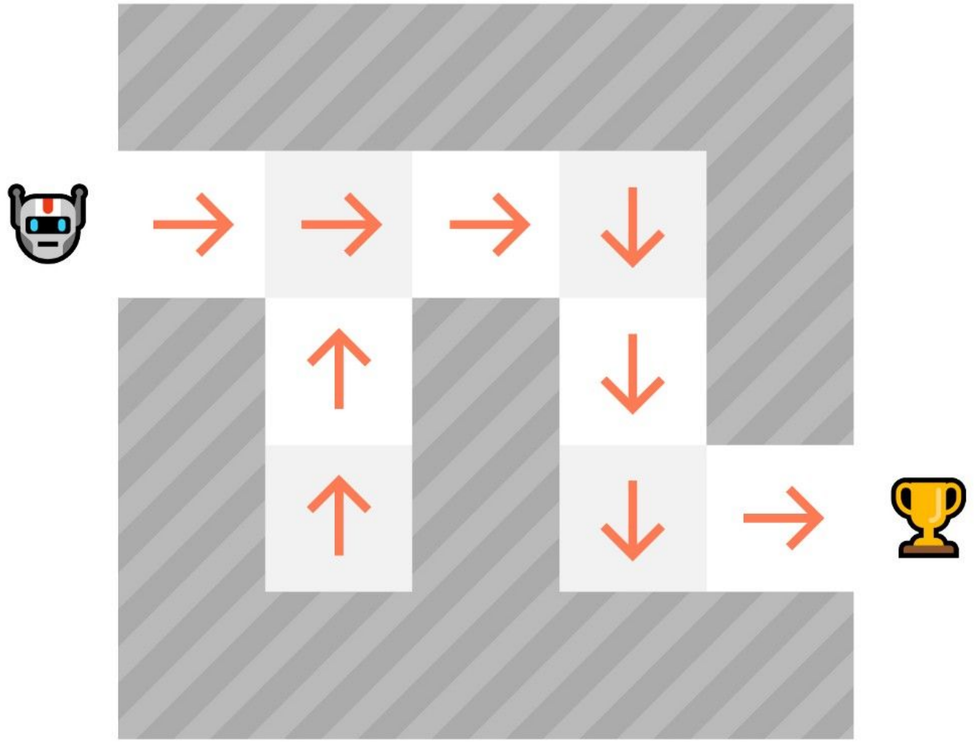
Policy-based RL



State s_0 → $\pi(s_0)$ → a_0 = Right

$$\pi(a|s) = \boxed{P[A|s]}$$

Probability Distribution over the set of actions given the state



source : https://huggingface.co/blog/deep-rl-intro

Value vs Policy

Value-based RL	Policy-based RL
Agent uses a **value function** to estimate the expected future reward for each state-action pair	Agent find the policy parameters that result in the best possible expected cumulative reward based on **policy gradient**
Sample inefficient for continuous state and action spaces	**Sample efficient** for continuous state and action spaces
Convergence is slower	Relatively faster convergence
Explicit exploration	Innately explores
Typically use a **deterministic policy** representing a mapping of states to actions	Have **stochastic policy** representing a probability distribution over actions

Bellman Equation

Policy Evaluation

Bellman Expectation Equation

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s\right]$$
$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a\right]$$

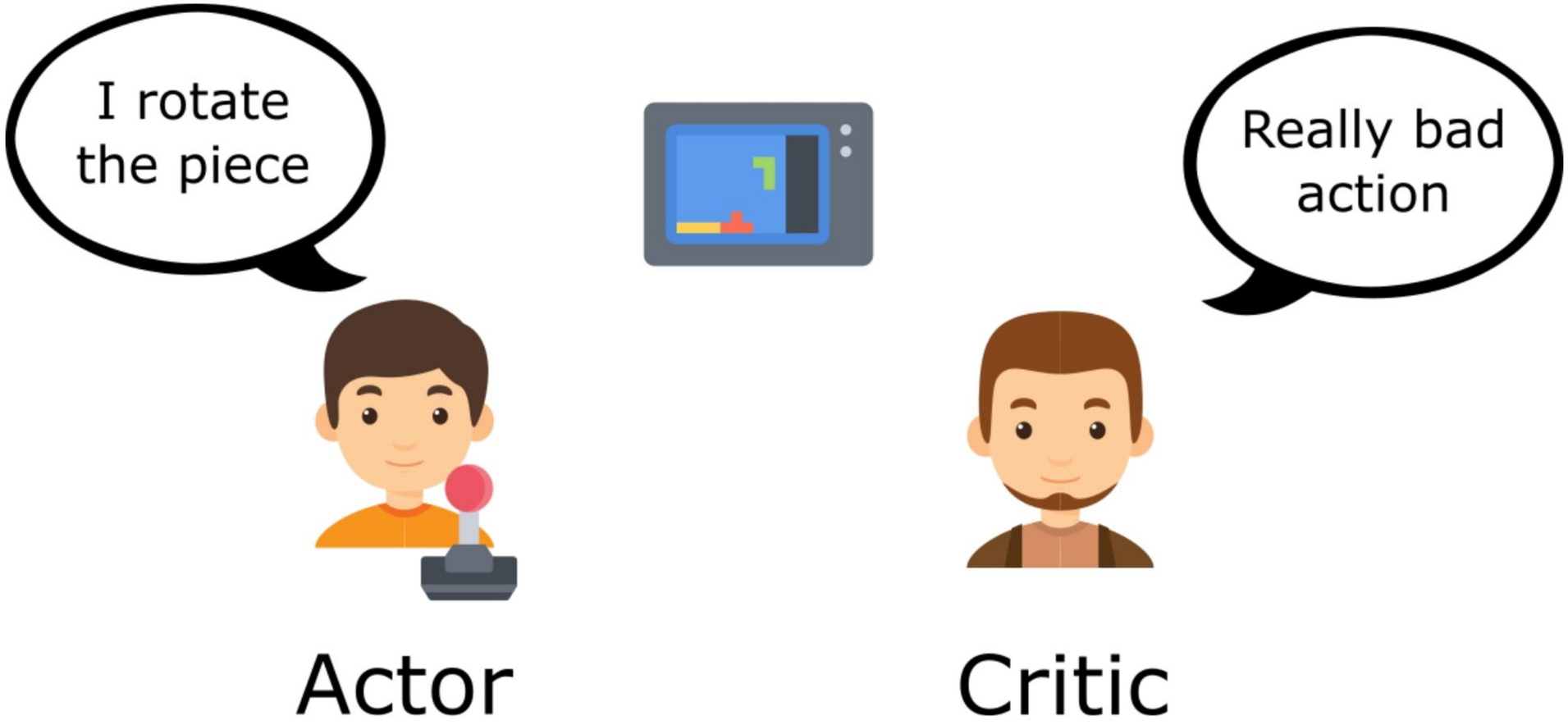
$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(s)}\left[R(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} V^{\pi}(s')\right]$$
$$Q^{\pi}(s,a) = R(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)}\left[\mathbb{E}_{a'\sim\pi(s')} Q^{\pi}(s',a')\right]$$

Proof :

$$Q^{\pi}(s,a) = R(s,a) + \mathbb{E}_{\pi}\left[\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a\right]$$
$$= R(s,a) + \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s') \cdot \gamma \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_{t+1}, a_{t+1}) | s_1 = s', a_1 = a'\right]$$
$$= R(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s') Q^{\pi}(s',a')$$

Matrix form: $Q^{\pi} = R + \gamma P^{\pi} Q^{\pi} \iff Q^{\pi} = (I - \gamma P^{\pi})^{-1} R$

Actor-Critic



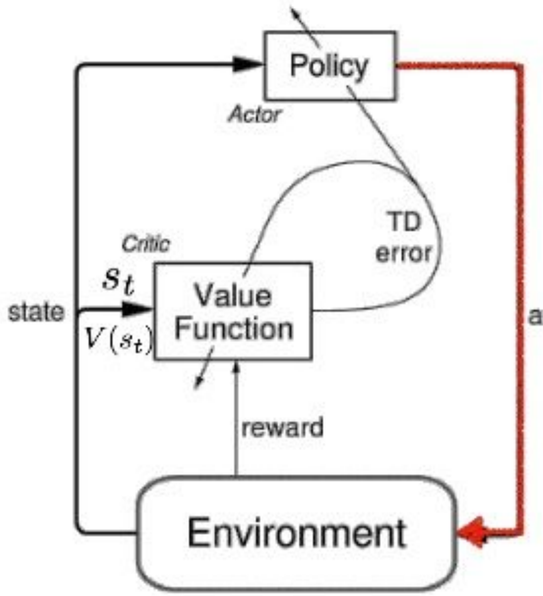
source : https://huggingface.co/learn/deep-rl-course/unit6/advantage-actor-critic

Actor-Critic

Algorithm 1 Actor-Critic Algorithm

for each episode **do**
  Observe initial state s
  Initialize empty episode buffer
  for each time step **do**
    Actor selects action a using the current policy: $a = \pi_\theta(s)$
    Take action a, observe reward r and next state s'
    Store transition (s, a, r, s') in the episode buffer
  end for
  for each time step in reverse order **do**
    Calculate discounted future rewards: $G_t = \sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k}$
  end for
  Update the Critic network
    Compute the temporal difference error: $\delta = G_t - V_w(s_t)$
    Update Critic parameters: $w \leftarrow w + \alpha_{\text{critic}} \cdot \delta \cdot \nabla_w V_w(s_t)$
  Update the Actor network
    Compute the advantage: $A_t = G_t - V_w(s_t)$
    Update Actor parameters: $\theta \leftarrow \theta + \alpha_{\text{actor}} \cdot A_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)$
end for

Actor-Critic

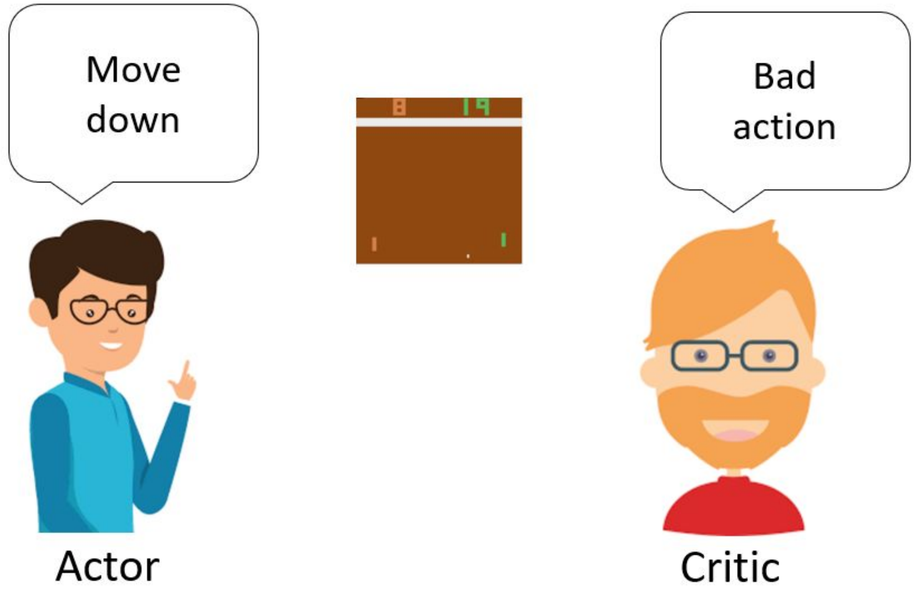


- Actor: decides which action to take
- Critic: tells the actor how good its action was and how it should adjust

(Figure from Sutton & Barto, 1998)

Why is Actor-Critic so good?

1. Combination of Policy and Value Learning
2. Variance Reduction and Stability
3. Effective Handling of Continuous Action Spaces
4. Real-time Online Learning
5. Asynchronous Updates



Actor-Critic

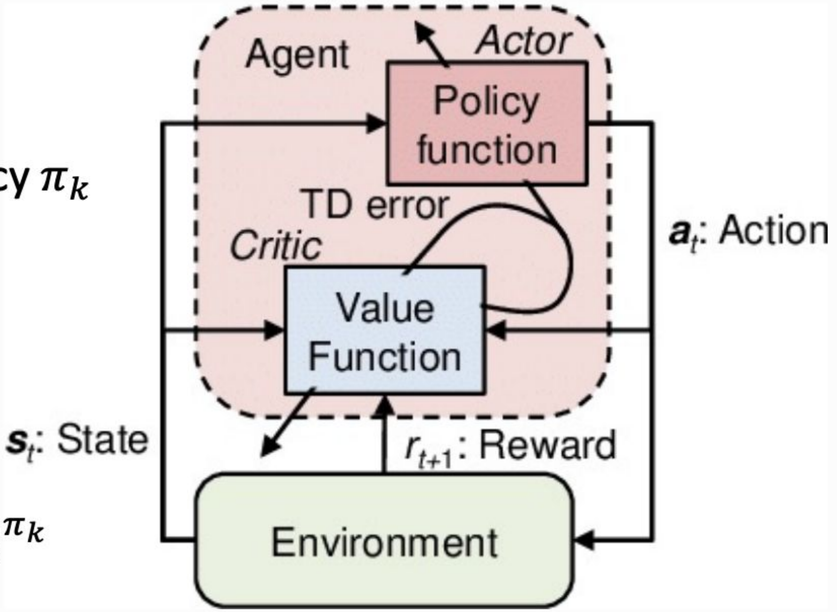
- Actor-critic Methods: combine value-based and policy-based methods
  - Critic update: estimate state (action)-value function under current policy π_k

$$V(s_k) \leftarrow V(s_k) + \alpha_k \left[r_k + \gamma V(s_{k+1}) - V(s_k)\right]$$

$$Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha_k \left[r_k + \gamma \cdot Q(s_{k+1}, a_{k+1}) - Q(s_k, a_k)\right]$$

  - Actor update: policy gradient based on value function estimator V^{π_k}, Q^{π_k}

$$\theta_{k+1} = \theta_k + \eta_k \nabla_\theta \hat{J}(\pi_{\theta_k})$$



[Sutton and Barto, 2018]

It’s time for your questions !

Quizz

https://red-field-0a6ddfd03.1.azurestaticapps.net/quiz/222

How does RL training algorithm knows how well it did?

- It achieves high accuracy
- Using perplexity metric
- Using reward function

Which problem(s) is RL is applicable to?

- With discrete environment
- With continuous environment
- Both

In Actor-Critic model, critic predicts

- Reward function
- Best next action
- Probability of next actions

Workshop

https://colab.research.google.com/drive/17XC6qcw3Bw_rLT3tLg2J6_RUlnwtiTb7?usp=sharing

CartPole gym

(a) Action Space

Number	Action
0	Push cart to the left
1	Push cart to the right

(b) Observation Space

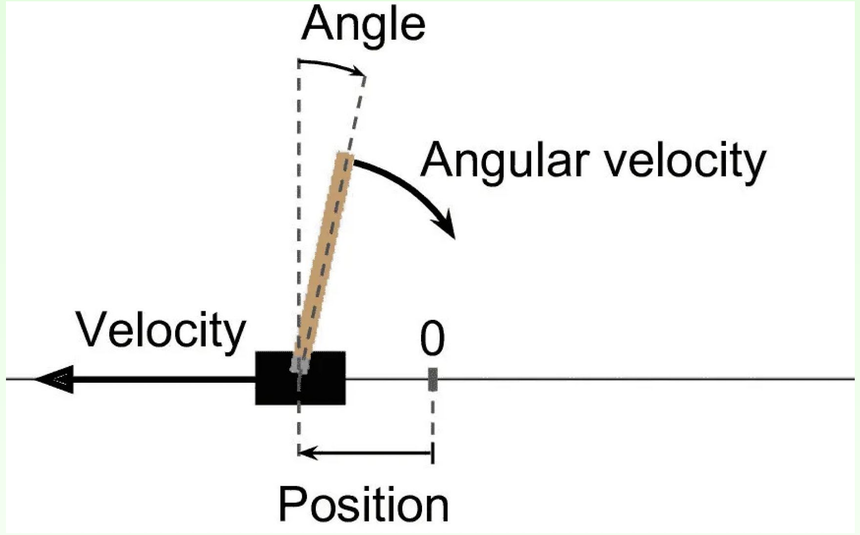
Number	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -0.418 rad (-24°)	~ 0.418 rad (24°)
3	Pole Angular Velocity	-Inf	Inf

(c) Rewards

Every Step	The Reward
To keep the pole upright for as long as possible	+1

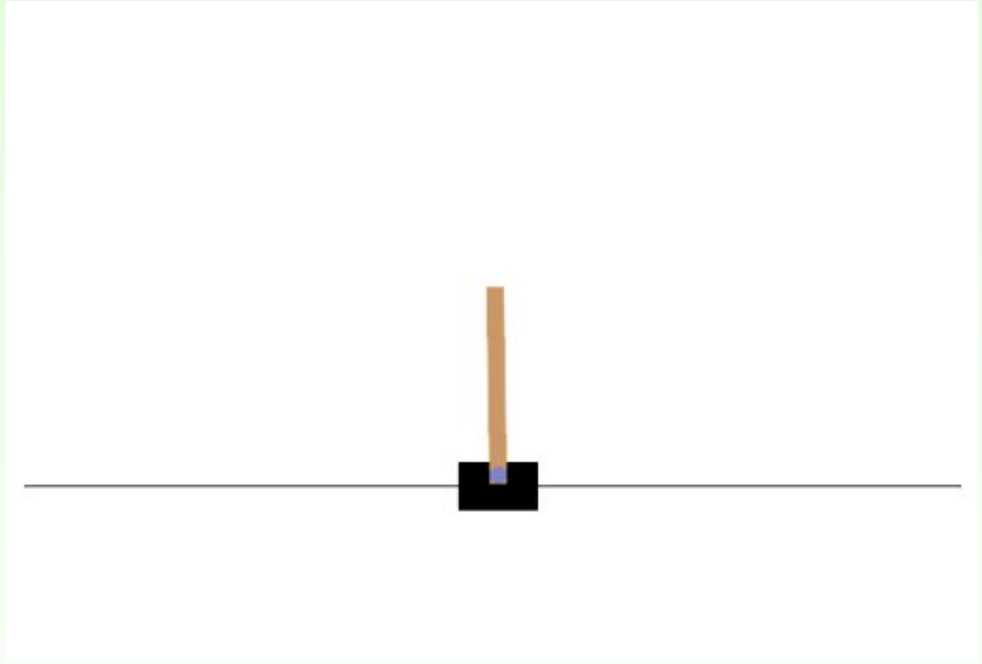
(d) Episode End

Cases	Termination or Truncation
1	Pole Angle is greater than ±12°
2	Cart Position is greater than ±2.4
3	Episode length is greater than 500 (200 for v0)

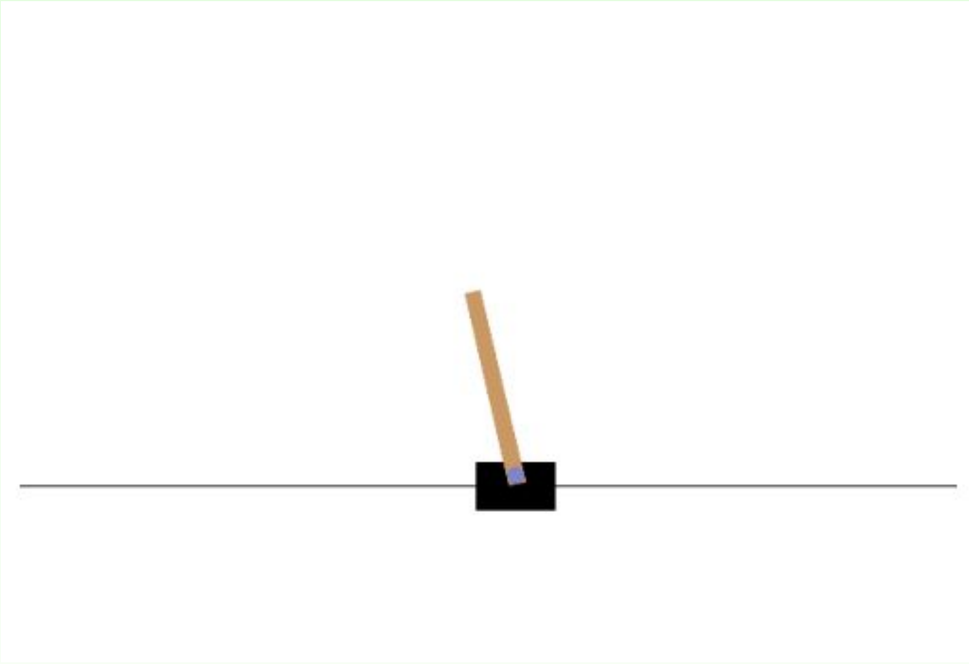


CartPole gym

Good position



Wrong position



CartPole gym

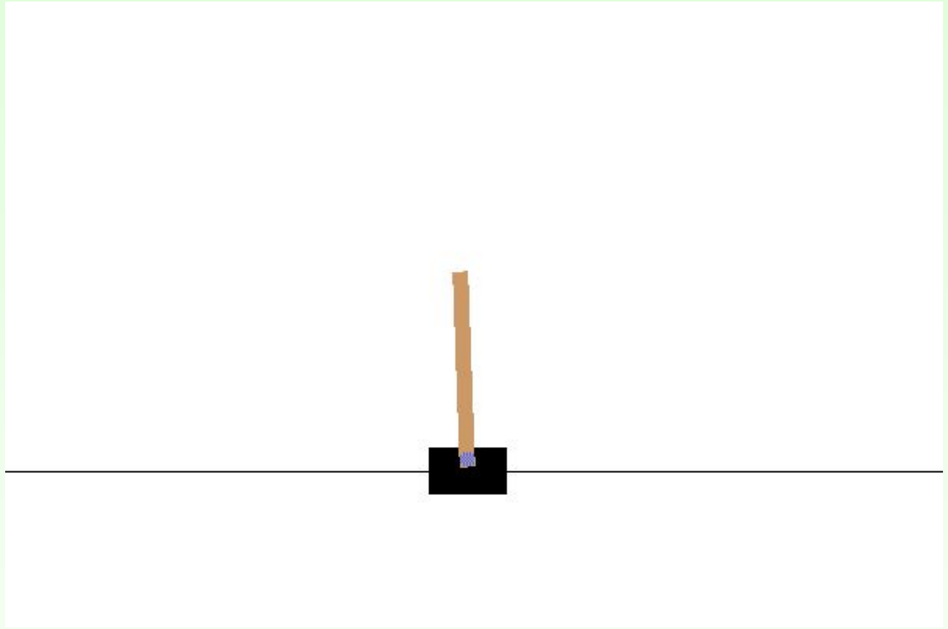
```python
# Import the OpenAI Gym library
import gym

# Create a CartPole environment instance
env = gym.make('CartPole-v1')

# Reset the environment to get the initial observation
observation = env.reset()

# Environment returns observation, reward, done flag end extra info.
observation, reward, done, info = env.step(env.action_space.sample())

# Render the current state of the environment (visualization)
env.render()
```



Real Life application

Guiding Pretraining in Reinforcement Learning with Large Language Models

Yuqing Du [*1] Olivia Watkins [*1] Zihan Wang [2] Cédric Colas [3 4] Trevor Darrell [1]
Abhishek Gupta [2] Jacob Andreas [3]

Efficient Object Detection in Large Images Using Deep Reinforcement Learning

Burak Uzkent Christopher Yeh Stefano Ermon
Department of Computer Science, Stanford University
buzkent@cs.stanford.edu, chrisyeh@stanford.edu, ermon@cs.stanford.edu

Deep Reinforcement Learning for Optimal Portfolio Allocation: A Comparative Study with Mean-Variance Optimization

Srijan Sood, [1] Kassiani Papasotiriou, [1] Marius Vaiciulis,[2] Tucker Balch [1].
[1] J.P. Morgan AI Research
[2] J.P. Morgan Global Equities; Oxford-Man Institute of Quantitative Finance
{srijan.sood, kassiani.papasotiriou, marius.vaiciulis, tucker.balch}@jpmorgan.com

Recent advances in reinforcement learning in finance

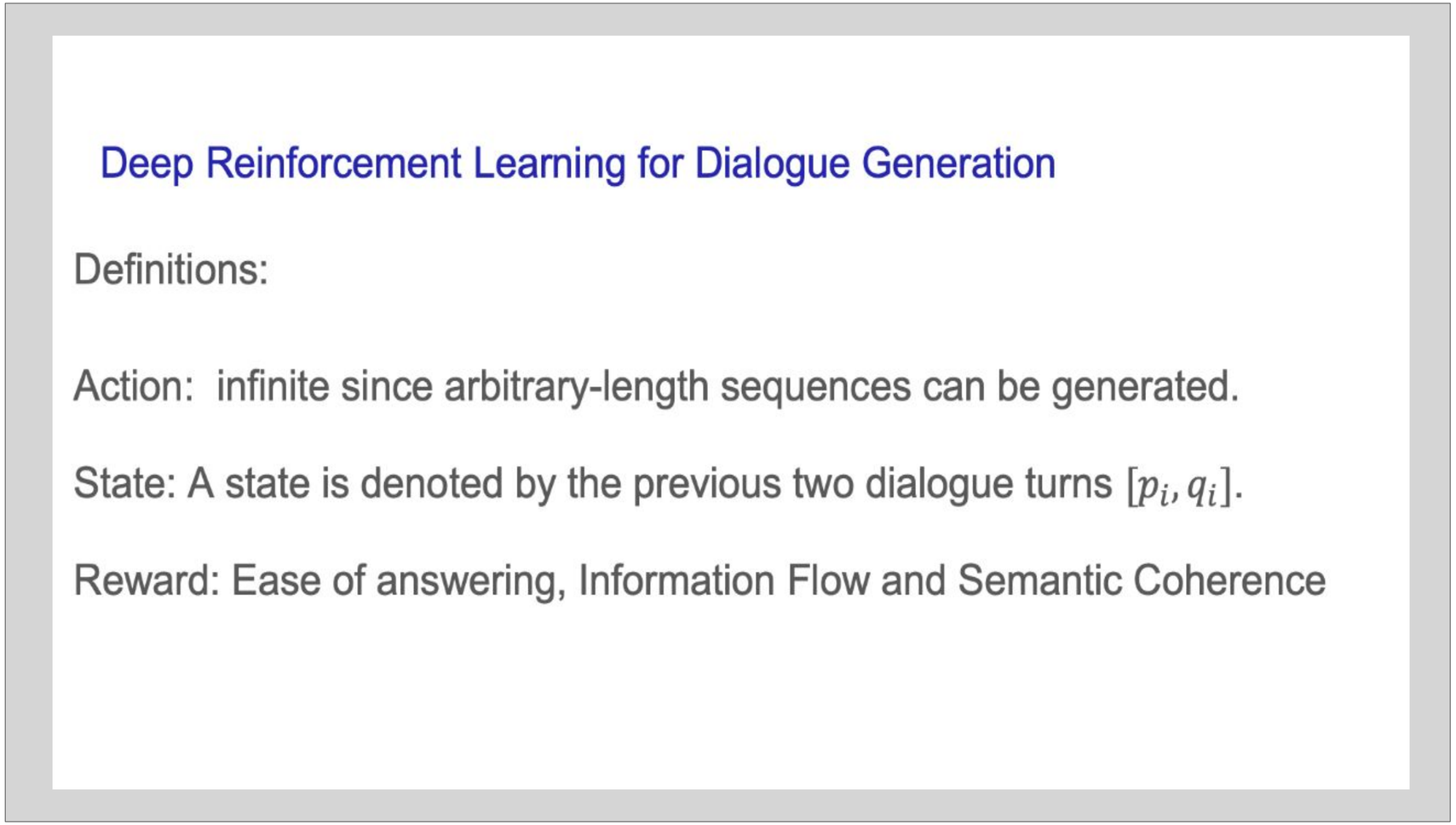
Ben Hambly[1] | Renyuan Xu[2] | Huining Yang[1]

Reinforcement Learning in Healthcare: A Survey

Chao Yu, Jiming Liu, *Fellow, IEEE*, and Shamim Nemati

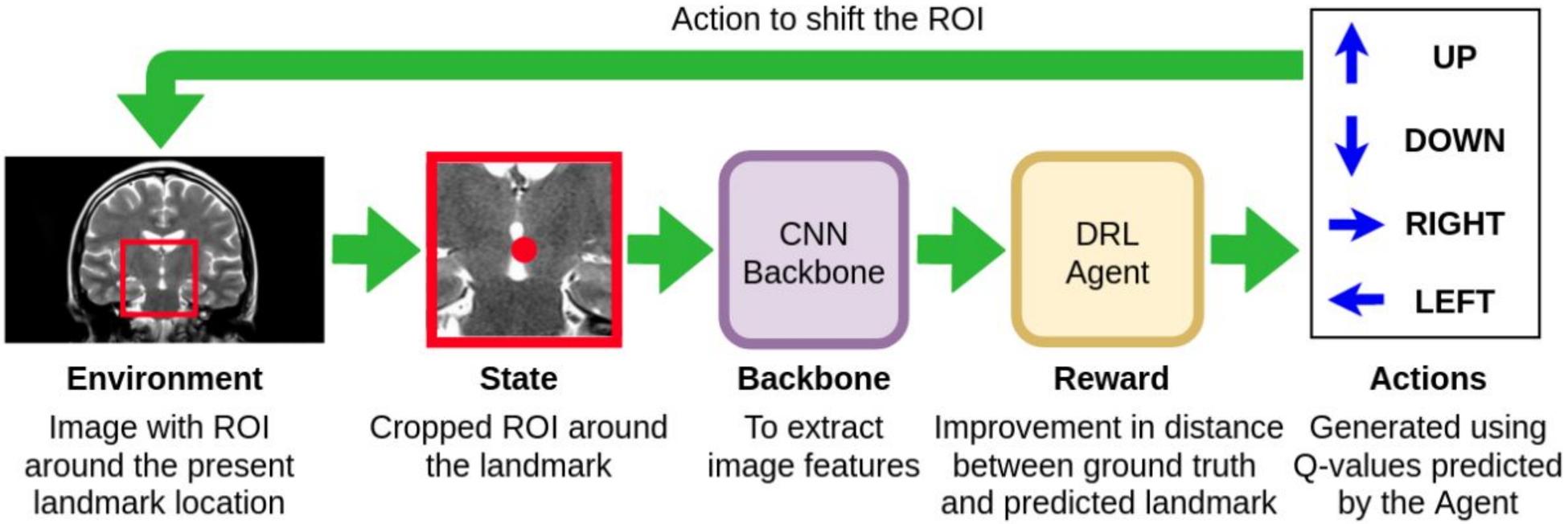
RL & NLP

- Article summarization
- Question answering
- Dialogue generation
- Dialogue System
- Knowledge-based QA
- Machine Translation
- Text generation



source : https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture16-guest.pdf

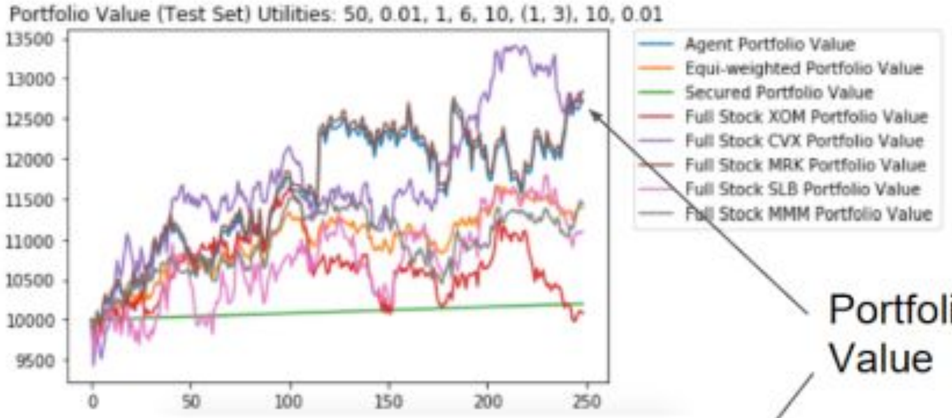
RL & Computer Vision



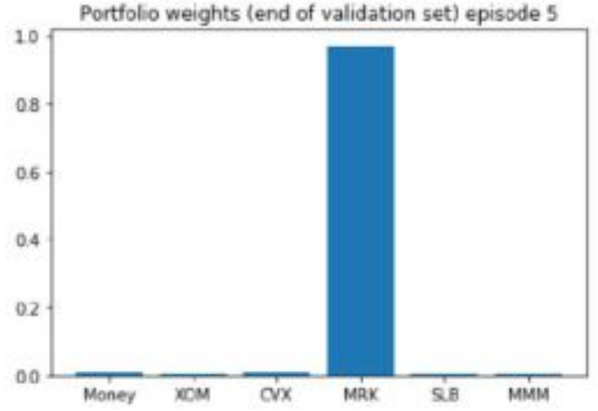
source : https://arxiv.org/pdf/2108.11510.pdf

RL & Finance

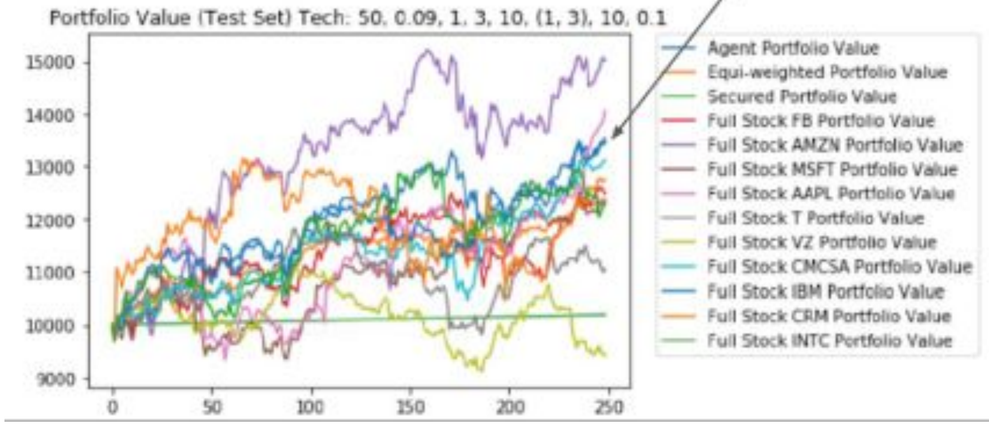
Single stock selection behaviour

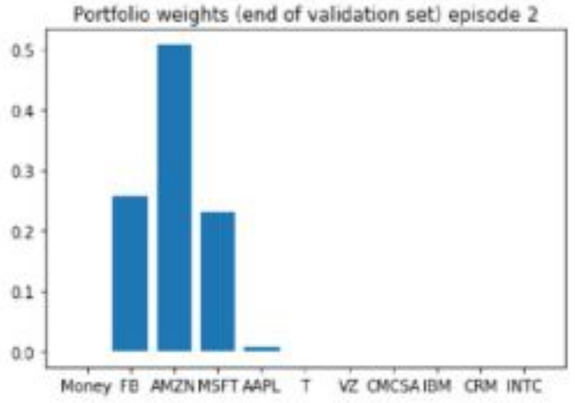


Portfolio Value



Multi stock selection behaviour





source : https://neptune.ai/blog/7-applications-of-reinforcement-learning-in-finance-and-trading

Pros	Cons
1. Versatility: RL can handle diverse tasks and domains.	**1. High Computational Complexity:** Training RL models can be computationally expensive.
2. Adaptability: RL agents can adapt to changes in the environment.	**2. Sample Inefficiency:** RL may require a large number of samples for effective learning.
3. Autonomous Decision-Making: RL enables autonomous decision-making in complex scenarios.	**3. Exploration Challenges:** Balancing exploration and exploitation can be challenging.
4. Learning from Interaction: Agents learn from interacting with the environment.	**4. Lack of Interpretability:** RL models can lack interpretability, making it challenging to understand their decisions.
5. Handles Partially Observable Environments: RL can deal with partially observable states.	**5. Sensitivity to Hyperparameters:** RL models are often sensitive to hyperparameter settings.
6. Suitable for Sequential Decision-Making: Ideal for tasks where decisions have a sequential impact.	**6. Risk of Suboptimal Solutions:** RL models may converge to suboptimal solutions.
7. Real-Time Decision-Making: RL can be applied to real-time decision-making scenarios.	**7. Ethical Concerns:** RL models may learn biased behaviors if not properly designed and supervised.
8. Potential for Generalization: RL models can generalize learning to new situations.	**8. Limited Transferability:** Learning in one environment may not easily transfer to another.

Please share your feedback !