04: Reinforcement Learning

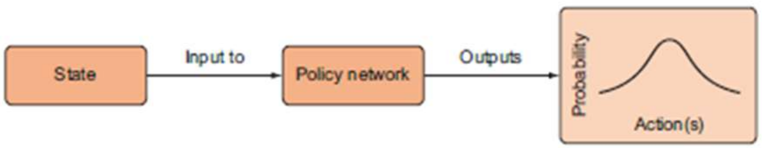
Antorweep Chakravorty

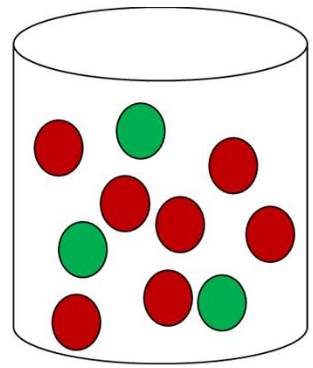
Topics

- Policy Gradient Methods
- OpenAI Gym
- CartPole
- REINFORCE Algorithm

Recap

- **Q Learning** - learns a **value function** that produces a Q-value for each action in a state
  - The Q-value is the **expected (weighted average) rewards**
  - Given the Q-values, an action is chosen based on a strategy or policy like epsilon greedy
- Alternatively, learning can be performed directly to predict the action
  - A **probability distribution** for all actions at a given state is generated
  - An action is sampled based on their likelihood
  - In this form of learning, the objective is to learn and improve the **policy function**





- An action is chosen based on its likelihood.
- This does not guarantee that the action with the highest probability is always chosen.
- It ensures that most of the time the action with the highest probability is chosen, however, it might also choose the action with the second-best probability or even the worse probability.
- Thus, introduces inherent exploration in the strategy for choosing actions

Policy Networks

- An **ANN** that learns the **policy function** and outputs a probability distribution for all actions given a state: $\pi(s) = P(A|S = s_t)$
- This class of algorithms are called as **Policy Gradient Methods**

Policy Networks

- Policy gradient methods may be constructed in multiple ways
- Methods, where outputs are a probability distribution over all actions, are called **Stochastic Policy Gradient**
- Alternatively, a **Deterministic Policy Gradient** may be used when the environment is stationary having a probability distribution that converges to a **degenerate probability distribution**
  - A degenerate distribution is one in which all the probability mass is assigned to a single possible outcome
- Deep ANN-based policy networks, fits better with Stochastics Policy Gradient methods due to the ability to differentiate in a straightforward manner

Policy Networks

- WHITEBOARD

Policy Gradient Algorithm

- Denoted using $\boldsymbol{\pi_{\theta}}$, where $\boldsymbol{\theta}$ is a vector representing the *weights and bias* of the network
- θ is initialized randomly at the start of the learning process
- At each forward pass through the network, for a given state S_t = **s**, π_θ(s) returns the distribution of overall actions in actions space **A**
    - Having θ initialized randomly, the probability of the actions would be a uniform distribution
    - An action $\mathbf{A_t}$ = a is chosen from this distribution and a reward $\mathbf{R_{t+1}}$ **= r** is received
    - We continue actions by sampling from the action distribution until we reach the end of the **episode**
- An episode is a **sequence** of **states**, **actions**, and **rewards** from an *initial state to a terminal state*. It is represented as: $\varepsilon = \{(S_0, A_0, R_1),\ (S_1, A_1, R_2), \ldots, (S_{t-1}, A_{t-1}, R_t)\}$
- Given the states the objective is to encourage the policy network to make these actions more likely next time. **Actions** that lead to **positive rewards** should be **reinforced**.

Policy Gradient Algorithm

- WHITEBOARD

Action Reinforcement

- The parameters of the network need to **maximize** the probabilities of winning actions
  - The probability of an action **a** given a state **s** and parameters **θ** for the policy **π** is denoted by: $\boldsymbol{\pi_s(a|\theta)}$
  - A probability distribution needs to have a **sum of 1**. Maximizing the probability of one action requires minimization of probabilities of other actions
- A naïve approach might be to make a target action distribution *degenerative*. However, using such an approach would discard future exploration
- A proper strategy for reinforcing actions should
  - Maintain stochasticity in action sampling to adequately explore the environment
  - Provide weighted credit to each action
- Appropriate actions will be reinforced by minimizing loss function: $\boldsymbol{1- \pi_s(a|\theta)}$. As the loss approaches 0, $\pi_s(a|\theta)$ will come closer to 1. Thus, it will inturn ensure that the gradients maximize $\pi_s(a|\theta)$

Action Reinforcement

- WHITEBOARD

Log Probability

- Often probabilities may be extremely small which leads to issues related to numeric precisions
- Therefore, instead of working with the probabilities directly, we use its natural logarithm **$-\log \pi_s(a|\theta)$** since the log of probability space ranges from **$(-\infty, 0)$**
- Results generated using $-\log \pi_s(a|\theta)$ instead of $1 - \pi_s(a|\theta)$ abide by the objective of ensuring that as loss approaches 0, $\pi_s(a|\theta)$ approaches 1.

Credit Assignment

- RL addresses control tasks where the terminal / final state consists of a sequence of previous states and actions
- Having, the objective $-\log \pi_s(a|\theta)$ assigns equal weight to every chosen action in an episode
- This allocates that same weight to the first choice of action in the initial state, the last action in the terminal state, and everything in between
- However, depending on the problem either the first or the last action that led to the most rewards may be more significant than the others
- This is the problem of **credit assignment**

Credit Assignment

- The confidence uncertainty of chosen action in an episode is represented by multiplying the magnitude of the update by a **decay factor** gamma $\boldsymbol{\gamma} \in [\mathbf{0}, \mathbf{1}]$
- The action leading to a final positive reward will have a decay factor of 1, meaning it receives a full gradient update
- Other actions will be decayed by a fraction so the gradient step will be smaller
- Therefore, the final **objective** to **minimize** will be: $-\boldsymbol{\gamma_t G_t log \pi_s(a|\theta)}$

Here:

$\mathbf{Y_t}$ is the **decay factor** at timestamp **t.**

$\mathbf{G_t}$ is the expected **total return** at timestamp t: $G_t = r_t + r_{t+1} + \cdots + r_{T-1} + r_T$

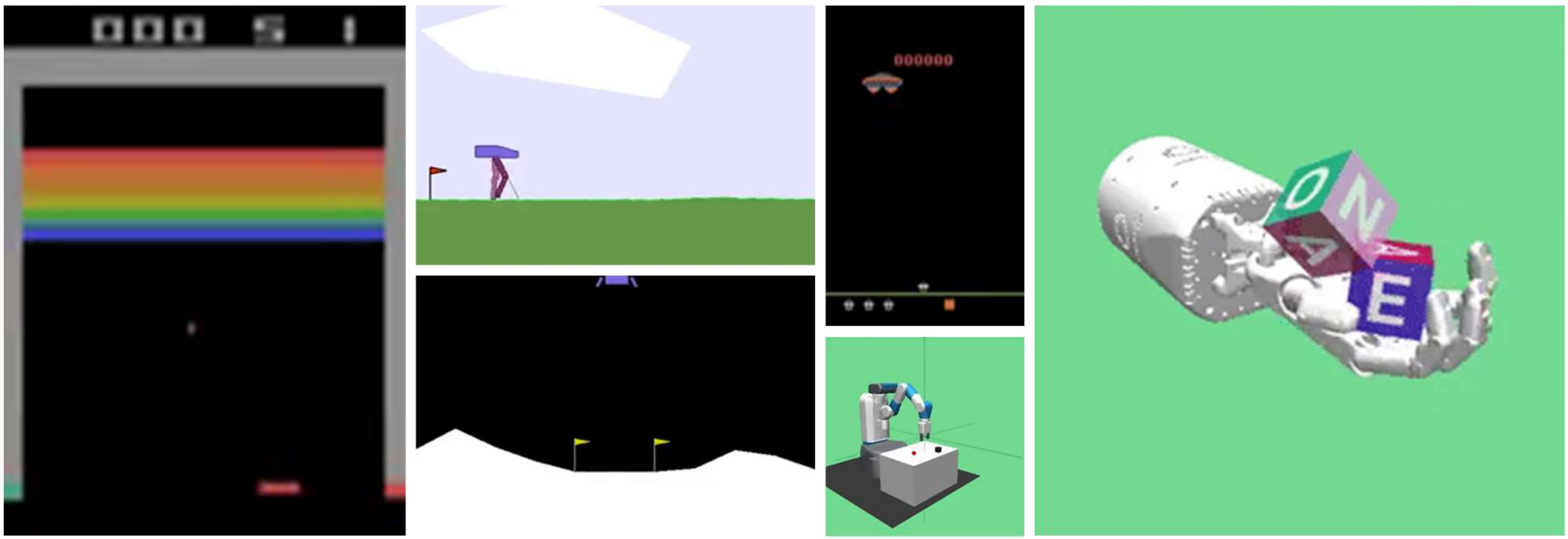
Credit Assignment

- WHITEBOARD

OpenAI gym

- An open-source interface.
- Gym is a toolkit for developing and comparing reinforcement learning algorithms.
- The library provides an easy-to-use suite of reinforcement learning tasks.
- It supports teaching agents everything from walking to playing games like Pong or Pinball.
- > Check it at https://gym.openai.com/

OpenAI gym

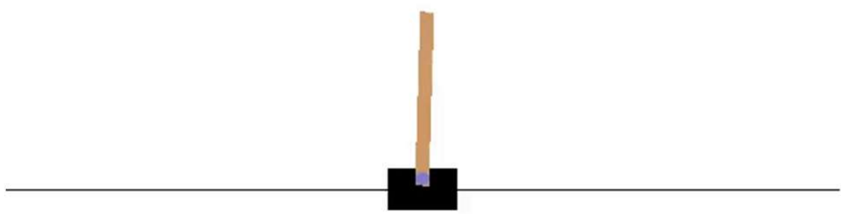


- Check it at https://gym.openai.com/envs

OpenAI gym

- CODE

CartPole



CartPole

- A Classic Control problem
- Aimed at **balancing** the **pole on a cart** by **moving** it either to the **left or right**
- In this environment there are only **two actions**
- The **state** is represented by a **vector** of size **four** indicating: *{cart position, cart velocity, pole angle, pole velocity}*
- A **reward** of **+1** is received at each timestamp the pole hasn't fallen
- The **episode ends or a terminal state** would be reached when the **pole falls**
- The **goal** is to maximize the length of the episode
- OpenAI states an **episode length of 200+** is considered as **solved**

CartPole

- CODE

REINFORCE Algorithm

- Create a feed-forward policy network
- Capture the episode information: Pass states through the network, use the model to predict and choose the next action. Continue until reaching terminal state or game over
- Training the model on the captured episode data
  - Calculate the probability of the action actually taken at each time step
  - Multiply the probability by the decay factor
  - Use this probability-weighted return to backpropagate and minimize the loss
- Continue with a new episode

REINFORCE Algorithm

- CODE