Deep Reinforcement Learning: Foundations and Practical Environment Setup for Real-World Applications

Franco Terranova

Université de Lorraine, CNRS, Inria, LORIA

franco.terranova@inria.fr

August 20, 2024

About Me

Name: Franco Terranova
PhD Student @ Université de Lorraine / INRIA / CNRS / LORIA
Current Project: Deep Reinforcement Learning for Cyber-Attack Paths Prediction
Previous Project: Deep Reinforcement Learning for a Self-driving Telescope
Website: `https://terranovafr.github.io`

FRANCE 2030
PROGRAMME DE RECHERCHE
CYBERSÉCURITÉ

Tutorial Material

https://terranovafr.github.io/teaching/2024-EASSS-Course

Overview

Paradigms of Machine Learning

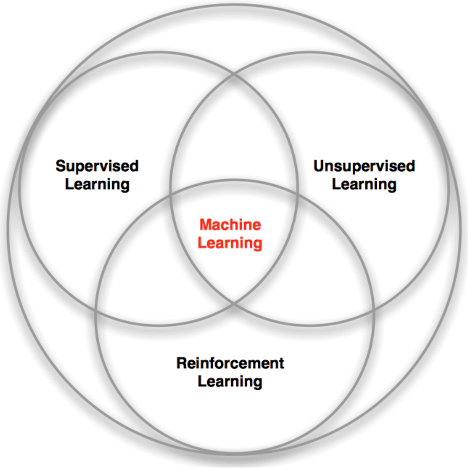


Image Source: `https://medium.com/dataseries/reinforcement-learning-mimics-human-learning-bc701d6ccc08`

Supervised Learning

- **Definition:** Learning from labeled data
- **Label:** The known output or correct answer for a given input
- **Data:** (x, y) — input and label
- **Goal:** Learn a function to map $x \rightarrow y$
- **Applications:** Classification, Regression, etc.

Example:

- Email spam detection: Emails with text labeled as "spam" or "not spam"
- Cat/Dog Image Classification: Images of cats and dogs labeled by type

Unsupervised Learning

- **Definition:** Finding patterns in unlabeled data
- **Data:** x — input data with no labels
- **Goal:** Discover hidden structures or patterns in the data
- **Applications:** Clustering, Dimensionality Reduction, etc.

Example:

- Customer segmentation in marketing: Grouping customers based on purchasing behavior
- Anomaly detection: Identifying unusual patterns in data, such as fraud detection in financial transactions

Reinforcement Learning

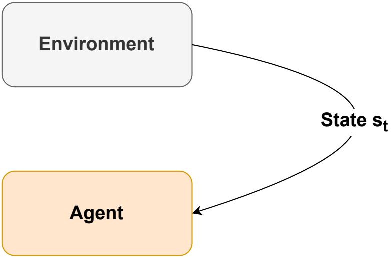
- **Definition:** Learning by interacting through trial and error with an **environment** that provides a **reward signal** (distinct from labels)
- **Goal:** Learn the optimal decision-making strategy in its context $\leftrightarrow$ maximize cumulative expected reward

Example:

- Game Playing: Agents learn strategies for games like Chess
- Drone Navigation: Agent navigating towards a destination by avoiding obstacles

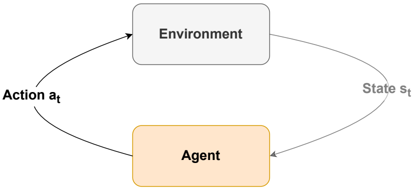
Interaction starts at timestep t

RL Elements - State



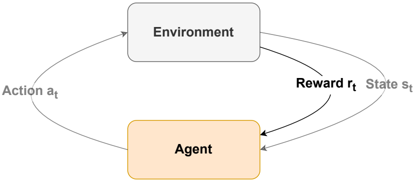
- Input for the agent
- Represents the context where the agent is located
- Agent must learn which elements of the state are relevant
- May not have full visibility of the environment

RL Elements - Action



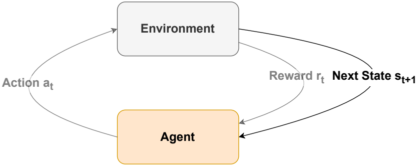
- Output for the agent
- Action produced conditional on the state provided as input $\leftrightarrow$ $a_t | s_t$
- Represents the modification the agent wants to make to the environment state

RL Elements - Reward



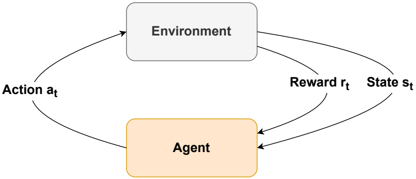
- Evaluation of the action taken in the previous state $\leftrightarrow r_t \mid a_t, s_t$
- Represents a prize or penalty for the action
- Guides the agent in adjusting its behavior for similar future states

RL Elements - Next State



- Resulting outcome of the action taken in the previous state $\leftrightarrow$ $s_{t+1} \mid a_t, s_t$
- Advances the timestep to $t + 1$
- Enables the sequential decision-making

RL Elements - Trial and error



- Agent learns through a loop of trial and error: state $\rightarrow$ action $\rightarrow$ reward, next state
- Sequential decision-making inspired by how humans learn

RL Elements - Simplistic Formulation

- The agent policy maps a state to an action:

$$\text{Action} = \pi(\text{State}) \quad \text{where } \pi \text{ is the policy}$$

- The environment maps an action and state to the next state and reward:

$$\text{Next State}, \text{Reward} = \mathcal{E}(\text{State}, \text{Action})$$

$$\text{where } \mathcal{E} \text{ represents the environment's dynamics}$$

- The overall loop can be summarized as:

$$\text{State}_{t+1}, \text{Reward}_t = \mathcal{E}(\text{State}_t, \pi(\text{State}_t))$$

What Makes RL Different?

- **No Supervisor:** Driven by reward signals, not explicit labeled data
- **Sequential Decision Making:** Optimization of long-term rewards through a series of actions over time
- **Delayed Feedback:** Feedback may be delayed, potentially sacrificing short-term rewards for long-term gains
- **Exploration vs. Exploitation:** Choosing between trying new actions or using known strategies

RL Definitions: Episodes, Trajectory, and Iterations

- **Episode:** A complete sequence of steps from the initial state to a terminal state or goal, after which the process restarts
- **Trajectory** τ **:** A sequence of states, actions, and rewards from the start to the end of an episode

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$$

Finite Episode: Episode length $T < \infty$

Infinite Episode: Episode length $T \to \infty$

- **Cutoff or Goal:** The condition or point at which an episode ends

Steps for RL Formulation

- Identify the State (s):
  - Determine what information defines the current situation of the agent
- Define the Actions (a):
  - Specify the possible decisions or moves the agent can take from each state
- Specify the Reward (r):
  - Decide how to quantify the feedback for each action in a given state
- Design the Environment Interaction:
  - Define state transitions and reward assignments based on actions
  - Specify episode structure, including length and termination criteria

Example - Breakout (Atari Game)

Image from https://www.coolmathgames.com/fr/0-atari-breakout

- **State Space:**
  - Raw pixel values from the game screen, 2D array of pixels
- **Action Space:**
  - Discrete actions: moving the paddle left, right, or no action
- **Reward Function:**
  - Positive reward for destroying bricks
  - Negative reward for losing the ball
- **Episode:**
  - Starts with the paddle at the bottom and the ball in motion
  - Ends when the ball falls below the paddle or all bricks are destroyed

AlphaGo (Go Game)

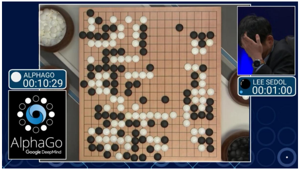


Image from Engadget

- **State Space:**
  - Board configurations, including the positions of black and white stones
- **Action Space:**
  - Placing a stone at any empty intersection on the board
- **Reward Function:**
  - Positive reward for winning the game
  - Negative reward for losing
- **Episode:**
  - Begins with an empty board
  - Ends when a winner is determined

Markov Decision Process (MDP) Formulation

$$
\begin{aligned}
\text{MDP:} \quad & (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma) \\
\mathcal{S}: \quad & \text{State space} \\
\mathcal{A}: \quad & \text{Action space} \\
\mathcal{P}: \quad & \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}, \text{ State transition function} \\
\mathcal{R}: \quad & \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}, \text{ Reward function} \\
\gamma: \quad & \text{Discount factor}
\end{aligned}
$$

Model-based Methods

One approach to finding optimal behavior in an environment involves approximating $\mathcal{P}$ and $\mathcal{R}$:

- **Know $\mathcal{R}$:** Determines the quality of a_t in s_t for each t
- **Know $\mathcal{P}$:** Predicts s_{t+1} based on a_t in s_t. Allows recalculation of $\mathcal{P}$ and $\mathcal{R}$ based on s_{t+1} and a_{t+1}
- **Challenges:** This approach can be impractical for real-world problems

Model-Free Methods

Indirectly approximate environment by approximating the policy:

- **Policy Approximation:** Learn a policy $\pi(s) \rightarrow a$, which maps states s to actions a
- **Objective:** Optimize the policy to maximize the cumulative expected reward (or return) over time

Return

Return (G_t): The total accumulated reward an agent expects to receive starting from time step t

- **Definition:**

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- **Finite Episode:** The sum is limited to a fixed number of steps, T, rather than extending to infinity
- **Components:**
  - $r_{t+1}, r_{t+2}, \ldots$: Rewards received at each time step
  - γ: Discount factor, $0 \leq \gamma \leq 1$

Discounting rewards

Role of Discount Factor (γ):

- **Trade-off Decision:** Determines the trade-off between immediate rewards and future rewards
- $\gamma = 0$**:** Focuses only on immediate reward

$$G_t = r_{t+1}$$

- $\gamma = 1$**:** Values future rewards as much as immediate rewards

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots = \sum_{k=0}^{\infty} r_{t+k+1}$$

- **Practical Use:** strictly between 0 and 1

Summary

- RL is a ML paradigm involving a loop of trial and error
- The reward signal guides the learning process
- Environment modeled as MDP
- Model-free methods preferred for real-world problems
- Maximization of $\mathbb{E}_\pi[G_t]$ properly setting γ

Model-Free Methods

We do not approximate $\mathcal{P}$ and $\mathcal{R}$, we approximate them indirectly:

- **Policy-Based:** The policy $\pi(a|s)$
- **Value-Based:** Value functions $V(s)$ or $Q(s, a)$
- **Actor-Critic:** Both policy $\pi(a|s)$ and value functions

Policy-based methods

- **Objective:** Directly learn a policy $\pi(a|s)$ representing the agent
- The policy (π) outputs the probability of taking action a in state s

$$\pi(a|s) = P(a_t = a|s_t = s)$$

- **Optimization Problem:** Train the policy to maximize the cumulative expected reward:

$$J(\pi) = \mathbb{E}_{\pi}[G_t]$$

Value-based methods

- **Objective:** Approximate a function that provides the quality (value) of each action in a given state
- **Potential Options:**
  - **State Value Function ($V(s)$):** Estimates the expected return starting from state s and following a particular policy π:

$$V(s) = \mathbb{E}_\pi[G_t | s_t = s]$$

  - **Action Value Function ($Q(s, a)$):** Estimates the expected return of taking action a in state s and following a particular policy π:

$$Q(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$$

Interconnection of π, V, and Q

- **From Q to π:** Optimal policy can be derived directly by selecting actions with the highest Q-value:

$$\pi(s) = \arg\max_a Q(s, a)$$

- However, we have no direct mapping from V to π
- Same from π to V or Q

Actor-critic methods

- **Objective:** Combine policy-based and value-based methods to improve learning
- **Components:**
  - **Actor:** Learns the policy $\pi(a|s)$ to select actions
  - **Critic:** Estimates the value function $V(s)$ to evaluate the quality of the states
- Leveraging the value estimate to inform the policy updates

RL Intersections

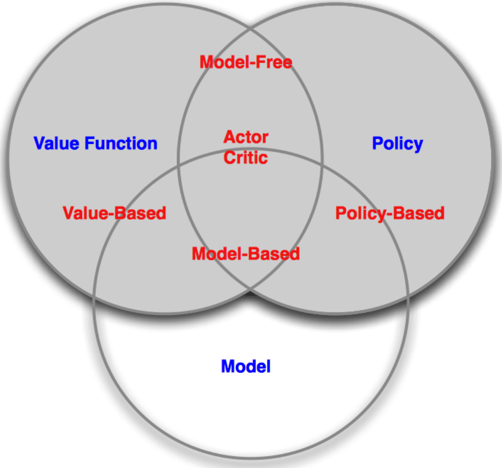


Image Source: Odonkor, Philip & Lewis, Kemper. (2018). Control of Shared Energy Storage Assets Within Building Clusters Using Reinforcement Learning. 10.1115/DETC2018-86094.

Algorithms for Learning

- **Objective:** Find functions that maximize $\mathbb{E}_\pi[G_t]$ using iterative optimization methods
- **Value-Based Methods:**
  - Update Formula: Bellman Equation
- **Policy-Based Methods:**
  - Update Formula: Policy Gradient Theorem, ...
- **Actor-Critic Methods:**
  - Combination of update formulas to reach the approximation of actor and critic

Optimality

Derive Optimal Policy in RL Problems:

- From Policy-Based / Actor-Critic Methods: directly derive the optimal policy (π^*)
- From Value-Based Methods:
  - Approximate the optimal action-value function (Q^*)
  - Derive the optimal policy (π^*) from this function:

$$\pi^*(a|s) = \arg\max_a Q^*(s, a)$$

How to choose?

Choosing the right RL method:

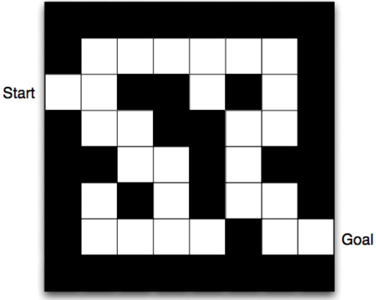
- **Policy-Based / Actor-Critic Methods:**
  - Converge to probabilistic (stochastic) policies
  - Reason: Optimize a policy $\pi(a|s)$, that inherently approximate a probability distribution:

$$\pi^*(a|s) = P(a|s)$$

- **Value-Based Methods:**
  - Converge to deterministic policies
  - Reason: Derive the policy by selecting the unique maximum action in each state:

$$\pi^*(a|s) = \arg\max_a Q^*(s, a)$$

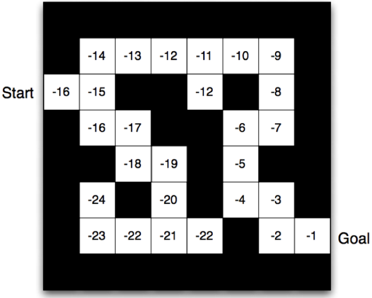
Grid World



Slide credit: D. Silver

- State: (x,y) position
- Action: up, down, left, right
- Rewards: -1 per time-step
- Episode termination: Reach goal

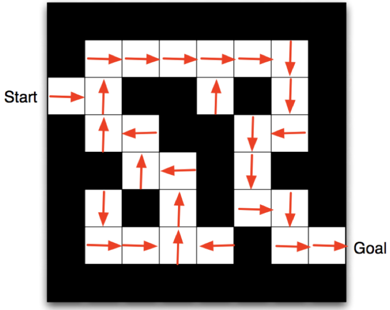
Grid World



Slide credit: D. Silver

- Optimal value function $V_\pi^*(s)$
- Expected return in each state

Grid World



Slide credit: D. Silver

- Optimal policy function $\pi^*(a|s)$
- The optimal policy is deterministic
- Actions that maximize expected return in each state

Tabular RL

Tabular RL Overview:

- **Definition:** Uses tables (arrays) to represent and approximate policies and value functions
- **Tabular Representation:**
  - Value Function Table
  - Action-Value Function Table
  - Policy Function Table

Tabular RL

	Value
State 1	$V^{\pi}(S1)$
State 2	$V^{\pi}(S2)$
....	
State M	$V^{\pi}(SM)$

Tabular representation of the value function

	Action 1	Action 2	....	Action N
State 1	$Q^{\pi}(S1, A1)$	$Q^{\pi}(S1, A2)$		$Q^{\pi}(S1, AN)$
State 2	$Q^{\pi}(S2, A1)$	$Q^{\pi}(S2, A2)$		$Q^{\pi}(S2, AN)$
....				
State M	$Q^{\pi}(SM, A1)$	$Q^{\pi}(SM, A2)$		$Q^{\pi}(SM, AN)$

Tabular representation of the action-value function

	Action 1	Action 2	....	Action N
State 1	$P(A1\|S1)$	$P(A2\|S1)$		$P(AN\|S1)$
State 2	$P(A1\|S2)$	$P(A2\|S2)$		$P(AN\|S2)$
....				
State M	$P(A1\|SM)$	$P(A2\|SM)$		$P(AN\|SM)$

Tabular representation of the policy

Limitations of Tabular RL

Challenges and Limitations:

- **Scalability:**
  - **Optimization:** Slow convergence and inefficient learning in large environments
  - **Memory:** Tables become impractical with large state or action spaces due to memory constraints
- **Generalization:**
  - No ability to generalize across unseen states or actions
- **Continuous Spaces:**
  - Inapplicable for environments with continuous state and action spaces

RL Elements - Discrete vs Continuous

Discrete vs Continuous Spaces:

- **Discrete:**
  - **Definition:** Finite/countable states/actions
  - **Example:** Board games
- **Continuous:**
  - **Definition:** Infinite states/actions
  - **Example:** Robot arm angles
  - **Note:** Requires specialized algorithms
- **Mixed:**
  - **Definition:** Both discrete and continuous elements
  - **Example:** Video games with levels and player control
  - **Note:** May need hybrid approaches

- **Parameterized Models:**
  - Represent the policy $\pi(a|s)$, action-value function $Q(s,a)$, or value function $V(s)$ using a parameterized function:

$$\pi_\theta(a|s) \quad Q_\theta(s,a) \quad V_\theta(s)$$

  - Here, θ represents the parameters of the function to be optimized for the return maximization

Function Approximators

$$\begin{aligned} \pi_\theta :& \quad \mathcal{S} \times \theta \to \mathcal{A} \\ V_\theta^\pi :& \quad \mathcal{S} \times \theta \to \mathbb{R} \\ Q_\theta^\pi :& \quad \mathcal{S} \times \mathcal{A} \times \theta \to \mathbb{R} \\ & \quad \theta \in \Theta, \text{ parameter space} \end{aligned}$$

Function Approximators

$$
\begin{aligned}
\pi_\theta : &\quad \mathcal{S} \times \theta \rightarrow \mathcal{A} \\
V^\pi_\theta : &\quad \mathcal{S} \times \theta \rightarrow \mathbb{R} \\
Q^\pi_\theta : &\quad \mathcal{S} \times \theta \rightarrow \mathbb{R}^{|A|} \\
&\quad \theta \in \Theta, \text{ parameter space}
\end{aligned}
$$

Function Approximators

- **Advantages:**
  - **Generalization:** Handle large or continuous state and action spaces by generalizing across similar states and actions
  - **Efficiency:** Reduce memory usage compared to tabular methods
  - **Optimization:** Efficients algorithms for iterative optimization
- **Deep RL** relies on deep learning, using neural networks (NN) as function approximators

Deep Learning

- **Definition:** A subset of machine learning involving NNs with multiple layers
- **Architecture:** Composed of input, hidden, and output layers
- **Universal Function Approximator:** NNs are capable of approximating any continuous function to a desired level of accuracy, given enough neurons and layers
- Uses **backpropagation** to adjust weights, with **gradients** computed to minimize error through optimization algorithms

Neural Networks as Function Approximators

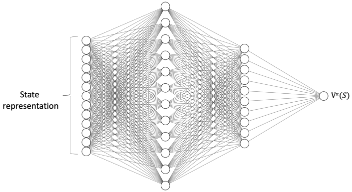
General Update Rule:

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$$

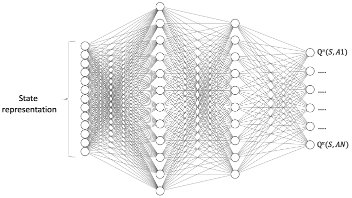
where

- θ denotes the model parameters
- α is the learning rate
- $\nabla_\theta J(\theta)$ represents the gradient of the loss function $J(\theta)$
- Iterative update formulas will be used to define the loss function for updating the function parameters θ

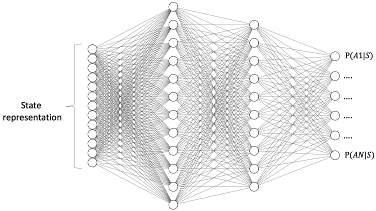
Deep Reinforcement Learning



NN approximating the value function



NN approximating the action-value function



NN approximating the policy function.

Images generated with `https://alexlenail.me/NN-SVG/`

Established DRL Algorithms

- **Value-Based**
  - Deep Q-Network (DQN)
  - ...
- **Policy-Based**
  - Proximal Policy Optimization (PPO)
  - Trust Region Policy Optimization (TRPO)
  - ...
- **Actor-Critic**
  - Advantage Actor Critic (A2C)
  - ...

Deep Q-Network (DQN)

- **What is DQN?**
  - Combines Q-learning with deep NNs
  - Approximates the Q-value function

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

  - Uses an experience replay to store and reuse experiences
  - Widely used version incorporates additional strategies to improve learning

DQN Algorithm (Part 1)

1. Initialize Q-network $Q_\theta(s, a)$ with random weights θ
2. Initialize an empty replay buffer
3. Collect experience (s, a, r, s') from the environment using the Q-network and store it in the replay buffer
4. Randomly sample a mini-batch of k transitions (s_i, a_i, r_i, s_i') from the replay buffer

DQN Algorithm (Part 2)

5. Compute target Q-values using the Bellman equation:

$$y_i = r_i + \gamma \max_{a'} Q_\theta(s'_i, a')$$

6. Compute the loss over the mini-batch:

$$L(\theta) = \frac{1}{2k} \sum_{i=1}^{k} (y_i - Q_\theta(s_i, a_i))^2$$

7. Update the Q-network by minimizing the loss:

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$$

8. Repeat from step 3 until convergence

Target Network

- **Concept:** A target network is a separate Q-network $Q_{\theta^-}(s, a)$ that provides stable Q-value estimates for the Bellman equation
- **Purpose:** Avoid instability in learning due to rapidly changing Q-values
- Periodically update the target network weights to match the online network weights $Q_\theta(s, a)$

DQN Algorithm (Part 1)

1. Initialize Q-network $Q_\theta(s, a)$ with random weights θ
2. **Initialize target network $Q_{\theta^-}(s, a)$ with the same weights as** $Q_\theta(s, a)$
3. Initialize an empty replay buffer
4. Collect experience (s, a, r, s') from the environment using the Q-network and store it in the replay buffer
5. Randomly sample a mini-batch of k transitions (s_i, a_i, r_i, s_i') from the replay buffer

DQN Algorithm (Part 2)

5. **Compute target Q-values using the target network Q_{θ^-}:**

$$y_i = r_i + \gamma \max_{a'} Q_{\theta^-}(s'_i, a')$$

6. Compute the loss over the mini-batch:

$$L(\theta) = \frac{1}{2k} \sum_{i=1}^{k} (y_i - Q_\theta(s_i, a_i))^2$$

7. Update the Q-network by minimizing the loss:

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$$

8. **Periodically update the target network weights to match the online network weights**
9. Repeat from step 3 until convergence

ϵ-greedy Strategy

- **Concept:** Balances exploration and exploitation in action selection
- **Strategy:**
  - With probability ϵ, select a random action (exploration)
  - With probability $1 - \epsilon$, select the action that maximizes the Q-value (exploitation)
- **Equation:**

$$a_t = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg\max_a Q_\theta(s_t, a) & \text{with probability } 1 - \epsilon \end{cases}$$

ϵ Decay in ϵ-Greedy Strategy

- **Purpose of Epsilon Decay:**
  - Start with high exploration to gather diverse experiences
  - Gradually shift towards exploitation to refine the policy
- **Epsilon Linear Decay Function:**

$$\epsilon_t = \max(\epsilon_{\min}, \epsilon_0 \cdot \text{decay\_rate}^t)$$

  - ϵ_t: Epsilon value at time t
  - ϵ_0: Initial epsilon value
  - decay_rate: Rate at which epsilon decreases
  - $\epsilon_{\min}$: Minimum value epsilon can decay to

DQN Algorithm (Part 1)

1. Initialize Q-network $Q_\theta(s, a)$ with random weights θ
2. Initialize target network $Q_{\theta^-}(s, a)$ with the same weights as $Q_\theta(s, a)$
3. Initialize an empty replay buffer
4. Collect experience (s, a, r, s') from the environment
    - **with probability ϵ, select a random action**
    - **otherwise, select the action that maximizes $Q_\theta(s, a)$**
5. Randomly sample a mini-batch of k transitions (s_i, a_i, r_i, s_i') from the replay buffer

DQN Algorithm (Part 2)

5. Compute target Q-values using the target network Q_{θ^-}:

$$y_i = r_i + \gamma \max_{a'} Q_{\theta^-}(s'_i, a')$$

6. Compute the loss over the mini-batch:

$$L(\theta) = \frac{1}{2k} \sum_{i=1}^{k} (y_i - Q_\theta(s_i, a_i))^2$$

7. Update the Q-network by minimizing the loss:

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$$

8. Periodically update the target network weights to match the online network weights
9. Repeat from step 3 until convergence

DQN Improvements

- **Dueling DQN:** Improve value estimation
- **Double DQN:** Reduces overestimation bias
- **Prioritized Replay:** Changes sampling strategy
- **Noisy DQN:** Noisy networks instead of ϵ-greedy
- **Distributional DQN:** From expected Q-value to distribution

Proximal Policy Optimization (PPO)

- **What is PPO?**
  - An optimization algorithm aimed to approximate a policy function

$$\pi_\theta(a|s) \approx \text{Optimal Policy Distribution}$$

  - Optimizes the policy using a clipped surrogate objective (here simplified):

$$L(\theta) = \mathbb{E}_t \left[\text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right]$$

  - Uses clipping to ensure stable and reliable updates by preventing large policy changes
  - Uses the advantage of the action $\hat{A}_t$ for the update

PPO Algorithm

1. Initialize the policy network π_θ with random weights
2. Collect data by interacting with the environment using the current policy
3. Compute the advantage $\hat{A}(s, a)$ for each time step
4. Update the policy network by maximizing the PPO objective (simplified):

$$L(\theta) = \mathbb{E}_t \left[\text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right]$$

5. Repeat steps 2 to 4 until convergence

Recap

- **Model-Free Methods:**
  - Do not require a model of the environment
  - Suited for real-world complexities
- **Classes of Methods:**
  - Value-Based (e.g., DQN), Policy-Based (e.g., PPO), and Actor-Critic
- **Tabular RL Limitations:**
  - Struggle with large or continuous state/action spaces
- **Deep NNs:**
  - Address scalability issues by approximating functions in complex environments

Extra

Best Practices for RL Experiments

Reporting and Analysis

- Plot reward versus steps/episodes to visualize learning progress and convergence
- Use relevant metrics, such as reward or domain-specific measures, for evaluation
- Document experimental settings and results for reproducibility and future reference

Generalization and Robustness

- Assess how well the policy generalizes to new or unseen environments
- Evaluate the algorithm's robustness to different conditions or noise

Best Practices for RL Experiments

Comparison

- Compare multiple RL algorithms to identify the most effective approach
- Explore hyper-parameters or use hyperparameter optimization techniques
- Conduct multiple runs with different random seeds to ensure result robustness and reproducibility
- Report confidence intervals (CIs) to improve reliability with few runs

Best Practices for RL Experiments

Others

- Consider normalization of the observation space and reward signal
- Consider the sample efficiency of algorithms
- Consider the NN size or function approximator used
- Determine which environment parameters affect learning and how

Partially Observable Markov Decision Process

$$
\begin{aligned}
\text{POMDP:} \quad & (S, A, T, R, \Omega, O) \\
S: \quad & \text{State space (hidden states)} \\
A: \quad & \text{Action space} \\
\Omega: \quad & \text{Observation space} \\
O: \quad & S \times \Omega \to [0,1], \text{ Observation function} \\
P: \quad & S \times A \to S, \text{ State transition function} \\
R: \quad & S \times A \to \mathbb{R}, \text{ Reward function}
\end{aligned}
$$

Challenges and Algorithms for POMDPs

- More suited for modeling realistic scenarios
- Some information may not be available at deployment phase
- **Challenges:**
  - Incomplete or noisy information
  - Hidden states complicate decision-making
- Need DRL to converge also in front of partial observability

References - Part 1

[Sutton & Barto, 2018] Richard S. Sutton and Andrew G. Barto (2018)
Reinforcement Learning: An Introduction
The MIT Press, ISBN: 978-0262039246.

[Puterman, 1994] Martin L. Puterman (1994)
Markov Decision Processes: Discrete Stochastic Dynamic Programming
Wiley-Interscience.

[Rumelhart et al., 1986] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams (1986)
Learning representations by back-propagating errors
Nature 323, 533–536.

[Watkins & Dayan, 1992] Christopher J.C.H. Watkins and Peter Dayan (1992)
Q-Learning
Machine Learning 8, 279–292.

[Mnih et al., 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013)
Playing Atari with Deep Reinforcement Learning
NIPS Deep Learning Workshop 2013, arXiv preprint arXiv:1312.5602.

References - Part 2

[van Hasselt et al., 2015] Hado van Hasselt, Arthur Guez, and David Silver (2015)
Deep Reinforcement Learning with Double Q-learning
arXiv preprint arXiv:1509.06461.

[Sutton, 1998] Richard Sutton (1988)
Learning to predict by the methods of temporal differences
Machine Learning 3, 9–44.

[Wang et al., 2016] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando Freitas (2016)
Dueling Network Architectures for Deep Reinforcement Learning
In *Proceedings of The 33rd International Conference on Machine Learning*, 1995–2003. PMLR.

[Fortunato et al., 2019] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg (2019)
Noisy Networks for Exploration
arXiv preprint arXiv:1706.10295.

References - Part 3

[Hessel et al., 2017] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver (2017)
Rainbow: Combining Improvements in Deep Reinforcement Learning
arXiv preprint arXiv:1710.02298.

[Williams, 1992] Ronald J. Williams (1992)
Simple statistical gradient-following algorithms for connectionist reinforcement learning
Machine Learning 8(3-4), 229–256.

[Mnih et al., 2016] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016)
Asynchronous Methods for Deep Reinforcement Learning
arXiv preprint arXiv:1602.01783.

[Bellemare et al., 2017] Marc G. Bellemare, Will Dabney, and Rémi Munos (2017)
A Distributional Perspective on Reinforcement Learning
arXiv preprint arXiv:1707.06887.

References - Part 4

[Akiba et al., 2019] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama (2019)

Optuna: A Next-generation Hyperparameter Optimization Framework

In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[Schulman et al., 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017)

Proximal Policy Optimization Algorithms

arXiv preprint arXiv:1707.06347.

[Bellman, 1957] Richard Bellman (1957)

Dynamic Programming

Princeton University Press, Princeton Mathematical Series, Volume 1.

See you soon at the lab session!

Installation Toolkit

- **Installation Toolkit:**
  - **Conda Environment:** Anaconda or Miniconda
  - **IDE:** Install an Integrated Development Environment (IDE) like PyCharm or VSCode for coding
  - **Requirements File:** Download the `environment.yml` file from `https://terranovafr.github.io/teaching/2024-EASSS-Course`.
    - Use the command: `conda env create -f environment.yml` to install the necessary libraries