Lecture 12: Reinforcement Learning

https://github.com/kaopanboonyuen/SC310005_ArtificialIntelligence_2025s1

Teerapong Panboonyuen
https://kaopanboonyuen.github.io

Reference

- https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf
- https://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture14.pdf
- https://rail.eecs.berkeley.edu/deeprlcourse/deeprlcourse/static/slides/lec-4.pdf
- https://www.cs.toronto.edu/~rgrosse/courses/csc2515_2019/slides/lec10-slides.pdf
- https://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L5.pdf
- https://magnet99.creatorlink.net/Reinforcement-Learning

What is Reinforcement Learning?

Learning from Experience

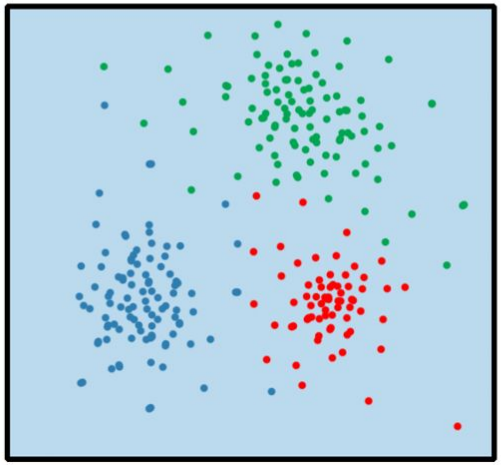
Reinforcement Learning (RL) is a subfield of machine learning where an **agent** learns to make decisions by performing **actions** in an **environment**. The agent receives **rewards** or penalties based on its actions, and its goal is to learn a strategy (or "policy") to maximize the cumulative reward over time.

Think of it like training a pet: you give it a treat (reward) for a desired behavior, and nothing (or a gentle "no") for an undesired one. Over time, the pet learns which behaviors lead to treats.

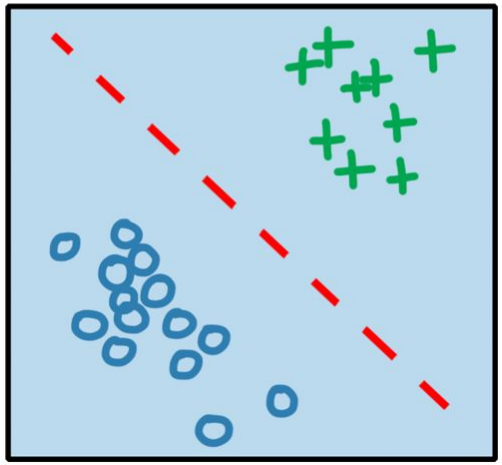
- **Agent:** The learner or decision-maker.
- **Environment:** The world the agent interacts with.
- **Actions:** The moves the agent can make.
- **Reward:** The feedback the agent receives for its actions.

machine learning

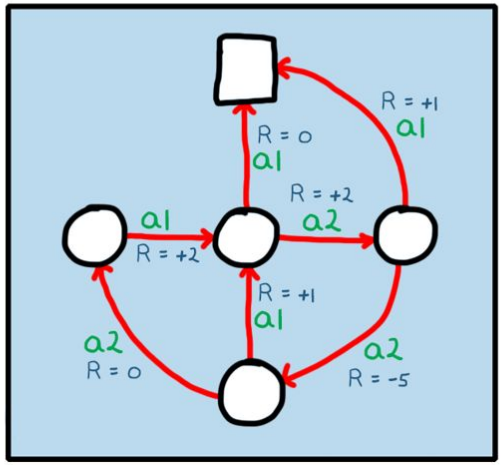
unsupervised learning



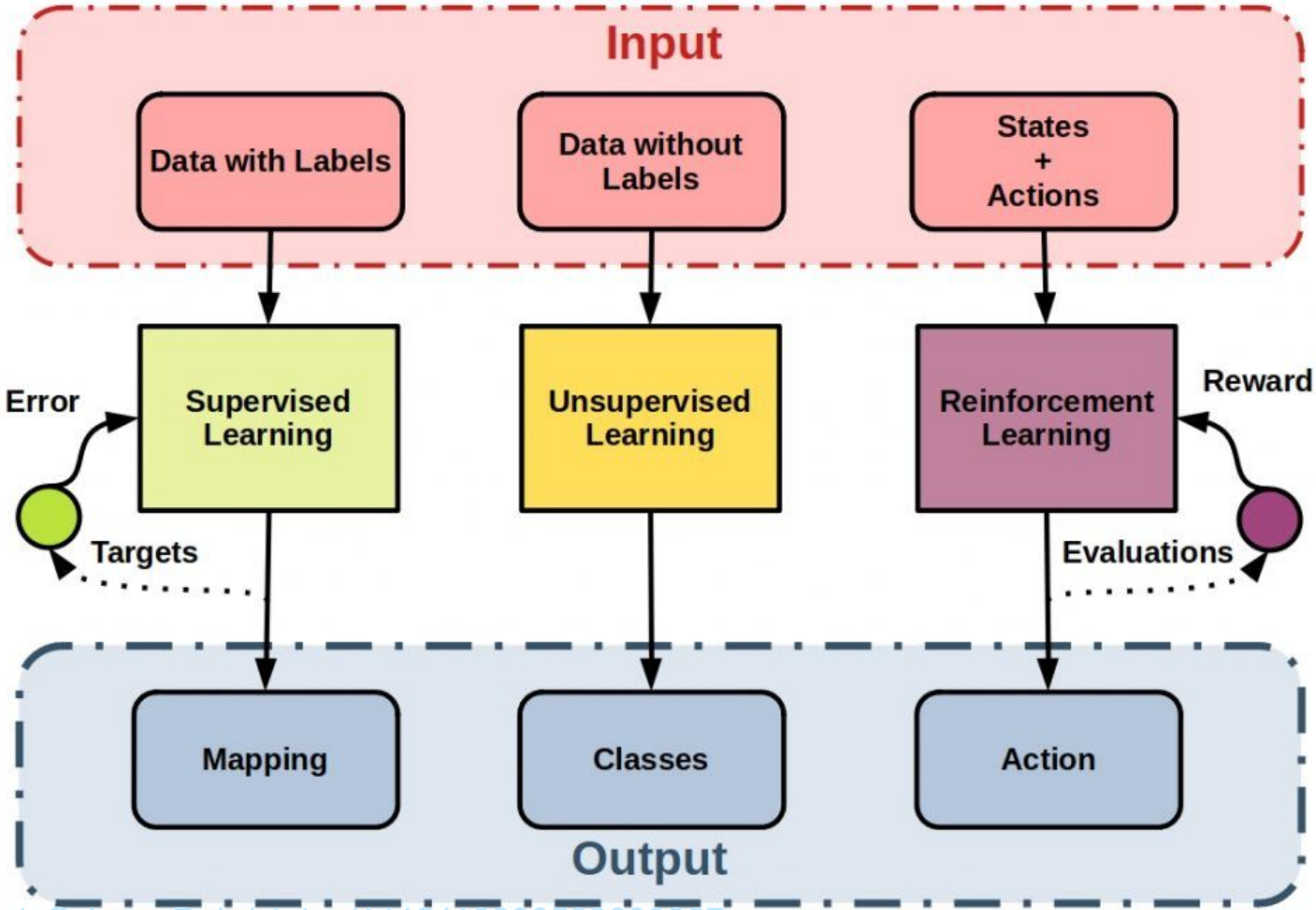
supervised learning



reinforcement learning



https://www.mathworks.com/discovery/reinforcement-learning.html



https://x.com/DataScienceDojo/status/1412110800593039367

Three classes of learning problems

Supervised Learning

Data: (x, y)
x is an input data, y is a label (e.g. photo with label "cat")

Goal: Learn to map input to output i.e. $\boldsymbol{x} \rightarrow \boldsymbol{y}$

An example: to classify

This is a cat

Unsupervised Learning

Data: x
x is data, there's no labels!

Goal: Learn an underlying structure of the data.

An example: Comparison

The two things are alike

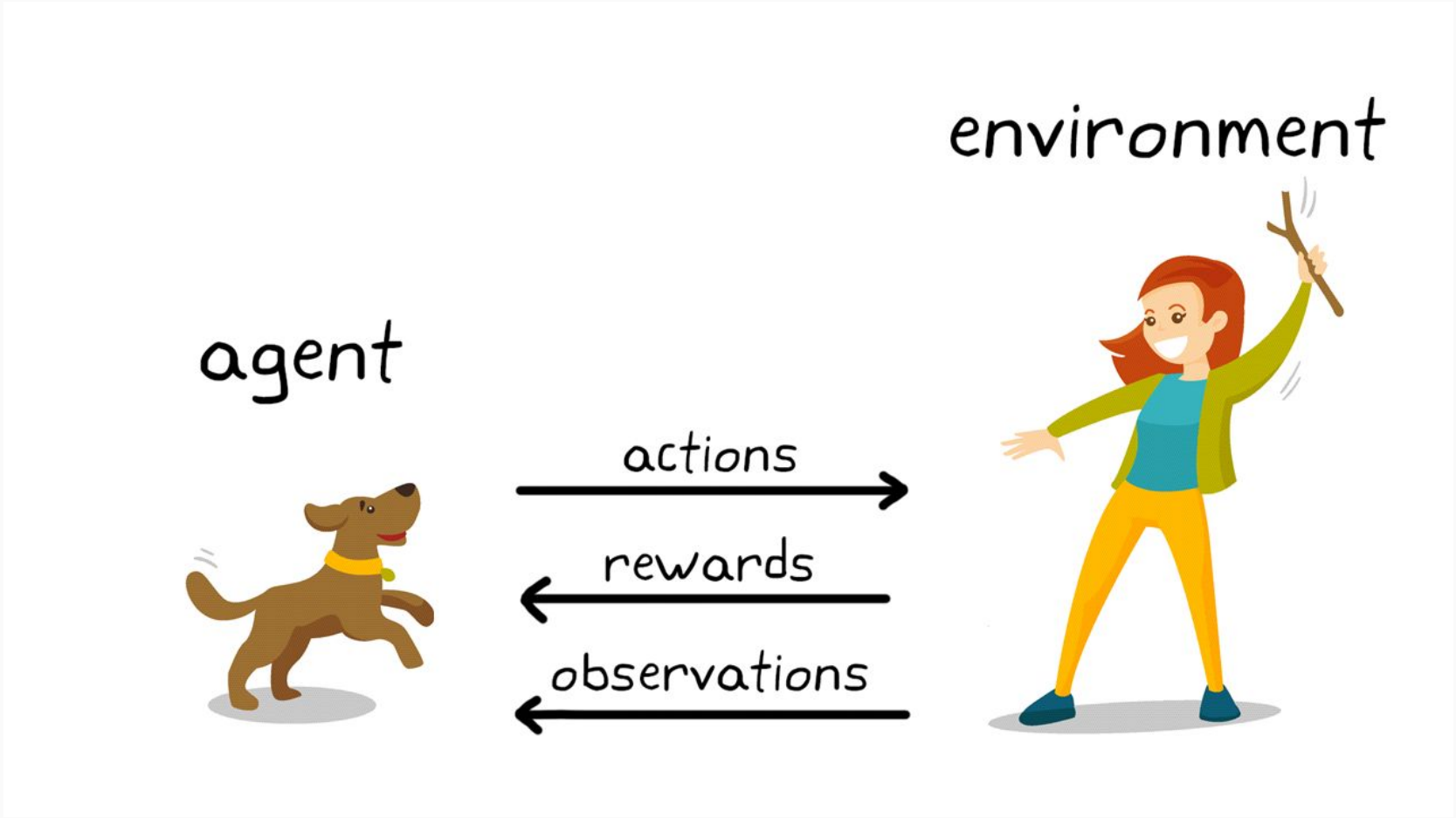
Reinforcement Learning

Data: No data, Only state-action pairs (s, a).

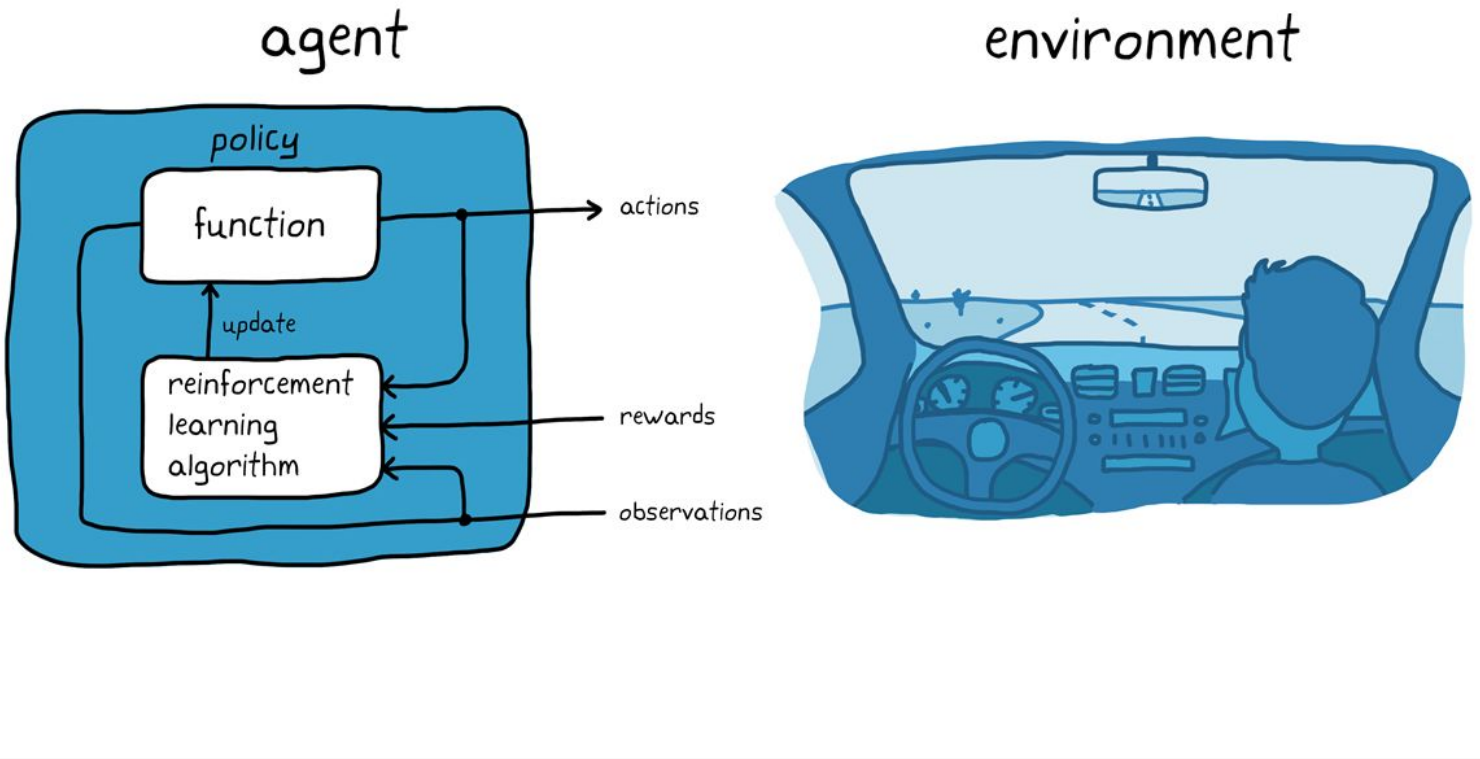
Goal: Maximize future reward over many time steps.

An example: reward = joy

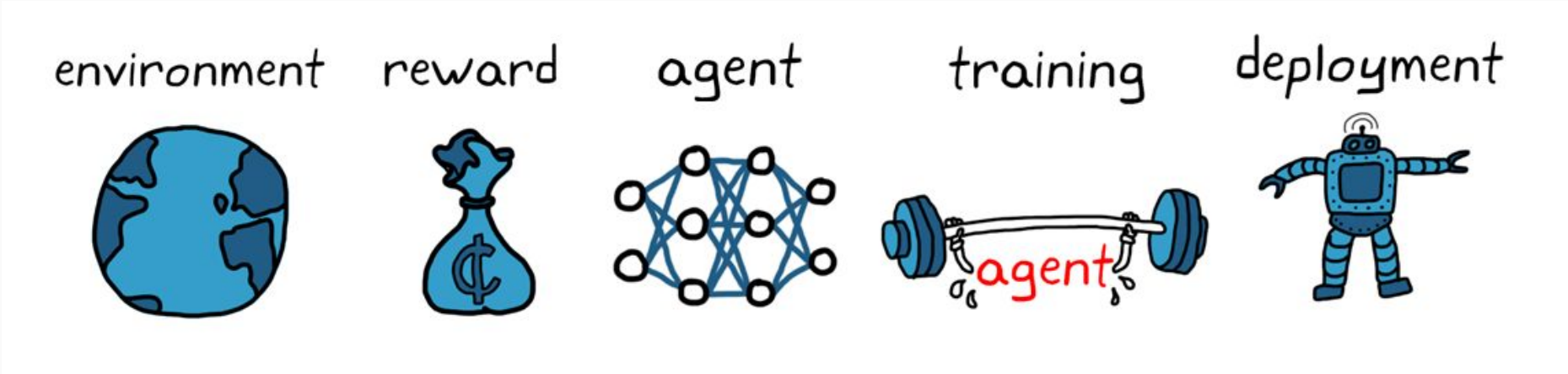
Interaction with the cat gives joy



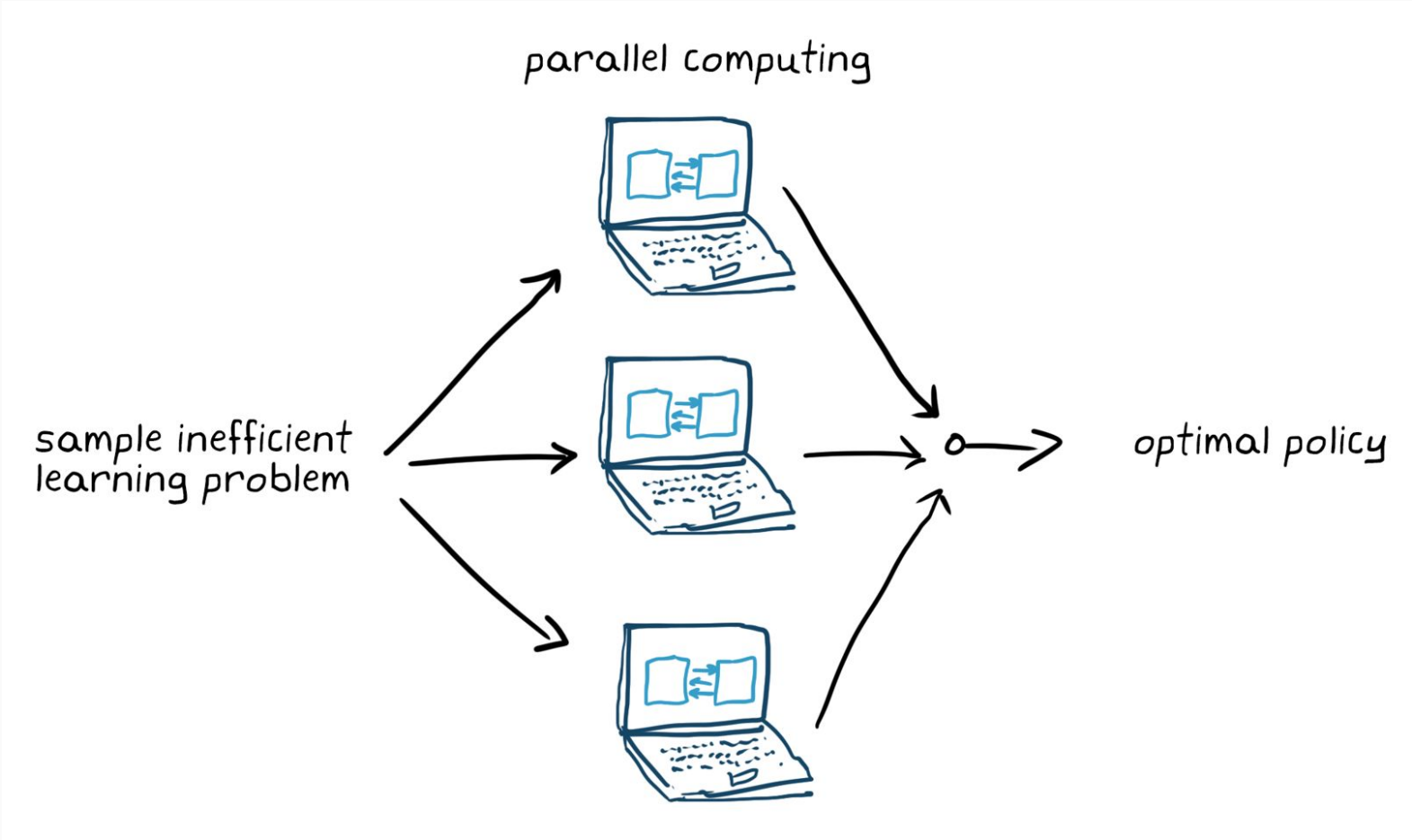
https://www.mathworks.com/discovery/reinforcement-learning.html



https://www.mathworks.com/discovery/reinforcement-learning.html



https://www.mathworks.com/discovery/reinforcement-learning.html



https://www.mathworks.com/discovery/reinforcement-learning.html

What is Reinforcement Learning (RL)?

- RL is a type of **machine learning** where an agent learns to make decisions by interacting with an **environment**.
- The agent aims to **maximize cumulative reward** over time.
- Applications: robotics, games, recommendation systems, autonomous driving.

Expectations and stochastic systems

$$\theta^\star = \arg\max_\theta E_{(\mathbf{s},\mathbf{a})\sim p_\theta(\mathbf{s},\mathbf{a})}[r(\mathbf{s},\mathbf{a})]$$

infinite horizon case

$$\theta^\star = \arg\max_\theta \sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)]$$

finite horizon case

In RL, we almost always care about *expectations*

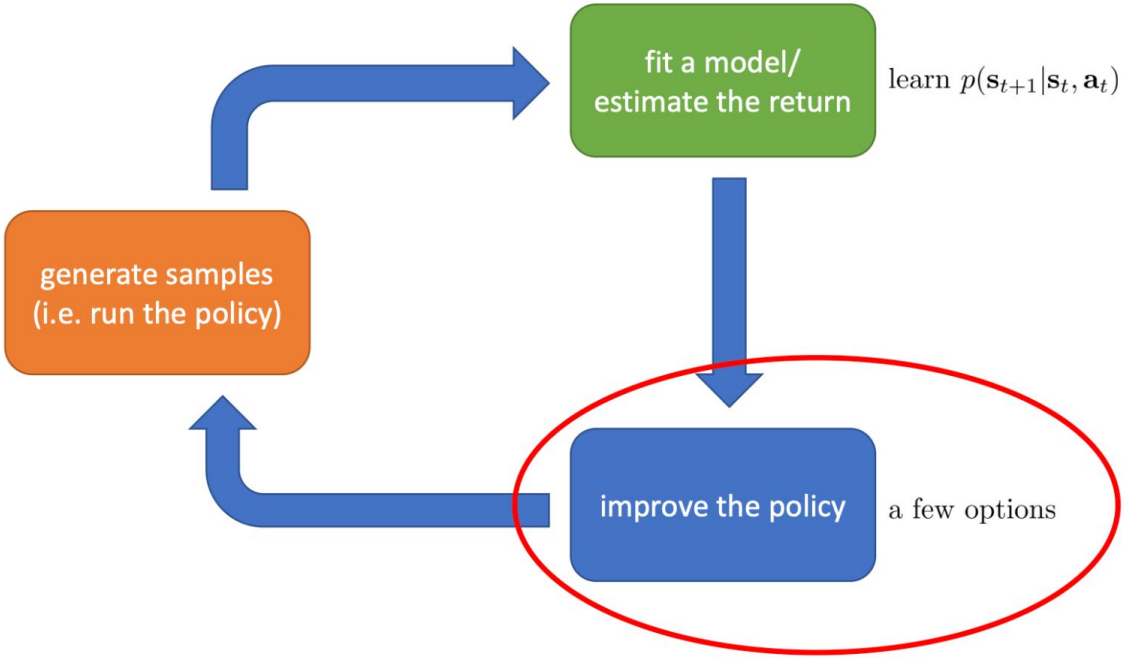
$r(\mathbf{x})$ – *not* smooth

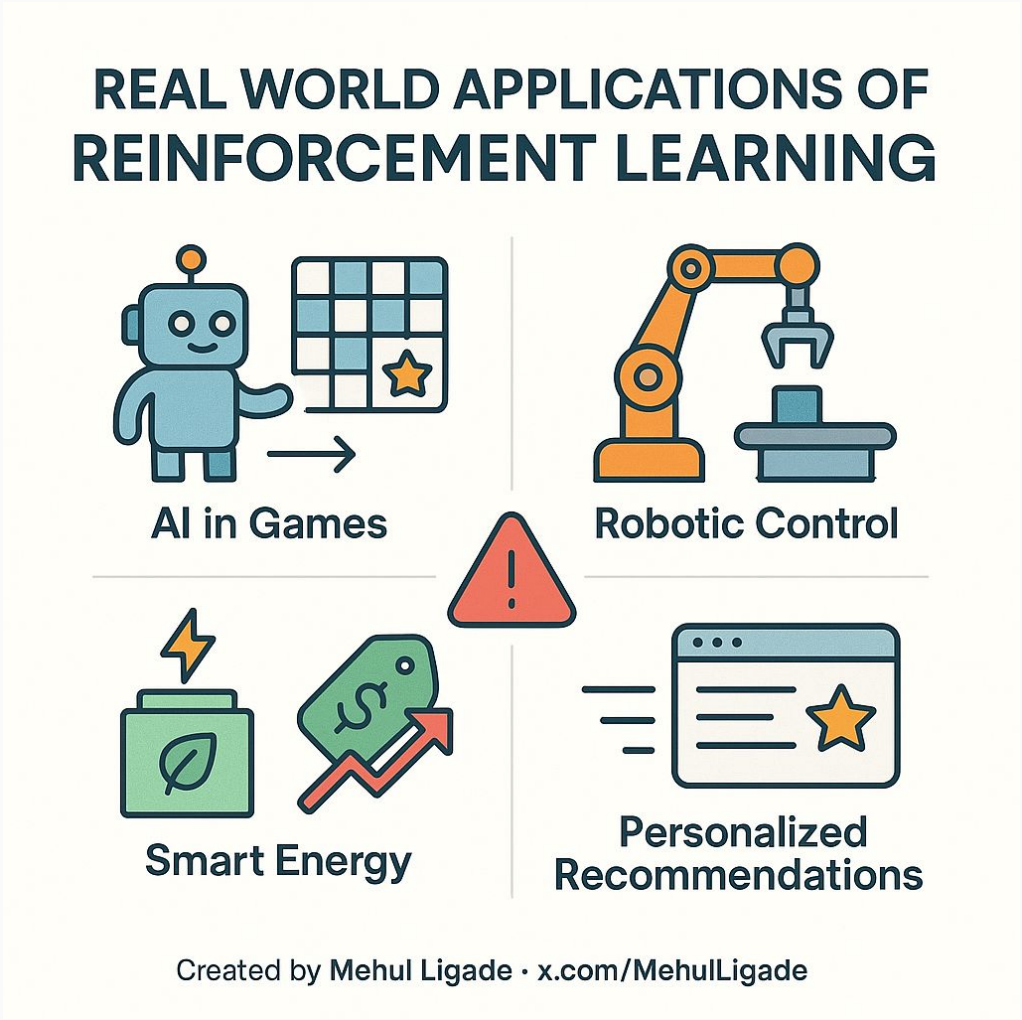
$\pi_\theta(\mathbf{a} = \text{fall}) = \theta$

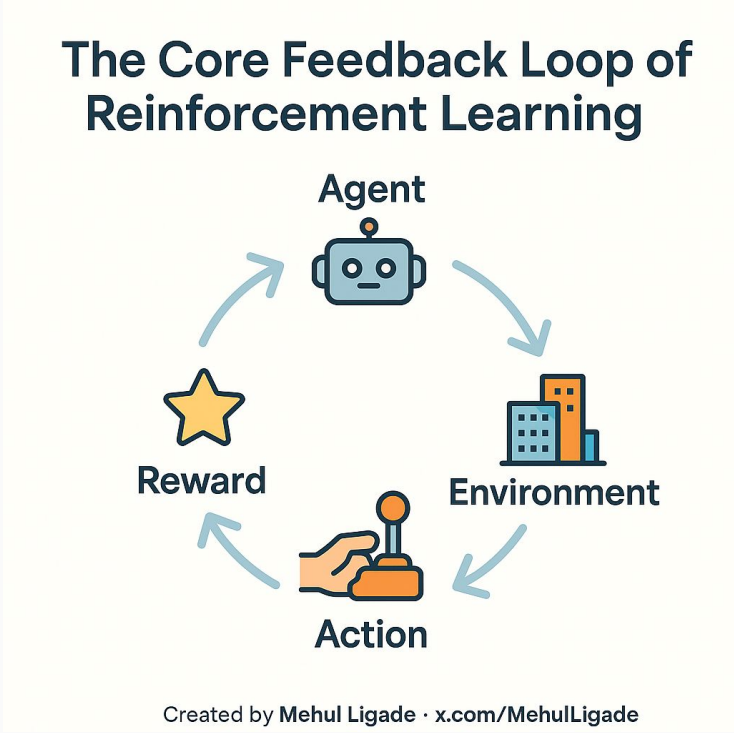
$E_{\pi_\theta}[r(\mathbf{x})]$ – *smooth* in θ!

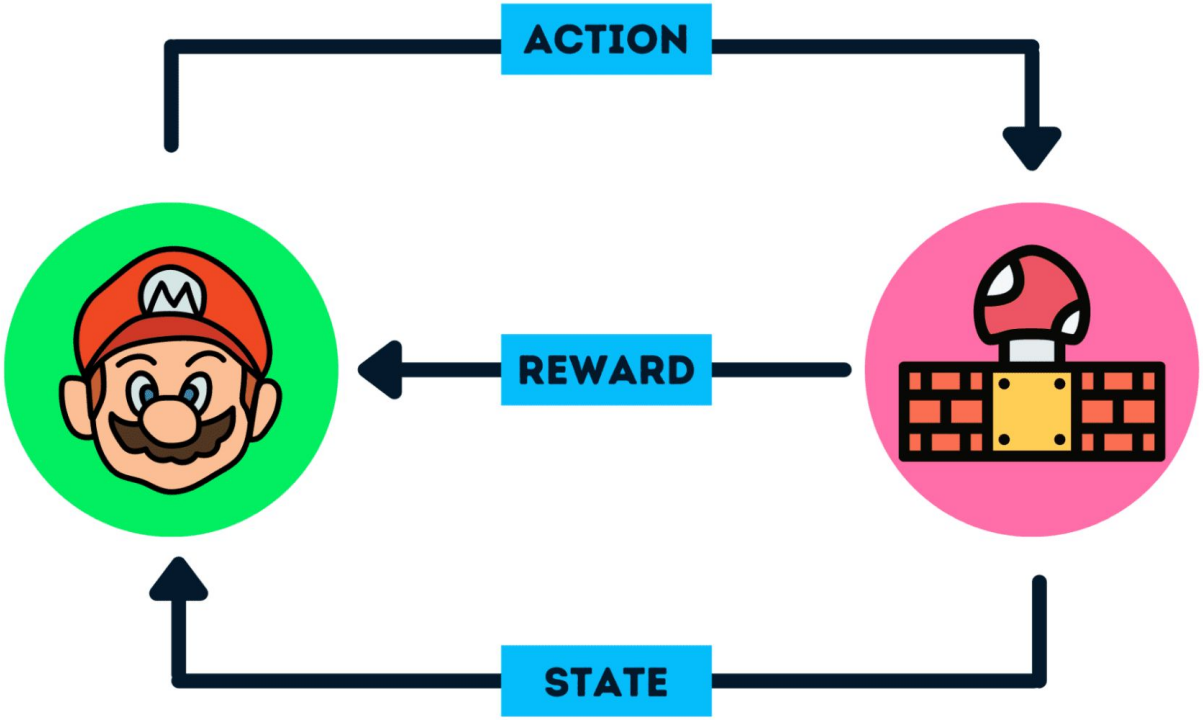
Credit: CS 285 Instructor: Sergey Levine UC Berkeley

Model-based RL algorithms

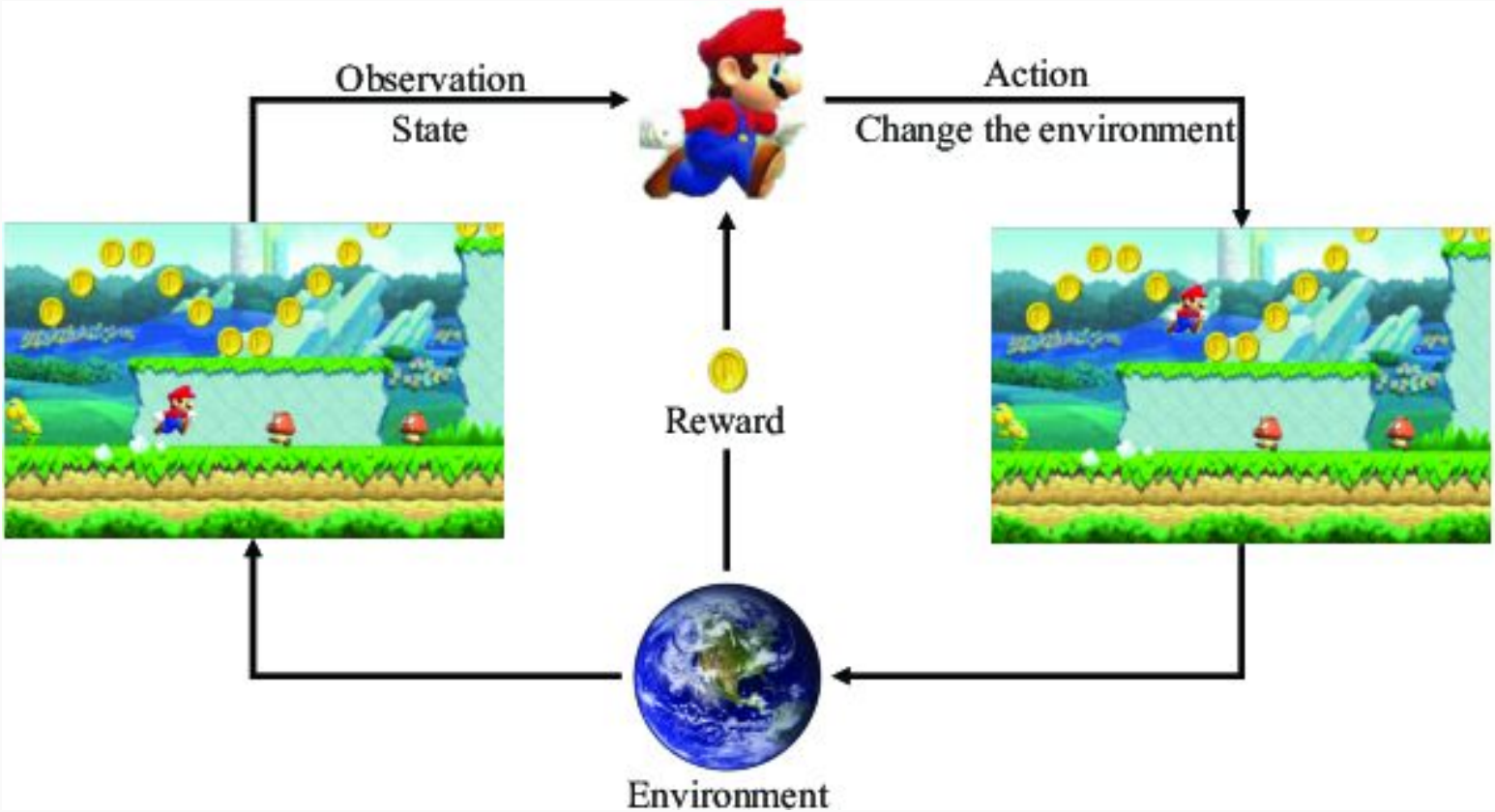



REAL WORLD APPLICATIONS OF
REINFORCEMENT LEARNING
AI in Games
Robotic Control
Smart Energy
Personalized
Recommendations
Created by Mehul Ligade · x.com/MehulLigade

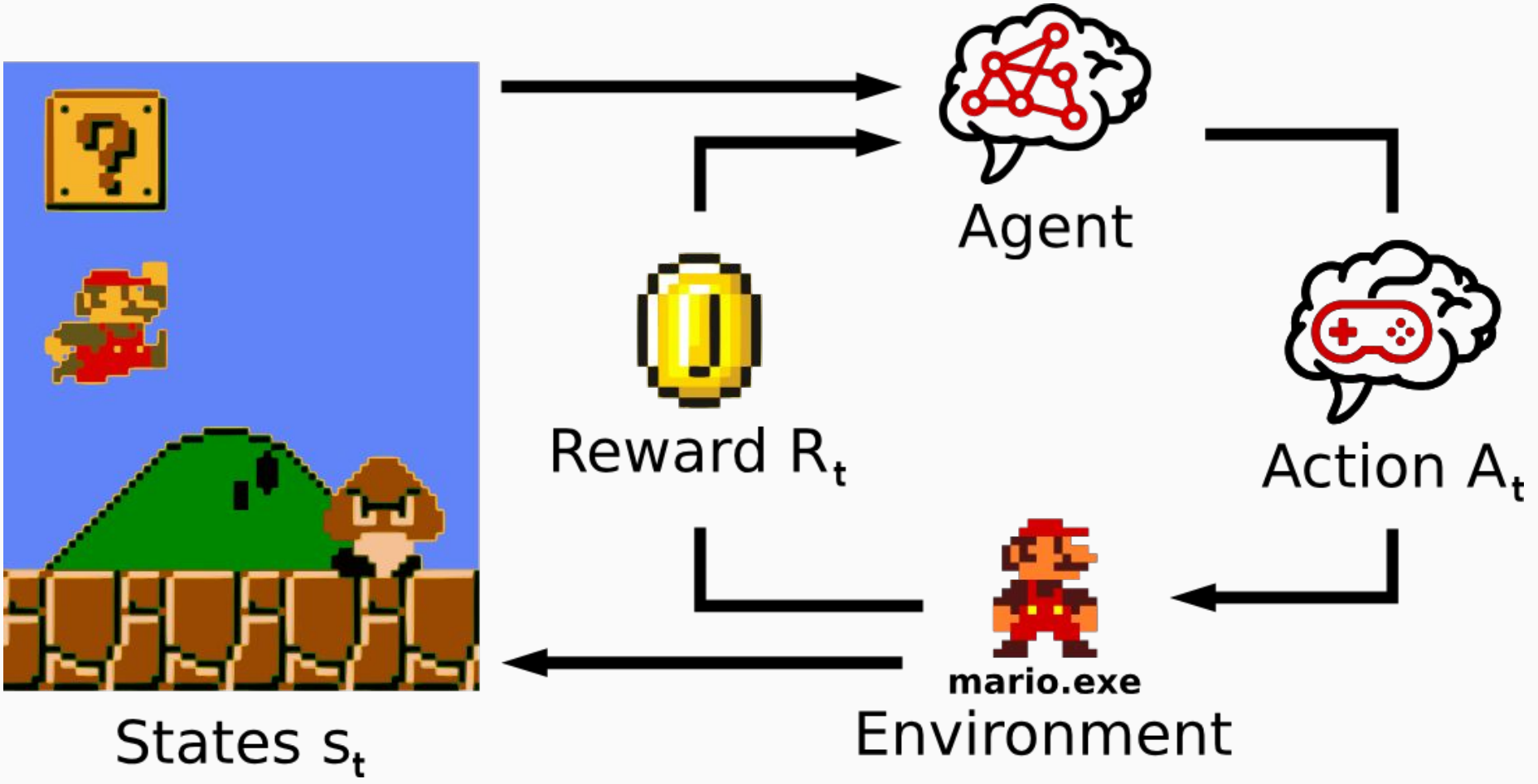

The Core Feedback Loop of
Reinforcement Learning
Agent
Environment
Action
Reward
Created by Mehul Ligade · x.com/MehulLigade


ACTION
REWARD
STATE

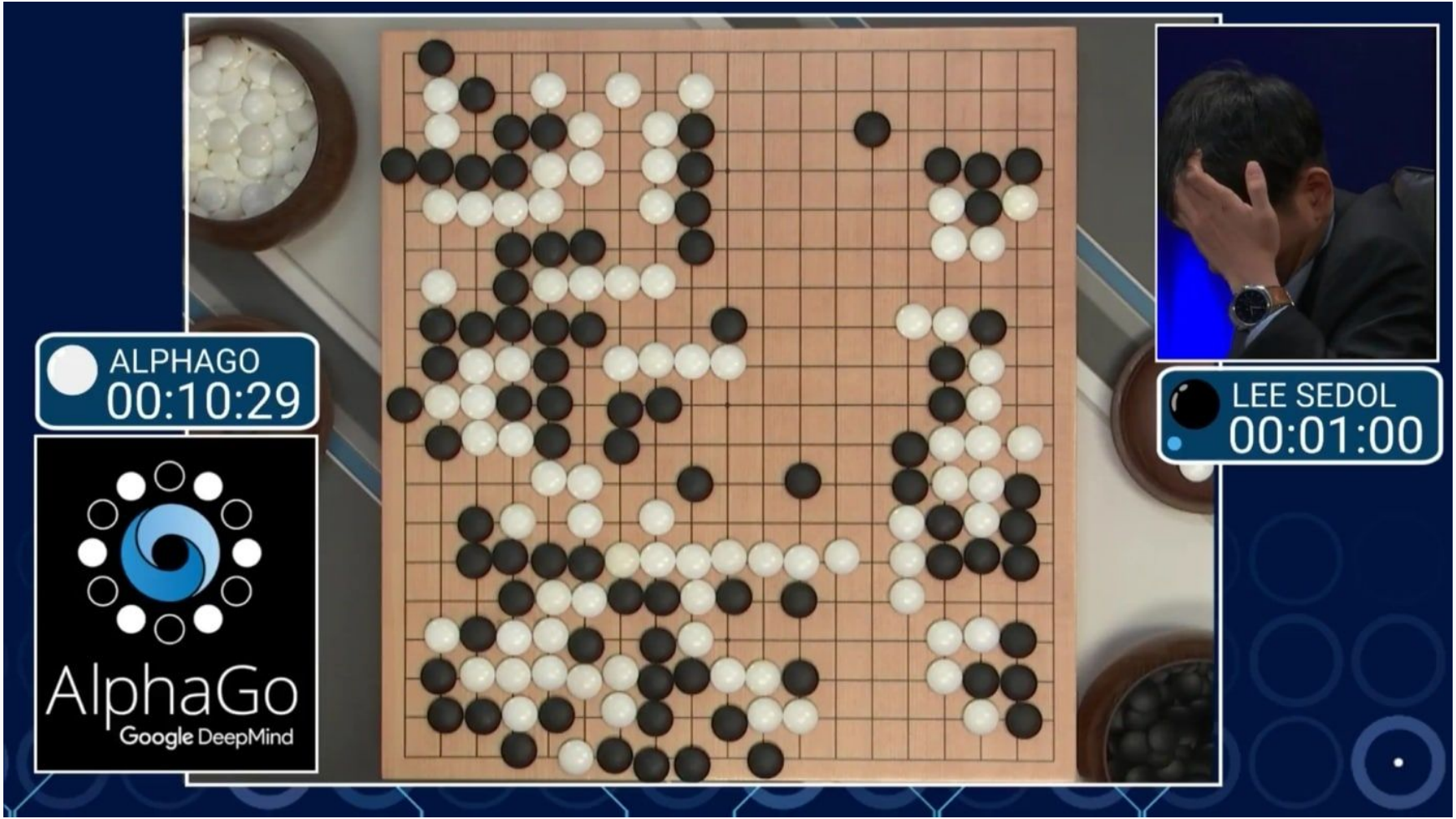
https://www.researchgate.net/figure/The-framework-of-reinforcement-learning-The-agent-performs-an-action-after-observing-the_fig1_365965695

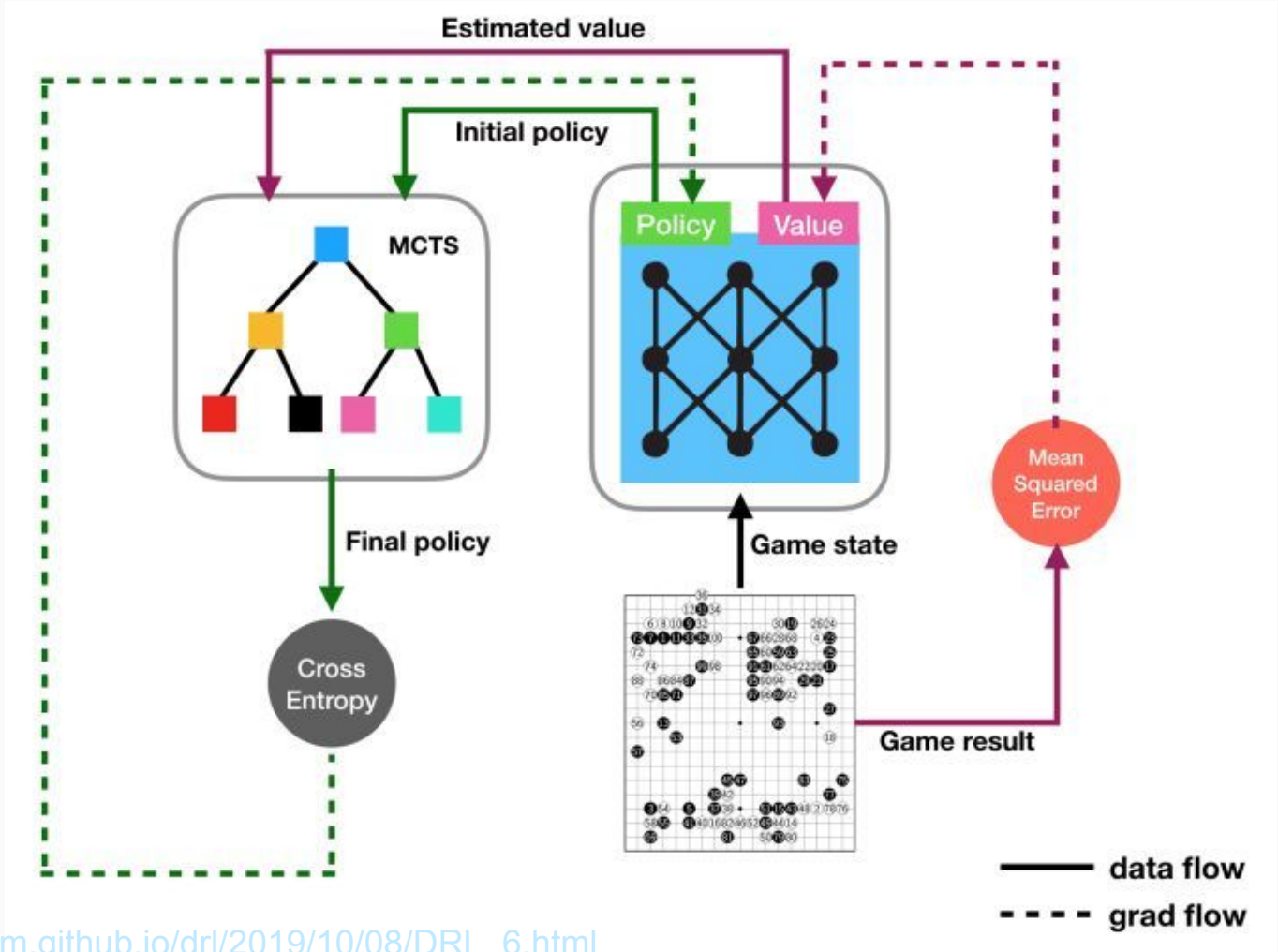


https://siegel.work/blog/RLModelBased/

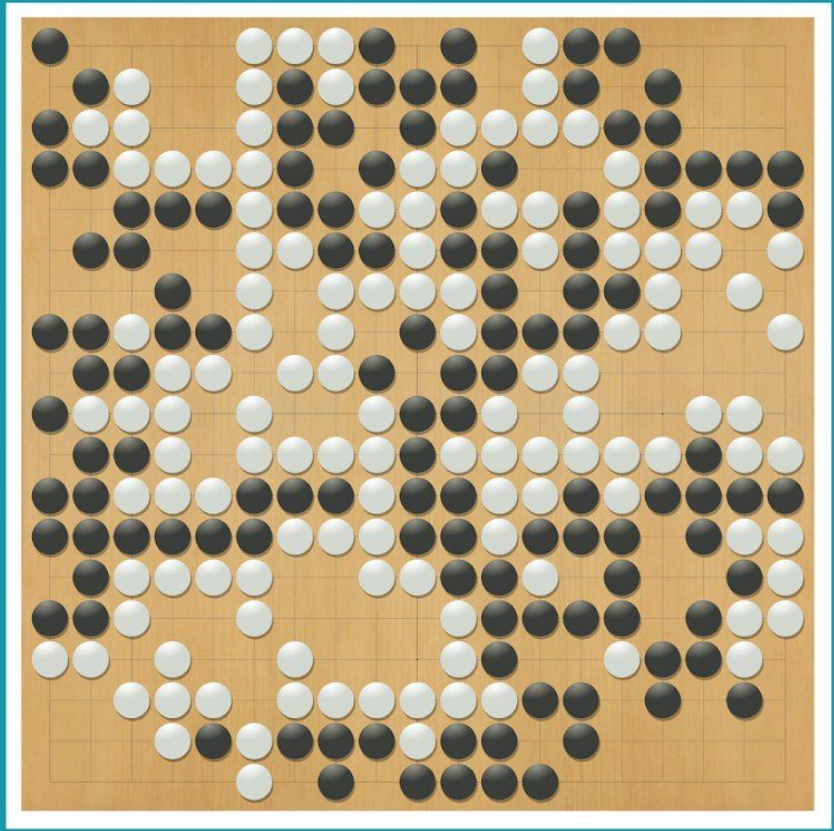



ALPHAGO
00:00:54
AlphaGo
Lee Sedol


ALPHAGO
00:10:29
AlphaGo
Google DeepMind
LEE SEDOL
00:01:00



https://antkillerfarm.github.io/drl/2019/10/08/DRL_6.html



THE ULTIMATE GO CHALLENGE

GAME 5 OF 5

15 MARCH 2016

AlphaGo
Won 4 of 5

vs

Lee Sedol
Won 1 of 5

RESULT	NUMBER OF MOVES	TIME WHITE	TIME BLACK
W+ Res	280	2h+	2h+

Introduction to RL

- Agent ↔ Environment interaction
- **Key idea:** Learn from experience (trial & error)
- No explicit labeled data like supervised learning
- Uses **feedback signals (rewards)** to improve decisions

Diagram:

mathematica

Copy code

```
Environment → State → Agent → Action → Environment → Reward → State
```

Core RL Concepts

- **State (s):** Representation of the environment at a time step
- **Action (a):** Choice made by the agent
- **Reward (r):** Feedback signal; positive or negative
- **Policy (π):** Strategy to select actions
- **Value Function (V):** Expected cumulative reward from a state
- **Q-Function (Q):** Expected cumulative reward from state-action pair

Simple Math for RL

- **Cumulative Reward:**

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

- γ = discount factor (0 ≤ γ ≤ 1)
- **Q-Learning Update:**

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_a Q(s',a) - Q(s,a)]$$

- **Policy Gradient (for Actor-Critic / PPO / A2C):**

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(a|s) \cdot \text{Advantage}]$$

Key Vocabulary

Term	Meaning
Agent	Learner or decision maker
Environment	System the agent interacts with
State	Snapshot of environment
Action	Decision taken by agent
Reward	Feedback signal
Policy	Action-selection strategy
Value Function	Expected cumulative reward from state
Exploration	Trying new actions
Exploitation	Using known best actions

Popular RL Algorithms

- **Q-Learning:** Tabular, model-free, off-policy
- **A2C (Advantage Actor-Critic):** Policy gradient + value function
- **PPO (Proximal Policy Optimization):** Policy gradient with stability improvements
- **Dyna-Q:** Model-based Q-Learning with planning

Sample Dataset Mapping

- Dataset: `RL_Online_Retail.csv` or `RL_Netflix_Users.csv`
- **States:** Customers (`CustomerID` or `User_ID`)
- **Actions:** Items / Genres (`StockCode` or `Favorite_Genre`)
- **Reward:** 1 if item matches purchase/favorite, 0 otherwise

Algorithm	How it interacts with data
Q-Learning	Update Q-table using observed state-action-reward-next_state
A2C	Learn policy & value function simultaneously
PPO	Optimize stochastic policy to maximize expected reward
Dyna-Q	Update Q-table & simulate experiences using a model

Evaluating RL Models

Metric	Definition
Average Reward	Mean reward per episode
Max Reward	Maximum reward achieved
Min Reward	Minimum reward achieved
Std Reward (Stability)	Standard deviation of rewards across episodes
Success Rate (%)	% of episodes where reward > threshold
Convergence (Episodes)	Episode when agent first reaches threshold reward

- **Purpose:** Measure effectiveness, stability, and efficiency of RL algorithms

Training RL Agents

- Reset environment → observe state
- Select action using policy or Q-table
- Apply action → get reward and next state
- Update policy/Q-table
- Repeat until episode ends
- Repeat for many episodes

Inference / Recommendation

- Load trained RL agent
- Pick a user (state)
- Predict action (item/genre)
- Map action to actual item/genre
- Evaluate reward or success

Key Teaching Points

- RL is about **learning from interaction**, not supervised labels
- **Exploration vs Exploitation** is critical
- Algorithm choice affects **sample efficiency, stability, and reward**
- Evaluation metrics help **compare models and interpret results**
- Using **datasets like Netflix or Online Retail**, students can practice **recommender-style RL**

Optional Extensions

- Model-based vs model-free comparison
- Multi-step episodes / sequential recommendation
- Hyperparameter tuning (learning rate, gamma, epsilon)
- Visualize Q-tables or policy probabilities

RL for Retail: dataset → environment (quick map)

- **Dataset (online retail)**: transactions table (InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID, Country). ~541,909 rows — use `pd.read_csv(...)`.
- **Teaching goal**: show RL as *decision-making over customers* (a recommender / business decision agent).
- **Mapping (core RL components)**
  - **Agent**: recommender/decision maker.
  - **Environment**: historical transaction dataset (serves observations & gives reward).
  - **State / Observation**: current customer index (one `CustomerID`) → can enrich with features (RFM: recency, frequency, monetary; country; past-basket).
  - **Action**: choose one item index from `self.items` (`spaces.Discrete(n_actions)`).
  - **Reward (current code)**: binary match

```py
reward = 1 if self.items[action] in customer_bought else 0
```

  - **Episode flow**: `reset()` → iterate customers; `step(action)` returns `(next_obs, reward, done, truncated, info)`.
- **Code variables -> meaning**: `self.customers` = list of states; `self.items` = action space; `action_space` = discrete items; `observation_space` = discrete customers.

- **What RL is predicting**: the *best action* for a given state (e.g., which intervention or recommendation for this customer) to **maximize cumulative reward** over customers/episodes.
- **Better reward designs (beyond binary)**
  - **Revenue**: `UnitPrice × Quantity` for matched item.
  - **Quantity-based**: number of units purchased if recommended.
  - **Conversion/Retention**: +1 if customer returns (after promotion action).
  - **Top-K/Rank metrics**: use precision@K / recall@K when recommending baskets.
  - **Cost-aware**: revenue – promotion cost (useful for coupons/discount decisions).
- **Policy families to teach**
  - Baselines: **random**, **popularity** (most-frequent item for customer group).
  - Exploration: **ε-greedy** (simple, teach exploration vs exploitation).
  - Learning: **value-based** (Q-learning / DQN), **policy gradient** (REINFORCE / PPO) or **offline RL** on logged data.
  - Rule-based experiments: RFM thresholds → deterministic actions.

- **Evaluation metrics & experiments**
  - **Cumulative reward** (sum across episode) — core RL metric.
  - Business metrics: **revenue uplift**, **conversion rate**, **precision@K**, **recall@K**.
  - Experiments: ablation over reward functions; compare policies (random / heuristic / learned); visualize learning curve (reward vs episode).
- **Class assignment ideas (short)**
  1. Implement `RetailEnv` → run random + popularity policies → compare cumulative reward.
  2. Replace reward with revenue → re-run policies, report change in metrics.
  3. Implement ε-greedy + Q-table (small subset of items) and show learning.
  4. Design a top-K recommender and evaluate precision@K on held-out customers.

```
url = 'https://raw.githubusercontent.com/kaopanboonyuen/panboonyuen_dataset/main/public_dataset/RL_Online_Retail.csv'
```

```
1 df
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
...	...	...	...	...	...	...	...	...
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	12680.0	France
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France

541909 rows × 8 columns

The RL Environment: `RetailEnv`

- RL requires an **environment** where the agent interacts.
- The environment provides:
  - **State** (what the agent observes)
  - **Reward** (feedback signal)
  - **Done flag** (episode termination)
- `RetailEnv` is a **discrete recommendation environment** based on retail transactions.
- Each **customer** is a state, each **item** is a possible action.
- Goal: Recommend items that match customers' purchases.

Class Structure: `RetailEnv`

python

Copy code

```python
class RetailEnv(gym.Env):
    metadata = {"render_modes": ["human"]}
```

- Inherits from `gym.Env` → standard RL interface.
- `metadata` defines optional render modes (human, rgb_array, etc.).
- Key takeaway: Every RL environment needs a **state space, action space, reset(), and step()**.

Initializing the Environment

python Copy code

```python
def __init__(self, data):
    super().__init__()
    self.data = data
    self.customers = data['CustomerID'].dropna().unique()
    self.items = data['StockCode'].unique()
    self.n_actions = len(self.items)
    self.action_space = spaces.Discrete(self.n_actions)
    self.observation_space = spaces.Discrete(len(self.customers))
    self.current_customer_idx = 0
```

- `data` : raw retail transaction dataframe
- `customers` : all unique customers → each customer is a **state**
- `items` : all unique stock codes → each item is an **action**
- `n_actions` : total number of actions (items)
- `action_space` : discrete actions → gym standard
- `observation_space` : discrete states → gym standard
- `current_customer_idx` : tracks which customer is active in episode

Resetting the Environment

python

Copy code

```python
def reset(self, seed=None, options=None):
    super().reset(seed=seed)
    self.current_customer_idx = 0
    return self.current_customer_idx, {}
```

- `reset()` starts a **new episode**
- Sets the **current customer index** to 0
- Returns **initial state**
- Key RL concept: Each episode starts from a fresh state
- Gym API: returns `(state, info)` tuple

Taking a Step in the Environment

python

Copy code

```python
def step(self, action):
    customer_id = self.customers[self.current_customer_idx]
    customer_bought = self.data[self.data['CustomerID']==customer_id]['StockCode'].values
    reward = 1 if self.items[action] in customer_bought else 0
    self.current_customer_idx += 1
    done = self.current_customer_idx >= len(self.customers)
    return (self.current_customer_idx if not done else 0, reward, done, False, {})
```

- `step(action)` → main interaction between agent and environment
- Input: `action` → index of item agent recommends
- Output: `(next_state, reward, done, truncated, info)`

Reward Mechanism

python

Copy code

```
reward = 1 if self.items[action] in customer_bought else 0
```

- If agent recommends an item **purchased by the current customer**, reward = 1
- Otherwise, reward = 0
- Simple binary reward encourages **matching recommendations**
- Key RL concept: Reward guides the agent's learning

Tracking State and Episode End

python

Copy code

```python
self.current_customer_idx += 1
done = self.current_customer_idx >= len(self.customers)
```

- Move to next customer
- `done = True` → all customers have been served
- Maps to **episode termination** in RL
- Allows agent to **experience all states once per episode**

Step Returns Explained

python Copy code

```python
return (self.current_customer_idx if not done else 0, reward, done, False, {})
```

- `next_state` : next customer index (or 0 if episode done)
- `reward` : 1 or 0 based on item match
- `done` : boolean → episode finished
- `truncated` : unused here (False)
- `info` : optional info dictionary for debugging

Rendering the Environment

python Copy code

```python
def render(self):
    pass
```

- `render()` displays environment state for human understanding
- Optional for learning; can implement visualization:
  - Current customer
  - Recommended item
  - Reward feedback
- Not essential for training

Q-Learning Overview

- **Type:** Tabular, model-free, off-policy
- **Key Idea:** Learn a Q-table mapping `(state, action)` → expected cumulative reward
- **Update Rule:**

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_a Q(s',a) - Q(s,a)]$$

- **Workflow:**
  1. Initialize Q-table with zeros
  2. For each episode:
     - Start at state `s`
     - Select action `a` (ε-greedy)
     - Receive reward `r` and next state `s'`
     - Update Q-table
     - Move to `s'`
  3. Repeat for all users

Dyna-Q Overview

- **Type:** Tabular, model-based
- **Key Idea:** Combines Q-Learning with planning via a learned model of environment
- **Workflow:**
  1. Observe `(s,a,r,s')` and update Q-table
  2. Store transition in model
  3. Perform `k` planning steps by sampling from model
  4. Update Q-table based on simulated transitions
- **Benefit:** Faster learning through simulated experience

PPO (Proximal Policy Optimization) Overview

- **Type:** Policy-gradient, model-free
- **Key Idea:** Optimize stochastic policy with stability constraints
- **Workflow:**
  1. Observe state `s`
  2. Sample action `a` from policy π
  3. Collect reward `r` and next state `s'`
  4. Compute advantage estimate
  5. Update policy network parameters
- **Why PPO:** Handles large action spaces & ensures stable updates

A2C (Advantage Actor-Critic) Overview

- **Type:** Actor-Critic, model-free
- **Key Idea:** Learns policy (actor) & value function (critic) simultaneously
- **Workflow:**
  1. Actor selects action based on policy π
  2. Critic evaluates value function `V(s)`
  3. Compute advantage `A(s,a) = r + γV(s') − V(s)`
  4. Update actor & critic parameters
- **Benefit:** Combines policy gradient with value estimation for efficiency

Mapping Dataset to Algorithms

Algorithm	How it interacts with `RL_Netflix_Users.csv`
Q-Learning	Update Q-table with (User_ID, Favorite_Genre, Reward, Next User)
Dyna-Q	Update Q-table + simulate user-genre interactions using model
PPO	Train policy network to select genres based on user state
A2C	Actor: genre policy, Critic: value of recommending a genre to user

Training Workflow

1. Initialize environment & agent
2. For each episode:
    - Reset environment → start with first user
    - While not done:
        - Agent selects action (genre)
        - Environment returns reward & next user
        - Agent updates Q-table or policy
3. Track **total reward per episode**
4. Repeat for `N` episodes

Tip: Episodes = one pass through all users

Evaluation Metrics

Metric	Definition
Average Reward	Mean reward per episode
Max Reward	Maximum reward achieved
Min Reward	Minimum reward achieved
Std Reward (Stability)	Standard deviation of rewards across episodes
Success Rate (%)	% of episodes where reward > threshold
Convergence (Episodes)	Episode when agent first reaches threshold reward

- **Purpose:** Compare effectiveness, stability, and sample efficiency of RL algorithms

Example Scenario (Student Exercise)

- Pick a user → state
- Recommend a genre → action
- Receive reward (1 if matches favorite, else 0)
- Repeat for all users → compute episode reward
- Apply Q-Learning, Dyna-Q, PPO, and A2C
- Track evaluation metrics

Students can compare how quickly each algorithm learns effective recommendations.

Key Teaching Points

- RL = learning from interaction, not labeled data
- **Exploration vs Exploitation** is critical for learning
- Algorithm choice affects:
  - Learning speed (convergence)
  - Stability (reward variance)
  - Average cumulative reward
- Metrics allow **quantitative comparison** between algorithms

Q&A
in RL?