AI Robotics

Reinforcement Learning

Overview

Robot Model

Sensor Measurements:

$Z_{0:t} = \{z_0, \ldots, z_t\}$

Robot State (or pose):

$X_{0:t} = \{X_0, \ldots, X_t\}$

sense

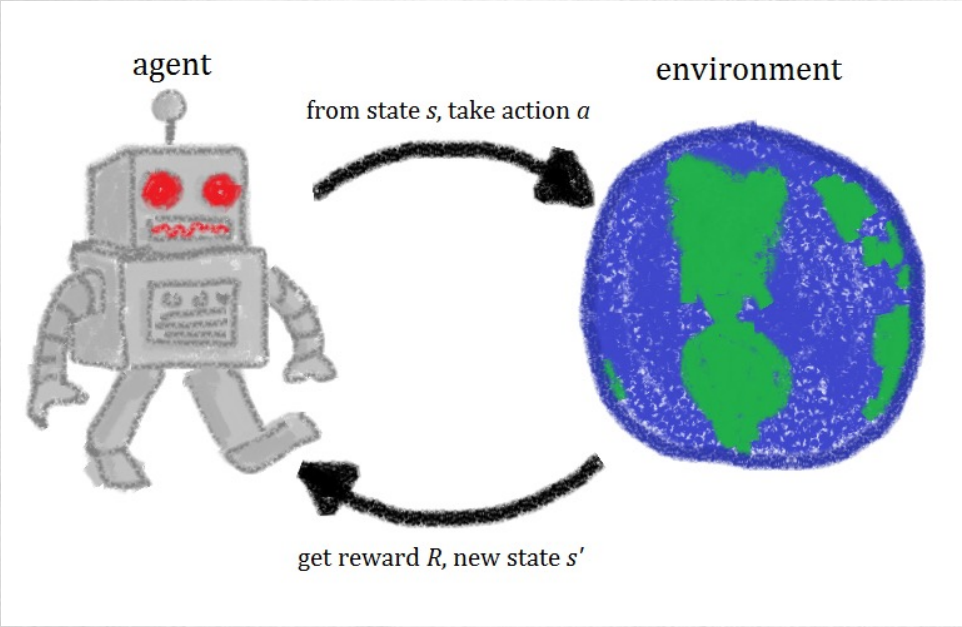
Environment

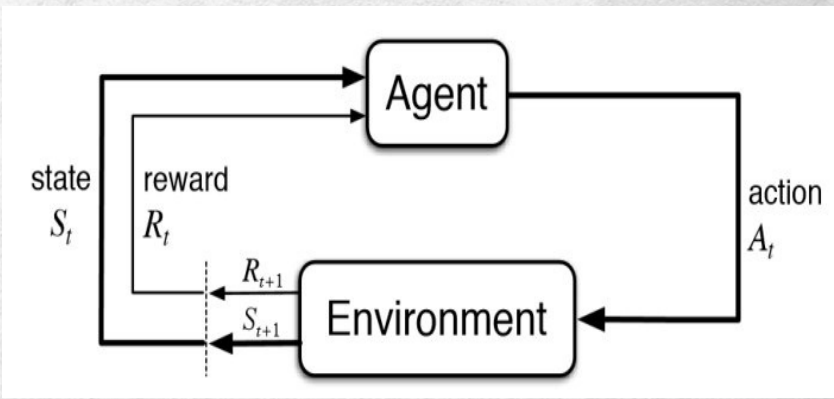
act

Robot Controls:

$u_{0:t} = \{u_0, u_1, \ldots, u_t\}$

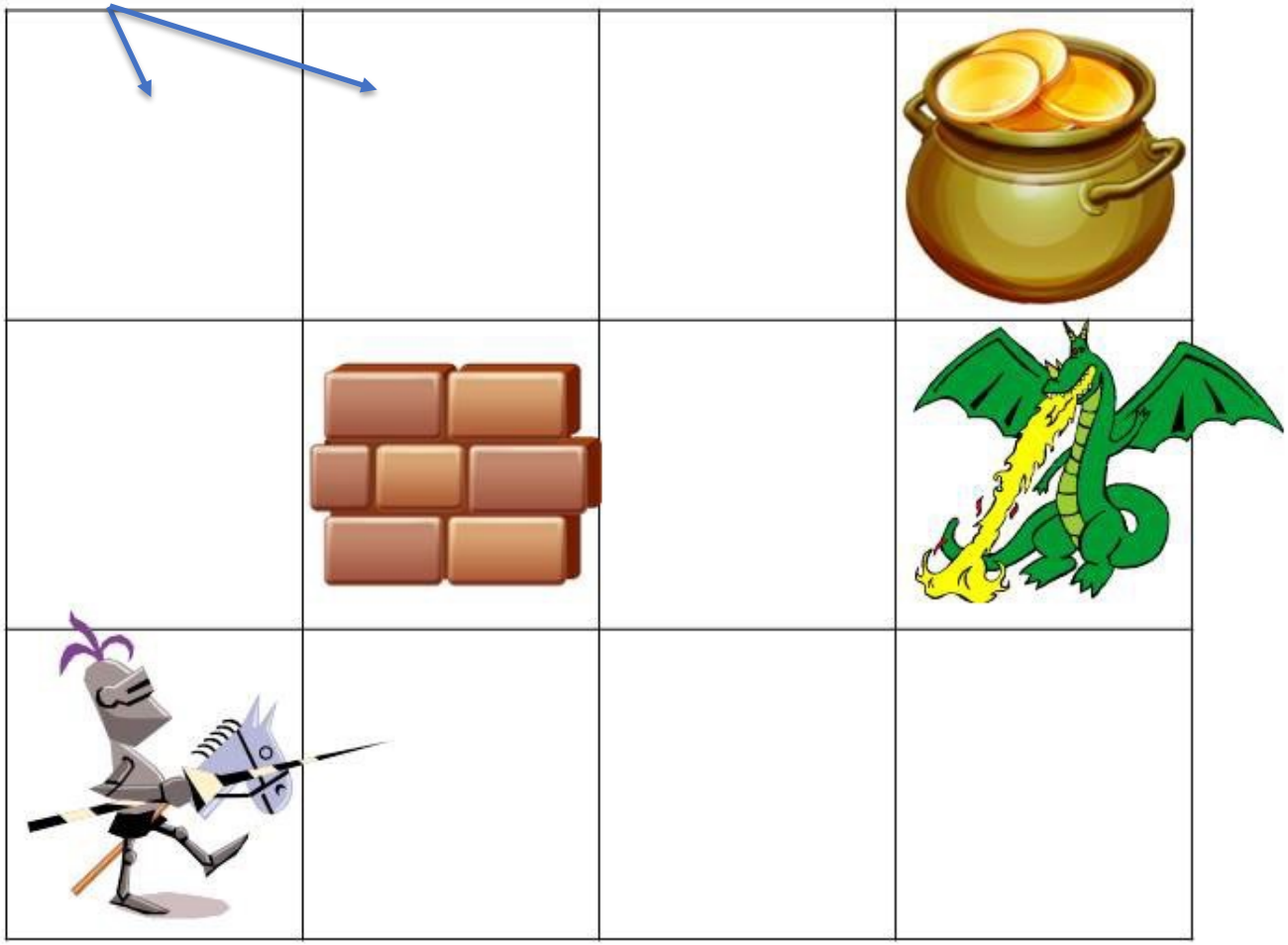
Simplified Model



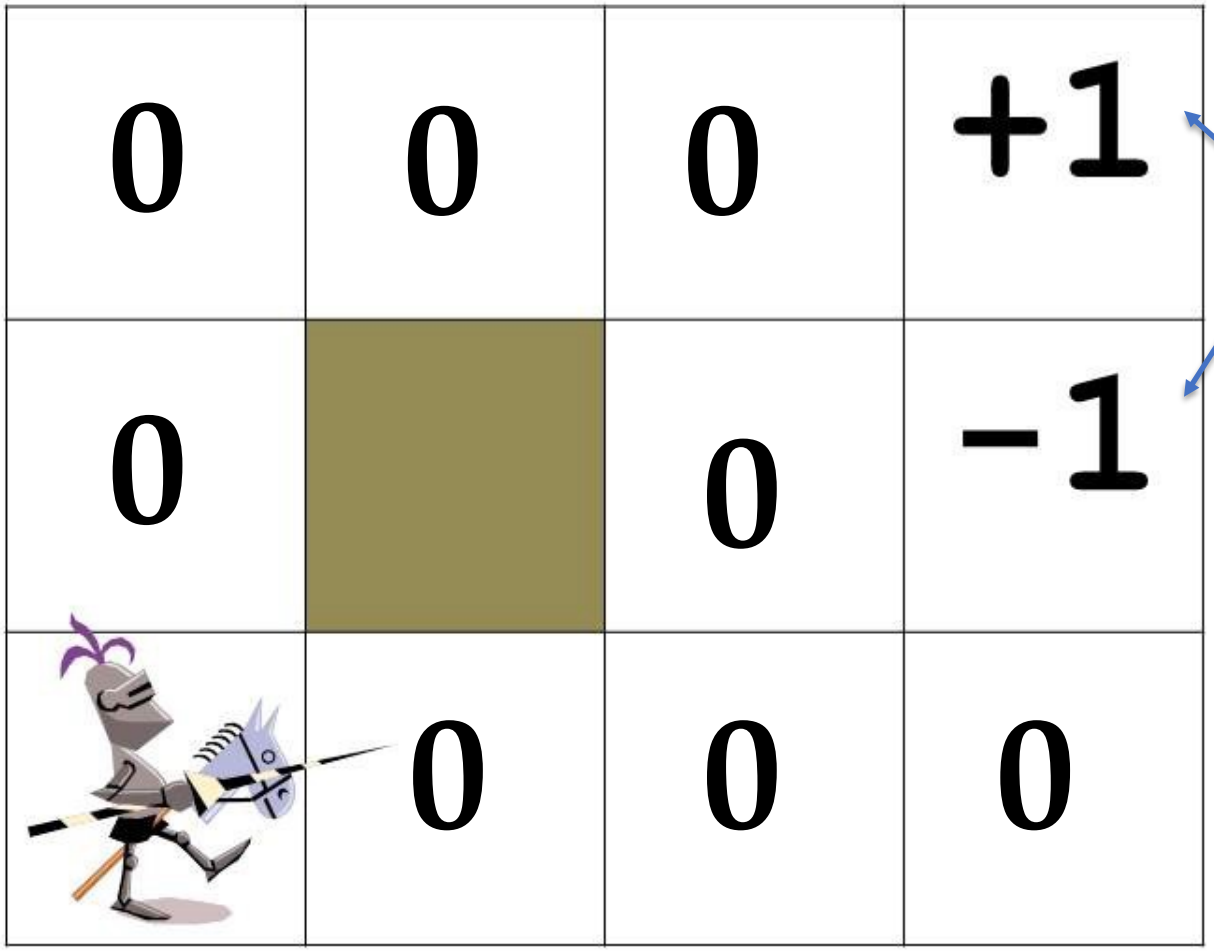


Example: Navigation on a Grid

States, x

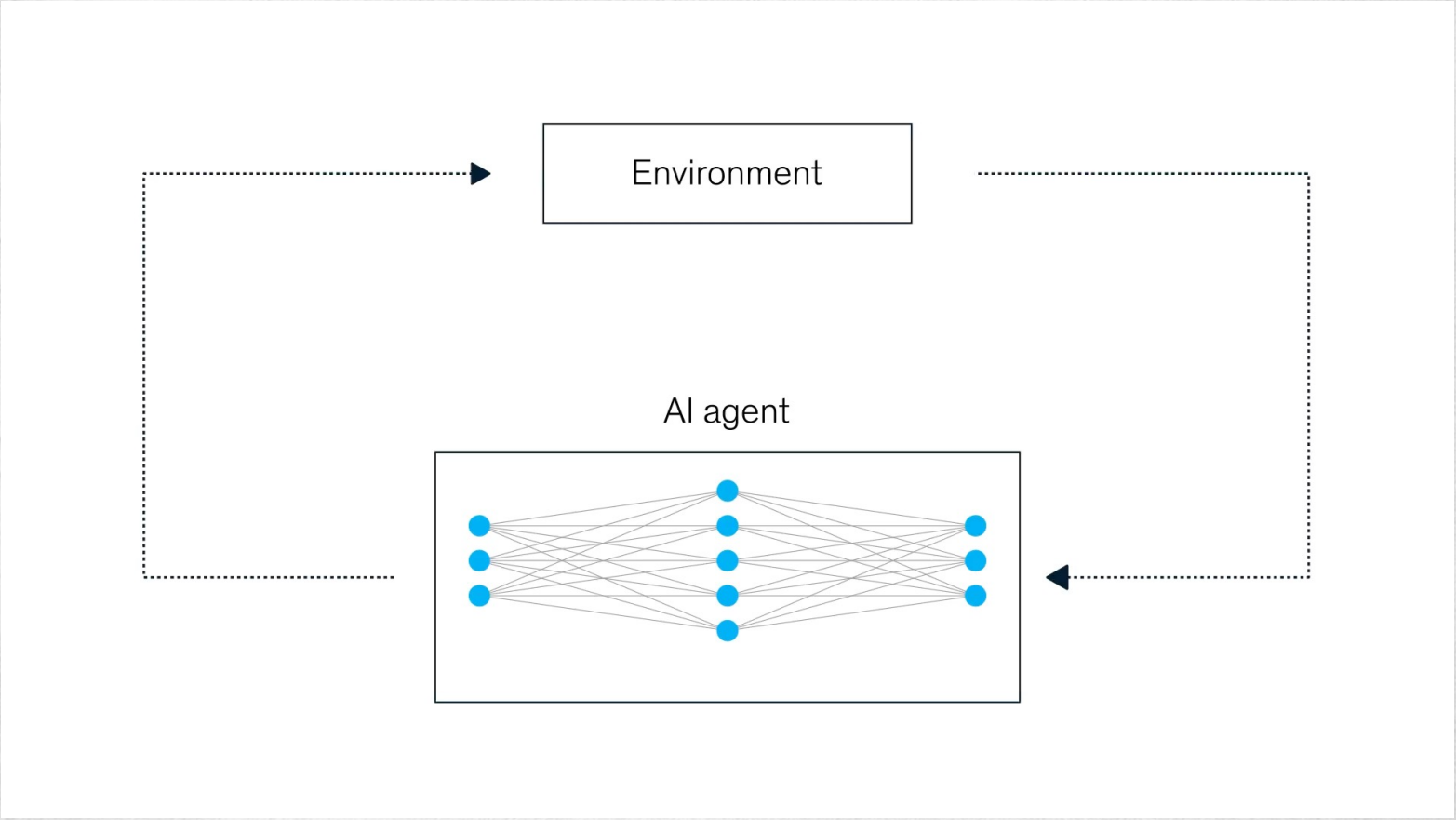


Example: Navigation on a Grid



Reinforcement Learning

Learning from Experience



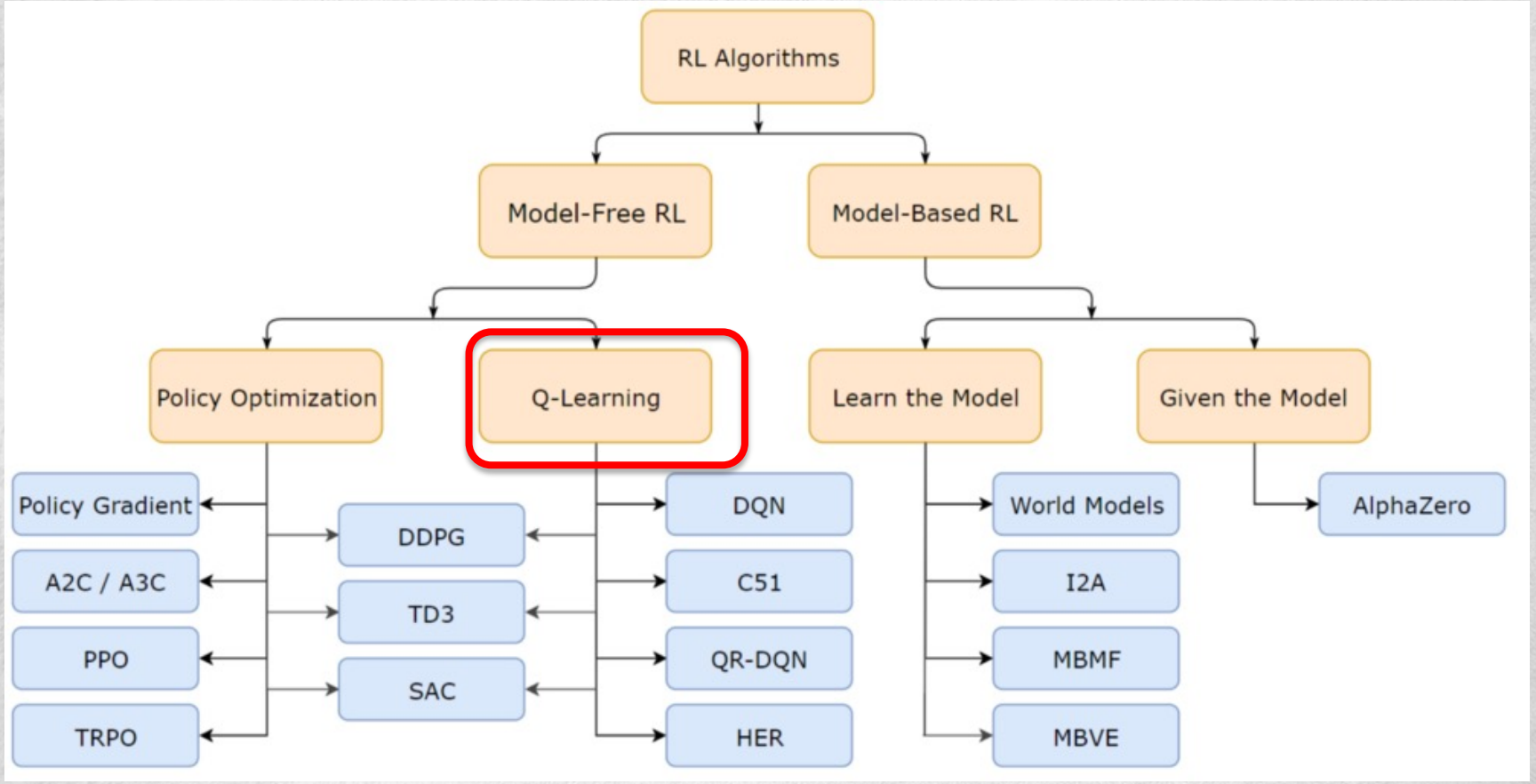
Reinforcement Learning

Reinforcement Learning

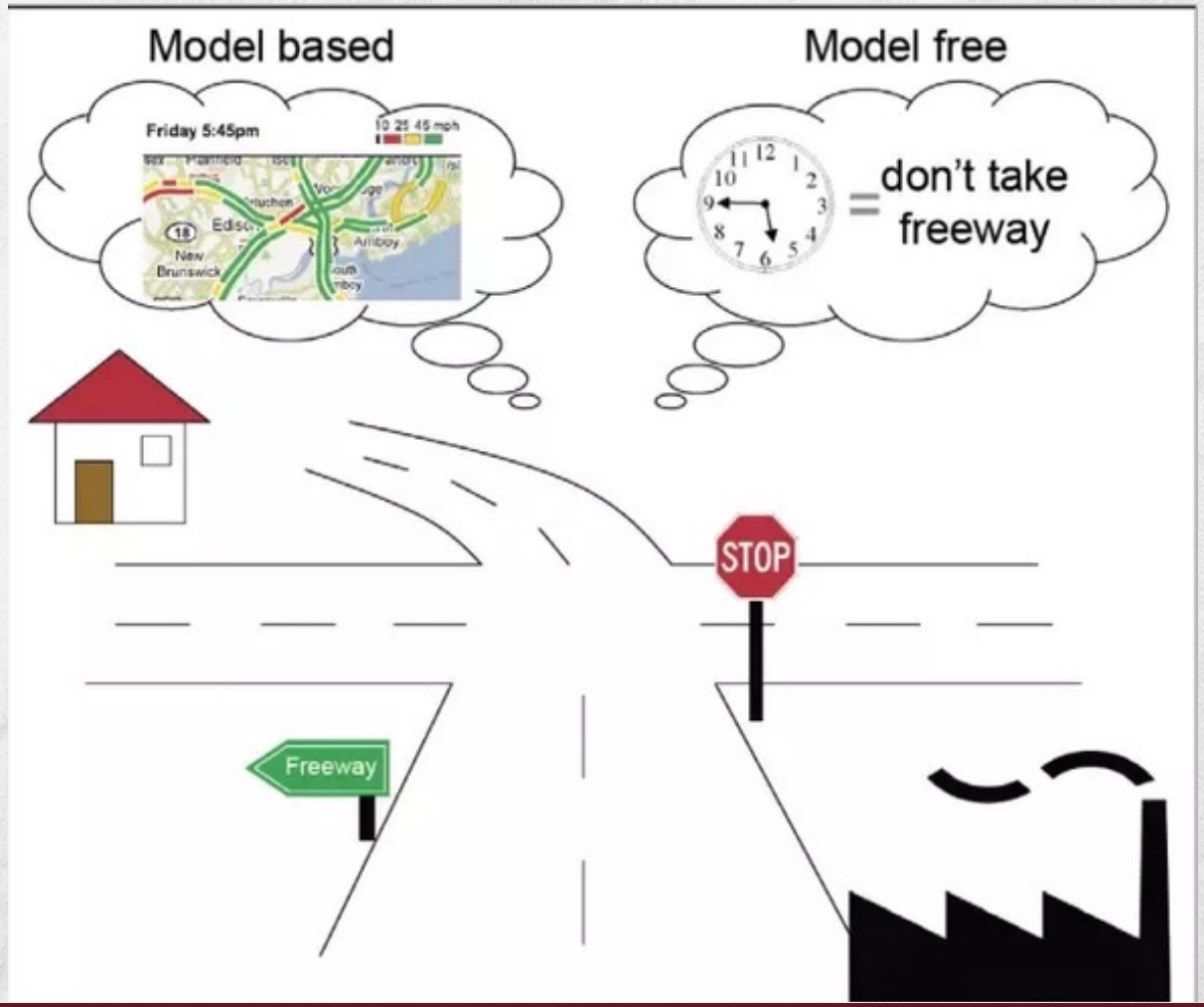
Properties

- The robot learns from ***experience*** by taking ***actions*** in an ***environment***
- Usually involves *sequential decision-making* tasks, each action may have long-term consequences on the overall goal
- Goal is to select actions to maximize total future reward.
- Motion model (state transition model) is unknown
- Sensor model is unknown
- Rewards are only known upon taking actions

Reinforcement Learning



Reinforcement Learning



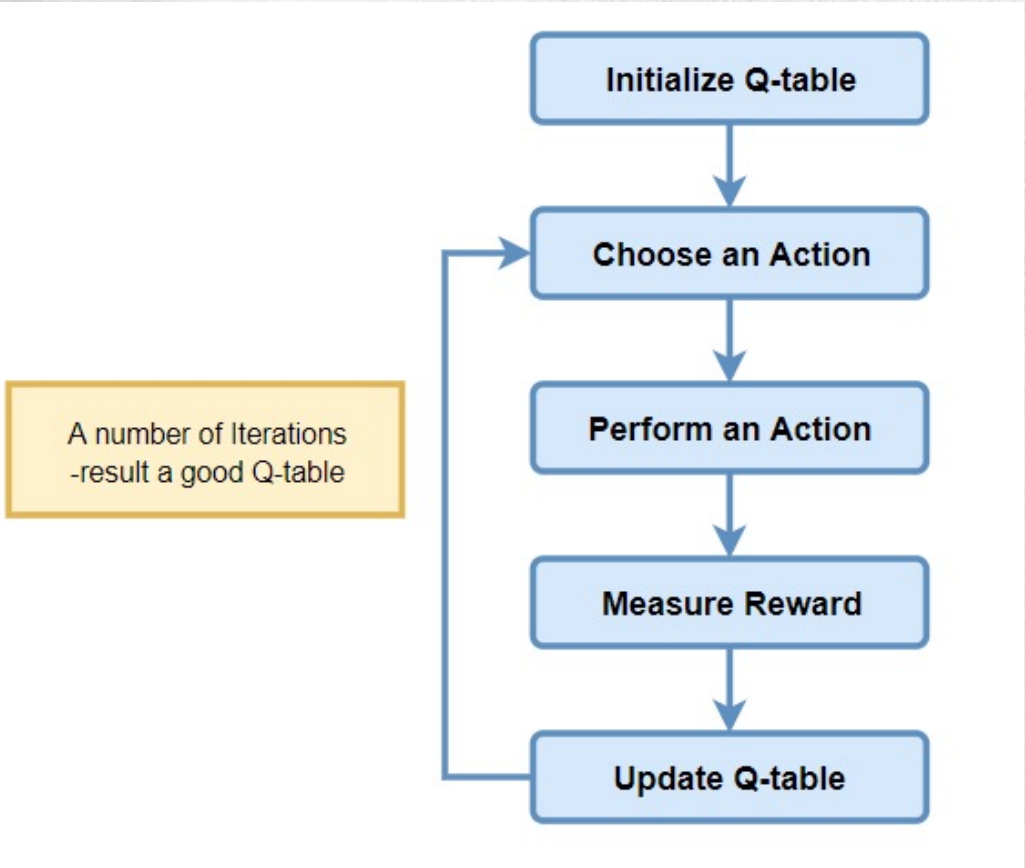
Q-Learning

Markov Decision Processes

The components of an MDP are:

- Set of **states** X (we will assume this is finite)
- An **initial state** $x_0 \in X$.
- A set of **actions** for each state $x \in X$, we call this $U(x)$.
- ~~A **transition model** P, where $P(x' \mid x, u)$ is the probability of reaching state x' after performing control action u in state x.~~
- A **reward function** R specifying the reward for each state action combination, where $R(x, u)$ is the reward for state x.

Q-Learning



- An algorithm for model-free reinforcement learning
- Does not require motion model / state transition model
- Uses a Q-function to estimate the expected rewards for an action taken in a given state

Q-Learning

Value function: Estimates the current and future expected rewards for a given state

Value iteration:

$$V(x) \leftarrow \gamma \max_u [\, R(x,u) + \sum_{x'} V(x')\, P(x' \mid x,u)]$$

Q-function: Estimates the current and future expected rewards for a given state and a given action

Q iteration:

$$Q(x,u) \leftarrow Q(x,u) + \alpha\, [\, R(x,u) + \gamma \max_u Q(x',u) - Q(x,u)]$$

Q-Learning

$$Q(x,u) \leftarrow Q(x,u) + \alpha\,[\,R(x,u) + \gamma \max_u Q(x',u) - Q(x,u)]$$

```
Q[state, action] = Q[state, action] + lr * (reward + gamma *
np.max(Q[new_state, :]) - Q[state, action])
```

Optimal value function: $V^*(x) = \max_u Q^*(x,u)$

Optimal policy: $\pi^*(x) = \operatorname{argmax}_u Q^*(x,u)$

Q-Learning

Game Board:

Current state (*s*): 0 0 0
0 1 0

Q Table: $\gamma = 0.95$

	0 0 0 1 0 0	0 0 0 0 1 0	0 0 0 0 0 1	1 0 0 0 0 0	0 1 0 0 0 0	0 0 1 0 0 0
↑	0.2	0.3	1.0	-0.22	-0.3	0.0
↓	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
→	0.21	0.4	-0.3	0.5	1.0	0.0
←	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

Q-Learning

Problem #1

- Q(x, u) can only be updated if that particular state and action has been experienced before
- ➢ Solution: Start with *exploration*, then gradually transition to *exploitation*

Problem #2

- The model converges slowly because an action has to be taken each time before learning about the resulting state and rewards
- ➢ Solution: Use *experience replay*, where the Q table is additionally updated using previous experiences

Problem #3

- The state space is extremely large for some problems (e.g. if the state is an image)
- ➢ Solution: Use a Deep Q Network

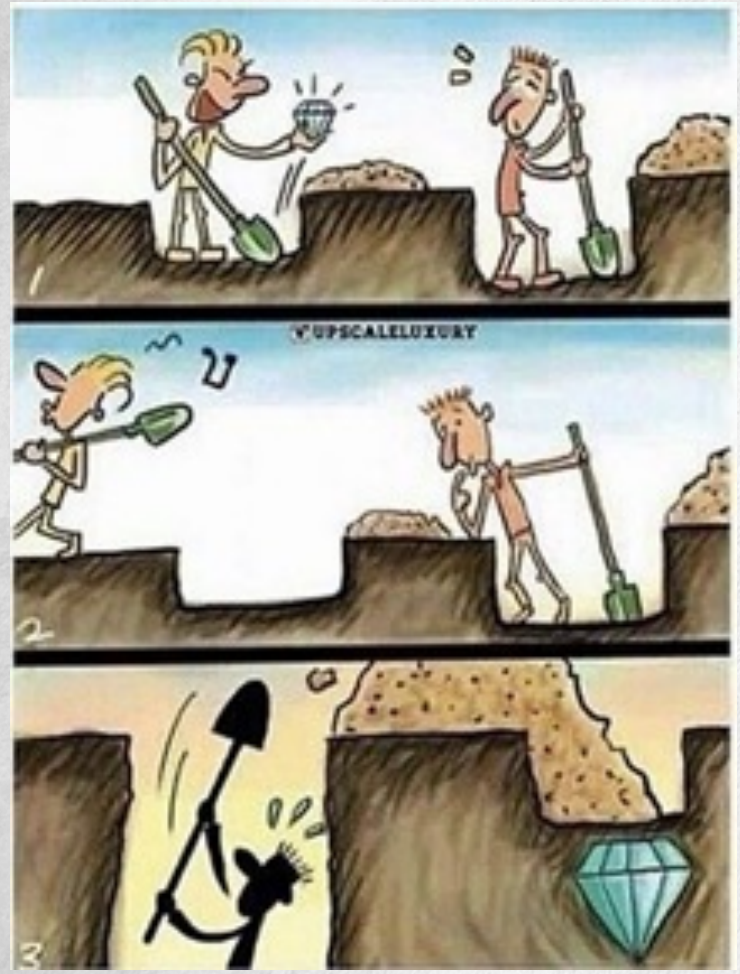
Exploration vs Exploitation

- **Exploration:** take a new action with unknown consequences
  - Pros:
    - Get a more accurate model of the environment
    - Discover higher-reward states than the ones found so far
  - Cons:
    - Not maximizing rewards when exploring
    - Something bad might happen
- **Exploitation:** go with the best strategy found so far
  - Pros:
    - Maximize reward as reflected in the current utility estimates
  - Cons:
    - Might prevent from discovering the true optimal strategy

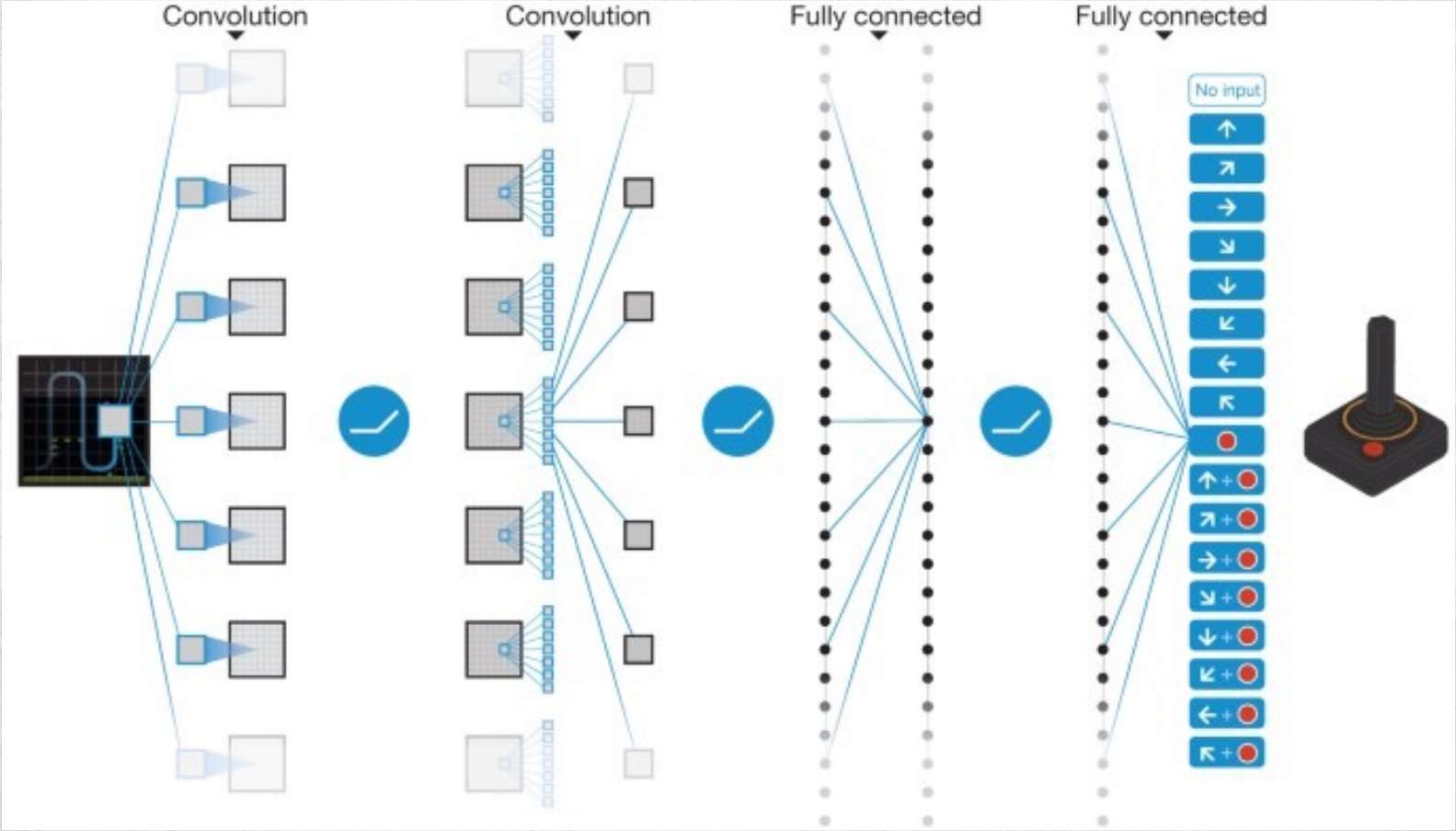
Exploration vs Exploitation

- **Idea:** explore more in the beginning, become more and more greedy over time
- **ε-greedy:** with probability 1–ε, follow the greedy policy, with probability ε, take random action
  - Possibly decrease ε over time
- More complex **exploration functions** to bias towards less visited state-action pairs
  - E.g., keep track of how many times each state-action pair has been seen, return over-optimistic reward estimate if a given pair has not been seen enough times

Exploration vs Exploitation



Deep Q Network

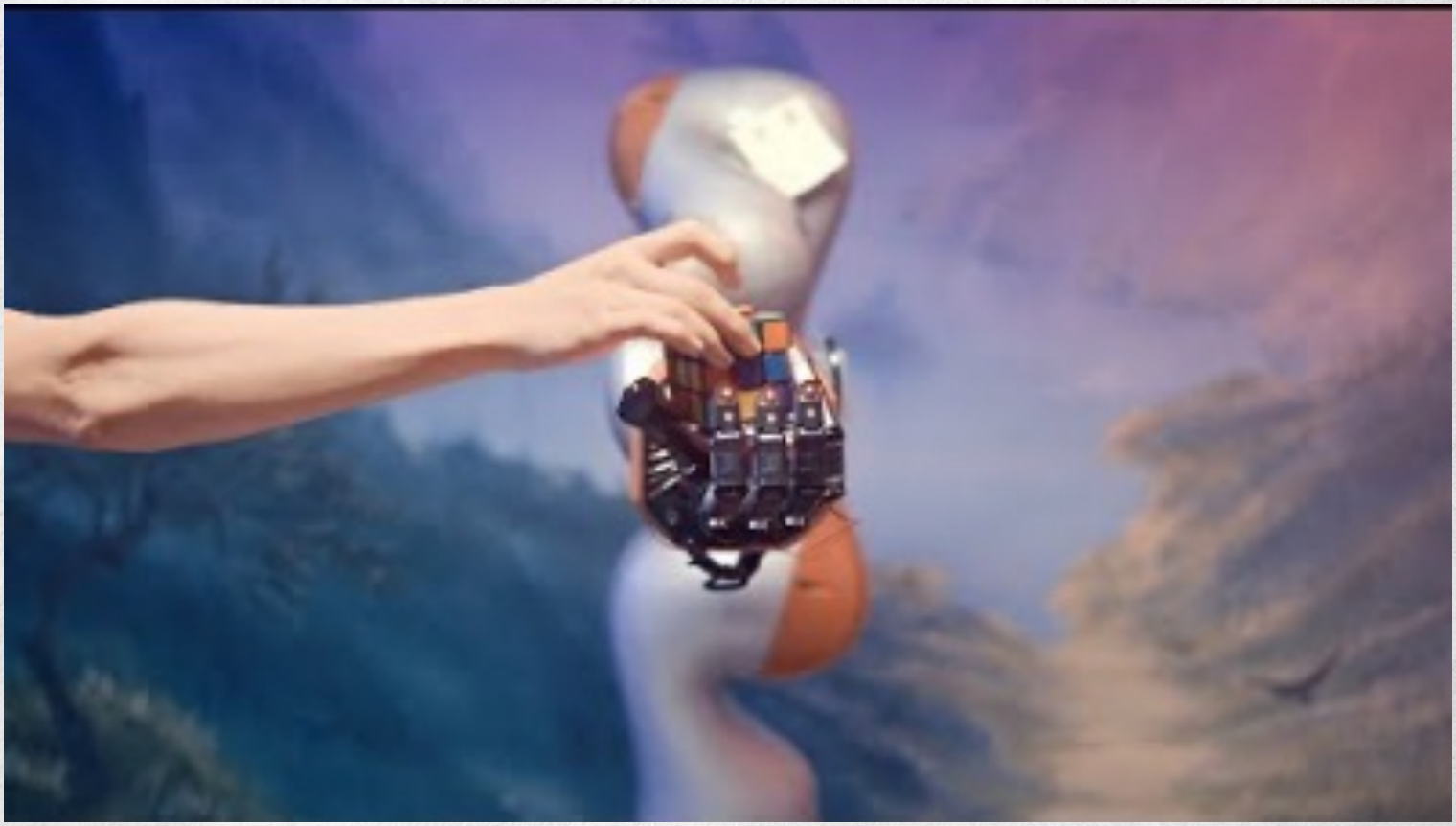


Examples with Reinforcement Learning or Q Learning

Game Playing

Drone Flight

Solving Rubik's Cube



References

1. https://towardsdatascience.com/reinforcement-learning-an-introduction-to-the-concepts-applications-and-code-ced6fbfd882d
2. https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/its-time-for-businesses-to-chart-a-course-for-reinforcement-learning#
3. https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c
4. https://www.nature.com/articles/nature14236