Deep Q learning

(Deep Reinforcement Learning)

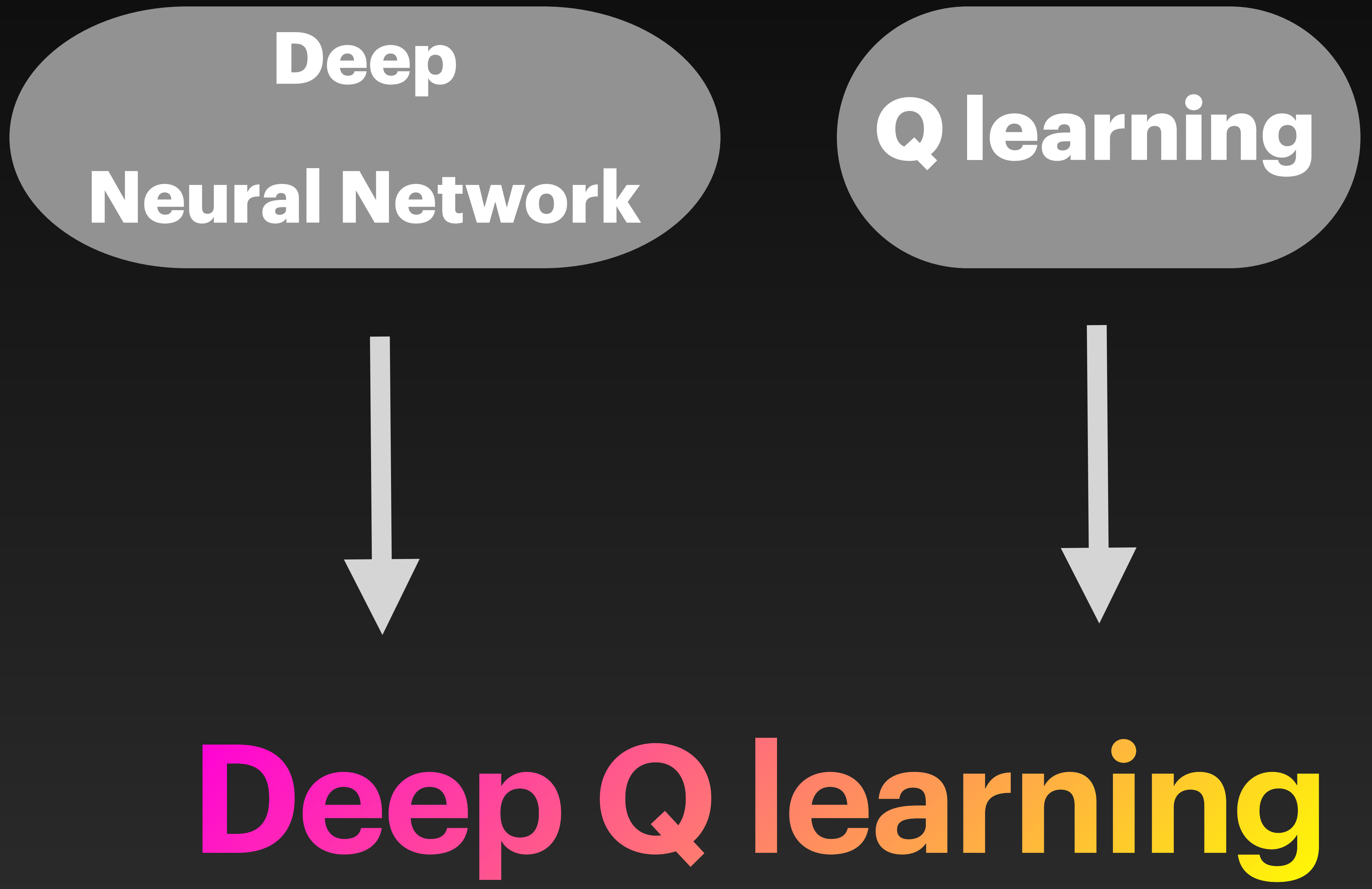
Artyom Voronin

Content:

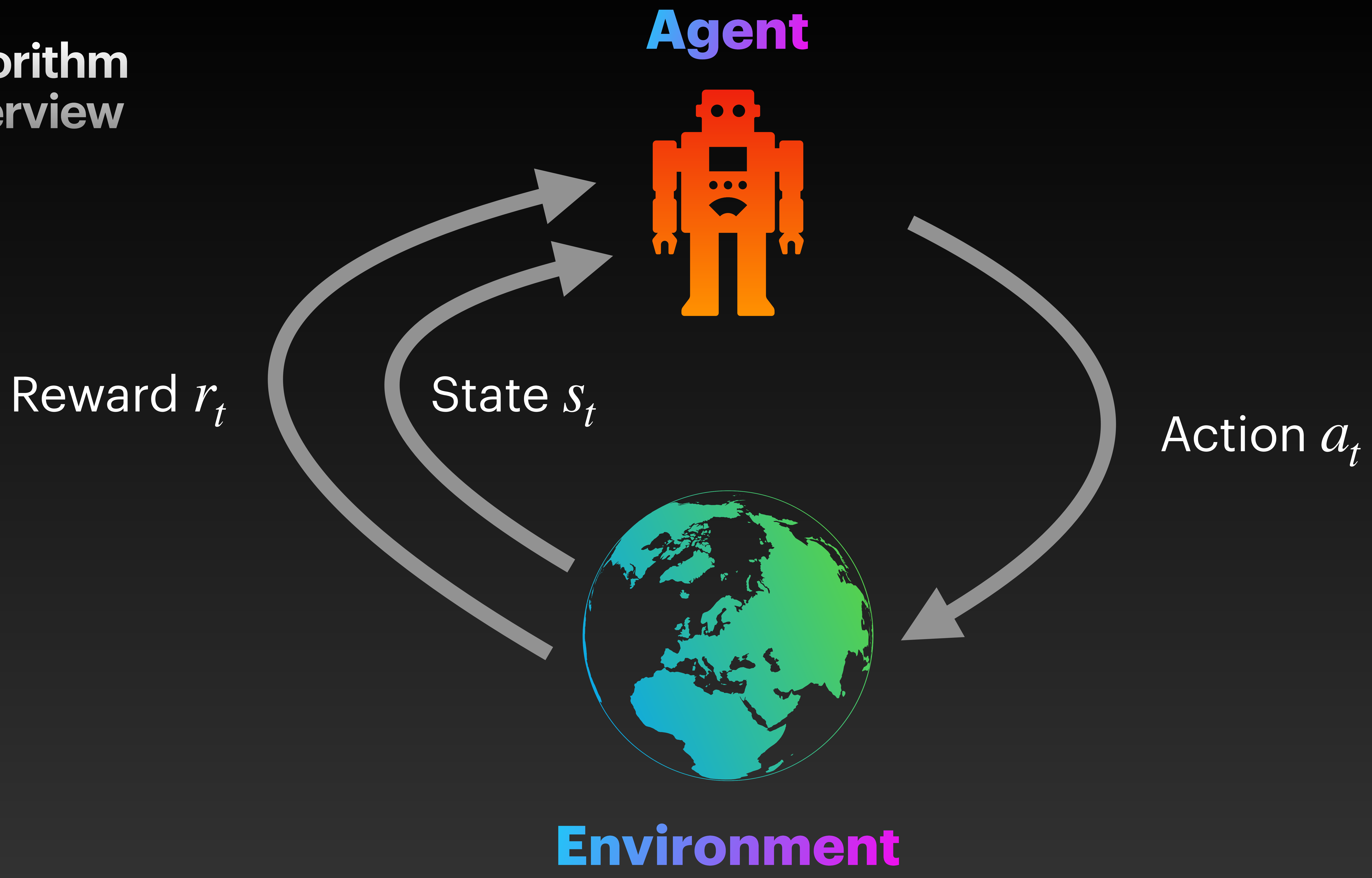
- Terminology
- Algorithm Overview
- Environment
- Agent
- Bellman equation
- Optimize DNN
- Algorithm Pseudocode
- Application
- Resources

Terminology

- **Reinforcement learning** - “teach by **experience** not **examples**”.
- **Q learning** is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state.
- **Deep Neural Network** is an artificial neural network with multiple layers.



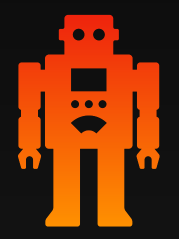
Algorithm Overview



One episode contain: $s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, \ldots, s_{n-1}, a_{n-1}, r_n, r_{terminate}$

*Episode - everything that happens between the first state and the last.

Algorithm Overview

Agent  interacts with the **environment**. The goal of the **agent** is to maximize its cumulative reward, called **return.**

Environment  provides information about its **state** and **rewards** received by the **agent** during one episode.

States represent by a real-valued vector, matrix, or higher-order tensor.

The set of all valid **actions** in a given **environment** is often called the **action space**.

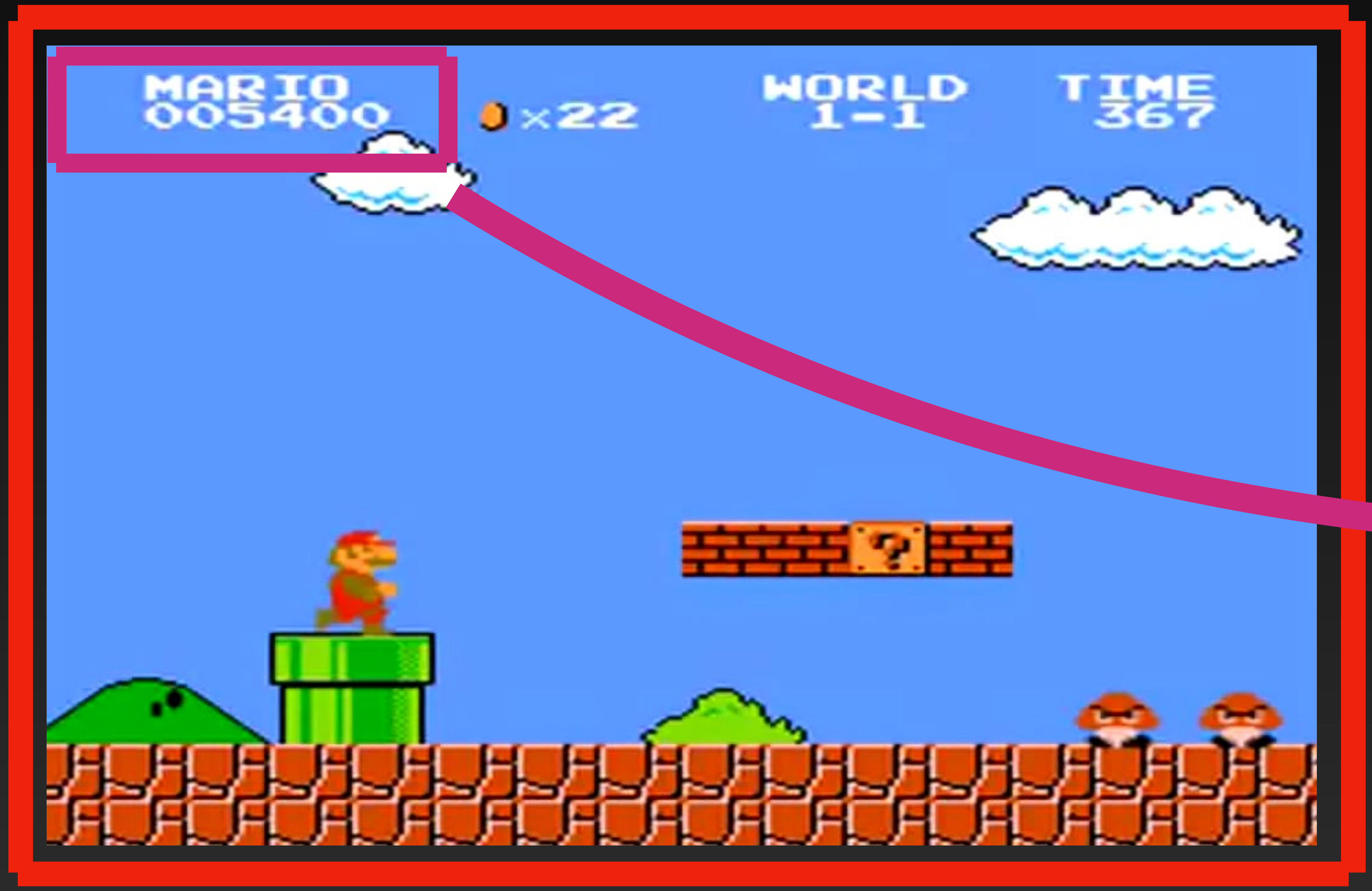
Reward is a number that tells it how good or bad the current world state is.

Environment

- Action
  - Keyboard press
  - Actuators movement
- State
  - Image
  - Sensor data
- Reward
  - Score
  - Goals done

Example:

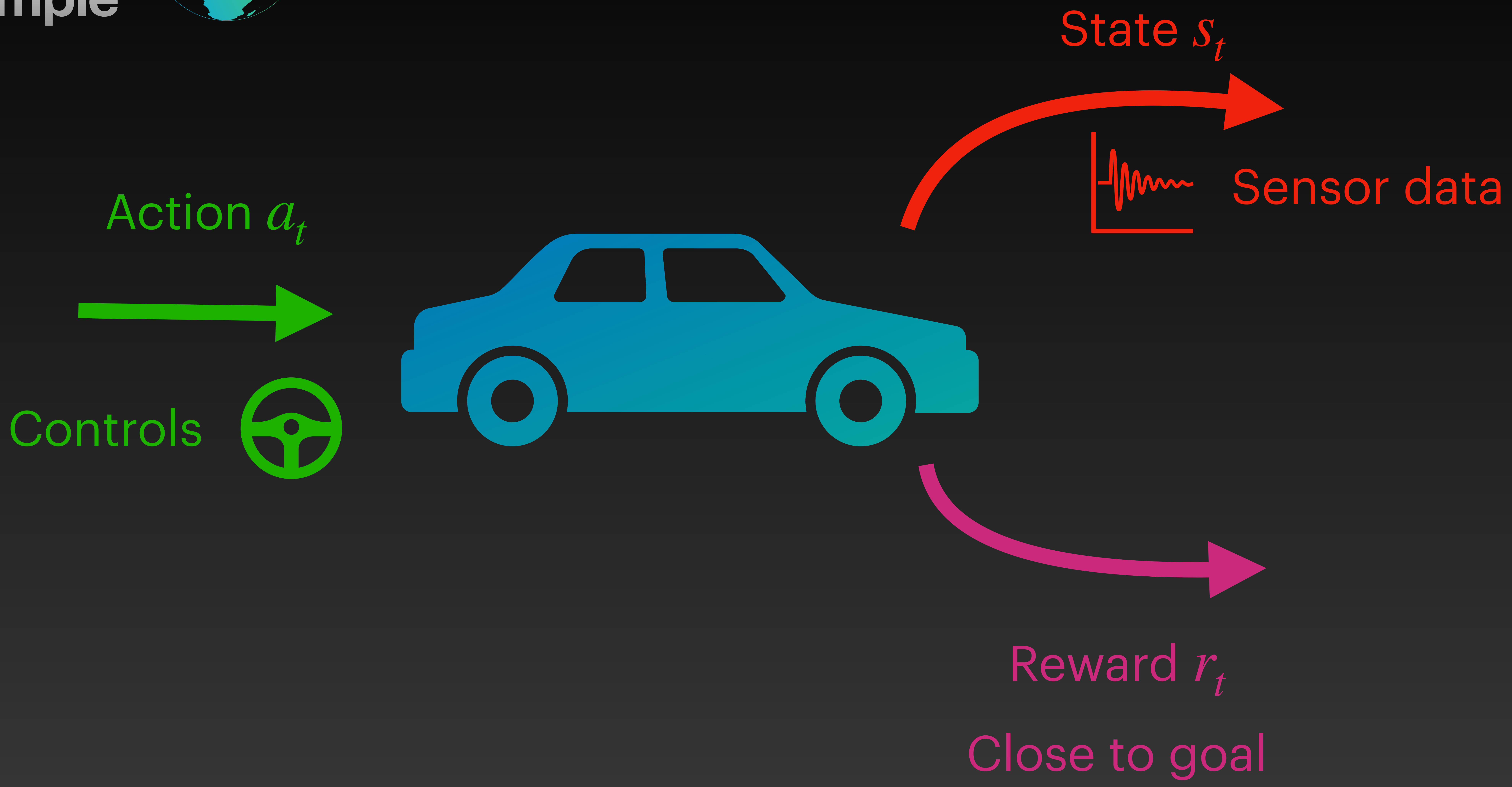
Action a_t



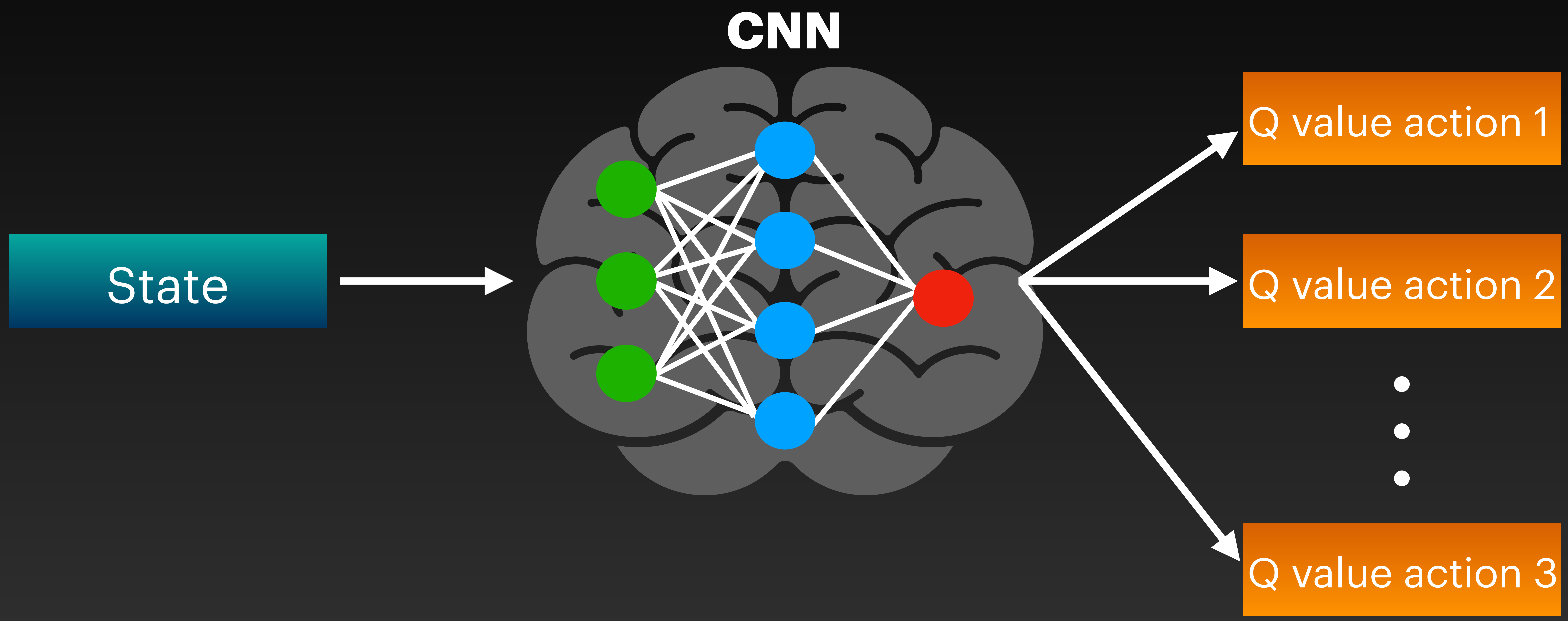
State s_t

Reward r_t

Environment Example



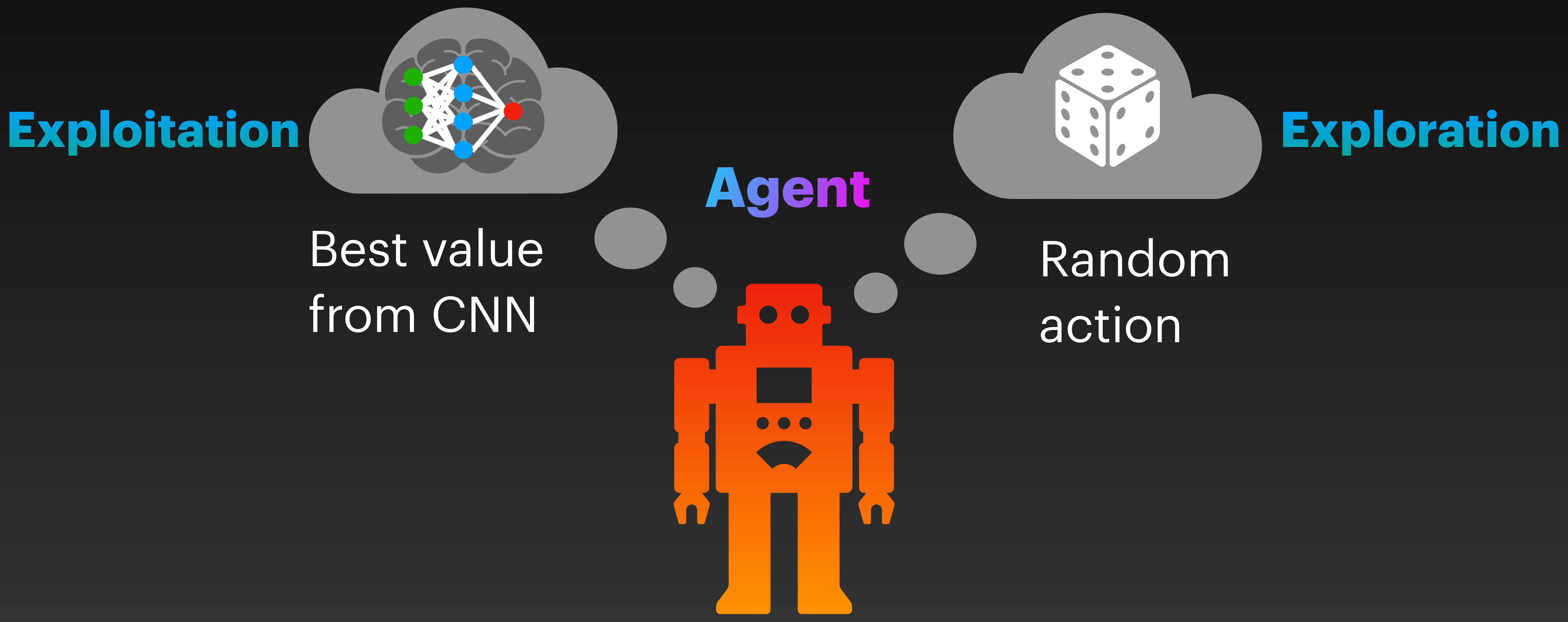
Agent



*The main idea of the Deep Q learning algorithm is the same as in the Q-learning algorithm, except using a CNN (Convolution Neural Network) instead of a Q-table.

Agent

ϵ-greedy algorithm



* With the probability ϵ, we select a random action a and with probability $1 - \epsilon$, we select an action that has a maximum Q-value.

Bellman Equation

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1}(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

New Q value for that state and that action

Current Q value

Learning rate

Reward for taking that action at that state

Discount factor

Maximum expected future reward given the new s_{t+1} and all possible action at that new state

Bellman Equation

Q-value - maximum total reward performed by agent’s sequence of action.

The **learning rate** α determines to what extent newly acquired information overrides old information.

The **discount factor** γ determines the importance of future rewards.

Optimize DNN

In case **Deep Q learning**, we replace **Q value** $Q(s, a)$ to **approximation function** $Q(s, a, \theta)$, where θ represents the trainable weights of the network.

Using Bellman equation as **Cost function** we will get:

$$Cost = [Q(s_t, a_t, \theta) - (r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a_t, \theta))]^2$$

Target network / Double Deep Q learning:

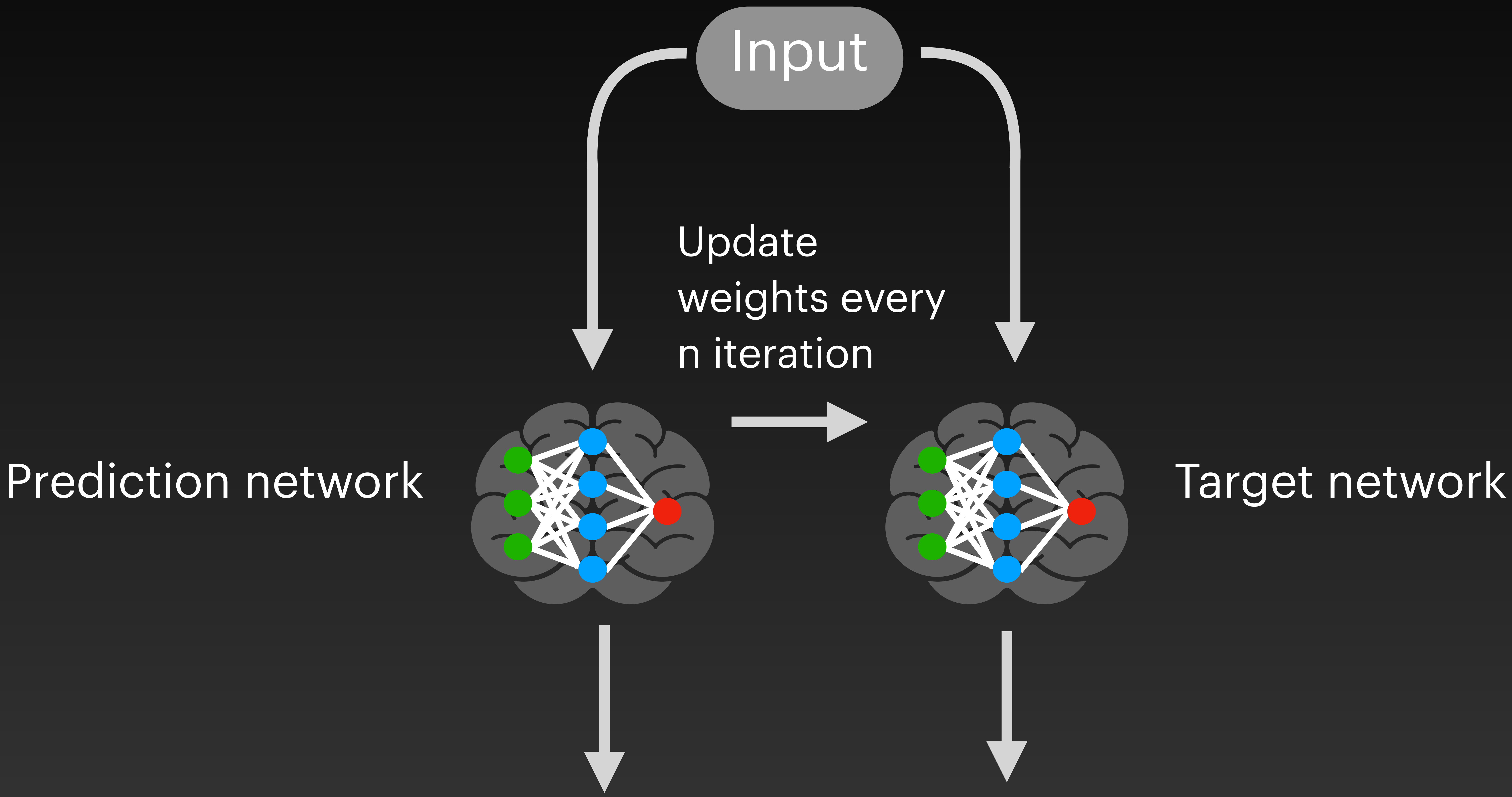
For better convergence and performance, instead of using one neural network for learning, we can use two:

$$Cost = [Q(s_t, a_t, \theta) - (r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a_t, \vartheta))]^2$$

where ϑ represents the **target network** weights, so $Q(s_{t+1}, a_t, \vartheta)$ means the Q Value predicted by the **target network.**

Optimize DNN

Target network / Double Deep Q learning



$$Cost = [Q(s_t, a_t, \theta) - (r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a_t, \vartheta))]^2$$

Algorithm Pseudocode

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

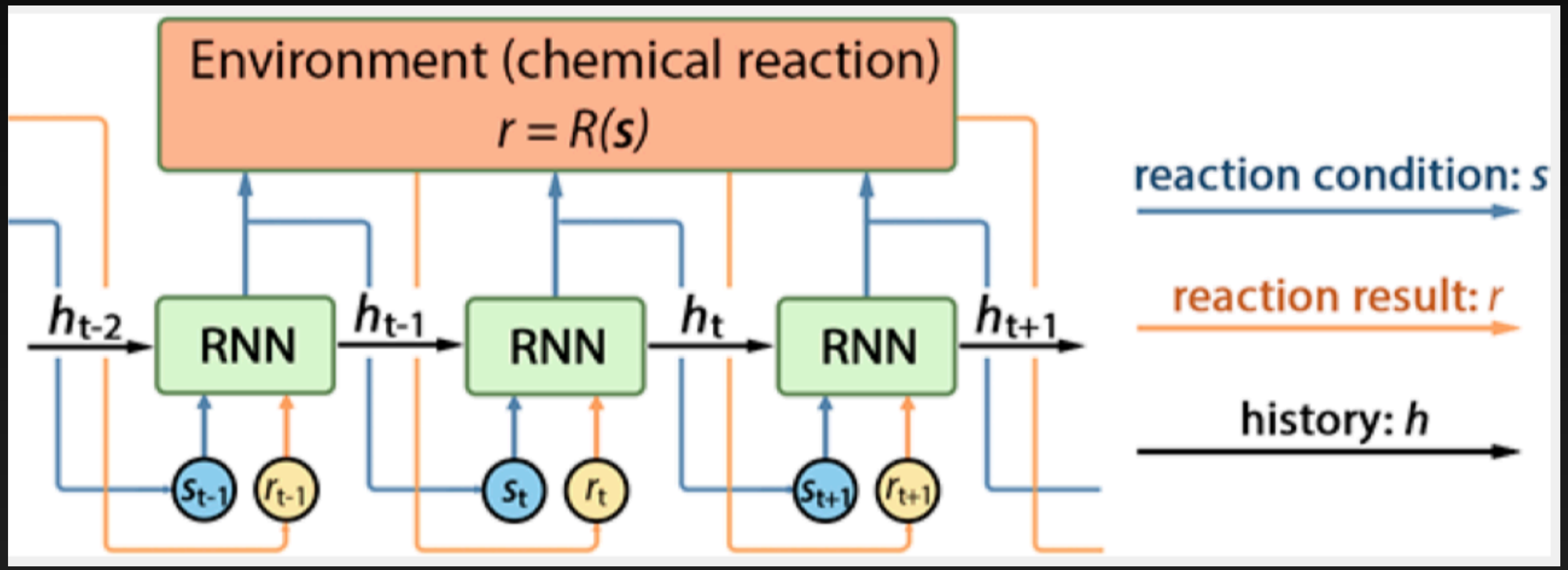
Applications

- Computer games
- Robotics
- Chemistry
- Web System Configuration
- Traffic Light Control

Google DeepMind plays Atari

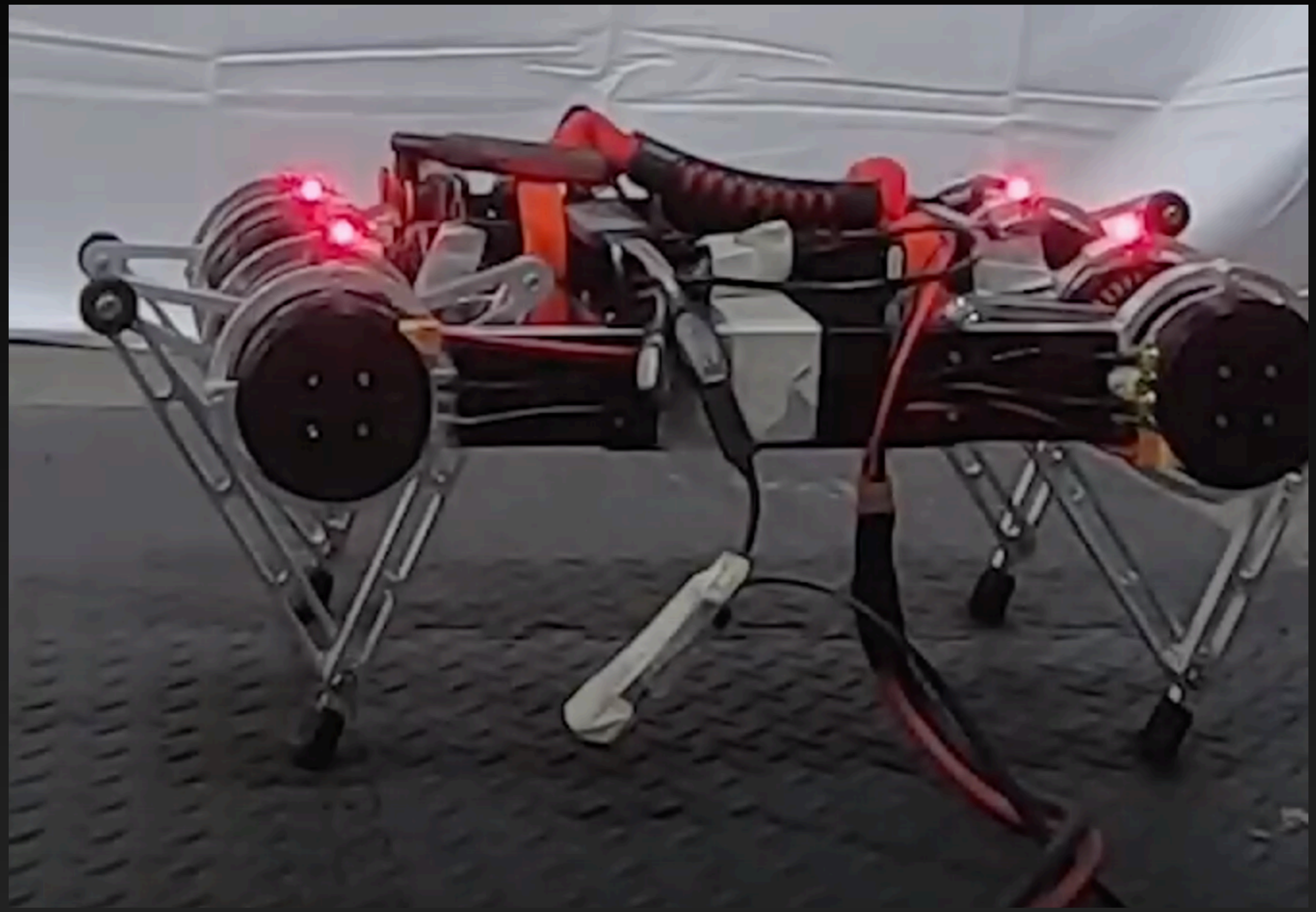
https://www.youtube.com/watch?v=V1eYniJ0Rnk

Application example



Optimizing Chemical Reactions with Deep Reinforcement Learning

https://pubs.acs.org/doi/pdf/10.1021/acscentsci.7b00492



Robot learning to walk by DQ-Learning algorithm

https://www.youtube.com/watch?v=n2gE7n11h1Y

Resources

- https://arxiv.org/pdf/1509.06461.pdf
- https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/
- https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf
- https://pathmind.com/wiki/deep-reinforcement-learning
- https://arxiv.org/pdf/1811.12560.pdf
- https://github.com/keras-rl/keras-rl
- https://openai.com