MATHEMATICS
FOR MACHINE LEARNING

Nagoya University, Fall 2023

Lecture 10

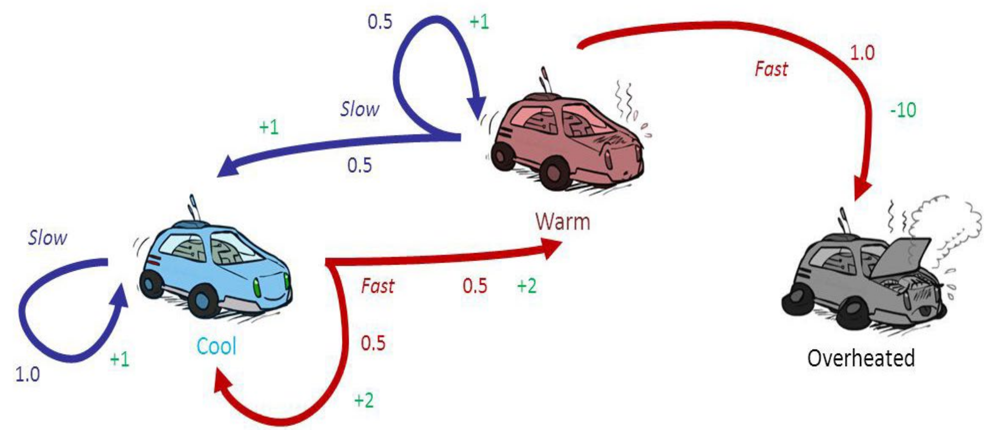
Reinforcement Learning: Deep Q Network

This week Tutorial: Wednesday 13th Dec. 5th period

https://www.henrikbachmann.com/mml2023.html

Deep reinforcement learning: Milestones

Q-Learning
(Chris Watkins, 1989)



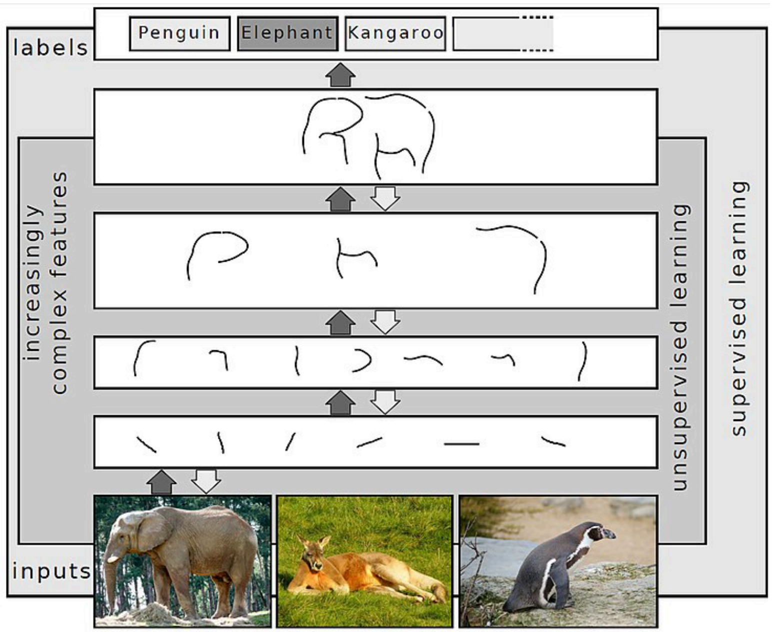
Deep Learning
(AlexNet, 2012)

Deep Q-Network
(DeepMind, 2015)

Deep reinforcement learning: Milestones

Q-Learning
(Chris Watkins, 1989)

Deep Learning
(AlexNet, 2012)



Sven Behnke, [CC BY-SA 4.0](), via Wikimedia Commons

Deep Q-Network
(DeepMind, 2015)

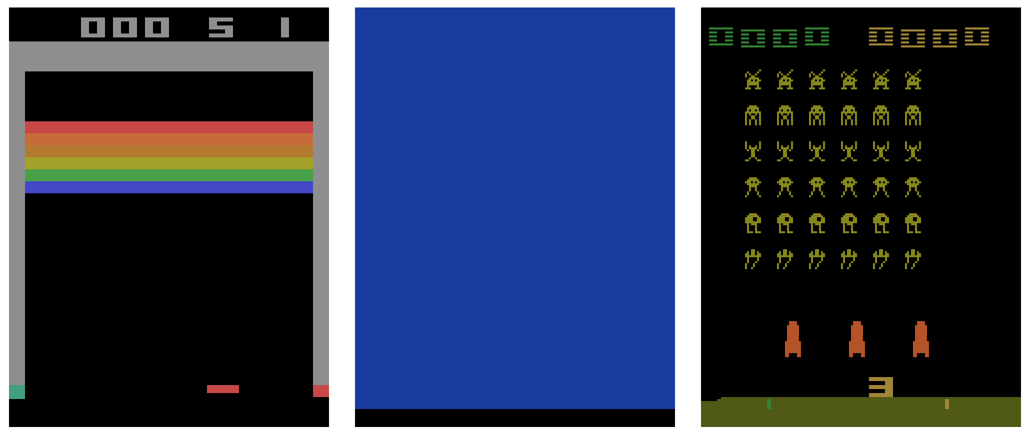
Deep Reinforcement Learning: Milestones

Q-Learning

(Chris Watkins, 1989)

Deep Learning

(AlexNet, 2012)

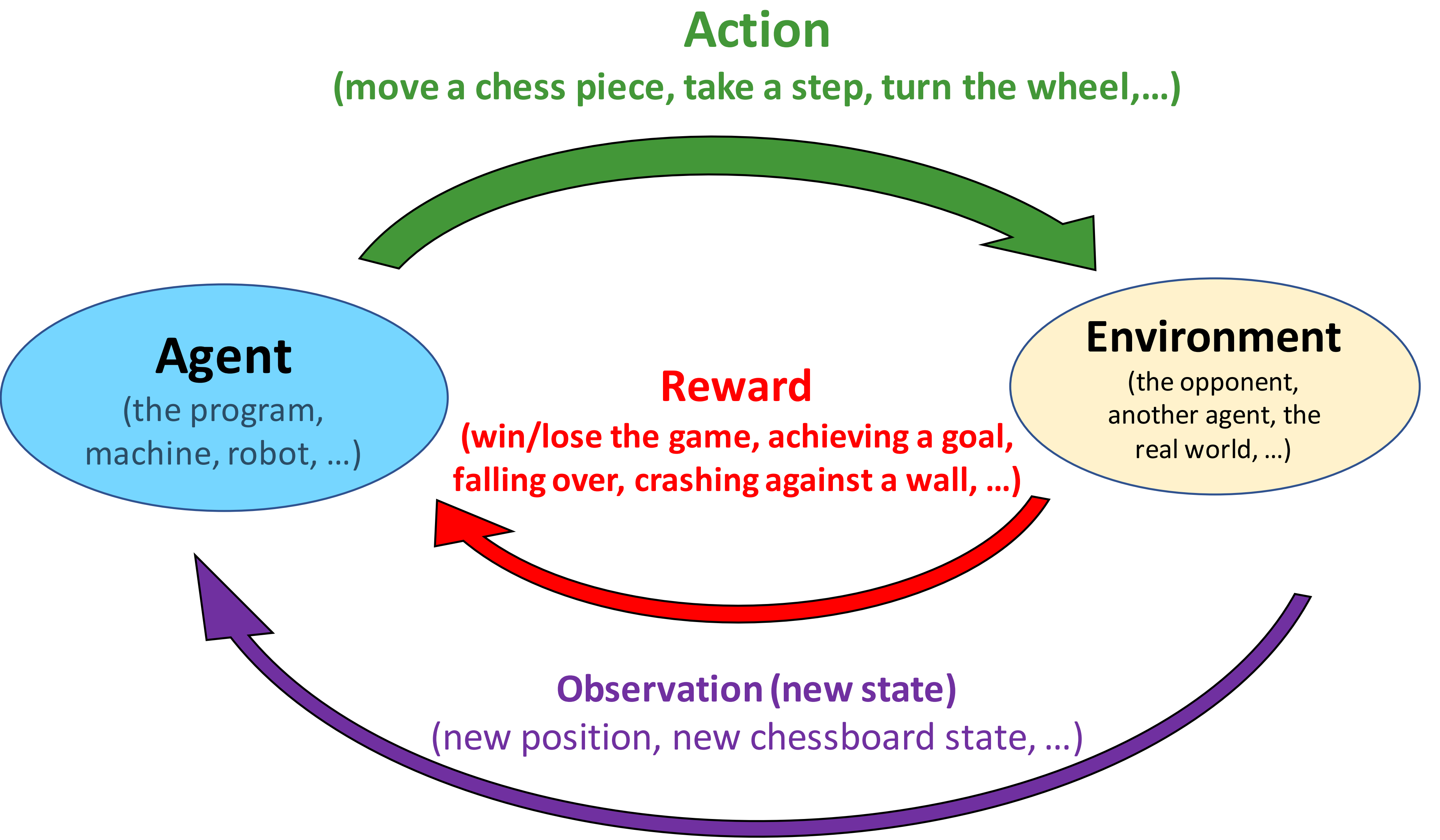


https://github.com/Farama-Foundation/Gymnasium

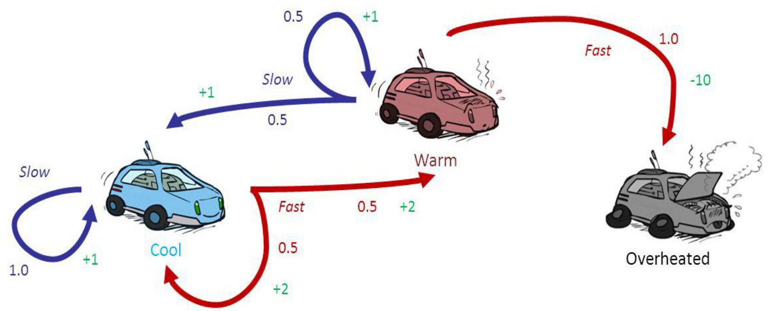
Deep Q-Network

(DeepMind, 2015)

Recall reinforcement learning: Basic idea



Recall Q-learning



$T(s, a, s')$: The probability that one reaches state s' when taking action a in state s

$R(s, a, s')$: The reward that one gets by going from state s to s' by doing action a

Goal: Choose the actions $a_0, a_1, a_2, \ldots$ at each state such that

$$\sum_{j \geq 0}^{N} \gamma^j R(s_j, a_j, s_{j+1})$$

gets big.

Recall Q-learning: Policy and Q function

A **policy** is a function $\pi : S \to A$.

The **value of a policy** π *at state* $s \in S$ *is* $V_\pi(s)$

A policy π^* is called **optimal** if it has maximal value for all states $s \in S$:

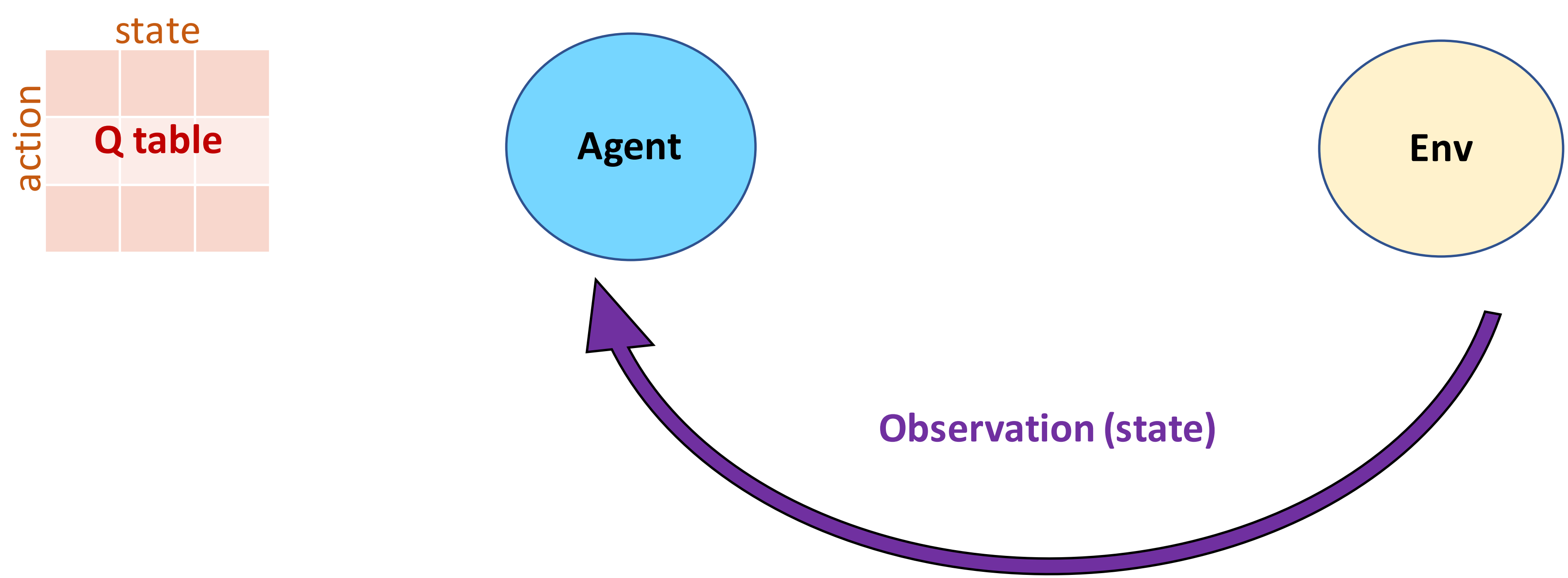
$$V_{\pi^*}(s) = \max_\pi V_\pi(s)\,.$$

The **state-action value function** Q^* is defined for all $(s, a) \in S \times A$ as the expected total reward for taking action $a \in A$ at state $s \in S$ following the optimal policy π^*.

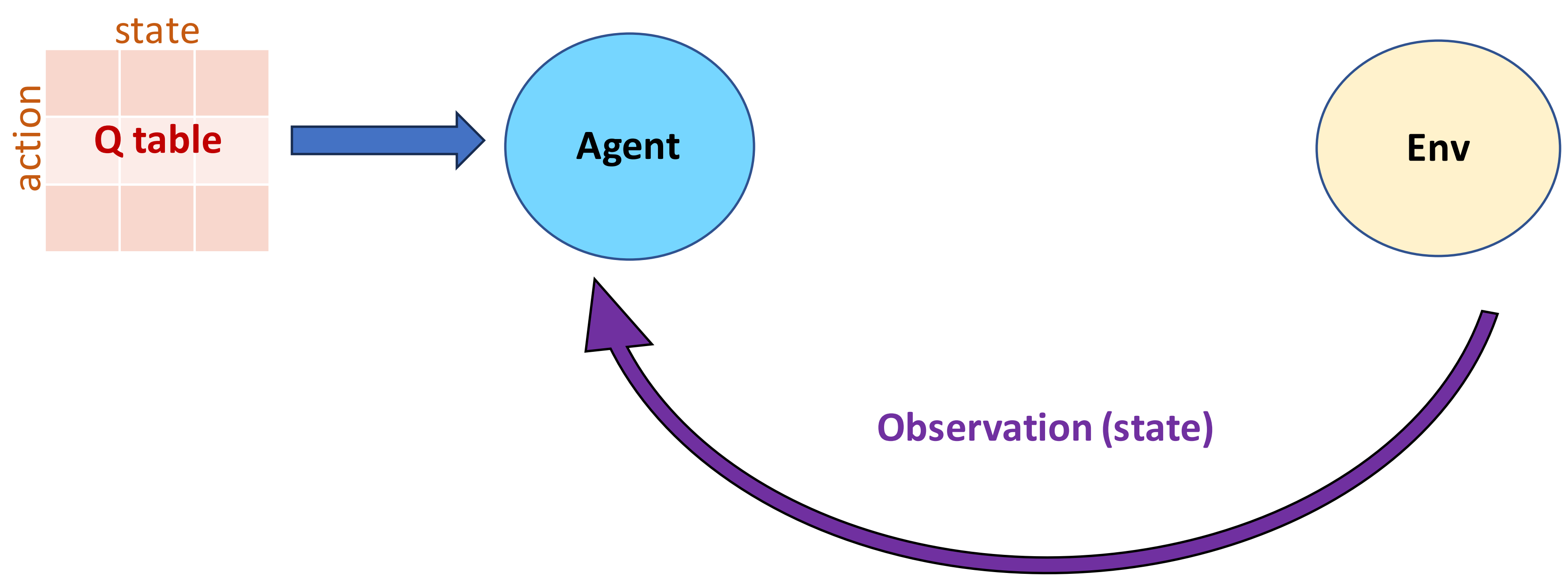
Having the state-action value function Q^* we can derive the optimal policy by

$$\pi^*(s) = \mathrm{argmax}_{a \in A}\, Q^*(s, a)\,.$$

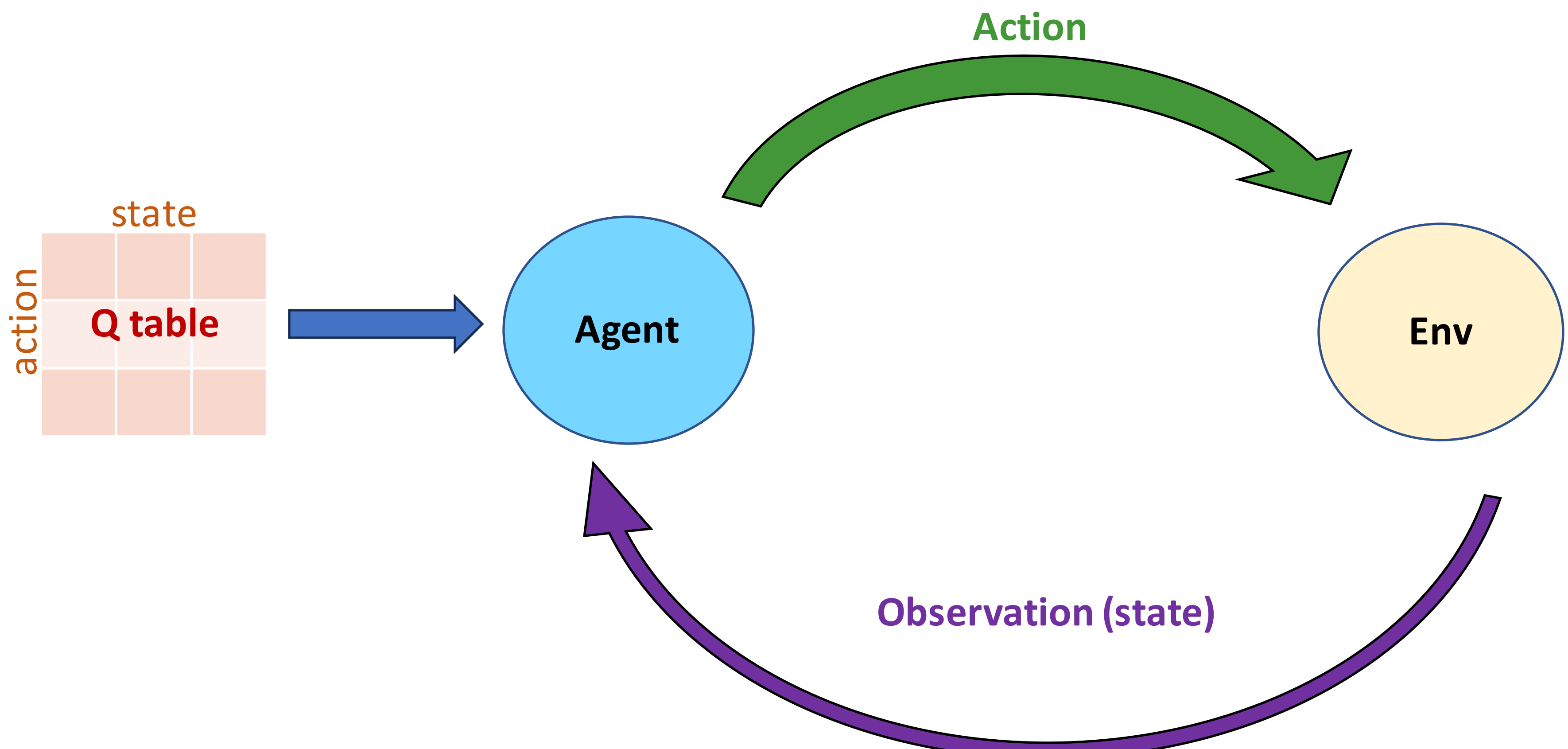
Recall Q-Learning: Basic idea



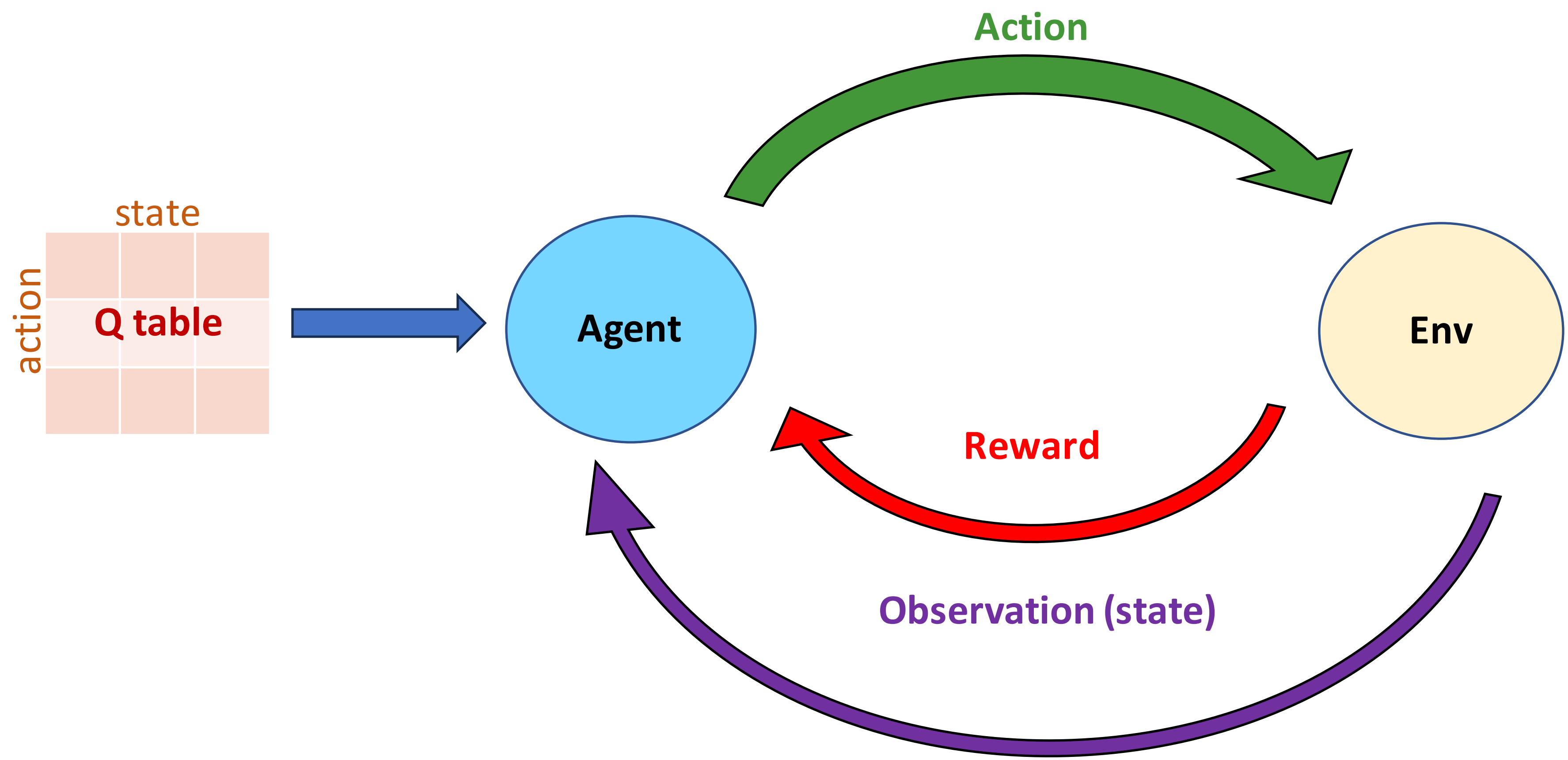
Recall Q-learning: Basic idea



Recall Q-learning: Basic idea



Recall Q-learning: Basic idea



Recall Q-learning + epsilon-greedy

Q-learning algorithm: Find for all $s \in S$ and $a \in A$ a function $Q(s,a)$, which gives a good approximation for $Q^*(a,s)$.

1. Start with random values for $Q(s,a)$. (e.g. all zero)
2. Choose a starting state $s_0 \in S$.
3. Look up the current best action in that state, i.e. $a_0 = \text{argmax}_{a \in A} Q(s_0, a)$ or choose a random action $a_0 \in A$ with probability $\epsilon \in [0,1]$ (Epsilon-Greedy Algorithm).
4. Apply this action and get a new state s_1 and reward $r_0 = R(s_0, a_0, s_1)$.
5. Update the value $Q(s_0, a_0)$ as follows (**Bellman equation**)

$$Q(s_0, a_0) = (1-\alpha)Q(s_0, a_0) + \alpha \left(r_0 + \gamma \max_{a \in S} Q(s_1, a) \right) .$$

   Here $\alpha \in [0,1]$ is the **learning rate**.

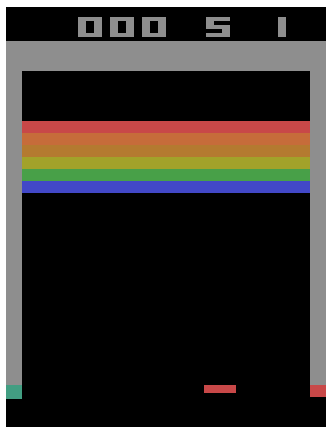
6. If s_1 is not a terminal state repeat with step 3.

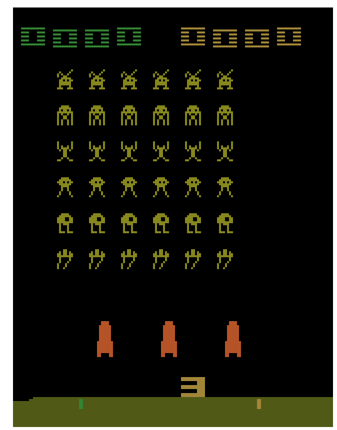
Recall: Convolutional neural network

Demo: https://tensorspace.org/html/playground/alexnet.html

Deep Q-network

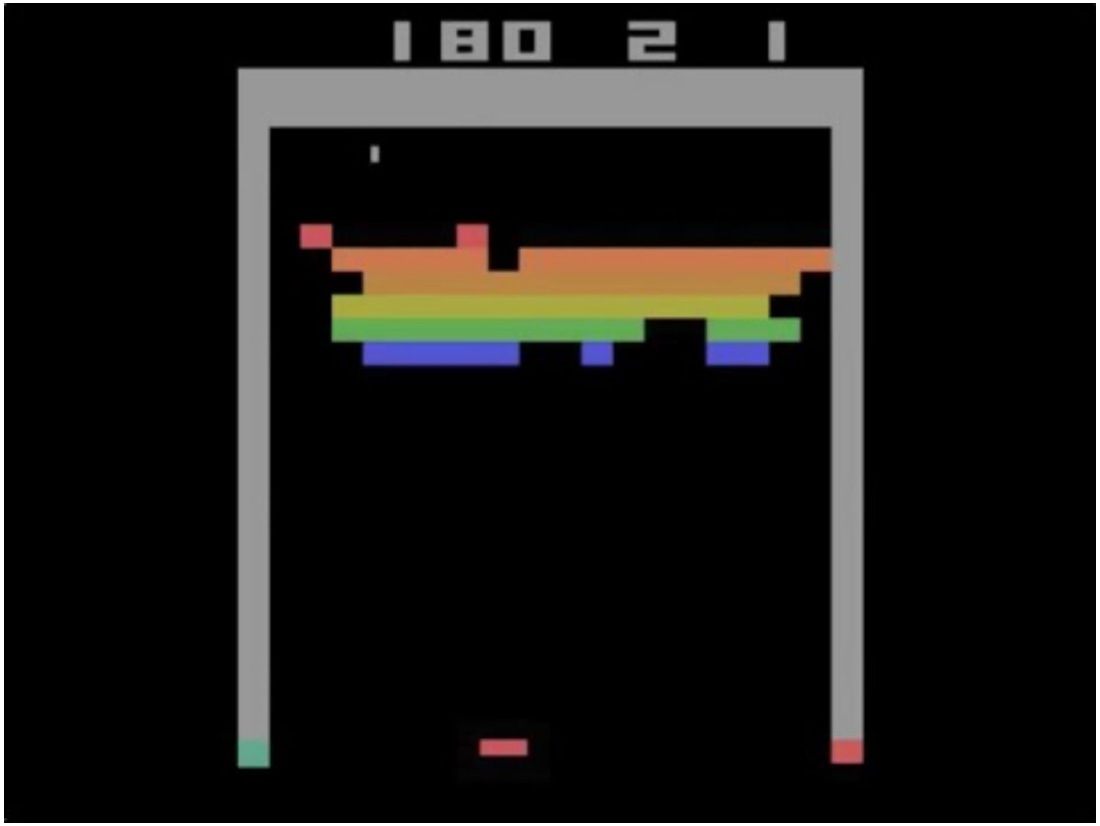
Mnih, Volodymyr; et al. (2015)
"Human-level control through deep reinforcement learning"
Nature. **518** (7540): 529–533





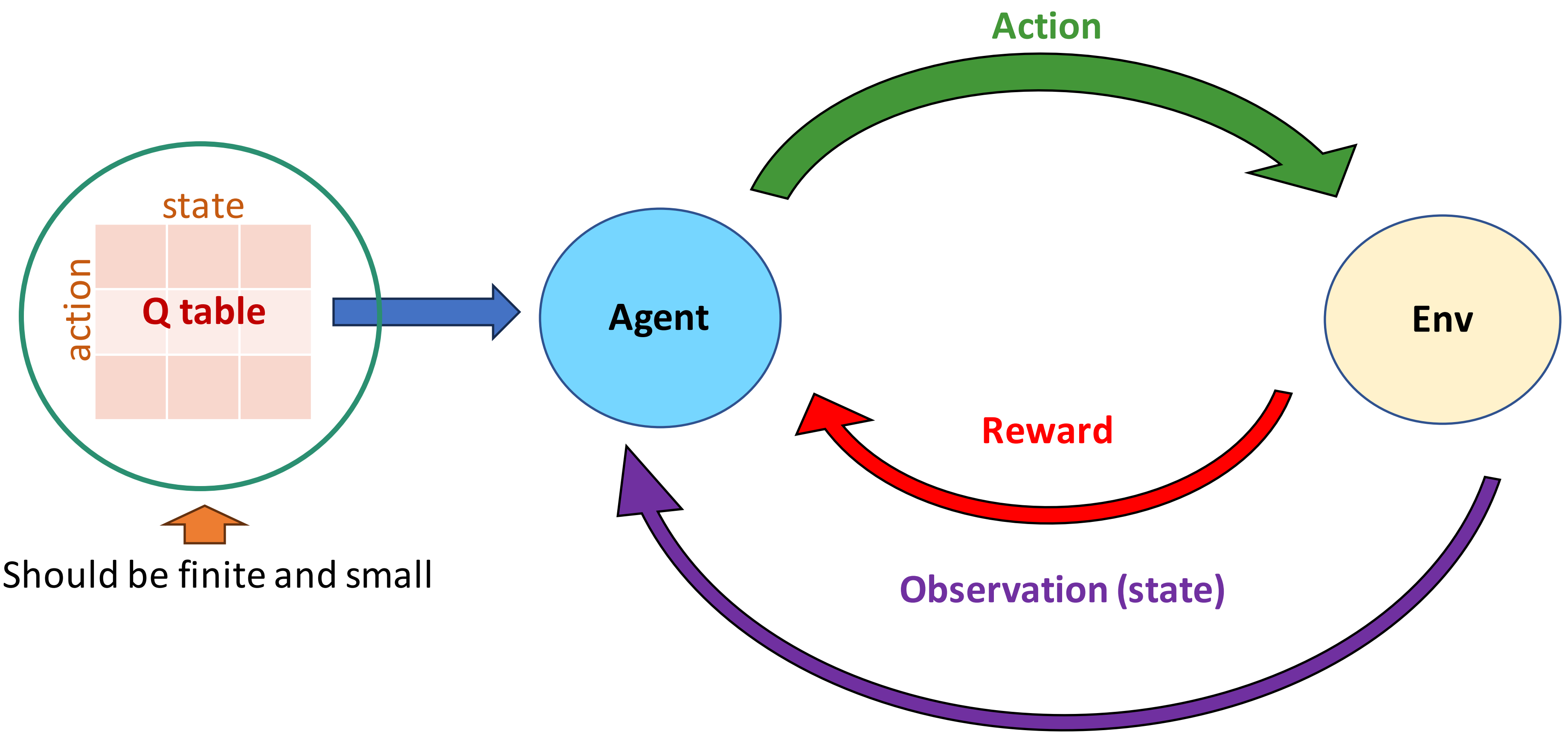
https://github.com/Farama-Foundation/Gymnasium

Deep Q-network

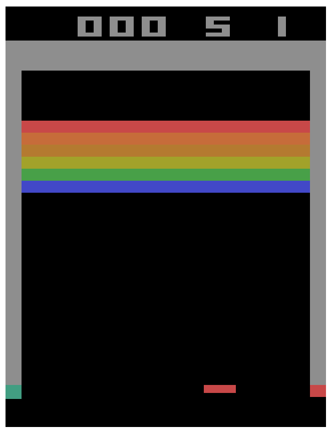


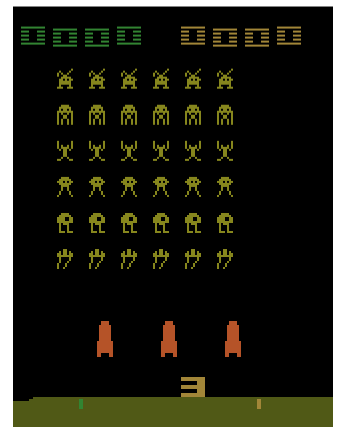
https://www.youtube.com/watch?v=TmPfTpjtdgg

Deep Q-network: Q-learning limitation



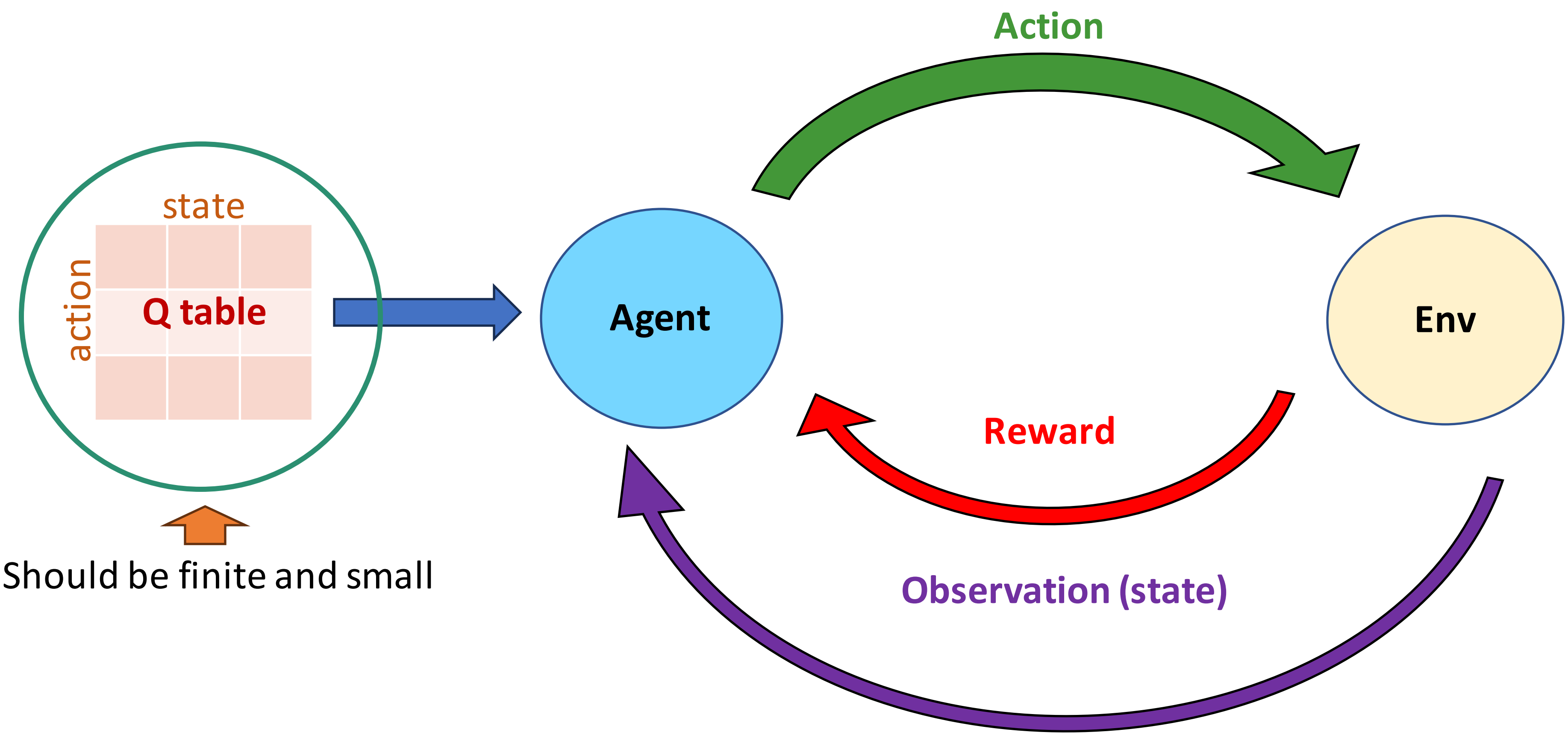
Deep Q-network: How many states?



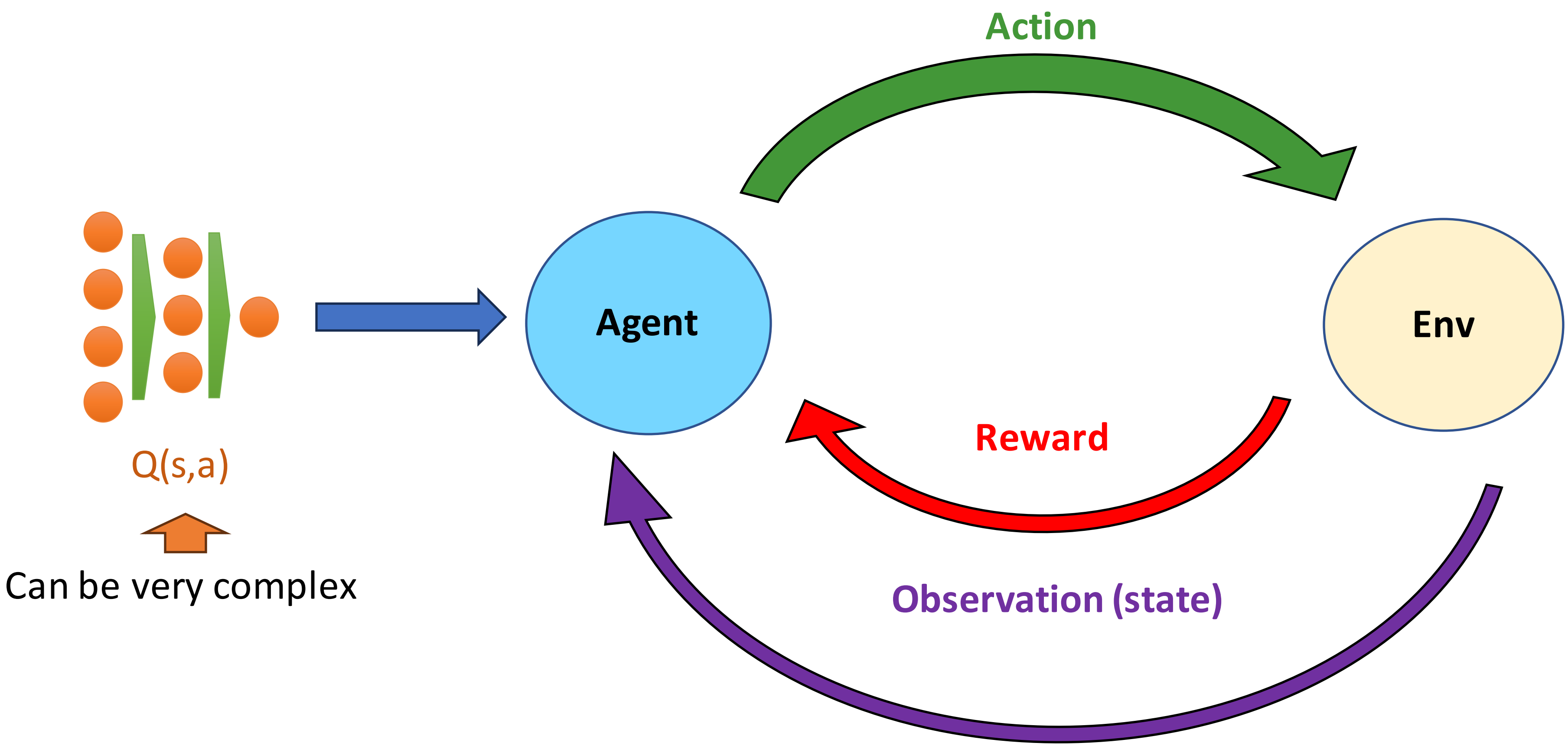



https://github.com/Farama-Foundation/Gymnasium

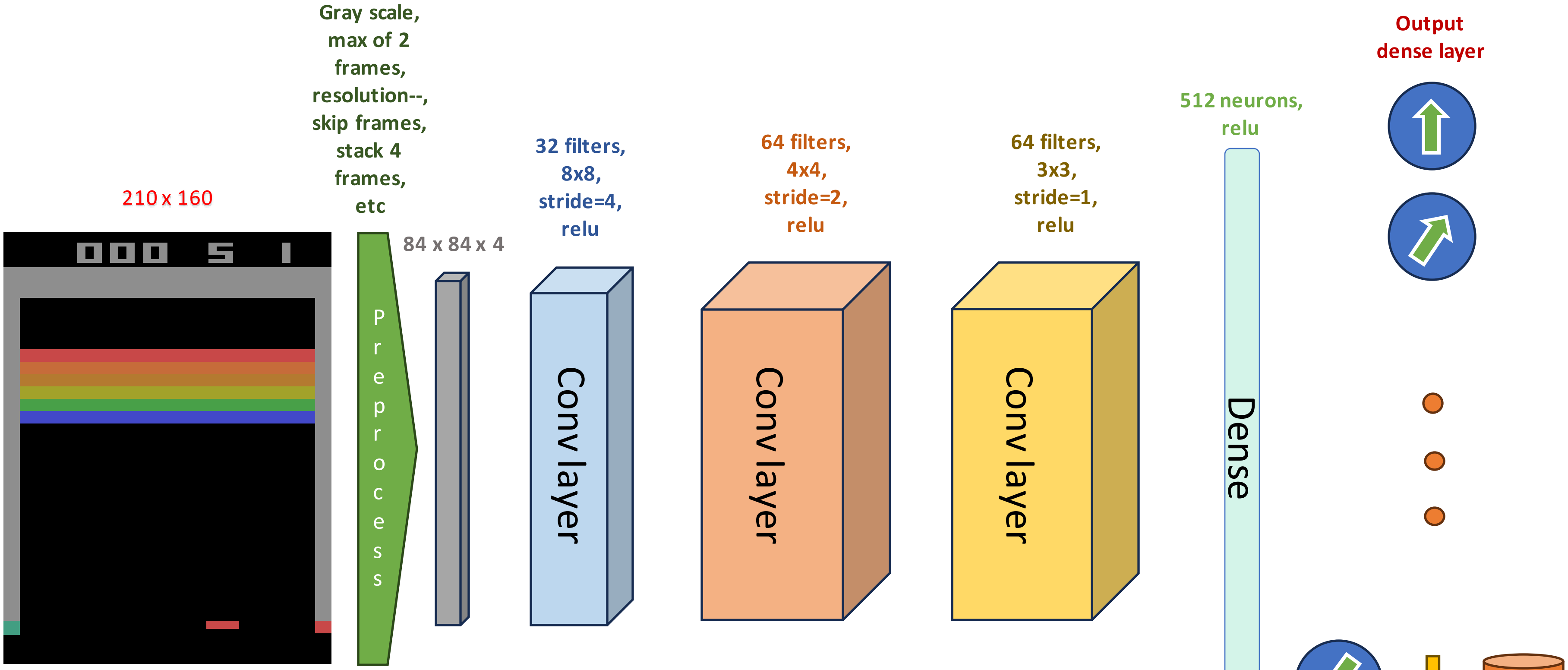
Deep Q-network: Solution?



Deep Q-network: Neural network



Deep Q-network: Architecture (DeepMind, 2015)



https://github.com/Farama-Foundation/Gymnasium

$$L_i(\theta_i) = E_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

$$\frac{\partial L_i(\theta_i)}{\partial \theta_i} = E_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)\frac{\partial Q(s,a;\theta_i)}{\partial \theta_i}\right]$$

Deep Q-network: Problem

- Replacing Q-table with a neural network was not revolutionary.

Let's sketch our algorithm so far.

```
for each episode:
  while not done:
    batch = []
    for several frames:
      reward, next_frame = env.step(action)
      batch.append(current_frame, reward, action, next_frame)
    QNN.train(batch)
```

Deep Q-network: Problem

- Replacing Q-table with a neural network was not revolutionary.
- Being able to train the neural network stably and at scale was the revolutionary idea.

Deep Q-network: Problem

- Replacing Q-table with a neural network was not revolutionary.
- Being able to train the neural network stably and at scale was the revolutionary idea.

Spot a problem…

```
for each episode:
  while not done:
    batch = []
    current_frame = env.reset()
    for several frames:
      action = QNN.predict(current_frame)
      reward, next_frame = env.step(action)
      batch.append(current_frame, reward, action, next_frame)
      current_frame = next_frame
    QNN.train(batch)
```

Deep Q-network: Problem

- Replacing Q-table with a neural network was not revolutionary.
- Being able to train the neural network stably and at scale was the revolutionary idea.

Spot a problem…

```
for each episode:
  while not done:
    batch = []
    current_frame = env.reset()
    for several frames:
      action = QNN.predict(current_frame)
      reward, next_frame = env.step(action)
      batch.append(current_frame, reward, action, next_frame)
      current_frame = next_frame
    QNN.train(batch)
```

! Our QNN is trained with highly correlated batch.

! A neural network should be trained with training data that represents the actual data.

Deep Q-network: Experience Replay

Replay buffer: all actions from all episodes

state	Action	Next state	Reward

Sample

Batch: small sample for QNN training.

state	Action	Next state	Reward

Deep Q-network: Experience Replay

```
replay_buffer = []
for each episode:
  while not done:
    current_frame = env.reset()
    for several frames:
      action = QNN.predict(current_frame)
      reward, next_frame = env.step(action)
      replay_buffer.append(current_frame, reward, action, next_frame)
      current_frame = next_frame
    batch = replay_buffer.sample(batch_size)
    QNN.train(batch)
```

Deep Q-network: Chasing a moving target

The target value in the loss function depends on QNN's weight and keep changing.

$$L_i(\theta_i) = E_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right]$$

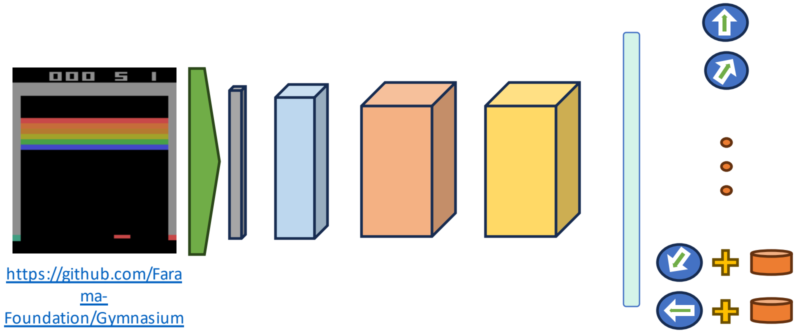
$$\frac{\partial L_i(\theta_i)}{\partial \theta_i} = E_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)\frac{\partial Q(s, a; \theta_i)}{\partial \theta_i}\right]$$

Deep Q-network: Chasing a moving target

The target value in the loss function depends on QNN's weight and keep changing.

$$L_i(\theta_i) \approx E_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} \hat{Q}(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

$$\frac{\partial L_i(\theta_i)}{\partial \theta_i} \approx E_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} \hat{Q}(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)\frac{\partial Q(s,a;\theta_i)}{\partial \theta_i}\right]$$



https://github.com/Farama-Foundation/Gymnasium

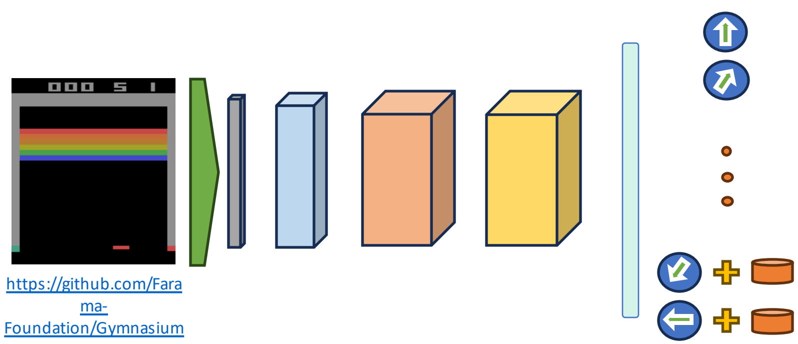
$$Q(s,a;\theta)$$

Deep Q-network: Chasing a moving target

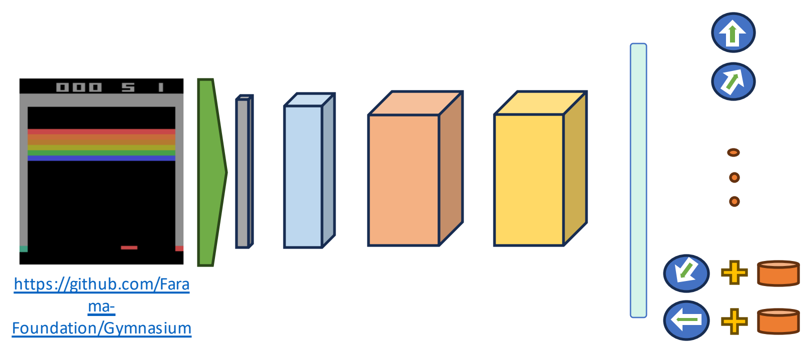
The target value in the loss function depends on QNN's weight and keep changing.

$$L_i(\theta_i) \approx E_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} \hat{Q}(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

$$\frac{\partial L_i(\theta_i)}{\partial \theta_i} \approx E_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} \hat{Q}(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)\frac{\partial Q(s,a;\theta_i)}{\partial \theta_i}\right]$$



$Q(s,a;\theta)$



$\hat{Q}(s,a;\theta^-)$

Deep Q-network: Implementations

Implementations:

- https://keras.io/examples/rl/deep_q_network_breakout/,
- Lecture 10 Colab (trained on a non-Atari game).