Reinforcement Learning

Different learning approaches

Supervised learning

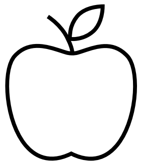
Data:

Labelled inputs (x, y)

Goal:

Learn a function to map $x \to y$

Example:



"This is an apple"

Applications:

Finding correlations

Unsupervised learning

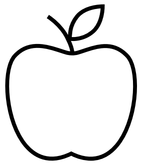
Data:

Unlabelled inputs (x)

Goal:

Learn the underlying structure of x

Example:



"These are similar"

Applications:

Categorizing

Feature recognition and exploitation

Reinforcement learning

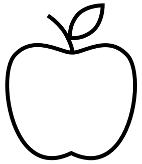
Data:

State-action pairs (s_t, a_t) and rewards r_t

Goal:

Maximise future rewards

Example:



"Collect as many as you can of these to win"

Applications:

Interacting with complex systems

Definitions and terminology

Agent: The entity taking actions.

Environment: The world in which the agent exists and operates.

Action a: Something the agent can perform in the environment.

Action space A: The set of all the actions.

Observation: Data accessible from the environment.

State s: Data describing the environment.

Reward r: A measure of the success (or failure) of the agent's action.

Total future reward R_t: the sum of all future rewards

$$R_t = \sum_{i=t}^{T} r_i = r_t + r_{t+1} + r_{t+2} + \cdots + r_T$$

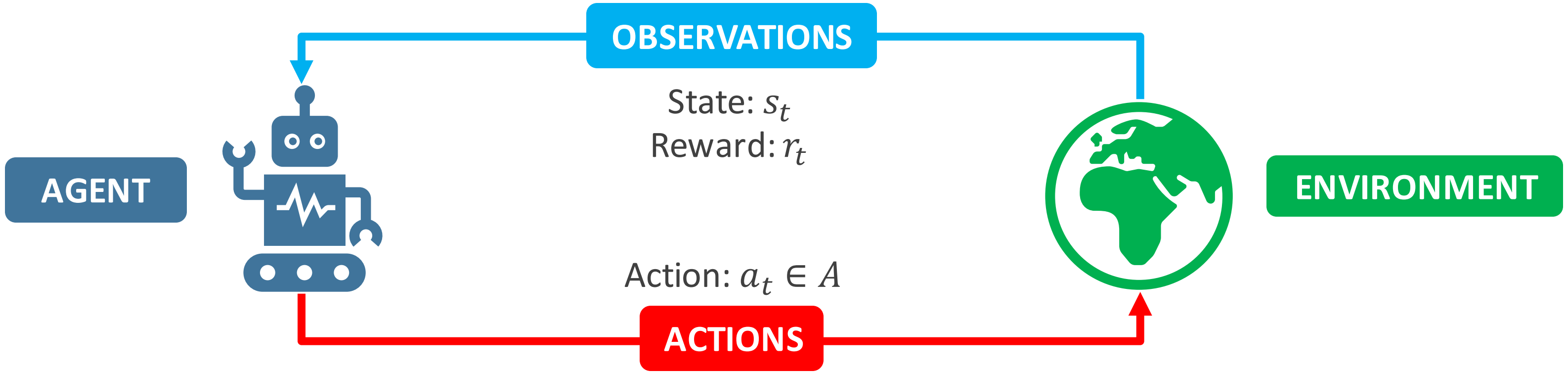
Total discounted future reward R_t: the sum of all future rewards, favouring short term rewards

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i = r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} + \cdots + \gamma^{T-t}\, r_T$$

Q-function Q: the expected future reward for taking an action in a given state.

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

Policy $\pi(s)$: A strategy to choose an action for a given state.



A practical RL problem – CartPole-v1

Environment

- A cart moves on a frictionless horizontal surface with an inverted pendulum attached.

State

- Position and velocity of the cart.
- Angular position and velocity of the pendulum.

Action space

- Push the cart to the left.
- Push the cart to the right.

Discrete action space of size 2.

Reward

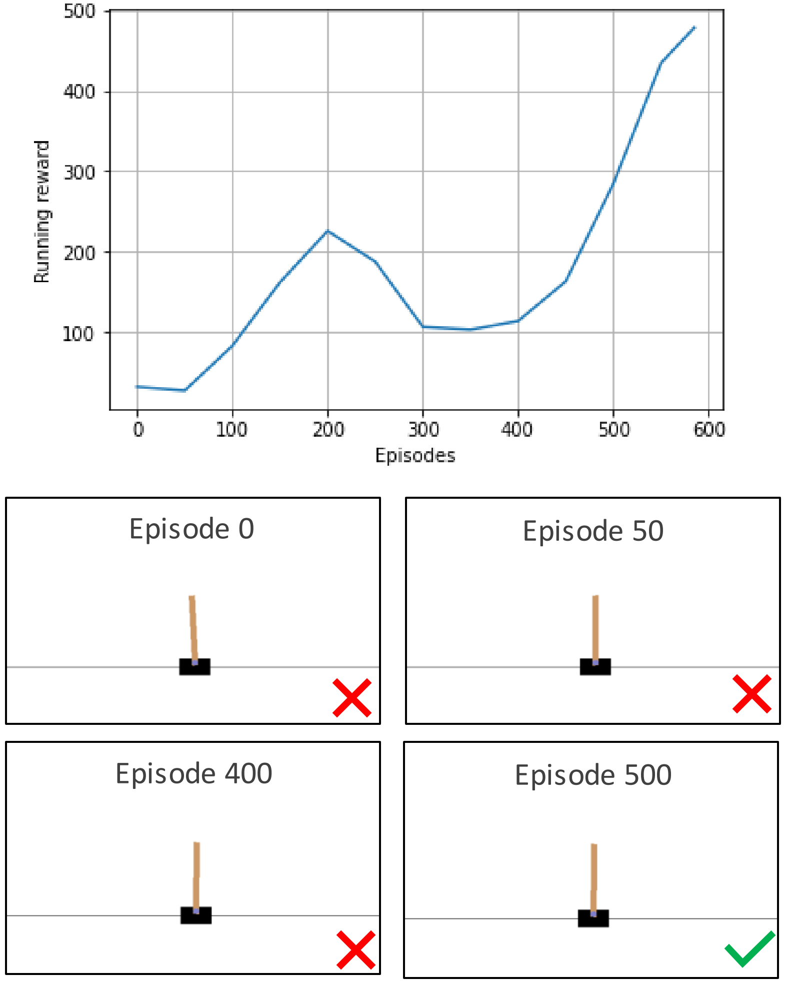
- +1 for each timestep the pendulum is kept upright.

Stop

- If the cart hits the boundary of the space.
- If the pole reaches 15°.
- If 500 timesteps have passed.

Success

- The average reward over the last 100 episodes is greater than 475.



Training progress of CartPole-v1 [2, 6]

This is one of the environments provided by the OpenAI Gym [6].

Q-value learning

Core idea

If we have $Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$ the optimal policy is easily defined

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a)$$

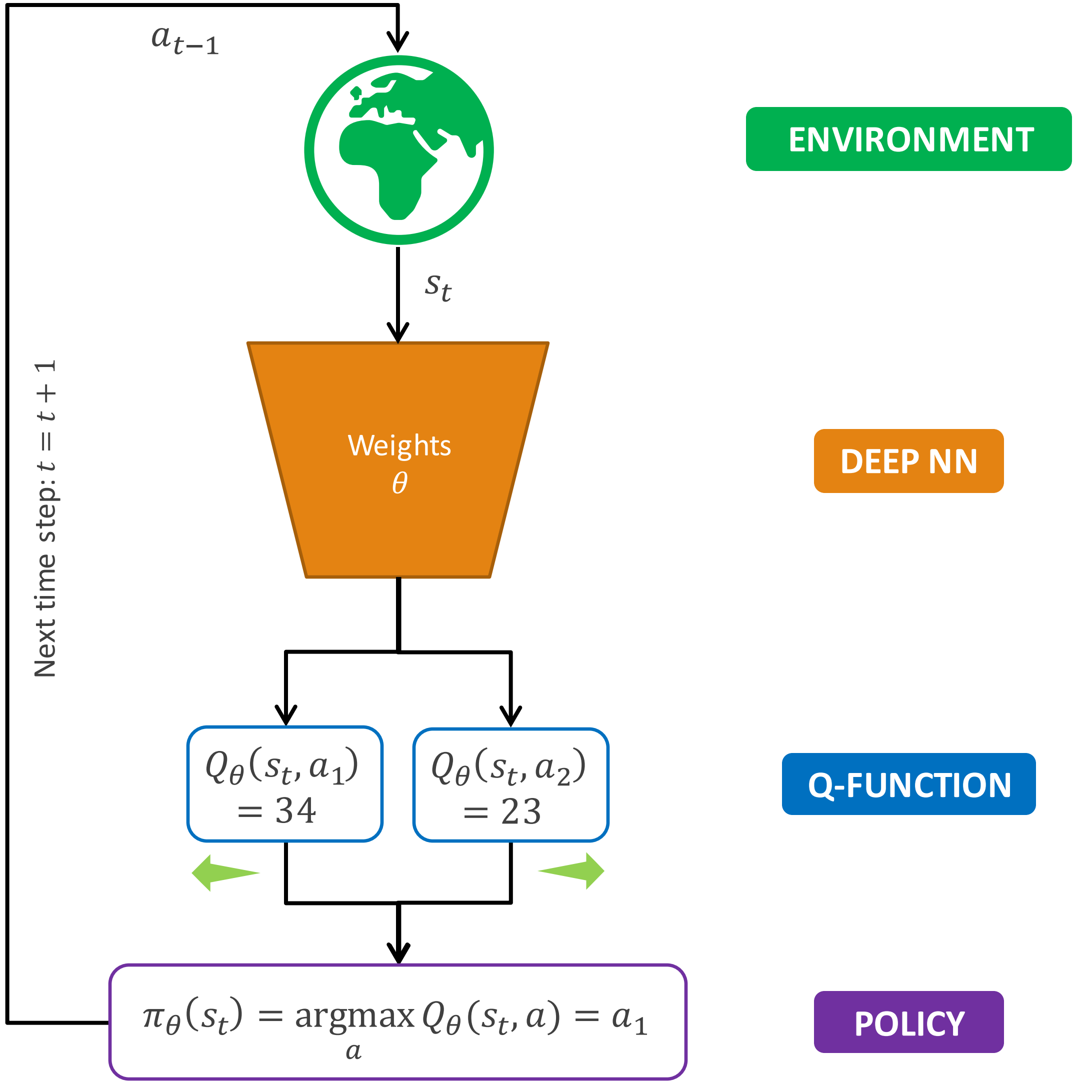
We can use a NN to predict $Q(s,a)$: $Q_\theta(s,a)$ (Deep Q Network – DQN)

Pros

- Works well with small, discrete action spaces

Cons

- Can't work in a continuous action space (limited # of outputs)
- Struggles with big action spaces or complex Q-functions
- The policy is fully deterministic: cannot work with stochastic events



Policy learning

Core idea

Instead of predicting $Q(s, a)$ and extrapolating $\pi(s)$ from it, just predict $\pi(s)$ directly.

Additionally, let's make $\pi(s)$ stochastic.

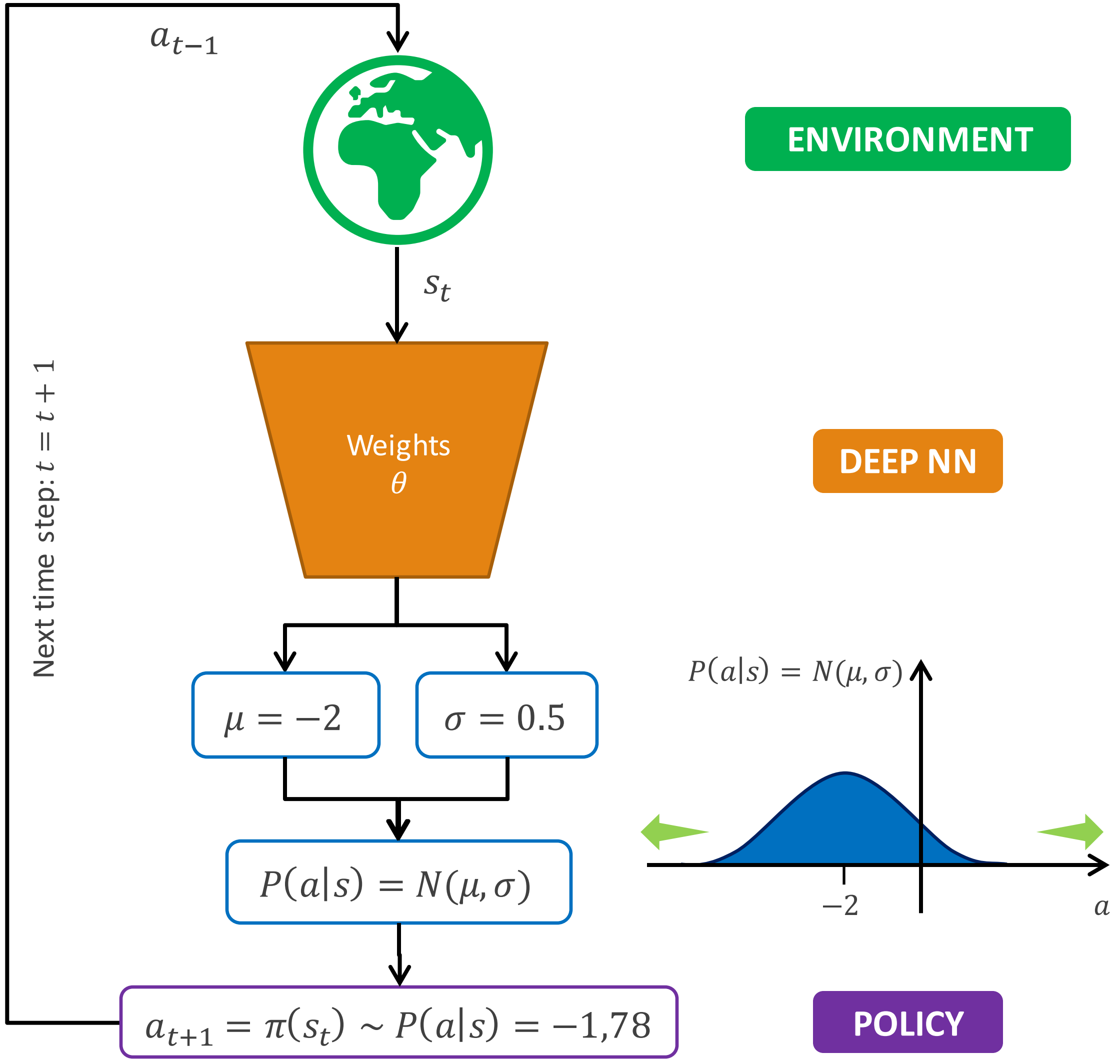
We can use a NN to predict (μ_π, σ_π): the action is decided by sampling $a = \pi(s) \sim N(\mu_\pi, \sigma_\pi)$.

Pros

- Can manage a continuous action space
- Better performance with big actions spaces
- Works with stochastic events

Cons

- Can still have problems with complex Q-functions



Policy Gradient training

Training algorithm

1. Initialise the agent (e.g. with a random policy π_0)
2. Run the policy until termination ⚠
3. Record all (s_t, a_t, r_t) and calculate all R_t (a posteriori)
4. Modify the policy (the weights θ of the NN):
   - Increasing the probability of actions which resulted in low rewards
   - Decreasing the probability of actions which resulted in high rewards
5. Repeat from step 2

- Can be problematic for a robot.
- Task termination may imply failure, damage or danger.
- Simulations can help, up to a point (simulation-to-real transfer)

How can we do this in practice?

$$\text{Loss: } L = -\underbrace{\boxed{\log P(a_t|s_t)}}_{\text{Log-likelihood of an action}}\underbrace{\boxed{R_t}}_{\text{Reward}}$$

$$\theta_{k+1} = \theta_k - \nabla L$$
$$\theta_{k+1} = \theta_k + \underbrace{\boxed{\nabla \log P(a_t|s_t)\, R_t}}_{\text{Policy gradient}}$$

Actor-critic agents

Some methods combine value and policy learning.

Core idea

- Lower the variance of the loss by removing a baseline from the reward [5]
- The agent runs two models:

1. Critic

- Predicts the Value baseline V_θ^π
- Over many episodes learns to align $V_\theta^\pi(s_t)$ with R_t
- Huber loss [4] is less sensitive to outliers than MSE

$$L_{\text{critic}}(s_t) = L_{\text{Huber}}(R_t, V_\theta^\pi(s_t))$$

2. Actor

- Defines the policy π_θ
- Over many episodes optimizes the policy π_θ using the Policy Gradient
- Similar loss to standard Policy Gradient, but the Reward is replaced by the Advantage (i.e. Reward w.r.t. the Baseline)

$$L_{\text{actor}}(s_t) = -\log \pi_\theta(a_t|s_t) A_t \ \text{ with } \ A_t = R_t - V_\theta^\pi(s_t)$$

Pros

From the critic:

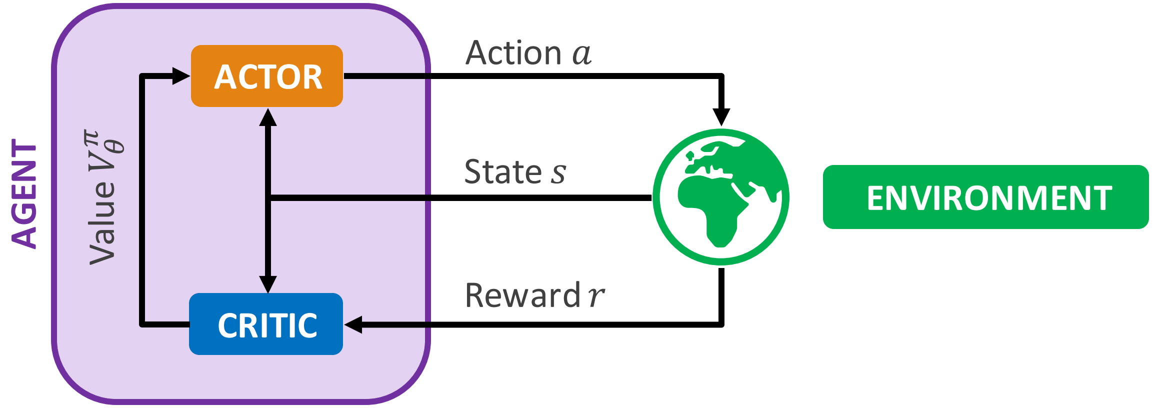
- efficient value estimation, with low variance

From the actor:

- smoother gradient thanks to Policy Learning

Cons

- Complex both computation and implementation wise
- Convergence is not guaranteed: the critic trains itself without ground truth data



References and resources

[1] Alexander Amini, “MIT 6.S191: Reinforcement Learning”, MIT 6.S191 lessons, https://www.youtube.com/watch?v=93M1l_nrhpQ

[2] “Playing CartPole with the Actor-Critic Method”, Tensorflow Tutorials, https://www.tensorflow.org/tutorials/reinforcement_learning/actor_critic

[3] Nguyen, T. T., Nguyen, N. D., Nahavandi, S. (2018). “Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications”. *IEEE Transactions on Cybernetics*, *50*(9), 3826–3839. https://doi.org/10.1109/tcyb.2020.2977374

[4] Huber Loss, Wikipedia, https://en.wikipedia.org/wiki/Huber_loss

[5] Chris Yoon, “Understanding Actor Critic Methods and A2C”, Towards Data Science, https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

[6] “CartPole-v1”, OpenAI Gym, https://gym.openai.com/envs/CartPole-v1/

[7] Giorgio Bonvicini, "Reinforcement Learning", https://github.com/GioBonvi/MachineLearning/tree/main/RL