AI Enabled Control Engineering

Lecture 5: CartPole, Reinforcement Learning, and DQN Practice

Professor Xun Huang

Aeronautics and Astronautics
College of Engineering
Peking University

huangxun@pku.edu.cn
https://xunger99.github.io/xunger/

Recap

- Frequency response, Bode plot, and Nyquist plot.
- State-space modelling perspective.
- Controllability and observability.
- State feedback and the limitations of purely classical methods.

Lecture 5

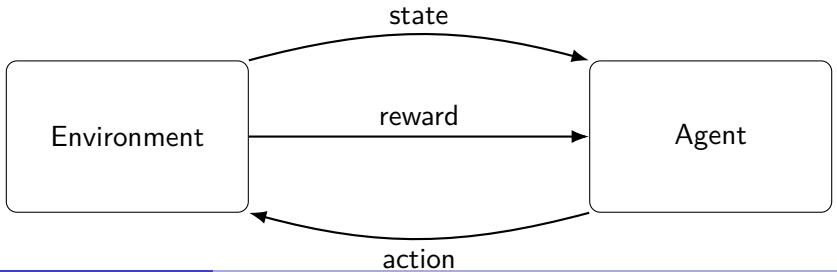
- Reinforcement learning overview.
- Q-learning and why Q-tables fail for CartPole.
- Deep Q-Network (DQN): replay buffer and target network.
- What Torch, Gymnasium, and Stable-Baselines3 do.
- Practical workflow of this lab.
- Step-by-step setup on macOS and Windows.

What is reinforcement learning?

Reinforcement learning (RL) studies how an agent learns by interacting with an environment. The objective is to maximize long-term cumulative reward.

At each step:

- the agent observes a state,
- chooses an action,
- receives a reward,
- and moves to the next state.



Three broad families of RL methods

- **Value-based methods**
  - estimate a value function or action-value function,
  - examples: Q-learning, SARSA, DQN.
- **Policy-based methods**
  - learn a policy directly.
- **Actor-Critic methods**
  - combine a policy model and a value model.

Today we follow the value-based route that leads from Q-learning to DQN.

Q-learning idea

The action-value function is

$$Q(s, a),$$

which measures how good action a is at state s.

Q-learning updates the estimate by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big].$$

Here

- α is the learning rate,
- γ is the discount factor,
- $\max_a Q(S_{t+1}, a)$ is the greedy next-state estimate.

Why a Q-table fails for CartPole

In a small discrete problem, we may store a table $Q[s][a]$.

States \ Actions	a_1	a_2	a_3
s_1	$Q(s_1, a_1)$	$Q(s_1, a_2)$	$Q(s_1, a_3)$
s_2	$Q(s_2, a_1)$	$\cdots$	$\cdots$
s_3	$Q(s_3, a_1)$	$\cdots$	$\cdots$

But CartPole states are continuous: $s = [x, \dot{x}, \theta, \dot{\theta}]$.
So we cannot enumerate all states with a finite exact table.
Therefore, we need a function approximator: $Q(s, a; \theta)$, where θ now denotes neural-network parameters.

Neural-network view of $Q(s, a)$

For CartPole:

- input: the state vector
$$[x, \dot{x}, \theta, \dot{\theta}],$$
- output: one Q-value for each discrete action.

If the action space is

$$a = 0 \quad \text{(push left)}, \qquad a = 1 \quad \text{(push right)},$$

then the network outputs

$$[Q(s, 0),\ Q(s, 1)].$$

So the control decision becomes:

$$a^* = \arg\max_a Q(s, a; \theta).$$

Deep Q-Network (DQN)

DQN replaces the explicit Q-table with a neural network.

Two key ideas make training much more stable:

- **Experience replay**
  - store transitions (s_t, a_t, r_t, s_{t+1}) in a replay buffer,
  - sample mini-batches randomly during training.
- **Target network**
  - maintain a second network with delayed parameters,
  - use it to compute more stable targets.

DQN target and loss

The standard DQN target is

$$y_i = r_i + \gamma \max_{a'} Q(s_i', a'; \theta^-),$$

where θ^- denotes the target-network parameters.

The loss is based on temporal-difference error:

$$L(\theta) = \mathbb{E}\Big[\big(y_i - Q(s_i, a_i; \theta)\big)^2\Big].$$

In words:

- the online network predicts current Q-values,
- the target network provides a delayed learning target,
- replay-buffer sampling breaks strong temporal correlation.

DQN workflow

1. Initialize the replay buffer D.
2. Initialize the online Q-network $Q(s, a; \theta)$.
3. Copy the target network: $Q(s, a; \theta^-) \leftarrow Q(s, a; \theta)$.
4. Repeat:
   - observe the current state s,
   - choose an action by ε-greedy policy,
   - execute the action and obtain (r, s'),
   - store (s, a, r, s') in D,
   - sample a mini-batch from D,
   - compute the target $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$,
   - update θ by minimizing $\big(y - Q(s, a; \theta)\big)^2$.
5. Periodically update the target network: $\theta^- \leftarrow \theta$.

CartPole training rules

In CartPole, the state is

$$[x, \dot{x}, \theta, \dot{\theta}].$$

The action space is discrete:

$$0 = \text{push left}, \qquad 1 = \text{push right}.$$

A reward of $+1$ is usually given for every valid step.

An episode terminates if

- the cart moves too far from the center,
- or the pole angle becomes too large,
- or the maximum episode length is reached.

So during training, the agent tries to keep the pole upright for as many steps as possible.

Why CartPole is a good first RL task

CartPole is a very good first task because

- the state is low-dimensional,
- the action space is discrete,
- the reward is easy to understand,
- the physical meaning is intuitive,
- training is much cheaper than in large-scale RL tasks.

It is therefore an ideal bridge from control theory to practical RL coding.

What does each software layer do?

In this lab, three layers appear repeatedly:

- **Torch**
  - tensor computation,
  - neural networks,
  - automatic differentiation,
  - optimizer and training backend.
- **Gymnasium**
  - environment interface,
  - CartPole dynamics and rendering,
  - reset/step/action-space/state-space definitions.
- **Stable-Baselines3**
  - RL algorithm implementation,
  - here mainly the DQN training pipeline.

Our practical route in this lab

We do not ask students to install official Gymnasium or Stable-Baselines3 packages from scratch.

Instead, the lab package already contains local copies of:

- `gymnasium/`
- `stable_baselines3/`

So the practical route becomes

1. prepare Python,
2. download the package from the WeChat group,
3. unzip it,
4. install the minimum external libraries,
5. run `Pole_cart.py`.

What is inside the lab package?

The package sent in the WeChat group should contain at least:

- `Pole_cart.py`
- `gymnasium/`
- `stable_baselines3/`

Students mainly modify only

`Pole_cart.py`.

The two folders

- `gymnasium/`
- `stable_baselines3/`

are treated as supporting source code and usually do not need to be edited in the first lab.

Main idea of the CartPole script

A typical CartPole DQN script has four parts:

- add local package paths,
- create the environment,
- define the neural-network architecture,
- create, train, test, and save the DQN model.

Typical imports are:

```
import gymnasium as gym
from stable_baselines3.dqn.dqn import DQN
from stable_baselines3.common.env_util import make_vec_env
from torch import nn
```

Minimal DQN configuration

```
policy_kwargs = dict(
    net_arch=[64, 64],
    activation_fn=nn.ReLU
)

model = DQN(
    policy="MlpPolicy",
    env=train_env,
    learning_rate=1e-3,
    buffer_size=100000,
    learning_starts=1000,
    batch_size=64,
    gamma=0.99,
    train_freq=4,
    target_update_interval=1000,
    verbose=1,
    policy_kwargs=policy_kwargs
)
```

- `net_arch`: hidden-layer sizes
- `learning_rate`: optimizer step size
- `buffer_size`: replay-buffer capacity
- `learning_starts`: start training after enough samples
- `batch_size`: mini-batch size
- `gamma`: discount factor
- `train_freq`: update frequency
- `target_update_interval`: target-network copy interval

Practical goal of Lecture 5

The first goal of this lab is not advanced algorithmic improvement.

The first goal is simply:

make the full CartPole training-testing pipeline run successfully on your own laptop.

- correct Python installation,
- correct package installation,
- successful training logs,
- successful testing episodes after training.

macOS: Step 1 — prepare Python

On macOS, first make sure that `python3` works in Terminal.
Run:

```
python3 --version
```

If Python is not available, install a recent Python 3 release first.

After downloading the package from the WeChat group and unzipping it, enter the folder named `class5`.
If it is on Desktop, run

```
cd ~/Desktop/class5
```

If it is in Downloads, run

```
cd ~/Downloads/class5
```

macOS: Step 2 — download and unzip the lab package

Download the compressed package from the WeChat group.
After downloading, unzip it and enter the folder.
A typical structure is:

```
class5/
 Pole_cart.py
 gymnasium/
 stable_baselines3/
```

Check that these three components exist before continuing.

macOS: Step 3 — create a virtual environment

In Terminal, run

```
cd ~/Desktop/class5
python3 -m venv rl_env
```

or, if your folder is in Downloads,

```
cd ~/Downloads/class5
python3 -m venv rl_env
```

Then use the Python inside that environment directly:

```
rl_env/bin/python3 -m pip install --upgrade pip setuptools wheel
```

macOS: Step 4 — install the required libraries

Install the external libraries by

```
rl_env/bin/python3 -m pip install numpy
rl_env/bin/python3 -m pip install torch torchvision
rl_env/bin/python3 -m pip install pygame
rl_env/bin/python3 -m pip install cloudpickle
rl_env/bin/python3 -m pip install matplotlib
rl_env/bin/python3 -m pip install pandas
```

We do **not** require you to install official

- Gymnasium
- Stable-Baselines3

because local copies are already included in the lab package.

macOS: Step 5 — test the environment

Before running the main script, verify the main dependencies:

```
rl_env/bin/python3 -c "import numpy"
rl_env/bin/python3 -c "import torch"
rl_env/bin/python3 -c "import pygame"
```

If all three commands return without errors, the basic environment is ready.

macOS: Step 6 — run the CartPole script

If your folder is on Desktop, run

```
cd ~/Desktop/class5
rl_env/bin/python3 Pole_cart.py
```

If your folder is in Downloads, run

```
cd ~/Downloads/class5
rl_env/bin/python3 Pole_cart.py
```

If everything is correct, you should see

- training logs in Terminal,
- testing episode scores after training,
- optionally a rendering window during evaluation.

Windows: Step 1 — prepare Python

On Windows, first make sure Python 3 is installed.
Open **Command Prompt** or **PowerShell** and check:

```
python --version
```

If Python is not installed, install a recent Python 3 release first.

Then download the class package from the WeChat group and unzip it.

Windows: Step 2 — enter the class folder

After unzipping, the folder should contain at least

```
Pole_cart.py
gymnasium\
stable_baselines3\
```

If the folder is directly under drive D, enter it by

```
cd D:\class5
```

If it is on Desktop, first locate the full path and then enter that folder.

Windows: Step 3 — create a virtual environment

Create a local virtual environment:

```
python -m venv rl_env
```

Then use the Python inside it directly:

```
rl_env\Scripts\python.exe -m pip install --upgrade pip setuptools wheel
```

Again, direct use of the local interpreter is usually more robust than relying on shell activation.

Windows: Step 4 — install the required libraries

Install the external libraries:

```
rl_env\Scripts\python.exe -m pip install numpy
rl_env\Scripts\python.exe -m pip install torch torchvision
rl_env\Scripts\python.exe -m pip install pygame
rl_env\Scripts\python.exe -m pip install cloudpickle
rl_env\Scripts\python.exe -m pip install matplotlib
rl_env\Scripts\python.exe -m pip install pandas
```

The local package already includes the source code for:

- Gymnasium
- Stable-Baselines3

so these do not need to be installed separately.

Windows: Step 5 — test the environment

Before running the main script, test the key packages:

```
rl_env\Scripts\python.exe -c "import numpy"
rl_env\Scripts\python.exe -c "import torch"
rl_env\Scripts\python.exe -c "import pygame"
```

If no errors appear, the main environment is ready.

Windows: Step 6 — run the CartPole script

If your folder is directly under drive D, run

```
cd D:\class5
rl_env\Scripts\python.exe Pole_cart.py
```

If your folder is elsewhere, replace `D:\class5` with your actual folder path.

A successful run should show

- training logs,
- no import errors,
- testing scores at the end.

Important practical details

Several details are especially important:

- Use the Python inside the local virtual environment.
- Do not mix different Python installations by accident.
- Do not install official Gymnasium and Stable-Baselines3 unless required.
- Make sure `Pole_cart.py` is in the same folder level as `gymnasium/` and `stable_baselines3/`.
- If a package import fails, fix the missing dependency first.

What should you mainly modify?

For the first lab, you should mainly work on

`Pole_cart.py`.

Typical follow-up tasks include:

- tuning the current DQN hyperparameters,
- changing the neural-network size,
- changing the total training steps,
- adjusting the ε-greedy exploration parameters,
- designing a modified reward function,
- recording testing logs such as

$$x, \dot{x}, \theta, \dot{\theta}, \text{ action, reward,}$$

- plotting curves such as the test θ–t trajectory,
- comparing performance under different settings.

The folders

Summary

- Reinforcement learning overview.
- Q-learning, function approximation, and DQN.
- Replay buffer and target network.
- CartPole training rules.
- Roles of Torch, Gymnasium, and Stable-Baselines3.
- Complete practical workflow for the lab package.
- Detailed setup steps for macOS and Windows.