Laboratory - Deep Reinforcement Learning: Foundations and Practical Environment Setup for Real-World Applications

Franco Terranova

Université de Lorraine, CNRS, Inria, LORIA

franco.terranova@inria.fr

August 20, 2024

Tutorial Material

https://terranovafr.github.io/teaching/2024-EASSS-Course

Overview

1. DRL Agents - Practical Setup
   - StableBaselines3
2. Environment - Practical Setup
   - OpenAI Gym
3. Grid World
   - POMDP
   - Matrix Representation
   - Generalization

Deep RL Agents

Components:

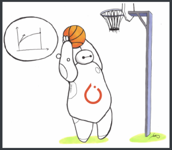
- **Libraries for Well-Established Algorithms:**
  - **Stable Baselines3:** https://stable-baselines3.readthedocs.io/en/master/
  - **Ray Rllib:** https://docs.ray.io/en/latest/rllib/index.html
  - **TF-Agents:** https://www.tensorflow.org/agents
  - **Keras-RL:** https://github.com/keras-rl/keras-rl
- **Neural Network Architecture:**
  - Number of layers and neurons per layer
  - Activation functions (e.g., ReLU, Tanh)
  - Network type (e.g., feedforward, convolutional, recurrent)

Deep RL Agents: Setup

- **Algorithm Hyperparameters:**
  - Learning rate
  - Batch size
  - Discount factor (γ)
  - Exploration strategy (e.g., ϵ-greedy for the DQN)
  - ...
- **Optimizer Selection:**
  - Adam
  - Stochastic Gradient Descent
  - ...
- **Additional Considerations (RL Specific):**
  - **Reward Shaping:** Modify rewards to guide the agent towards desired behaviors
  - **Training Stability:** Use techniques to enhance training stability
  - ...

Stable Baselines3

Stable Baselines3



master

Search docs

USER GUIDE

- Installation
- Getting Started
- Reinforcement Learning Tips and Tricks
- Reinforcement Learning Resources
- RL Algorithms
- Examples
- Vectorized Environments
- Policy Networks
- Using Custom Environments
- Callbacks
- Tensorboard Integration
- Integrations

/ DQN

Edit on GitHub

DQN

Deep Q Network (DQN) builds on Fitted Q-Iteration (FQI) and make use of different tricks to stabilize the learning with neural networks: it uses a replay buffer, a target network and gradient clipping.

Available Policies

`MlpPolicy`	alias of `DQNPolicy`
`CnnPolicy`	Policy class for DQN when using images as input.
`MultiInputPolicy`	Policy class for DQN when using dict observations as input.

Notes

- Original paper: https://arxiv.org/abs/1312.5602
- Further reference: https://www.nature.com/articles/nature14236

Note

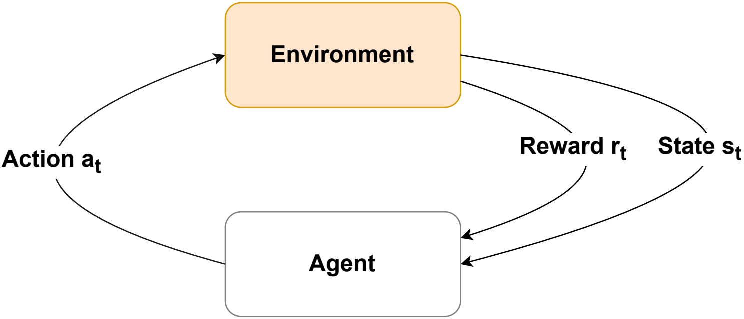
This implementation provides only vanilla Deep Q-Learning and has no extensions such as Double-DQN, Dueling-DQN and Prioritized Experience Replay.

Can I use?

- Recurrent policies: ❌
- Multi processing: ✔
- Gym spaces:

Website: `https://stable-baselines3.readthedocs.io/en/master/index.html`

Environment Focus



Environment Setup

- The environment maps an action and state to the next state and reward:

$$\text{Next State}, \text{Reward} = \mathcal{E}(\text{State}, \text{Action})$$

where $\mathcal{E}$ represents the environment's dynamics

- Define internal dynamics so that agent can learn from it

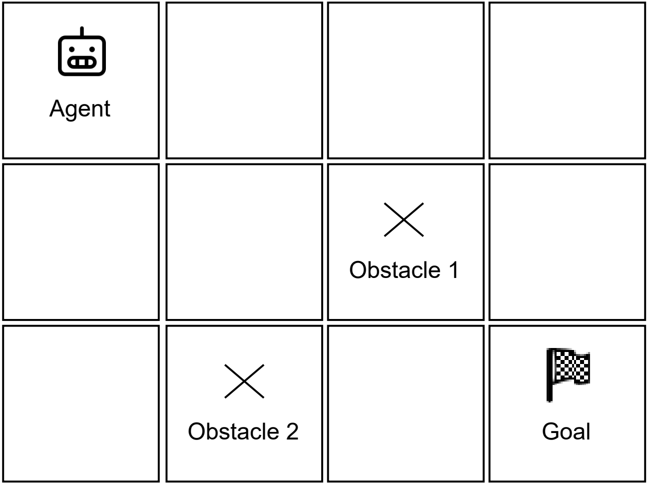
OpenAI Gym Library

- **Standard Library:** Widely used for creating and testing RL environments
- **Class-Based:** Provides a Python class with attributes and methods to define and manage environments
- **Attributes:** Includes state and action spaces, attributes for rewards calculation, . . .
- **Methods:** Simulate the dynamics and should adhere to the standard
- **Gymnasium:** Currently maintained version of Gym

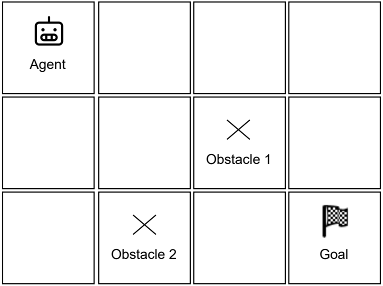
OpenAI Gymnasium Methods

- `__init__`: Initializes the environment, setting up its initial dynamics and attributes
- `reset`: Resets the environment and returns:
  - **State**: The initial state
  - **Info**: A dictionary with additional information (optional)
- `step(action)`: Takes an action, updates the environment, and returns a tuple containing:
  - **Next State**: The state after the action
  - **Reward**: The reward received after taking the action in the state
  - **Done**: A boolean indicating if the episode has ended
  - **Truncated**: A boolean indicating if the episode has been truncated
  - **Info**: A dictionary with additional information (optional)
- `render`: Displays a visual representation of the environment (optional)
- `close`: Cleans up and closes the environment when done (optional)

Grid World



Grid World - Cell View



- **Observation:** (x, y) — The current position in the grid
- **Action:** Movement directions — up, down, left, right
- **Reward:**
    - Small Penalty — If the agent moves to an empty cell
    - Bigger Penalty — If the agent moves into an obstacle or outside
    - Prize — If the agent reaches the goal
- **Episode:** Terminates when the goal or cut-off is reached

Grid World - POMDP

- **Static Environment:**
  - The environment does not change over time
  - Finding the optimal policy is straightforward
  - The cell view is a MDP for the task
- **General Case:**
  - The grid evolves or changes
  - Application to another grid with different parameters
  - A generalizable policy may be complex to determine with this observation, being a POMDP for the task
- **Challenges on using this POMDP:**
  - Same state leading to different action outcome
  - Exposing multiple training environments may lead to instable learning

Matrix - MDP

- **Full visibility:**
  - Grid is translated into a 1D array
  - Agent, Obstacles, Goal: Encoding choice
    - E.g. Current position represented with 1, Obstacles represented with 2s, and the goal with 3
- Now each observation will have a deterministic (reward, next state) when selecting an action
- **Assumption:** Fixed width and height for now

Switching Module

- **Periodic Changes:**
  - The grid environment can be periodically switched (updated) with another version
  - New environments can have different obstacle positions, and goal locations
- **Dynamic Training:**
  - During training switch periodically based on a switch interval
  - Ensures the agent can generalize across different environments

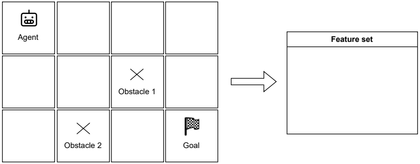
Experiments

- What is the impact of ... ?
  - Episode cut-off
  - Rewards and their scale
  - Value of γ
  - Number of training iterations
  - Relation between grid size and number of iterations
  - Epsilon decay
  - Number of environments

Generalize across grid sizes

- **Grid size** determines the number of input and output neurons of the agent's NN
- Agent's NN specialized to a given grid size
- Possible solutions:
  - Re-train a NN for every grid size
  - Padding techniques to the maximum grid size
    - Need to set a maximum
    - Waste of resources for small grids
    - May require a larger NN

Other solutions



1. Trade-off observability range
    - E.g. surrounding pixels in the 1-hop neighborhood
2. Features describing the environment
    - Requires manual features engineering
3. Potential combination of these solutions

Convolutional Neural Networks

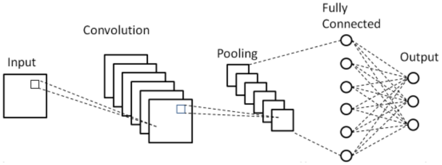


Image Source: Mesuga, Reymond Bayanay, Brian. (2021). A Deep Transfer Learning Approach to Identifying Glitch Wave-form in Gravitational Wave Data.

- Avoid feature engineering
- CNN will automatically determine a feature representation
- Updates driven by the DRL loss

Well-Established Environments

Popular Gym Environments:

- `CartPole-v1`: Balancing a pole on a moving cart
- `MountainCar-v0`: Driving a car up a hill
- `LunarLander-v2`: Landing a spacecraft on the moon
- `Atari` environments: Classic arcade games (e.g., `Pong-v0`, `Breakout-v0`)

Example of Utilization:

```
import gymnasium as gym
env_cartpole = gym.make('CartPole-v1')
```

Open Problems

- **State and Action Space Encodings:** Efficient representation that allows learning
- **Markovian Property:** Ensures that observation encodes all information needed
- **Exploration vs. Exploitation:** Balancing the trade-off between exploring new strategies and exploiting known ones
- **Sample Efficiency:** Improving the efficiency of learning algorithms to require fewer samples

New Directions

- Addressing catastrophic forgetting with **continual reinforcement learning**
- Exploring scenarios with multiple agents (**Multi-agent RL**) and incorporating game theory
  - Independent learners
  - Cooperation games
  - Competitive games
- **Inverse Reinforcement Learning** to derive realistic reward functions
- **Meta Reinforcement Learning** enables fast learning across different tasks

References

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann (2021)
Stable-Baselines3: Reliable Reinforcement Learning Implementations.
Journal of Machine Learning Research, 22, 268:1-8.
`http://jmlr.org/papers/v22/20-1364.html`.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016)
OpenAI Gym.
arXiv:1606.01540, `https://arxiv.org/abs/1606.01540`.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton (2015)
Deep learning.
Nature, 521(7553), 436–444.

Advance Your RL Agents to New Horizons!