Deep Reinforcement Learning

Introduction and Hands-on in Simple Environments

Gianluca Aguzzi gianluca.aguzzi@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna

Talk @ Fondamenti di Intellingenza Artificiale

25/05/2023

Me

- PhD student in Computer Science and Engineering
- Research interests:
  - Multi-agent systems
  - Distributed Collective Intelligence
  - Deep Reinforcement Learning
  - Multi-agent Reinforcement Learning
  - Distributed Macro-programming
- Lead developer of ScaFi
- Scala Lover & Functional Programming enthusiast

Contents

Road to **Deep** Reinforcement Learning

Overview

- **Reinforcement Learning (RL)** ➔ learning how to act in order to maximize a numerical reward signal
- Key features:
  - **trial-and-error search** (no supervisor)
  - **delayed reward** (no immediate feedback)
- Someone argue(Silver et al. 2021) that reward maximization is the sole principle that suffices to explain all aspects of intelligent behavior

Application

- Robotics
- Game playing
- Resource management
- Finance
- Chatbots
- Intelligent transportation systems

Question

Can standard RL (e.q., Q-Learning) solve these complex problems?

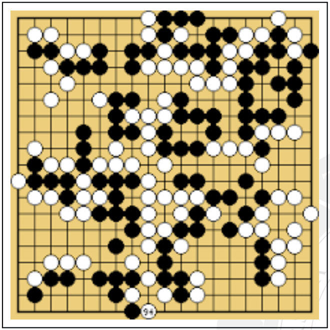
Reinforcement Learning Pitfalls: Large state space

Problem

- **State space** ➔ set of all possible states
- **State space explosion** ➔ the number of states is too large to be stored in memory

Example (Go)

- 10^{170} possible states **(!!!!)**
- 10^{80} atoms in the universe
- 10^{16} seconds since the Big Bang



Example (Chess)

- 10^{46} possible states
- total space required $\sim 10^{35}$ terabytes

Question

How to deal with large state space?

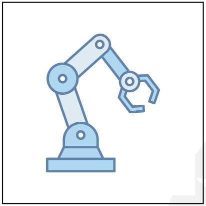
Reinforcement Learning Pitfalls: Continous Action Space

Problem

- **Action space** ➔ set of all possible actions
- **Continous action space** ➔ the actions are real numbers (e.g. $[0, 1]$) ➔infinite number of actions

Example (Robotics)

- **Action space** ➔ the set of all possible joint angles
- **Continous action space** ➔ the set of all possible real joint angles



Question

How to deal with continous action space?

Reinforcement Learning Pitfalls: Generalization

Problem

- **Generalization** ➔ the ability to perform well on previously unseen environments
- Can be also seen as **transfer learning** ➔ the ability to transfer knowledge from one environment to another
- **Generalization gap** ➔ the difference between the performance on the training environments and the performance on the test environments

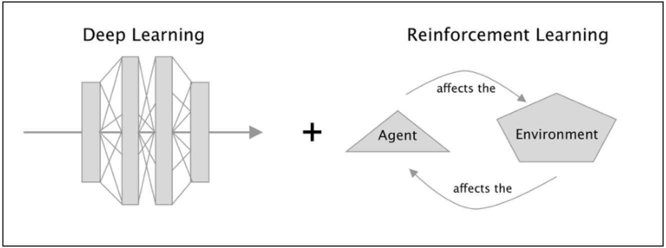
Example (Go)

- **Generalization** ➔ the ability to play well with different opponents
- **Generalization gap** ➔ the difference between the performance on the training set and the performance on the test set

Question

How to deal with generalization?

Deep Reinforcement Learning



Overview

- **Deep Reinforcement Learning (DRL) ➔** the use of deep neural networks to approximate the value function / policy

Key features

- **value function approximation** (instead of table) **➔ handle large state space**
- **policy gradient** (instead of Q-Learning) **➔ handle continous action space**
- **deep neural networks ➔ handle generalization** (Representation learning)

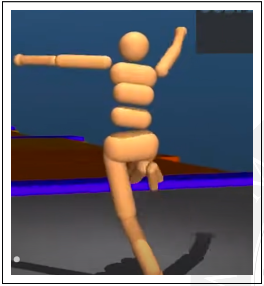
Algorithms types

Value-based

- the agent learns the value function
- **Example** ➔ *Deep Q-Learning*

Policy-based

- the agent learns the policy
- **Example** ➔ REINFORCE, PPO

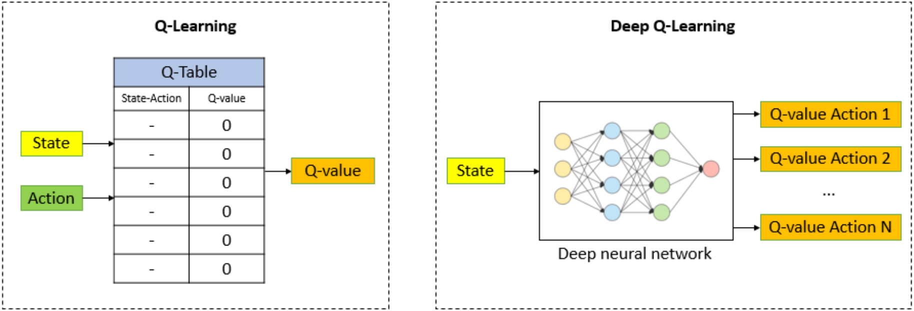


Contents

Deep Q-Learning

Q-Learning but q-function is approximated by a neural network

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



Deep Q-Learning

Loss function

- Bellman equation: $Q(s, a) = (r + \gamma \max_{a'} Q(s', a'))$
- Treating $r + \gamma \max_{a'} Q(s', a')$ as a target value
- Regression problem: $L(\theta) = (r + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a, \theta))^2$

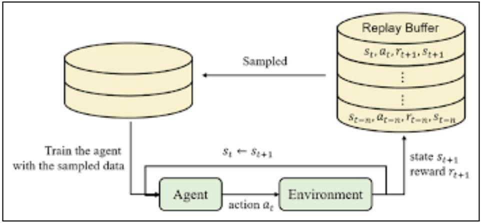
Issues

- **Correlation** ➔ the samples are not independent
- **Non-stationary** ➔ the target value changes over time

Solutions

- **Replay Buffer** ➔ store the transitions (s, a, r, s') and sample them randomly
- **Target Network** ➔ used to compute the target value

Deep Q Learning: Replay Buffer



How

- Store the transitions (s, a, r, s') in $\mathcal{D}$ of prior experience
- During Backpropagation, sample a batch of transitions (s, a, r, s')

Loss computation

- Sample a random batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute the target value $y = r + \gamma \max_{a'} Q(s', a', \theta)$
- Use the target value to compute the loss $L(\theta) = \mathbb{E}[(y - Q(s, a, \theta))^2]$

Deep Q Learning: Fixed Target Network

How

- Use a separate network to compute the target value
- The target network is updated every C steps

Loss computation

- Let θ^- be the parameters of the target network
- Sample a random batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute the target value $y = r + \gamma \max_{a'} Q(s', a', \theta^-)$
- Use the target value to compute the loss $L(\theta) = \mathbb{E}[(y - Q(s, a, \theta))^2]$
- After C steps, update the target network parameters $\theta^- \leftarrow \theta$

Benefits

- **Stable** ➔ the target value is fixed for C steps, avoiding the non-stationary issue (dipendece on target and prediction cause)

Deep Q Learning: Epsilon decay

How

- ϵ is the probability of selecting a random action
- ϵ is decayed over time (or steps or episodes)
- **(!!!)** Off-policy nature of DQL ➔ the agent can learn from random actions

Why

- **Exploration vs Exploitation** ➔ the agent needs to explore the environment to learn the optimal policy
- **Exploitation** ➔ the agent needs to exploit the learned policy to maximize the reward

Deep Q Learning: Algorithm

Algorithm

- Initialize the replay buffer $\mathcal{D}$
- Initialize the target network parameters θ^-
- Initialize the Q-network parameters θ
- **for** episode $= 1$, M **do**
  - Initialize the initial state s_1
  - **for** t $= 1$, T **do**
    - With probability ϵ select a random action a_t
    - otherwise select $a_t = argmax_a Q(s_t, a, \theta)$
    - Execute action a_t in the environment and observe reward r_t and next state s_{t+1}
    - Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    - Sample a random minibatch of transitions (s, a, r, s') from $\mathcal{D}$
    - Set $y_i = r + \gamma \max_{a'} Q(s', a', \theta^-)$
    - Perform a gradient descent step on $(y_i - Q(s, a, \theta))^2$ with respect to the network parameters θ
    - Every C steps reset $\theta^- \leftarrow \theta$

Deep Q Learning: Extensions and Limits

Limits

- Works only for discrete action spaces
- Sample inefficiencient
- Overestimation of the action value due to the max operator

Extensions

- **Double DQN** ➔ use two separate networks to select and evaluate the action
  - Pro: avoid overestimation of the action value
- **Prioritized Experience Replay** ➔ sample the transitions from the replay buffer according to their TD-error
  - Pro: better exploration of the state space
- **Raindow DQN** ➔ combination of the previous extensions

Contents

Deep Reinforcement Learning in Python

Components Required

- **Environment** ➔ the environment in which the agent operates (also called *gym*)
- **Neural Network** ➔ the neural network used to approximate the Q-function
- **Learning Agent** ➔ the agent that learns the optimal policy

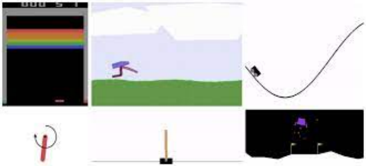
Reference Libraries

- **OpenAI Gym** `https://gymnasium.farama.org/content/basic_usage/` ➔for the environment definition
- **PyTorch** `https://pytorch.org/` ➔for the neural network definition
- **Stable Baselines** `https://stable-baselines.readthedocs.io/en/master/` ➔for the DRL algorithms

Code repository

`https://github.com/cric96/intro-deep-reinforcement-learning-python`

OpenAI Gym



What

- OpenAI Gym is a toolkit for developing and comparing RL algorithms
- It supports teaching agents everything from walking to playing games
- It provides a diverse suite of environments that range from easy to difficult and involve many different kinds of data

Why

- **Standardized** ➔ the environments are standardized ➔ algorithms can be compared
- **Easy** ➔ the environments are easy to use, so that the focus is on the algorithms
- **Flexible** ➔ the environments are flexible, so that new environments can be added

OpenAI Gym I

Main Concepts

- Env is a Python class with the following methods:
  - reset() ➔ reset the environment and return the initial state
  - step(action) ➔ execute the **action** and return the next state, the **reward** and a boolean flag indicating if the episode is terminated
  - render() ➔ render the environment
  - close() ➔ close the environment
  - action_space ➔ the action space (i.e. the set of possible actions)
  - observation_space ➔ the observation space (i.e., the set of possible observations)
- Env is a **black box** ➔ the agent can only interact with it through the methods

Typical usage

```
import gym
env = gym.make('your-env')
env.reset()
for _ in range(episodes):
        env.render()
        env.step(env.action_space.sample())
env.close()
```

OpenAI Gym II

Environment Definition

```
class MyEnv(gym.Env):
        metadata = {'render.modes': ['human']}
        def __init__(self):
                ## Init observation, action spaces
        def step(self, action):
                # Execute one time step within the environment,
                # return observation, reward, done, info
                ...
        def reset(self):
                # Reset the state of the environment to an initial state
                ...
        def render(self, mode='human', close=False):
                # Render the environment to the screen
                ...
```

PyTorch

What

- PyTorch is an open source machine learning framework
- It provides a flexible and efficient library for deep learning
- It provides a seamless path from research prototyping to production deployment

Why

- **Pythonic** ➔ the code is Pythonic ➔ it is easy to use
- **Dynamic** ➔ the code is dynamic ➔ it is easy to debug
- **Fast** ➔ the code is fast ➔ it is easy to scale

PyTorch – Super-fast overview

Tensors

- `torch.Tensor` is the central class of the package
- `torch.Tensor` is a multi-dimensional matrix containing elements of a single data type
- `torch.Tensor` provides a lot of methods for manipulating the data

Autograd

- `torch.autograd` is a package for automatic differentiation
- `torch.autograd` uses a tape-based system for automatic differentiation

Neural Networks

- `torch.nn` is a package for building neural networks
- `torch.nn` provides classes and functions implementing neural networks
- `torch.nn` provides a lot of modules for building neural networks

PyTorch – API Usage

```python
## Main imports
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# Create a tensor
x = torch.tensor(
        [1, 2, 3],
        dtype=torch.float32
)

# Create a neural network
class Net(nn.Module):
        def __init__(self):
                super(Net, self).__init__()
                self.fc1 = nn.Linear(10, 20)
                self.fc2 = nn.Linear(20, 10)
        def forward(self, x):
                x = F.relu(self.fc1(x))
                x = self.fc2(x)
                return x


## Network initialization
net = Net()

# Create an optimizer
optimizer = optim.SGD(
        net.parameters(),
        lr=0.01,
        momentum=0.9
)

# Create a loss function
criterion = nn.MSELoss()

# Training loop
for _ in range(epochs):
        # Perform a forward pass
        output = net(x)
        # Zero the gradients
        optimizer.zero_grad()
        # Compute the loss
        loss = criterion(output, target)
        # Perform a backward pass
        loss.backward()
        # Update the weights
        optimizer.step()
```

Stable Baselines 3

What

- Stable Baselines 3 is a set of reliable implementations of reinforcement learning algorithms
- Stable Baselines 3 is a fork of Stable Baselines 2
- Stable Baselines 3 is based on PyTorch

Why

- State-of-the-art implementations of reinforcement learning algorithms
- Tested and documented codebase
- Easy to use, easy to extend

Main concepts

- **Policy** ➔ the agent's behavior
- **Environment** ➔ the task to be solved
- **Algorithm** ➔ the algorithm to train the agent

Stable Baselines 3 – Usage Overview

Create an environment

- `env = gym.make('CartPole-v1')`

Create a policy

- `policy = MlpPolicy(env.observation_space, env.action_space)`
- `policy = CnnPolicy(env.observation_space, env.action_space)`

Create an algorithm

- `model = PPO(policy, env, verbose=1)`
- `model = A2C("MlpPolicy", env, verbose=1)`
- `model = DQN(policy, env, verbose=1)`

Train the agent

- `model.learn(total_timesteps=10000)`

Contents

Conclusion

What we have learned

- Deep Reinforcement Learning is a powerful tool for solving complex tasks
- Deep Q Learning is a simple yet effective algorithm for solving reinforcement learning tasks

Just a scratch on the surface

- Deep Reinforcement Learning is a very active research field
- There are a lot of algorithms and techniques to learn
- There are a lot of interesting applications to explore

Resources

- *Reinforcement Learning: An Introduction*: intro to Reinforcement Learning (reference book)
- *Deep reinforcement learning in action*: pratice-oriented book on Deep Reinforcement Learning
- *Foundations of deep reinforcement learning*: intro to Deep Reinforcement Learning

Deep Reinforcement Learning

Introduction and Hands-on in Simple Environments

Gianluca Aguzzi gianluca.aguzzi@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna

Talk @ Fondamenti di Intellingenza Artificiale

25/05/2023

References I

[1] Laura Graesser and Wah Loon Keng. *Foundations of deep reinforcement learning*. Addison-Wesley Professional, 2019.

[2] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[3] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[4] David Silver et al. "Reward is enough". In: *Artificial Intelligence* 299 (2021), p. 103535. issn: 0004-3702. doi: https://doi.org/10.1016/j.artint.2021.103535. url: https://www.sciencedirect.com/science/article/pii/S0004370221000862.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. isbn: 0262039249.

[6] Alexander Zai and Brandon Brown. *Deep reinforcement learning in action*. Manning Publications, 2020.