Reinforcement Learning for Autonomous Systems Design

Introduction to Gym

Mirco Musolesi

mircomusolesi@acm.org

Introduction to Gym

- Gym is a tool for developing and comparing reinforcement learning algorithms.

- It is compatible with different numerical computational frameworks, such as TensorFlow, PyTorch or Theano.

- The gym library contains a collection of test problems (called environments) that can be used for working on reinforcement learning problems.

Goal of Gym

- Gym provides benchmarks for comparing different algorithms and helps in comparing algorithms that are proposed by different researchers.

Installation

- In order to install gym in the command line you need to use

```
> pip install gym
```

- If you want to install a particular collection of the environments you need to write

```
> pip install[name_of_the_environment_collection]
```

such as

```
> pip install[atari]
```

- To install all the environments you need to write:

```
> pip install[all]
```

Environments

- You can use simply load an environment as follows:

```
>> import gym

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

Cart Pole Problem

- Classic problem in reinforcement learning.
- A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track.
- The system is controlled by applying a force of +1 or -1 to the cart.
- The pendulum starts upright and the goal is to prevent it from falling over.
- A reward of +1 is provided for every timestep that the pole remains upright.
- The episode ends when the pole is more than 15 degrees from vertical or the cart moves more than 2.4 units from the centre.

Environments

```
import gym

env = gym.make(“CartPole-v1”)

env.reset()

for _ in range(1000)

    env.render()

    env.step(env.action_space.sample()) # random action

env.close()
```

Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems

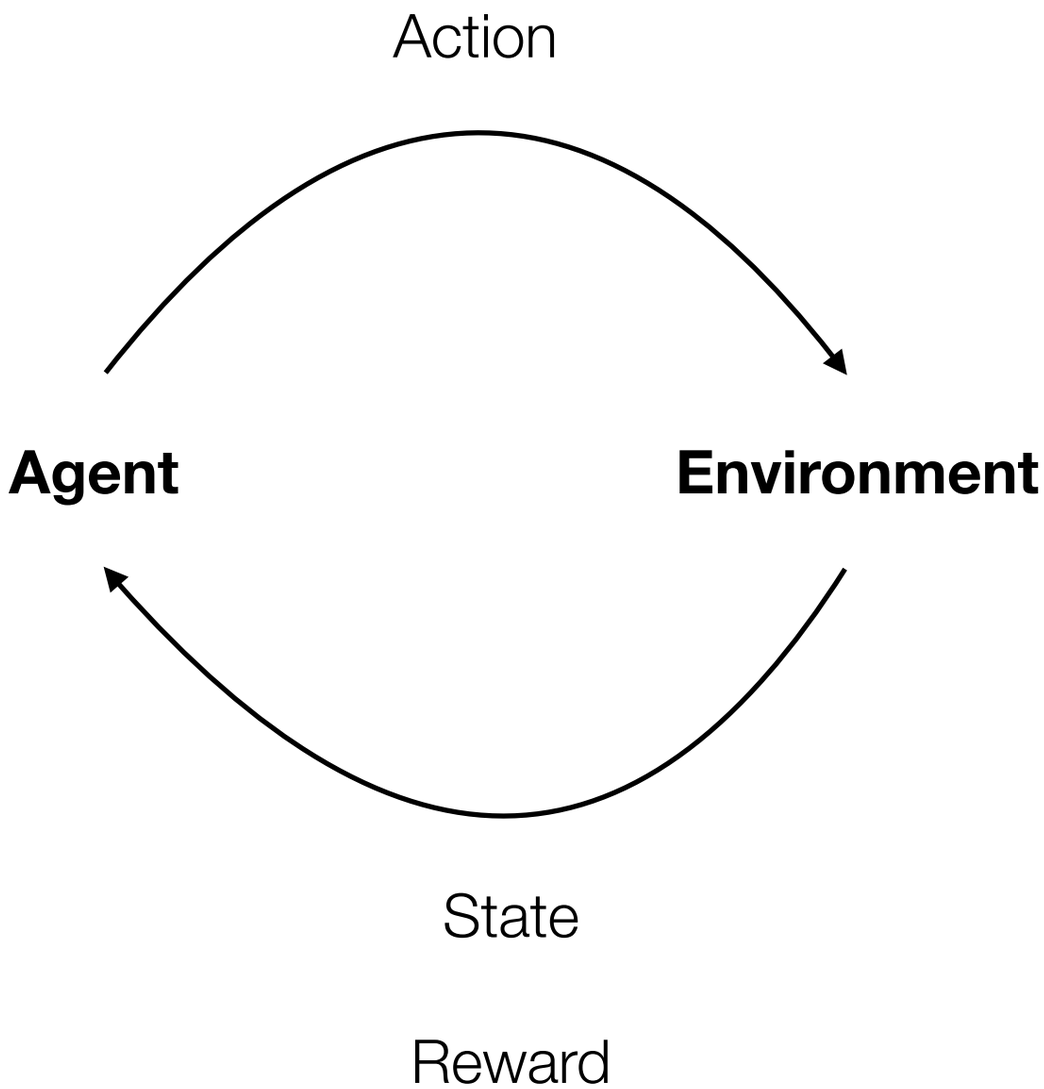
ANDREW G. BARTO, MEMBER, IEEE, RICHARD S. SUTTON, AND CHARLES W. ANDERSON

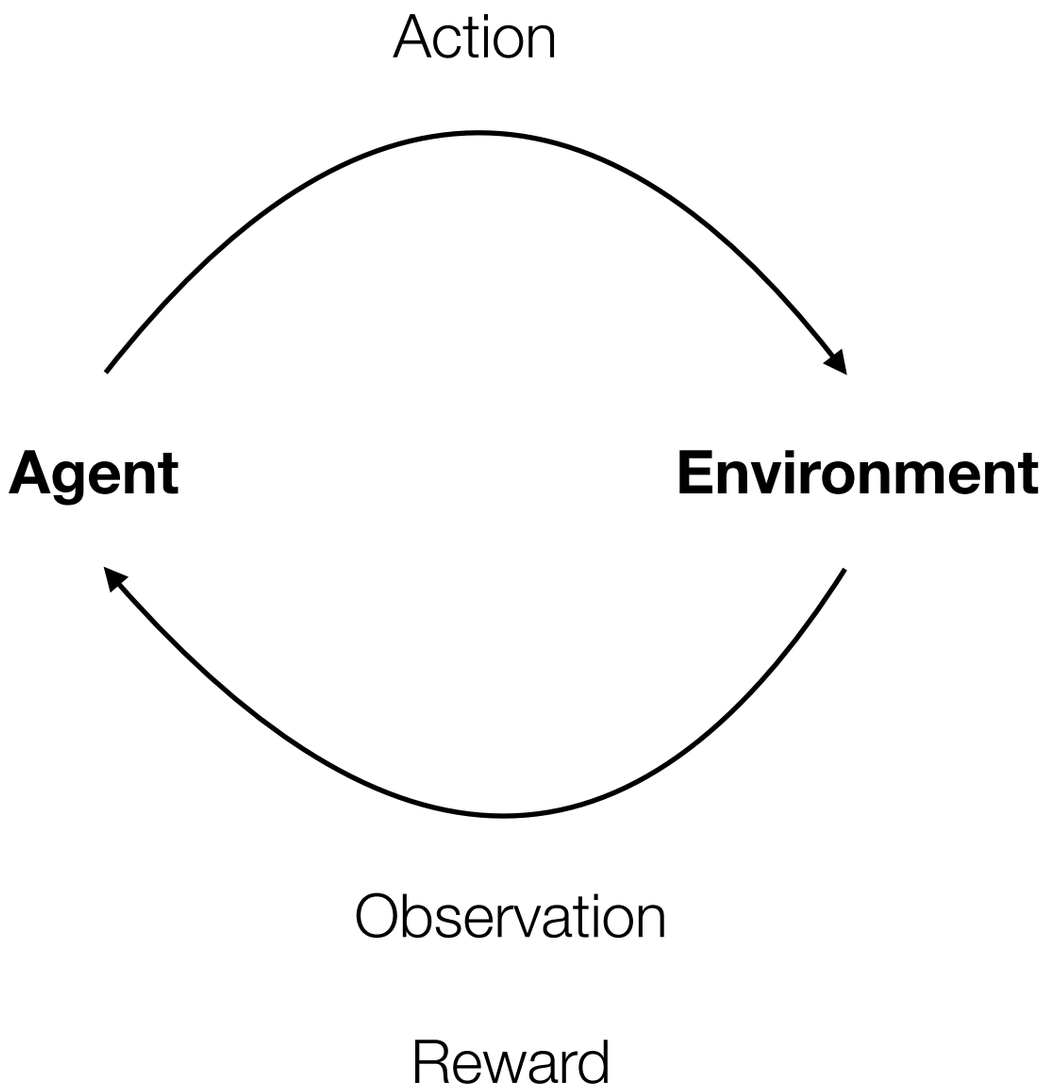
***Abstract*—It is shown how a system consisting of two neuronlike adaptive elements can solve a difficult leaning control problem. The task is to balance a pole that is hinged to a movable cart by applying forces to the cart's base. It is assumed that the equations of motion of the cart–pole system are not known and that the only feedback evaluating performance is a failure signal that occurs when the pole falls past a certain angle from the vertical, or the cart reaches an end of a track. This evaluative feedback is of much lower quality than is required by standard adaptive control techniques. It is argued that the learning problems faced by adaptive elements that are components of adaptive networks are at least as difficult as this version of the pole-balancing problem. The learning system consists of a single *associative search element* (ASE) and a single *adaptive critic element* (ACE). In the course of learning to balance the pole, the ASE constructs associations between input and output by searching under the influence of reinforcement feedback, and the ACE constructs a more informative evaluation function than reinforcement feedback alone can provide. The differences between this approach and other attempts to solve problems using neuronlike elements are discussed, as is the relation of this work to classical and instrumental conditioning in animal learning studies and its possible implications for research in the neurosciences.**

Manuscript received August 1, 1982; revised April 20, 1983. This work was supported by AFOSR and the Air Force Wright Aeronautical Laboratory under Contract F33615-80-C-1088.

The authors are with the Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003.

0018-9472/83/0900-0834$01.00


Action
Agent
Environment
State
Reward


Action
Agent
Environment
Observation
Reward

The `step()` Function

- The most important function in Gym is `step()`.

- `step()` receives in input the action to be taken and return a quadruple (`observation`, `reward`, `done`, `info`):

  - o`bservation` is an object representing the representation of the state of the environments. Examples are pixels from a camera, joint angles and joint velocities of a robot, current state of a board of a game, etc.

  - `reward` (which is implemented using a `float`) is the amount of reward achieved by the previous action. Remember that the goal of reinforcement learning is to maximise the total reward, i.e., this value.

The `step()` Function

- `done` is a `boolean` that is used to understand when it's time to reset the environment. The majority of tasks are divided into episodes. done equals to true indicates that the episode has terminated.
    - `info` is a dictionary (`dict`) and contains diagnostic information.
- The process get started by calling `reset()`.

Typical Structure of an Gym program

```
import gym

env = gym.make(“CartPole-v1”)

for i_episode in range(20):

   observation = env.reset()

   for t in range(100):

       env.render()

       print(observation)

       action = env.action_space.sample()

       observation, reward, done, info = env.step(action)

       if (done):

           print(“Episode finished after {} time steps”.format(t+1))

            break

env.close()
```

Spaces

- Every environment comes with an `action_space` and an `observation_space`.

```
>>> import gym

>>> env = gym.make(“CartPole-v1”)

>>> print(env.action_space)

Discrete(2)

>>> print(env.observation_space)

Box(4,)
```

Spaces

- The `Discrete` space allows a fixed range of non-negative numbers, so in this case valid actions are either 0 or 1.

- The `Box` space represents an n-dimensional box, so valid observations will be an array of 4 numbers.

- We can also check the value of the bounds using the techniques described in the following slide.

- You can find more information about other action spaces in the documentation.

Spaces

```
>>> print(env.observation_space.high)

[4.8000002e+00 3.4028235e+38 4.1887903e-01
3.4028235e+38]

>>> print(env.observation_space.low)

[-4.8000002e+00 -3.4028235e+38 -4.1887903e-01
-3.4028235e+38]
```

Environments

- Open Gym provides a variety different algorithms:
- Basic algorithms
- 2D environments (e.g., Lunar Lander).
- Atari games
- Control systems (also based on the Mujoco physics simulator)
- Text

References

- OpenAI Gym website http://www.gym.openai.com

- Chapter 18 of Aurelien Geron. Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow. Second Edition.2019. O'Reilly.