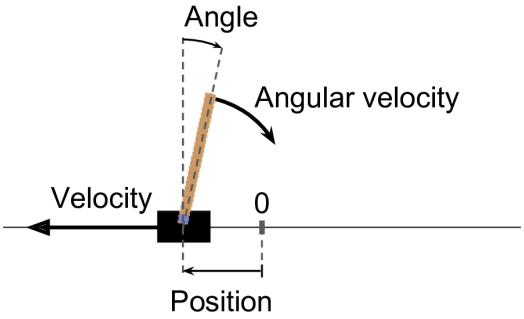
Reinforcement Learning, Policy Gradient

Last modified on Saturday, 21 May 2022 by f.maire@qut.edu.au

This week we explore RL algorithms based on the direct estimation of the gradient of the return with respect to the policy parameters.

We illustrate policy gradients methods with the application of the REINFORCE algorithm on a pole balancing task.



This prac requires the installation of *OpenAI gym*.
Download the file *pg_prac.py* from Blackboard.
Complete the lines tagged with "INSERT YOUR CODE HERE".

Exercise 1

Install *OpenAI gym*: $ python3 -m pip install -U gym

Then, check that the installation was successful by running the script *cart_pole_demo.py* (download this file from Blackboard).

Look up the attributes of the object *env* returned by the call *env = gym.make('CartPole-v0')*

```
>>> import gym
>>> env = gym.make("CartPole-v1")
>>> obs = env.reset()
>>> obsarray([-0.01258566, -0.00156614,  0.04207708, -0.00180545])
```

Here, we've created a CartPole environment. This is a 2D simulation in which a cart can be accelerated left or right in order to balance a pole placed on top of it (see Figure above).

Let's ask the environment what actions are possible:

```
>>> env.action_space
Discrete(2)
```

Discrete(2) means that the possible actions are integers 0 and 1, which represent accelerating left (0) or right (1).

After the environment is created, you must initialize it using the reset() method. This returns the first observation. Observations depend on the type of environment. For the CartPole environment, each

observation is a 1D NumPy array containing four floats: these floats represent the cart's horizontal *position* (0.0 = center), its *velocity* (positive means right), the *angle of the pole* (0.0 = vertical), and its *angular velocity* (positive means clockwise).

The step() method executes the given action and returns four values:

- **obs** This is the new observation.
  For example, assume obs is array([-0.01261699, 0.19292789, 0.04204097, -0.28092127]). The cart is now moving toward the right (obs[1] > 0). The pole is still tilted toward the right (obs[2] > 0), but its angular velocity is now negative (obs[3] < 0), so it will likely be tilted toward the left after the next step.
- **reward** In this environment, you get a reward of 1.0 at every step, no matter what you do, so the goal is to keep the episode running as long as possible.
- **done** This value will be True when the episode is over. This will happen when the pole tilts too much, or goes off the screen, or after 200 steps (in this last case, you have won). After that, the environment must be reset before it can be used again.
- **info** This environment-specific dictionary can provide some extra information that you may find useful for debugging or for training. For example, in some games it may indicate how many lives the agent has.

Once you have finished using an environment, you should call its close() method to free resources.

Let's hardcode a simple policy that accelerates left when the pole is leaning toward the left and accelerates right when the pole is leaning toward the right.
Complete the function def basic_policy(obs)

Run this policy to see the average rewards it gets over 500 episodes by calling
collect_stats_simple_strategy()

Exercise 2

- Create a sequential neural network model model = keras.models.Sequential(...)
- with a hidden layer with 5 neurons and the "elu" activation function and an output layer with a single neuron that should compute the probability of going right.
- See how the untrained network performs by calling view_one_episode_initial_policy_net()

Exercise 3

- Train the neural network by calling the do_training() function
- What is the performance of the trained network?