RL-glue and you

Why RL-glue?

- In RL we have agent's interacting with an environment
    - e.g., a bandit algorithm interacting with a 10-armed bandit problem
    - e.g., a Sarsa agent interacting with a grid world
- The agent and environment are both **programs** that must *exchange* data at each time step
- RL-glue manages this *interaction*, deciding many things: like who goes first? what information is stored?, etc

What would you do?

- Given the task of running an experiment with an agent and environment?
- You could do the *single-file approach*: write one big program that contains the agent code, the environment code, and some function to call each of them in turn
    - What are the limitations of this approach?
- You could make your own agent, environment software, architecture.

What is RL-Glue

- RL-glue is a *communication protocol*
  - *a system of rules that allow two or more entities to transmit information*
  - *…defining the rules syntax, semantics and synchronization of communication*

A standard interface

- The implementation of the RL-glue protocol is realized as a standard software **interface** that the agent and environment **must implement**

- An agent that is compatible with RL-glue is required to implement a set of standard functions:

  - *agent_init, agent_start, agent_step, agent_cleanup, agent_message*

  - a python program implements these functions is compatible with RL-glue. We call it an ***agent program***

Standard interface…

- An environment that is compatible with RL-glue is required to implement a set of standard functions:
  - *env_init, env_start, env_step, env_cleanup, env_message*

- a python program implements these functions is compatible with RL-glue. We call it an **environment program**

simple_agent.py

- We released some **example** code to show you how to write agents, and environments compatible with RL-glue

- simple_agent.py is an example of one such agent. It's a simple agent that does not learn, but instead takes random actions

  - but it does provide an **example** of an agent that is compatible with RL-glue: it implements the *agent interface* (a**ll the required agent functions!**)

simple_env.py

- simple_env.py is an **example** of an environment. It ignores the agent's action and returns a random reward on each time step

- But it does provide an **example** of an environment that is compatible with RL-glue: it implements the *environment interface* (**all the required functions!**)

Your task

- Make your own agent and environment programs:
    - Maybe you might call them: bandit_agent.py, and bandit_env.py
- Start by copying over the code in simple_agent.py and simple_env.py
    - that way you know you have implemented the required functions
- Modify the functions inside bandit_agent.py so that it implements the bandit algorithm described in the assignment
    - similarly for the bandit_env.py

Let's look at simple agent

```
#!/usr/bin/env python
Comments
imports …
globals …

def agent_init():
    # this is where you initialize agent data structures

def agent_start(this_observation):
    # first time step
    # this is where your agent selects an action and returns it. No learning happens here!
    return action

def agent_step(reward, this_observation):
     # the agent selects a new action
     # and does a learning update with the Q-values, based on the reward and action from previous time step
     return action

def agent_end(reward):
    # final learning update at end of episode
    # never gets called in a bandit problem
    return

def agent_cleanup():
    # clean up
    return

def agent_message(inMessage):
     # might be useful to get information from the agent
     return “some string"
```

Study simple_env.py

- It tells you what functions your bandit_env.py program should implement

- Can you figure out from simple_env.py and the REAME file what each function does?

Experiment programs

- Now we have an agent and environment program that are compatible with RL-glue!
    - How do we run an experiment?
- We must write an experiment program!
- The experiment program **calls functions** inside rl_glue.py
- These rl_glue functions provide different ways to run an experiment

Let's look at simple experiment (an **Example** experiment program)

```
#!/usr/bin/env python
Comments
imports …

from rl_glue import *  # Required for RL-Glue
RLGlue("simple_env", "simple_agent")

if __name__ == "__main__":
   maxSteps = 1000
    numRuns = 30
    # create some data structures to store performance

    for k in range(numRuns):
        RL_init()
        RL_start()
        for i in range(maxSteps):
            (reward, obs, action, terminal) = RL_step()
            # store some of this information
        RL_cleanup()
       # update statistics about agent performance, so we can average over runs

 # Save data to a file
```

What can you modify?

- Anything inside your agent, environment, and experiment programs:
    - bandit_agent.py, bandit_env.py, bandit_exp.py
- Your agent **must** implement all of the required agent functions
- And your environment **must** implement all the required environment functions
- And your experiment program **should** call functions in rl_glue.py

What can you *not* modify?

- Don’t touch rl_glue.py
- There will be a few lines of code that are special
  - for example, the experiment program must contain the following lines of code:
    - **from** rl_glue **import** * # Required for RL-Glue RLGlue("simple_env", “simple_agent”)
    - These cases are well documented in the example code!

Exercise: find the glue!!

A simple bandit algorithm

Initialize, for $a = 1$ to k:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$

Repeat forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

Agent program

Environment program

Experiment program