Preferred Networks Intern Screening 2017 Coding Task for Machine Learning / Mathematics Fields

Changelog

- 2017-05-12 : Initial version
- 2017-05-24 : Correct the sentence in problem 4.

Notice

- In problems 1-5, use only standard libraries. Usage of multi-dimensional arrays from NumPy, Eigen etc. is not allowed.
- In problem 6, feel free to use any library that you want.
- Please use either of the following programming languages:
- C, C++, Python, Ruby, Go, Java
- In the problems, there is a need to use pipe/fork. This should not be an issue on MacOS/Linux. On Windows, we have examined the code using cygwin or msys2.

Things to submit

- Please submit the code for problems 1-6, a dump of the parameters from problem 5, and the report for problem 6. The code can either be divided into separate files for each problem, or you can put the code for all problems in one single file. Either is fine.
- Please write your code so that it will be easy to read for other people. Also, please make sure that it is easy to re-run your program. As for the report, please make sure that it is concise and easy to understand.
- Please include instructions on how to compile/run your code (such as execution environment, compiler, and other necessary steps).
- It is appreciated if you also include an explanation of your program (at most 1 A4 page), if possible.

How to submit

Submit files should be uploaded to Google drive. After uploading them, please fill in the share URL on the submission form. See the following URL for how to upload files to Google drive:

- Submission form: https://docs.google.com/a/preferred.jp/forms/d/e/1FAIpQLSd_zC_XT2dHM-yRO9WQ-YuRU0sx2HeQIep-NBoqMWpN_j8KNw/viewform

- How to upload files : https://www.preferred-networks.jp/wp-content/uploads/2017/04/intern2017_GoogleUpload_3.pdf

Inquiry

If you have any questions about the problems, please also send these to intern2017@preferred.jp (the same address as used for applying to the internship)

Task description

In this task, you will solve a reinforcement learning problem by applying an optimization algorithm called the Cross Entropy Method (CEM) in order to train an *agent* for decision making. Reinforcement learning is a way of learning a good policy for the agent to make decisions when interacting with the environment. The agent acts upon the observations it received from the environment. As a result of the agent's action, the environment updates its internal state, and sends back a new observation and a *reward* to the agent. The reward indicates how good was the action that the agent took. Taking an action is also called taking a *step*. At one point in time, the environment will send a stop signal to indicate the end of the interaction. The interval between the start of the agent's interaction and the stop signal is called an *episode*. The sum of the rewards in one episode is called the *return*. The goal of a reinforcement learning algorithm is to improve the return of the agent.

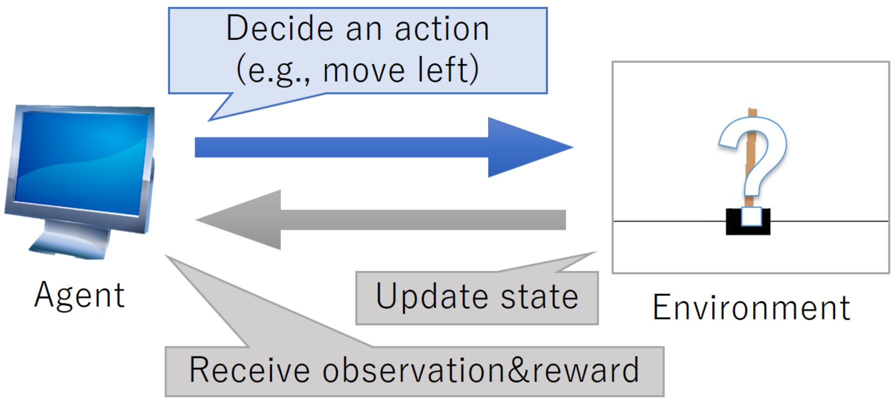

Figure 1:

In this task, you will solve a classic control system task called *cartpole*. In all the problems, please express the environment using a single class (if you use a programming language that has no classes, please use suitable functions instead).

The environment is an interface that has the following methods. As a side note, this interface is an imitation of the one used in OpenAI Gym.

- `reset()`: This method will reset the environment. The method takes no parameters. The return value is a vector that indicates the observation reward of the initial state of the environment.
- `obs_dim()`: This method returns the number of dimensions of the observation vector.
- `step(action)`: Applies an action to the environment. The method takes a single parameter `action` that indicates the action to apply to the environment. In this task, for the sake of simplicity, `action` is always of value either 1 or -1. The environment will update its internal environment according to the action, and finally return three values: a vector indicating the new observation, a boolean indicating the stop signal, and the reward.

Problem 1

First, for the sake of verification, we will create an environment where training the agent is very easy. Please create a class that implements an interface according to the following specification. Further, please write some simple code (a unit test) that verifies that the class you created is working correctly.

Class name: `EasyEnv` Methods:

- `reset()`: Returns a 1-dimensional vector sampled uniformly from the interval $[-1, 1]$.
- `obs_dim()`: Returns 1.
- `step(action)`: Call the previous step's observation (the component from the 1-dimensional vector) `prev_obs`. If `step` is called immediately after `reset`, `prev_obs` will take the value equal to the return value of `reset`. The observation for this step will be sampled uniformly from $[-1, 1]$. The reward is `action * prev_obs`. The environment will end after taking 10 steps.

Problem 2

There is a file called `cartpole.cc` in the problem directory. This source code describes the cartpole problem's environment. You can compile it using e.g. the following command.

```
g++ -std=c++11 cartpole.cc -o cartpole.out
```

We will call the output of this command, `cartpole.out`, as the *host program* for convenience. You can specify the program that corresponds to the agent as a command line parameter to the host program. Internally, the host program will run the agent and pipe its input and output so that it can interact with the host program. The repository includes two sample programs. One written in C++

called `random_action.cc`, and one written in python3 called `random_action.py`. You can run the sample agents by the following commands.

```
# python3
./cartpole.out "python3 random_action.py"
# C++
g++ -std=c++11 random_action.cc -o random_action.out
./cartpole.out ./random_action.out
```

In order to make the agent interact with the host program, it is necessary to write commands like the below to the standard output.

- Flush `r` to the standard output: Resets the environment and sends the initial observation to the standard input. The format is like `obs <x1> <x2> <x3> <x4>`. Each `xi` represents an actual real value.
- Flush `s <action>` to the standard output (here, `<action>` is either 1 or -1): Applies the action to the environment. The internal state of the environment will be updated, and the observation will be sent to the standard input in the same format as above. If the angle of the cartpole has reached a predetermined value, the environment will end, and instead of the observation, `done` will be sent to standard input.
- Flush `q` to the standard output: Stops the host program. If the agent quits without having sent this command to the host program, there is a risk that the host program will display an error, so please be careful.

Please write a class called `CartPoleEnv` that will handle the reading and writing of standard input and output. `CartPoleEnv` has the same interface as in Problem 1 (`reset`, `obs_dim`, `step`), and also has the following specifications:

- The reward is always 1.
- The environment will end after 500 steps.

If you want to print debug messages to the screen, you are recommended to write to `stderr`. Further, please write some simple code (a unit test) for checking that your class is working correctly.

Problem 3

The policy of the agent is represented using a model that has some parameters. In this problem, the model is a linear function. Please implement a class as described below. Further, please write some simple code (a unit test) to make sure that your code is working correctly.

Class name: `LinearModel` Methods:

- `<constructor>(initial-param)`: Initializes the model's parameters with `initial-param`, which is a vector. The dimension of `initial-param` is the same as returned by the environment's `obs_dim` method.
- `action(obs)`: Takes the inner product of the observation and the model parameters. Returns 1 if the inner product is positive, and -1 otherwise.

Problem 4

In reinforcement learning, the Cross Entropy Method (CEM) is an optimization algorithm that updates the policy in the following way.

- The initial policy parameters are arbitrary.
- Repeat the following until the return converges:
- Call the current policy parameters θ.
- For $i = 1, ...N$, sample a vector ε_i representing the noise (with the same number of dimensions as θ) and set $\theta_i = \theta + \varepsilon_i$.
- For $i = 1, ...N$, reset the environment, run the agent for one episode using θ_i as parameters, and calculate the return.
- Calculate the mean of the θ_i vectors whose return was among the top $100\rho\%$, and update θ to that value.

Here, N and ρ are hyperparameters. In problems 4 and 5, please use $N = 100$, $\rho = 0.1$. As for the noise vector ε_i, please sample each component from a normal distribution with mean 0 and variance 1. If the programming language you use does not include an implementation of normal distributions as a library, please sample uniformly and independently from $[-1, 1]$ three times, and then use the sum of them as the approximation of a normal distribution sample. Please implement the CEM, and confirm that it runs correctly in `EasyEnv`.

Problem 5

Please train the agent in `CartPoleEnv`. Write a function to evaluate the model. Run the agent continuously for 100 episodes, and if at least 95 episodes have a return that is at least 500, the training is considered successful. When the training succeeds, please write the parameters of `LinearModel` to a file in the following format:

```
w1
w2
w3
w4
```

Please submit these parameters when handing in your solution. Here, `wi` represents the actual value of the `i`-th parameter.

Problem 6

Please write a report on how the experimental results change when you adjust the parameters and implementation of Problem 5. The report should be 1-2 A4 pages long (including tables and figures).

Things to try can be for example the following (but feel free to add more):

- In reinforcement learning, the *sampling efficiency* (the number of episodes and steps required for the policy to converge) is an important indicator of the algorithm's quality. Please try and see how the sampling efficiency changes when you change N, ρ and the noise generation.
- Having only access to partial observations (that is, not being able to get all the observations from the environment) is common when running the algorithm in the real world, as opposed to on synthetic data. Please try to remove parts of the observations received from `cartpole.cc` and see if it is still possible to train the agent. If it doesn't go well, try to add more features (something you think might yield useful information) to the observation returned from the environment and see if training succeeds.
- In the real world, it is often the case that the observations contain noise. Also, the observations might arrive to the agent with some delay. Please investigate how these issues affects the training of the agent.
- Instead of using a simple linear model with few parameters for modelling the policy, it is possible to use a neural network with many more parameters. Please investigate how the sampling efficiency changes when using a neural network.