CS6700 : Reinforcement Learning
Programming Assignment #3

HRL and DQN — Deadline: 5^{th} May 2019, 11:55 pm

- This is an individual assignment. Collaborations and discussions are strictly prohibited. *Do **not** put up the code for any part of the assignment in public repositories online.*
- You have to turn in the well-documented code along with a detailed report of the results of the experiments. Typeset your report in LaTeX.
- Be precise with your explanations. Unnecessary verbosity will be penalized.
- Check the Moodle discussion forums regularly for updates regarding the assignment.
- *Any kind* of plagiarism will be dealt with extremely seriously. Acknowledge any and every resource used.
- **Please start early.**

1 Hierarchical Reinforcement Learning

SMDP Q-learning

For this problem, we would be referring to Sutton, Precup and Singh's 1999 paper on 'Between MDPs and semi-MDPs : A Framework for Temporal Abstraction in Reinforcement Learning'. Please do read the paper upto and including Section 3, it is self explanatory and a good reference leading up to the understanding and implementation of SMDP Q-learning. Section 3 of the paper first talks about SMDP planning and is a good reading exercise to build the intuition.

We would be working with a simple 4 room grid world environment (explained in the next section). Your task is to implement 1-step SMDP Q learning and intra-option Q learning on this environment.

The Environment

The environment for this task is the following grid world. The cells of the grid correspond to the states of the environment. There are four rooms numbered: 1, 2, 3 and 4. The room number 1 is the one with 'HALLWAYS' written in it. The second room is on the right to the first one, and so on. G_1 and G_2 are the goal states. States G_1 and G_2 give a reward of $+1$, transitions to all the other states give a reward of 0. The discount factor is taken to be $\gamma = 0.9$.

The actions and the options

There are four primitive actions: up, down, left and right. With probability 2/3, the actions cause the agent to move one cell in the corresponding direction. With probability 1/3, the agent instead moves in the other directions, each one with a probability 1/9. If the agent hits the wall, it remains in the same state.

There are 8 multi-step options : each one leading to a particular room's hallway. A hallway option's policy finds the shortest path within the room to its target hallway. The termination condition $\beta(s)$ for each hallway option is zero for states s within the room and 1 for the states outside the room, including the hallway states. The initiation step $\mathcal{I}$ comprises the states within the room plus the non-target hallway state leading to the room.

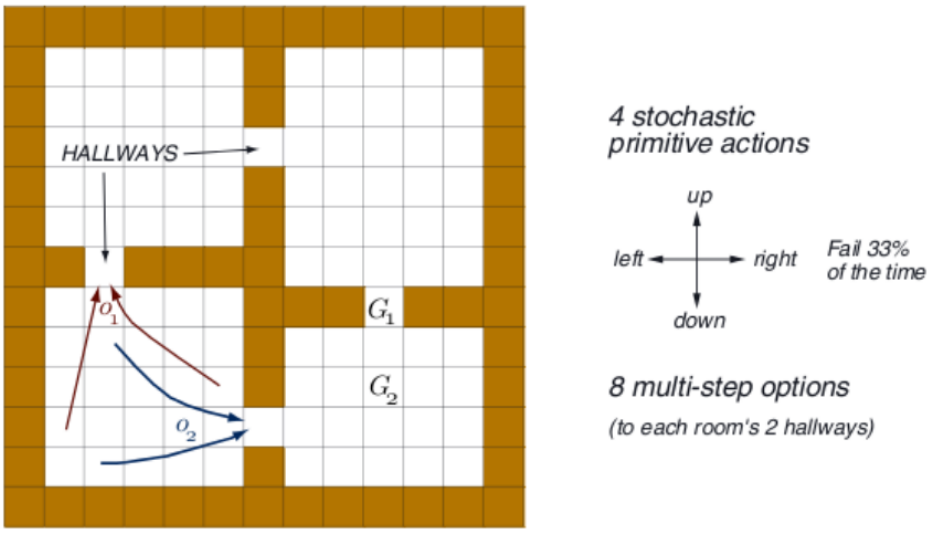


Figure 1: 4 Room Grid World

The code for the environment is provided in this link.

SMDP Q-learning implementation

Here we implement the single step SMDP Q-learning for each of the goals, as described in section 3.2 of the paper.

A rough sketch of the algorithm is as follows: Given the set of options,

1. Execute the current selected option (*e.g.* use epsilon greedy $Q(s, o)$) to termination.
2. Compute $r(s, o)$
3. Update $Q(s_t, o)$ using Q-learning update

Questions and deliverables

1. Visualize the learned Q values.
2. Change initial state to the centre of room 4. What do you observe?
3 [**Bonus**] Implement intra-option Q learning. Do you observe any improvement? Why is that so?

Note: The step function in the environment code provided is not defined for actions in pre-hallway states that do not lead the agent to a hallway state. Please fix this before proceeding.

2 (Not so) Deep RL

Overview

- This assignment requires you to implement and evaluate Q-Learning on a simple control task - the 'CartPole' environment.[1]
- The Q-learning algorithm and the nuts and bolts of a Deep Q-Network have been covered in the class, and you have been provided with starter code for this task.
- **You do not require a GPU (and hence, AWS) for this task**; the *D*QN can be trained easily within a few minutes on a decent CPU.
- While each of the runs won't take much time, depending on your implementation, you may have to play around with the parameters like learning rate, epsilon-annealing, batch size, hidden layer sizes, etc. which could take up the bulk of your time.
- The actual coding for this assignment might not even involve 100 lines of code, but the evaluation may take a very long time. So start early!

Installation

- [Important] We expect you to use **only** TensorFlow since it is the most widely used deep learning framework today.[2] You are **not allowed** to use any higher-level APIs or packages like Keras for this assignment.
- Install TensorFlow for your machine following the documentation. We recommend using the Virtualenv installation since it provides a virtual and isolated Python environment, incapable of interfering with or being affected by other Python programs, versions, and libraries being used by your other projects on your machine.

[1] Check out the details here : github.com/openai/gym/wiki/CartPole-v0

[2] Atleast in the next few years, TensorFlow and PyTorch will be the frameworks used for most, if not all, deep learning projects, so it is best you get familiar with them. We shall stick to TensorFlow for now since it has better documentation and support than PyTorch as of today. And once you learn TensorFlow, it is very easy to pick up PyTorch or Keras.

Implementation

- Please download the attached starter code. It is written in TensorFlow and gives a brief overview of each step and the flow of the code. While we encourage you to use the given starter template, *you do not necessarily have to stick to it* - use it as a reference. Feel free to make any design/flow choices that you feel is more appropriate to your methodology.
- An important component of any experiment, especially in Deep Learning, is visualization of some training signals, which serve as an indicator of the health of the experiment, and make it way easier to understand, debug, and optimize TensorFlow codes. TensorBoard, by the creators of TensorFlow, is one such suite of visualization tools. Check out the documentation of the features it offers. The starter code also uses TensorBoard. After setting up TensorBoard on your machine following the aforementioned link, you can point your favourite browser to a local link of the form `http://0.0.0.0:6006` to visualize the reward per episode or episode-length. You are also free to use alternative (real-time) plotting methods of your choice.

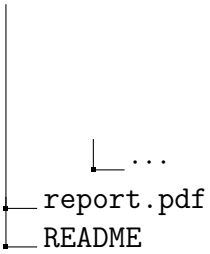
Expectations

1. Code up the DQN with the experience replay and target network, and turn in the learning curve of episodes on the x-axis versus the average total reward of a 100-episode window on the y-axis. The CartPole task is considered 'solved' if you get a reward of over 195 for 100 consecutive episodes.
2. Report the set of hyperparameters which work the best for you (even if you could not solve the task). *Beware - the search for the right hyperparameters could take longer than the coding up itself.* Note that the starter code is initialized with a decent set of hyperparameters.
3. Report any observations or inferences you make regarding the variation of certain hyperparameters like hidden layer size(s), epsilon, minibatch size.
4. [**Bonus**] Once you find the best set of hyperparameters, try removing the experience replay and/or the target network to see how the learning is affected. Report your observations and inferences.

Submission Guidelines

Submit a single tar/zip file containing the following files in the specified directory structure. Use the following naming convention: `rollno_PA3.tar.gz` or `rollno_PA3.zip`
A sample submission would look like this:

```
rollno_PA3
└── Code
    ├── Q1
    │   └── ...
    └── Q2
```


...
report.pdf
README