ECE448/CS440 Spring 2022

Assignment 6: Reinforcement Learning

Due date: Monday May 2nd, 11:59pm (We will accept late submissions up till 16th May, 11:59pm)

Updated 2022 By: Guannan Guo and Mohit Goyal

Starter code

You can download the starter code here: zip, tar

Run the main functions of "tabluar.py" and "dqn.py" to train your RL model. Then run "mp6.py" to run/test the resulting model. Note that "mp6.py" is not that important. It's only there to help you see how your model performs

General guidelines

Basic instructions are similar to previous MPs: For general instructions, see the main MP page and the course policies.

Code will be submitted on gradescope.

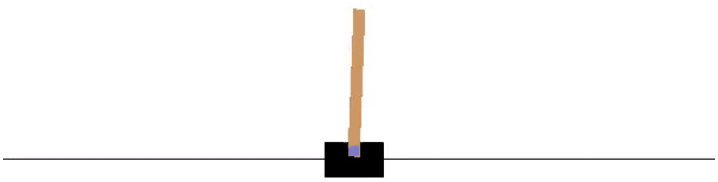
The extra credit portion will be submitted separately on gradescope. In your MP average, the extra credit is worth 10% of the value of the regular part of the assignment.

Objective

Create an RL agent to do cool things, like balancing a pole or swinging a multi-pendulum as high as possible.

Our RL agent will be implemented via Q-Learning in the tabular setting and function approximation setting

Environment - CartPole V1



In the CartPole environment, there is a pole standing up on top of a cart. The goal is to balance this pole by moving the cart from side to side to keep the pole balanced upright. A reward of +1 is provided for every timestep that the pole remains upright, and the episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

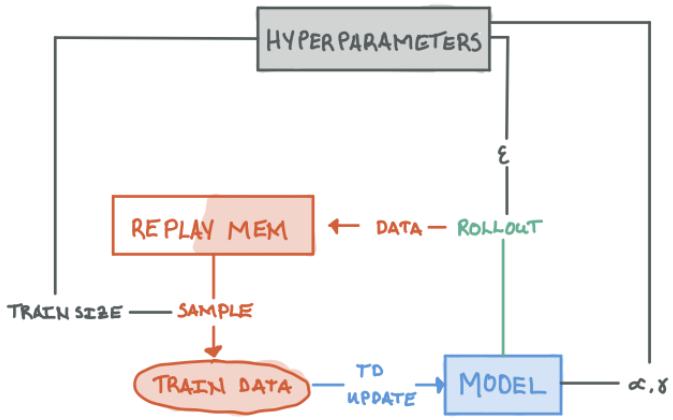
Observation space

Index	Meaning	Min	Max
0	Cart Position	`-4.8`	`4.8`
1	Cart Velocity	`-Inf`	`Inf`
2	Pole Angle	`-24 deg`	`24 deg`
3	Pole Tip Velocity	`-Inf`	`Inf`

Action Space

- 0 - Left
- 1 - Right

High Level Algorithm



1. Initialize Model
2. Rollout trajectories using current model and store it into the replay memory
3. Sample data from the replay memory to update the model's Q-value estimates
4. Repeat steps 2-4 until convergence

Your Job

Implement the initialization and training code of Q-learning for the tabular setting.

Most of the code is written for you, you just have to fill out some functions

Rollout

Implement the `rollout` function in `utils.py`. This function plays a given number of episodes and returns the replay memory and score. The replay memory is a list of `(state, action, reward, next_state, done)` at each timestep for all episodes, and score is a list of total rewards for each episodes.

We use the replay memory in our `train` function in `utils.py`. During training, the agent samples data randomly from the replay memory to reduce the strong temporal relationship between consecutive pair of `(state, action, reward, next_state, done)`.

Tabular Setting

- Implement the following functions of `TabQPolicy` in `tabular.py`
  - Initialization `__init__`: create the table to hold the Q-vals.
  - `qvals`: given the current state, return the value for each action.
  - `td_step`: look below for the description of temporal difference learning.
- Tune the discretization in main call of `tabular.py`
  - The state space for this environment is continuous. We have provided a function for discretizing the state space into bins for you. You just have to choose the granularity of division by tuning the `buckets` argument. The model takes as input - bins, which denotes the number of distinct bins for each dimension of the state space.

Training: `Temporal Difference`

You have to implement the `td_step` function in `tabular.py`.

The function takes in training data of the form `state`, `action`, `reward`, `next_state`, `terminal (done)`.

Recall the TD update for Q-vals is

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot (target - Q(s,a))$$

where $target = r + \gamma \cdot \max_{a'} Q(s', a')$ if the state is not terminal and $target = r$ otherwise.

Return the square error of the original q-value estimate, i.e. the square of the difference between that and `target`.

Hyperparameter Tuning

A large part of this assignment is tuning the hyperparameters to train a good model. See the `hyperparameters` function in the `utils.py` script to see what parameters you can control.

Extra Credit: Function Approximation Setting

In this extra credit part, you need to run DQN policy on CartPole-v1. You have to create a neural network that takes the state as input, and output the score for each action. The state is no longer discrete as in the tabular setting.

- Implement the following functions of `DQNPolicy` in `dqn.py`
  - Initialization `__init__`: create optimizer and loss
  - `td_step`: temporal difference learing update
- Implement `make_dqn`
  - This defines your neural network. You have the creative freedom to make it however you want as long as the returned object is a PyTorch `nn.Module`. Please make sure that your model can be trained and evaluated on a CPU.

Submission

You need to train and submit your models on gradescope. First, adjust/tune your parameters as needed in `utils.py` and the granularity of your Q-table in the bottom of `tabular.py`. The same set of hyperparameters does not necessarily work well for both Q learning and DQN simultaneously. Then, train your Q learning agent by running `python tabular.py` and your DQN agent with `python dqn.py`. At completion, the model will be saved to `tabular.npy` and `dqn.model` for Q learning and DQN respectively.

You can view the performance of your agents with `python mp6.py --model <saved_model.file> --episodes 8 --epsilon 0`

- For tabular Q learning, submit `tabular.py` and your trained model `tabular.npy` on gradescope.
- If you do the extra credit, submit your `dqn.py` and the trained model `dqn.model` to the separate extra credit assignment on gradescope.

Expected Scores

- For tabular Q learning, `python mp6.py --model tabular.npy --episodes 8 --epsilon 0` should yield 200+ score.
- If you do the extra credit, `python mp6.py --model dqn.model --episodes 8 --epsilon 0` should yield 300+ score.

You don't need to submit `utils.py` to the gradescope.

Package Requirements

This assignment requires some Python packages

- `gym` - RL environment
- `numpy` - numerical analysis package
- `tqdm` - progress bars

References

- Gym - A reinforcement library
- PyTorch - A deep learning library