Introduction to Deep Learning due on Jun 27, 2020

Seoul National University

Homework 8

INSTRUCTIONS

- The homework is due at 9:00am on Jun 27, 2020. Anything that is received after that time will be considered to be late and we do not receive late homeworks. We do however ignore your lowest homework grade.
- Answers need to be submitted electronically on ETL. Only PDF generated from LaTex is accepted.
- Make sure you prepare the answers to each question separately. This helps us dispatch the problems to different graders.
- Collaboration on solving the homework is allowed. Discussions are encouraged but you should think about the problems on your own.
- If you do collaborate with someone or use a book or website, you are expected to write up your solution independently. That is, close the book and all of your notes before starting to write up your solution.

1 Deep Q-Learning [50 points]

1.1 Introduction

This assignment requires you to implement and evaluate deep Q-learning with convolutional neural networks for playing Atari games. The deep Q-learning algorithm was covered in lecture, and you will be provided with starter code. You need to use TensorFlow as the given starter code does. It is highly recommended for you to run the code on GPU.

Please start early! The actual coding for this assignment will involve about 60 lines of code, but the evaluation may take a long time.

1.2 Implementation Guide

The first phase of the assignment is to implement a working version of deep Q-learning. To run the code, follow the instructions in `main.ipynb`. It will not work however until you finish implementing the algorithm in `dqn.py`.

You may want to look at `run_dqn_atari.py` before starting the implementation. This file defines the convolutional network you will be using for image-based Atari playing, defines which Atari game will be used (LunarLander is the default), and specifies the hyperparameters.

The default code will run the LunarLander game with reasonable hyperparameter settings. The starter code already provides you with a working replay buffer, all you have to do is fill in parts of `dqn.py`, by searching for `YOUR CODE HERE`. The comments in the code describe what should be implemented in each section. You may find it useful to look inside `dqn_utils.py` to understand how the replay buffer works, but you should not need to modify it. You may also look inside `run_dqn_atari.py` to change the hyperparameters or the particular choice of Atari game.

1.3 Evaluation

Run LunarLander for at least `500K` steps with the default hyperparameter settings, and include a learning curve plot in your report showing the performance of your implementation. The x-axis should correspond

to number of time steps and the y-axis should show the mean 100-episode reward as well as the best mean reward. You can generate the plots easily by following `main.ipynb`, passing the log directory created by `run_dqn_atari.py`.

In the log directory, you can also find out video files of your agent. You may examine these files to understand how your agent works as training goes on.

2 Policy Gradient [50 points]

2.1 Introduction

This assignment requires you to implement and evaluate Vanilla Policy Gradient, Policy Gradient with reward-to-go, Policy Gradient with neural network baseline for playing CartPole. The Policy Gradient algorithm was covered in lecture, and you will be provided with starter code. You need to use TensorFlow as the given starter code does.

2.2 Implementation Guide

The assignment is to implement a working version of Policy Gradient. To run the code, follow the instructions in `main.ipynb`. It will not work however until you finish implementing the algorithm in `train_pg_f18.py`.

The default code in `train_pg_f18.py` will run CartPole with reasonable hyperparameter settings. All you have to do is fill in parts of `train_pg_f18.py`, by searching for `YOUR CODE HERE`. The comments in the code describe what should be implemented in each section.

2.3 Evaluation

Run CartPole with the default hyperparameter settings, and include a learning curve plot in your report showing the performance of your implementation. The x-axis should correspond to number of parameter update iterations and the y-axis should show the mean reward of collected episodes at each iteration. You can generate the plots easily by following `main.ipynb`, passing the log directory created by `train_pg_f18.py`.

In the log directory, you can also find out video files of your agent. You may examine these files to understand how your agent works as training goes on.