COMPGI13/COMPM050 (Advanced Topics in ML): Assignment #3

Due on Tuesday, April 11, 2017

Thore Graepel, Koray Kavukcuoglu, Hado van Hasselt, Joseph Modayil

Setup and Logistics

Submissions

- **Due date**: **11th April 2017, 23:55** (Start date: 11th Mar 2017)
- Submit .zip via Moodle containing: **report, code, readme** (with instructions how to run the code), **trained weights** (final weights only).
- (Optional, but encouraged) If you want to use **github** to manage your code, please accept the following invitation. This will generate a private repo with your github id (make sure you are signed).
- Individual assignment.

In this assignment you will develop some simple reinforcement learning agents that use function approximation. We will test agents on the classic Cart-Pole problem and three games from the Atari Learning Environment, both of which are available by installing Open AIs Gym. The assignment will be primarily assessed on the written report. Turn in your report and links to your code to Diana.

Environments: Installing Gym

In order to access the environments used in this assignment, you will need to install `gym`. It is a simple pip install, but if you run into problem check you the official documentation

```
git clone https://github.com/openai/gym
cd gym
pip install -e . # minimal install
```

After installation:

- Initialize an environment and test it with a random agent. See documentation for examples.
- Make sure you have all the environments needed for the assignment, but checking the registry.

```
from gym import envs
print(envs.registry.all())
```

Now we are all set! Let's train some agents!

Problem A: Cart-pole [60 points]

Load the cart-pole environment ("CartPole-v0"). The task is to keep a pole balanced indefinitely. This environment has a four dimensional observation and two discrete actions. We modify the problem from the Gym implementation as follows. Set the reward to be 0 on non-terminating steps and -1 on termination. Set the maximum episode length to 300. Let the discount factor be 0.99.

1. Generate three trajectories under a uniform random policy. Report the episode lengths and the return from the initial state. [3 points]

2. Generate 100 episodes under the above random policy, and report the mean and standard deviation of the episode lengths and the return from the starting state. [2 points]

3. Collect 2000 episodes under a uniformly random policy and implement batch Q-learning to learn to control the cart-pole. To represent the value function, use i) a linear transformation and ii) a 1 hidden layer(100) network. Plot your performance and loss during training. Report results for learning rates in $[10^{-5}, 0^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 0.5]$. [2 x 5 points]

4. Implement an online Q learning algorithm with a small neural net for function approximation. Connect the input observation to a single hidden layer of 100 units followed by a ReLU, which is followed by a layer with one output per action. Note that with the automatic gradient computation in tensorflow, you must apply a stop gradient operation to avoid adapting the learning target.

$$\delta = R_{t+1} + \gamma \, \texttt{tf.stop\_gradient}\left(\max_a Q(S_{t+1}, a)\right) - Q(S_t, A_t)$$

$$Loss = \frac{1}{2}\delta^2$$

   Train an epsilon-greedy Q-learning agent with epsilon of 0.05 over 2000 episodes. Adjust parameters (such as step sizes in the optimizer) to improve performance. Plot learning curves of control performance per episode, averaged over 100 runs. Report the step sizes and optimizer you used, along with any other parameters that you modified. [10 points]

5. Make a network using a 30 unit hidden layer with ReLUs. Now try it with 1000 hidden units. Compare and report their performance over 2000 episodes. (Log and plot the performance and bellman residual every 20 steps). [10 points]

6. Start with the 100 unit network from question 4. Make an experience replay buffer for your agent that stores transitions of experience $\{(S_i, A_i, R_{i+1}, S_{i+1})\}_{i=0}^{t}$. Train your agent with the replay buffer and compare performance to your original agent. [10 points]

7. Start with the network from the previous part and add a target network, copying the current network parameters to the target network every 5 episodes. Train and compare. [10 points]

8. Try one of two modifications. Use either Sarsa or Double-Q learning instead of Q-learning. Compare the learning curves and final performance to your previous agent. [5 points]

Note: For control performance, please report the (averaged) empirical return obtained and the (average) length of the episodes.

Problem B: Atari games [40 points]

For the second part of the assignment, we will apply Q-learning to three Atari games. However, we will use a smaller network and shorter training times than is commonly used. We will use the Atari games *Pong*, *Ms. Pacman*, and *Boxing*.

State space

First, there are several pre-processing steps involved in-between the Atari environment and your learning agent. The environment observations must be reduced in size to **28x28** images and converted to greyscale. Stack four consecutive frames together to get a single agent state of size (28x28x4). Store these as bytes (tf.uint8) to conserve memory. *Note that there are a several variants of the Atari environment available in Gym, and we will use a version of the games that skips frames in a stochastic fashion (the environments are named "`Pong-v3`", "`MsPacman-v3`", and "`Boxing-v3`"). Also, make sure you are using these and not the "`*-ram`" versions.*

Agent

Implement a convolutional network to approximate the Q-function. For the first layer use a filter size of 6x6 with a stride of 2 and 16 channels followed by a ReLU. For the second layer use a filter size of 4x4 with a stride of 2 and 32 channels followed by a ReLU. Flatten the output and add a fully connected third layer with 256 units followed by a ReLU. Finally, we have a linear layer that predicts the state-action value function. The top of the network should have one output for each action.

Training

Train the network using Q-learning by adding appropriate losses. Use a fixed epsilon of 0.1, and set the discount factor to 0.99. Clip the environmental rewards to be -1, 0, or 1. Use minibatches of size 32. Use the RMSprop optimizer with a stepsize of .001, and adjust if needed. Use a target network (switching every 5000 steps), and an experience replay buffer (storing at least 100000 transitions). Adjust these parameters as needed – for instance, use a larger replay buffer if your machine can handle it.
Training this small network on Atari takes a portion of a day. Report performance when training for one million agent steps on a single run, and evaluate your agent every 50 000 steps.

1. Report the score and frame counts from the three games under a random policy, evaluated on 100 episodes. Report both the average and the standard deviation. [5 points]

2. Report performance on the three games from an initialized but untrained Q-network, evaluated on 100 episodes. Explain why the performance can be different from part one. [5 points]

3. Plot the losses of your agent during the course of training. Discuss the shapes of the curves and why they might differ from supervised learning. [3 x 5 points]

4. Report the final performance in terms of cumulative undiscounted rewards per episode of your trained agent, averaged over 100 episodes. Describe any modifications to the above by which you were able to improve performance in this limited data regime. [3 x 5 points]

General structure and notes on submissions

1. Include your **name** and **student no.** (+github username if used) in the submission: both in the report + zip. (Moodle automatically masks your names/ids)

2. **Submission** must be in **.zip** format (not .rar or any other). Name it:

   ```
   $studentno_assignment$nr_assignment.zip.
   ```

   Structure:

   ```
   code   #code folder
   models #trained_models folder
   report #report
   readme #instructions for the code and tf/python versions
   ```

3. **Report:** must be **present** :P to compile all of your results & questions and must be **.pdf** format (not .doc, .txt, .md ....etc). The report is the most important thing. This will be the thing you are graded on + a quick check of the trained models. Name convention for the report:

   ```
   studentno_report.pdf
   ```

   Do not use more than one file: everything should be contained in `studentno_report.pdf`.

4. **Code**: comment your code, don't put everything in the name of file :P

IMPORTANT: If **any of the above are not** in place, you will **not receive any marks** on your submission! Others:

- Please describe your experimental set-up and how you chose the hyperparameters (As I said I'm not really rigorous on that, thus you don't need to fully document that with plots & results, but I need to have a sense of what you did, especially when something seems wrong.)
- Don't include pages of logs on runs/screenshots of your terminals. Just plot the learning curves. **The x-axis in these should be either epochs(for supervised tasks), or number of updates.**