50.021 – AI

Alex

Week 08: A shallow deep Q

[The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.]

1 Disclaimer

I dont want you to take some code submitted to openAI gym and say: you have done it. Then you have saved your time and learned – absolutely nothing. Merely running code from others is below uni level. Hope your own aspirations are higher than that :) .

2 in class task: Shallow deep-Q-learning

Lets start with this code `https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html` and adapt it so that it works without images and thus learns much faster.

steps to do: Usually what always helps is to look into the code and make a plan how to change it. Goal is: getting over 195 average reward (for cartpole it is equal to episode length) over the last 100 rounds.

- take the code and put it into a .py file. Encapsule the actual running part starting with `num_episodes = 50` into a main-routine
- What are your next steps:
  - I. replace the sampling interface from images to kinematic inputs of the cartpole
  - II. replace the image-based DQN by a neural network suitable for the 4-dim kinematic inputs of the cartpole

- I – Google for the gym cartpole github to understand its interfaces
- I – outcomment everything related to `get_cart_location, get_screen` functions. Outcomment all calls of these functions, including the computation of state using these
- I – you will get your start state from the call to `env.reset()` inside the iteration over episodes. Google for the gym github to understand what `.reset()` does
- I – you have a call to step() which should get you the next state but right now it does not (because the code wants to get states from images) fix that.
- I – wrap your states and next states into a pytorch tensor of proper dtype (nope, not double) with a batch dimension of size 1
- II – replace the neural net by 3 fc layers with 24,24, and number of output space many outputs. Dont make the fc layers too fat, unless your home feels to cold and you want to employ thousands of hidden neurons to warm your place up. Use leaky_relu or selu for the first 2.
- III change parameters a little, 1000 episodes is a start. Personally (for cartpole only!!) i prefer to cap max episode length to 500 by forcing a done flag in that case – to avoid cheating by letting one single episode of 1000 length you solve the problem. Usually you wont reach with in 500 episodes the 195 average over last 100 replays. Lets change a few params.
  - increase reply mem to 50k, double the target update time (or to 30), tune your learning rate above $1e-4$, below $1e-2$, a mild weight decay maybe??
  - consider a linear eps decay to 0.02 over 12k to 25k steps instead of the given exponential one (not sure if it consistently helps).
  - What helps me a lot is to switch to double Q-learning `https://arxiv.org/abs/1509.06461`.
    replace (where L is the huber loss)

    $$\mathrm{L}(r + \max_{a'} \gamma Q_{\text{target}}(s', a'), Q_{\text{policy}}(s, a))$$

    by

    $$a'_p = \text{argmax}_{a'} Q_{\text{policy}}(s', a')$$
    $$\mathrm{L}(r + \gamma Q_{\text{target}}(s', a'_p), Q_{\text{policy}}(s, a))$$

    Note:
    - you take for argmaxing the state s' which you would use in your target net from your experience s, a, r, s'.

    - you may need to use no_grad() on the policy net (plus put the policy net into eval mode and after that put it back into train mode), and torch.gather on the target net.

- outcomment `env.render()` unless you want to see every iteration a psychotically dancing cartpole o_O. Better save a model and run it in prediction model

3 Homework task 1

That should take no more than one hour of your life.

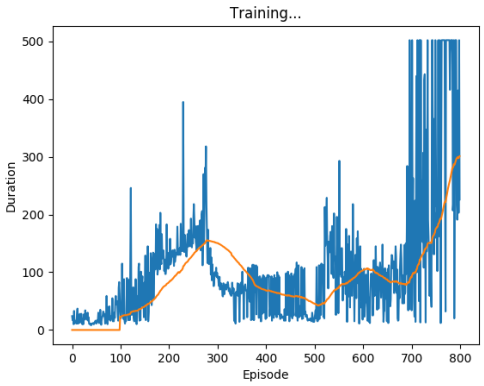
Take the code of `simpleq2_studentversion.py`, finish it. Submit it, and a plot of the $Q(s, a)$ for the learnt solution (e.g. you can plot one graphic for each of the five actions uldr, stay, or you use my one visual with the fences that plots them all). You may consult the free pdf of Sutton and Barto for how to implement.

4 Homework task 2

Take the cartpole from in class coding.

Try to find parameters such that you get – on average over the last 100 iterations – at least 195 reward . – this may take some time. Here the song when you are waiting for the curve to lift off: `https://www.youtube.com/watch?v=ExTar4dUZiQ`, the lyrics may fit every AI class.

Submit your reward curve during training along with your code for your best try (you need to reach 195 only at some time, usually it starts to overfit and degenerate again). Notice the instability of RL!!! Its often not a result which is trivially reproducible.



If you are lucky, then the peak at 300 gets already over 195 and you are done. Here it unlearned and recovered (a version with $lr = 5e - 4$, no Double DQN).

Run an ablation study on your implementation of DQN for cartpole . That is change several parameters and observe performance changes when you run the algorithm every time 5 times:

Report averaged reward curves over these 5 times when you change at least: loss (one alternative, a meaningful loss for regression, not $L = 0$, yah), some neural network parameter (one alternative), learning rate (3 values), the replay memory size (3 values). You can play with more.

Next week you will take that code and use it to implement policy-gradient reinforce, which learns slower but achieves better scores.

5 Homework task 3

take the deep-Q code from the in class task with the cartpole and adapt it to the lunar lander from the openAI gym. The modification – from cartpole to lander – should take you 15 minutes at most. Otherwise you are doing something wrong!

- check that your rewards have always the right dtype
- you cannot plot durations as you have done with cartpole, but you have to maintain an averaged reward over your episode for performance monitoring.

Run it for 2000 ... 2500 iterations. Try it until your average reward of the last 100 plays reaches at least 175 (for openAI one requires it to be 195 actually). Be warned: you may see a steep learning curve after 800 to 1200 iterations only. That one needs some patience

double Q-learning is highly suggested here. its 5 more lines and 30 minutes of thinking.

It is very insightful to run for every step an `env.render()` to see what the lander has learned in every stage.

Running this for lunar lander on a cpu for 2000 iters may take 1 hour or a few hours if your CPU is a Gurke (use google translate if you dont know what that means).

You save a model every 100-200 iters, then use in prediction mode the average of the 5 best models. Submit runnable code for that, too.