CS5180/7180: Special Topic - Artificial Intelligence (Spring 2025) Robert Platt
Northeastern University 11:59 PM, Mar 26, 2025

# Exercise 6: Deep Q-Network (DQN)

Please remember:

- Be mindful that this exercise may include question(s) requiring **significant compute time**, plan accordingly.
- Exercise due at **11:59 PM EST Mar 26, 2025**.
- Submissions should be made electronically on Canvas. Please ensure that your solutions for both the written and programming parts are present. You can upload multiple files in a single submission, or you can zip them into a single file. You can make as many submissions as you wish, but only the latest one will be considered.
- For **Written** questions, solutions should be typeset. If you write your answers by hand and submit images/scans of them, it will **not** be considered.
- The PDF file should also include the figures from the **Plot** questions.
- For both **Plot** and **Code** questions, submit your source code along with reasonable documentation.
- You are welcome to discuss these problems with other students in the class, but you must understand and write up the solution and code yourself. Also, you *must* list the names of all those (if any) with whom you discussed your answers at the top of your PDF solutions page.
- Each exercise may be handed in up to two days late (24-hour period), penalized by 10% per day late. Submissions later than two days will not be accepted.
- Contact the teaching staff if there are medical or other extenuating circumstances that we should be aware of.

1. **3 points.** *Nonlinear Function Approximation with Neural Networks*

   Universal approximation theorem tells us Neural Networks have a good deal of approximation capabilities. In this question, we will be approximating a low-dimensional non-linear function $f(x) = 1 + x^2$ using stochastic gradient descent. Please use PyTorch library (https://pytorch.org) for this homework.

   (a) **Code:** Use linspace to get 500 even spaced values in the range of $[-10, 10]$, complete your training set by obtaining the function outputs for each of the 500 inputs.

   (b) **Code:** Build your two-hidden-layer model with *relu* as activation, layers are of size 8. Setup the Adam (Adaptive Moment Estimation) optimizer.

   (c) **Code:** Randomly sample batches of size 32 from your training set, calculate your loss on the batch (mean squared error), and perform parameter update based on the gradient. Repeat until your loss is sufficiently low or stable. You will be setting your own learning rate.

   (d) **Plot:** Plot your approximated function and the original function using a line plot. Additionally, test and plot your model with layers with widths 16, 64 and 128.

   (e) **Written:** How accurate is your learned model? How accurate it is within the range of $[-10, 10]$? How about outside of the range? Can you notice any difference between models having different layer widths? What do you think that may have caused the difference?

2. **3 points.** *Four Rooms yet again but with DQN*

   In this question, you will implement DQN and test it on our favorite domain, Four Rooms, as implemented in ex4 and ex7. You will be in charge of your network architecture and other hyperparameters. However, you are recommended to start with a network with a single hidden layer of 64 units; replay size of 100000 transitions, $\epsilon = 0.1$, batch size of 64, discounter factor at 0.99.

   (a) **Code:** Setup your environment, replay memory buffer, value networks, and optimizers. Remember that although your value network models the $Q(s, a)$ value function ($s, a \rightarrow q(s, a)$), in the case of DQN it takes the state as input and output vector-form values for all actions ($s \rightarrow [q(s, a_1), q(s, a_2), ...]^T$).

(b) **Code:** Collect rollouts and store them in reply memory while performing batch updates in Q-Learning fashion based on the data in your reply memory. Your loss is the batch MSE of your TD error (the difference between your targets and your value predictions). Also, remember to use $\epsilon$-greedy in your rollouts.

(c) **Code:** Update your target network periodically (e.g. every 10000 steps). Repeat the learning process until there is little or no performance gain.

(d) **Plot, Written:** Plot your learning curve (averaged over 10 trails) with confidence bands. How does it compare to your tabular methods from previous exercises?

3. **2 points.** *Evaluate and tune your DQN on more environments*

In this question, you will evaluate your DQN on CartPole (https://gym.openai.com/envs/CartPole-v1/) and LunarLander (https://gym.openai.com/envs/LunarLander-v2/). You will need to schedule your $\epsilon$ so that it is annealed to a low value (e.g. 0.05) from 1.

(a) **Code:** Run and tune your DQN on CartPole and LunarLander.

(b) **Plot** Plot your learning curve (averaged over 5 trails) with confidence bands.

(c) What are the network architecture and hyperparameters you find works for CartPole and LunarLander respectively?

(d) **Written:** Render your runs at different stages of training, what kind of policy (or policies) do you see? What do you think the corresponding value function would be? You can inspect your value function to validate your intuition.

4. **2 points.** *DQN on Atari*

(a) **Code:** Evaluate your DQN on any of the Atari game provided with gym (we recommend start with Pong). You will need to add convolution layer. Additionally, you may want to use libraries like OpenAI Baselines for image processing (grayscaling, framestacking, etc.).

(b) **Plot, Written** Plot your learning curve (averaged over some trails or not). How long does your experiments take to complete? What did you have to do with the hyperparameters to make it work?