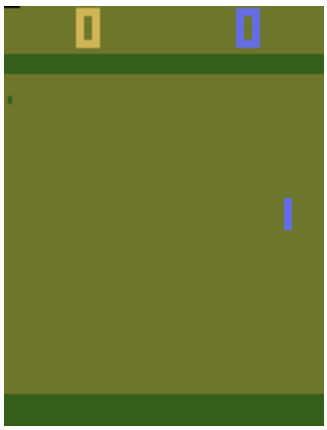
CSC 580: Artificial Intelligence II
Winter 2026

Homework #4: Deep Q-Learning Networks for Atari Pong

This assignment is to implement Deep Q-Learning Networks (DQN) for an RL agent that lean to play Atari games, in particular the game of **Pong**.



The goal is to **implement DQN**, and **train the model long enough that the agent plays as well as humans** (hopefully).

Preliminary:

You should do this assignment on platforms that provides GPU support (and high RAM ideally) because it involves intensive computation. Another issue is visualization (rendering color animation. The code provided will run on **Google CoLab** but should be fine on most other platforms.

Overview:

- We will implement a DQN model in a very similar way as the original Atari RL paper ("Human Level Control Through Deep Reinforcement Learning" (2015 Nature)):
  - two **Convolutional Neural Network (CNN)** networks (prediction and target)
  - four-frame stacking of input images
  - use of a replay buffer (although we use a smaller size, down from 1,000,000 to 100,000)
- We also add another feature in the environment, which reduces the action space (number of actions) from 6 to 4 (to ignore irrelevant actions)

- In this assignment, we start with a **checkpoint** **after the initial 700,000 steps have finished**. This is to (skip the warm-up phase where the replay buffer is filled and) focus on the training phase.
- **(*)** The checkpoint includes three files. You first load those files and continue further training.
  - pre-trained weights for the prediction network (called 'q_net')
  - pre-trained weights for the target network
  - content of the replay buffer -- 100,000 transactions (where each transaction is a tuple (observation/image, reward, terminated, truncated, info))
- Starting from the 700,000 step point, you further train the model until **at least the 2,500,000 (2.5M) step point or the model completely beat the system agent**.
- You record the training performance **every 300,000 steps** on the way, and create a chart of the progress.
- As for 'performance', in addition to **total_rewards** (which you calculate from the rewards returned from the environment), you also measure the **number of ball hits by the agent**. This is a custom feature in the assignment, and the code to detect a hit is given to you -- **you must incorporate the code in the training**

on your own, though.

- At the end of (long) training, you create a **video** of an episode using the last trained model as a part of the submission, along with source code files and a write-up.

Start-up Code:

- A few support code and data files are provided. They are posted on the **course Github**.
- The github site also includes two subfolders ('checkpoints' and 'videos'). Those subfolders are assumed in the code. So if you clone the site, you will get the subfolders as well.
- **(*)** The '700,000 step' pre-trained weights and the replay buffer files are placed under the 'checkpoints' subfolder.
    - Two pre-trained weights files are in the pt format ("q_net.pt" and "target_net.pt")
    - Note that the replay buffer file is in the ZIP format (**"replay_700k.zip"**). You must unzip it to obtain a pkl file (**"replay_700k.pkl"** -- it's zipped because Github does not allow larger files).
- Code to load the checkpoint data files and set up the environment is written in the beginning of the two Notebook files ("Pong_train.ipynb" and "Pong_eval.ipynb"). If you are coding on Colab, change the mount path to your folder organization. Then run the requirements.txt file (as already written).
- The main file is "Pong_train.ipynb". But some core functions are written in "dqn_core.py". Those files are partially filled -- you fill the places indicated with **"## (*) TODO"**.
- The other file "Pong_eval.ipynb" is complete. However you must complete "dqn_core.py" before running code in the eval file.

Your Task:

Your task consists of the following four steps. Step 1 through 3 (inclusive) is a **preliminary phase** for completing the code. Step 4 is the **formal experiment phase** to do further model training.

Step 1. Complete "dqn_core.py"

- There are a few places you are supposed to fill. Those places are indicated with **"## (*) TODO"**.
- For ReplayBuffer, read the docstring first and understand it. The buffer is fixed-length (of 'capacity'), and must be **circular**. The variable 'pos' indicates the next insertion point/index.
- Note the model DQN is based on **PyTorch**, in particular, the **torch.nn** package -- it provides neural network layers and utilities. So your code for 'def forward(self, x)' must be written using utility functions in the package.

  After you complete the file, you can run ***"Pong_eval.ipynb"*** (which is completely written) to check the correctness of your code.

Step 2. Complete "Pong_train.ipynb"

- There are several places you are supposed to fill, in particular in the cells under **code section 2 ("Further training") and 3 ("Main Training Loop")**. Those places are indicated with **"## (*) TODO"**. Again, remember that the code is written in **PyTorch**.
- Then execute the cell **"3.1 Run do_train() by executing the cell below"**. That will do further 300,000 steps by default (by 'TOTAL_STEPS'), but you can change it if you like .
- Visualize the learned model by running the **code section 4 ("Visualize the further-trained model ")**.

Step 3. Incorporate Hit_Count

- Incorporate the hit-count metric. It measures the number of paddle hits by the agent. The code to do so is written in ***"Pong_eval.ipynb"***. **Incorporate that in the training part in your code**.
- To do so, you will probably rewrite the function **"def do_train(start_step)"**. It's up to you how you do it. Design by your self.

- Then execute the cell **"3.1 Run do_train() by executing the cell below" again** and check the correctness of the code (by the output). That will do another 300,000 steps by default (by 'TOTAL_STEPS'), but you can change it if you like.

Step 4. Conduct formal further training and produce performance chart and video

- Now that your code is complete, you will **re-start** the code and load the 700,000 step checkpoint. Also make sure you set 'TOTAL_STEPS' back to 300,000 if you had changed in the previous steps.
- To do further training, you train 300,000 steps at a time, until **at least the 2,500,000 (2.5M) step point** (thus step points 700_000, 1_000_000, 1_300_000, 1_600_000, 1_900_000, 2_200_000 and 2_500_000) **or the model completely beat the system agent**.
- After training every 300,000 steps, you obtain the performance (total_rewards and hit_count) using the last model (and executing the cell **"3.1 Run do_train() by executing the cell below"**).
- Note that one 'run' will probably take 20-25 minutes on Colab.
- After you finish all runs, you create:
    - a chart that shows the performance progress during training. Show it in the Notebook.
    - a video that shows a game (one episode) using the last model. Save the video under the 'videos' subfolder.

Deliverables

Submit the following:

1. Shared link to the folder on your Google Drive account where you have your code and materials.
2. Write-up report.
    - **Minimum 2.0 pages** (in pdf of docx).
    - **Your name, student ID, course number and assignment number (HW#4)** at the top of the document. Failure to comply with this will result in significantly reduced grades.
    - **Collaborators** if you worked with other students or used **GenAI tools**. Claim any reference sites you consulted as well.

    - Your comments on DRL agents after learning. How long did it take to train? Did the learned agent play 'well'? Did it exceed your expectation? Write some observations of the agent learning and/or behavior.
    - Write your general reflections on the problem:
        - What you learned from this exercise.
        - How difficult you felt this exercise was.
        - Any particular difficulties you encountered.
        - How you would do/approach differently next time (if there was one).
        - and anything else.

Submissions

In the submission box 'HW#4'. Upload the write-up file. Type in your shared Google Drive folder link in the submission comment box.