Individual Assignment

Artificial Intelligence in Games

September 23, 2021

In this assignment, you will train a Deep Q-Network [Mnih et al., 2015] to play Atari Breakout. Please read this entire document and the paper before you start working on the assignment.

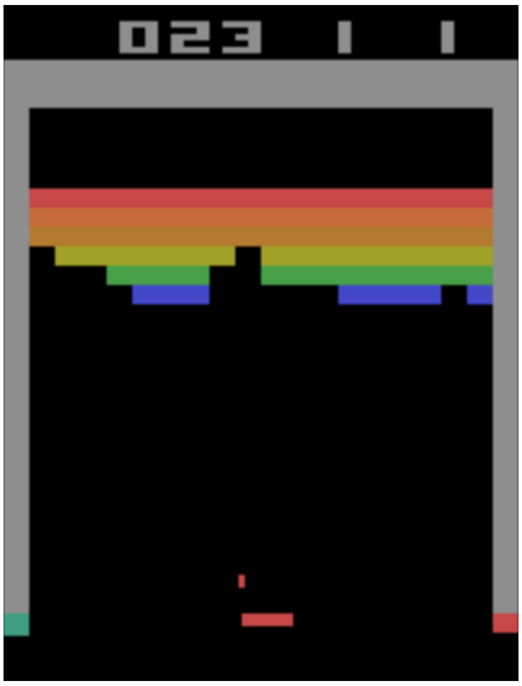

Figure 1: Atari Breakout.

1 Programming environment

You will use Google Colaboratory to train a Deep Q-Network on a graphics processing unit (GPU). If you are not familiar with Google Colaboratory, please read this tutorial. You will need a personal Google account.

As a first step, create a new notebook on Google Colaboratory. On the menu at the top, select *Runtime*, select *Change runtime type*, select *GPU* as the *Hardware accelerator*, and select *Save*. After this, your notebook will have access to a GPU.

The next step is to install OpenAI baselines. In a new code cell, type `!pip install baselines --no-deps`, and execute the cell (note the exclamation mark).

The next step is to mount your Google Drive so that it is accessible from your notebook. This can be accomplished by executing a code cell containing the code shown in Listing 1 and enabling access as required.

Listing 1: Mounting a Google Drive.

```
from google.colab import drive
drive.mount('/content/drive')
```

In order to import the Atari Breakout ROM, you need to download the file *Roms.rar* from the Atari 2600 VCS ROM Collection. You should extract this file and upload the two *zip* files that it contains to a folder called *game_ai/roms* in your Google Drive. You do not need to extract these *zip* files. Finally, you can import the ROMs by executing `!python -m atari_py.import_roms /content/drive/MyDrive/game_ai/roms` in a new code cell. If this process is successful, the *ROMs* will be listed as they are imported.

2 Baseline implementation

This assignment is based on a didactic but inefficient implementation of Deep Q-Networks that is available here.

As a first step, copy and paste this implementation into a code cell in your notebook. Although you could run this code cell and observe the training process, this would not be very useful: your virtual machine would run out of memory or time (a Google Colaboratory virtual machine dies after twelve hours, and so everything that is not saved to Google Drive will be lost), and the trained model (convolutional neural network) would not be saved to Google Drive.

This baseline implementation uses OpenAI Gym [Brockman et al., 2016] to create the Atari Breakout environment. If you are not familiar with the OpenAI Gym, please read this introduction. You should become familiar with the main OpenAI Gym environment methods, such as *reset* and *step*.

The implementation also uses several Gym wrappers to transform the original Atari Breakout environment for our purposes. You can find more information about the Atari wrappers here.

Finally, the implementation is also based on Keras, an open-source Python library for artificial neural networks based on TensorFlow. If you are not familiar with Keras, please read the introduction to Keras for researchers and the functional interface tutorial. If you are not familiar with NumPy, you may also read the NumPy quickstart tutorial and the NumPy broadcasting tutorial.

After reading the paper and completing the steps listed above, you should be able to understand the baseline implementation of Deep Q-Networks.

3 Improved implementation

This section lists some improvements that you should make to the baseline implementation.

The baseline implementation uses an optimizer called Adam [Kingma and Ba, 2014]. In order to follow the paper, your first task is to substitute the optimizer created by a call to *keras.optimizers.Adam* by an *optimizer* created by a call to *tf.keras.optimizers.RMSprop*. You should use a learning rate of 0.0001 and a discounting factor (rho) of 0.99.

The baseline implementation uses the variable *epsilon_random_frames* to control how many frames (states) should be observed before greedy actions are taken. You should eliminate this variable and remove it from the conditional test where it appears.

The baseline implementation has a replay buffer of size 100000. This will cause memory problems on Google Colaboratory. You should reduce the size of the replay buffer to 10000.

The baseline implementation will train the Deep Q-Network until the average return obtained during the last 100 episodes surpasses 40. This will take longer than the maximum lifetime of a virtual machine. You should change the outermost while loop so that training is interrupted after 2000000 frames (states) have been observed.

The baseline implementation draws a batch and updates the Deep Q-Network every four frames only after the replay buffer size is larger than the batch size (note that the corresponding comment found in the code is inaccurate). You should draw a batch and update the Deep Q-Network every four frames only after the replay buffer is *full*.

The baseline implementation stores the return of the last 100 episodes in a list called *episode_reward_history*. Instead, you should store the return of every episode. The variable *running_reward* should still only contain the average return of the last 100 episodes.

The baseline implementation computes the one-step return for each state in a batch incorrectly. Although this implementation works for Atari Breakout, it would not work for many other Atari games. You should substitute two assignment in the baseline implementation by a single assignment shown in Listing 2.

Listing 2: Corrected one-step return computation.

```
# Baseline implementation:
# updated_q_values = rewards_sample + gamma * tf.reduce_max(
#     future_rewards, axis=1
# )
# updated_q_values = updated_q_values * (1 - done_sample) - done_sample

# Correct implementation:
updated_q_values = rewards_sample + (1 - done_sample)*gamma*tf.reduce_max(
    future_rewards, axis=1
)
```

After the training loop, you should use Keras to save your model (convolutional neural network) to a Google Drive folder (for instance, */content/drive/MyDrive/game_ai/model*). You should also use NumPy to save the list *episode_reward_history* that contains the return for each training episode.

4 Training

As a next step, train your Deep Q-Network for 2000000 frames. This may take more than ten hours. The average return during the last 100 episodes should be close to 10.

Hint: You should test your implementation by training your Deep Q-Network for a few frames only. Once you are confident in your implementation, you should train your Deep Q-Network overnight.

5 Testing

In order to test your Deep Q-Network after training, you should create an additional code cell.

As a first step, load the trained model (convolutional neural network) from your Google Drive. After that, create the Atari Breakout environment and employ the same wrappers used for training the model.

You should use the wrapper *gym.wrappers.Monitor* to record videos of your trained agent interacting with the environment. An environment wrapped by this class behaves normally, except for the fact that the episodes are recorded into a video that is stored to disk. The wrapper *gym.wrappers.Monitor* is able to see past the other wrappers in order to record raw frames.

You should record ten episodes of interaction using a *greedy* (not *ϵ-greedy*) policy based on your trained Deep Q-Network. Listing 3 presents an incomplete version of the testing code.

Important: The videos will be written to your Google Drive only after the method *env.close* is called.

Listing 3: Recording videos of a Deep Q-Network agent.

```
from google.colab import drive
drive.mount('/content/drive')

from baselines.common.atari_wrappers import make_atari, wrap_deepmind
import numpy as np
import tensorflow as tf
from tensorflow import keras
import gym

seed = 42

model = keras.models.load_model('/content/drive/MyDrive/path/to/your/model')

env = make_atari("BreakoutNoFrameskip-v4")
env = wrap_deepmind(env, frame_stack=True, scale=True)
env.seed(seed)

env = gym.wrappers.Monitor(env, '/content/drive/MyDrive/path/to/your/videos',
                           video_callable=lambda episode_id: True, force=True)

n_episodes = 10
returns = []

for _ in range(n_episodes):
    ret = 0

    state = np.array(env.reset())

    done = False
    while not done:
        # FIXME: Incomplete

        ret += reward

    returns.append(ret)
```

```
env.close()

print('Returns: {}'.format(returns))
```

6 Submission instructions

This assignment corresponds to 20% of the final grade for this module. You will work individually. The deadline for submitting this assignment is Dec 17th, 2021. Penalties for late submissions will be applied in accordance with the School policy. The submission cut-off date is 7 days after the deadline. Submissions should be made through QM+. Submissions by e-mail will be ignored. Please always check whether the files were uploaded correctly to QM+. Cases of extenuating circumstances have to go through the proper procedure in accordance with the School policy. Only cases approved by the School in due time will be considered.

If you are unsure about what constitutes plagiarism, please ask.

The (individual) submission must be a single zip file. This file must contain a single folder named *[student_id]_[student_name]*. This folder must contain a report, a folder named *videos*, and a folder named *code*.

The *code* folder must contain two files: *dqn_train.py* and *dqn_test.py*. These files should contain the content of the code cells used for training and testing, respectively. Based solely on the correctness and clarity of your code, you will receive the following number of points for accomplishing each of the following tasks:

- Implementing the improvements listed in Section 3. [20/100]
- Implementing the testing described in Section 5. [10/100]

The *video* folder must contain 10 video files. These files should be the result of the testing procedure described in Section 5. You will receive the following number of points for accomplishing the following task:

- Recording videos that provide evidence of successful training (see Section 4). [10/100].

The report must be a single pdf file. Other formats are not acceptable. The report must be excellently organized and identified with your name, student number, and module identifier. You will receive the following number of points for answering each of the following questions:

1. During training, why is it necessary to act according to an ϵ-greedy policy instead of a greedy policy (with respect to Q)? [10/100, please write less than 100 words]
2. How do the authors of the paper [Mnih et al., 2015] explain the need for a target Q-network in addition to an online Q-network? [10/100, please write less than 100 words]
3. Explain why the one-step return for each state in a batch is computed incorrectly by the baseline implementation and compare it to the correct implementation. [10/100, please write less than 300 words]
4. Plot a moving average of the returns that you stored in the list *episode_reward_history*. Use a window of length 100. **Hint**: *np.convolve(episode_reward_history, np.ones(100)/100, mode='valid')*. [10/100]
5. Several OpenAI Gym wrappers were employed to transform the original Atari Breakout environment. Briefly explain the role of each of the following wrappers: *MaxAndSkipEnv*, *EpisodicLifeEnv*, *WarpFrame*, *ScaledFloatFrame*, *ClipRewardEnv*, and *FrameStack*. [20/100, please write less than 500 words]

You may also work on the following tasks for your own edification:

- Adapt your code to train a Deep Q-Network to play a different Atari game available in the OpenAI gym.
- Write your own wrapper for an Atari game. This wrapper should transform observations or rewards in order to make it much easier to find a high-performing policy.

References

[Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI gym.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.