Reinforcement Learning
Project work

November 9, 2020

1 Introduction

The project work for the course is about implementing a reinforcement learning agent that can play the game of Pong **from pixels** (see Figure 1). In this environment, the agent controls one paddle and can take one of three actions: moving up or down, or staying in place.

The assignment is supposed to be done in groups of 2 students. If you need to find a partner for the project, please join the `project-groups` channel on Slack and advertise yourself :). Each group can give a name to their agent (anything you like, keep it civil).

To get familiar with Pong and reinforcement learning from pixels, you can read the blogpost by Andrej Karpathy: `http://karpathy.github.io/2016/05/31/rl/`.

You are allowed to get inspiration from other sources, **as long as you reference them and clearly state which parts you used, why, and how they work**.

2 Report

Your report should be structured as follows:

1. Introduction (motivation and problem statement),
2. Review of external sources you used,
3. Design of the agent architecture,
4. Training methodology,
5. Evaluation of results,
6. Conclusions.

The report should be submitted as a PDF file. For more details, refer to Section 4.

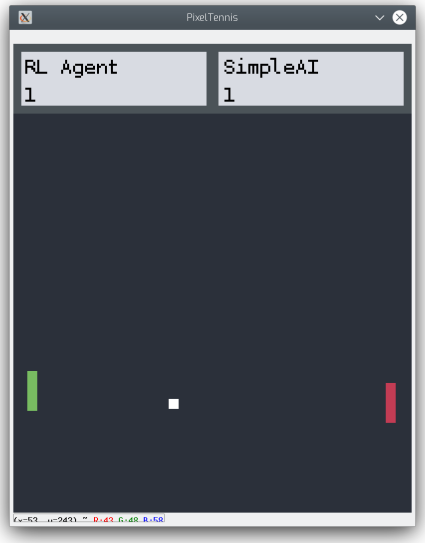

Figure 1: The Pong environment

3 Technical requirements

Your agent **has to be trained using some kind of reinforcement learning method**. Hard-coded algorithms or other machine learning techniques (such as supervised learning) will not be accepted.

3.1 The Pong game

Components:

- `wimblepong.py`: Contains the implementation of the pong game. It provides the following public methods: `step(actions)`, `reset()`, `render()`, `set_names(name player 1, name player2)`. An example implementation of how these functions are used can be found in `test_pong_simple_ai.py`, which demonstrates an implementation for 2 simple hard-coded AI players,
- `simple_ai.py`: A simple hard-coded pong agent,
- `test_pong_ai.py`: This file contains an example implementation of two simple AI agents playing against each other,
- `test_agent.py`: This file tests your agent against Simple AI or against another agent (and can also be used to verify that your agent meets the interface requirements),
- `mass_test_simple_ai.py`: Tests all agents in the given directory against simple AI,
- `epic_battle_royale.py`: Tests all agents in the given directory against each other.

3.2 Interfacing with the pong game

Your agent should be contained in a separate file called `agent.py`, and defined as a separate class called `Agent`. The `Agent` class must implement the following interface:

- `load_model():void` - a method that loads the saved model from a file,
- `reset():void` - a function that takes no arguments and resets the agent's state after an episode is finished,
- `get_action(frame:np.array):int` - a function that takes a `raw` frame from Pong (as a numpy array), and returns an action (integer),
- `get_name():str` - a function that returns the name of the groups agent (max 16 characters, ASCII only).

The agent has to be able to run both on the CPU and the GPU, depending on what is available (you can use `torch.cuda.is_available()` to check if a GPU is available). To test it, you can run the script with `CUDA_VISIBLE_DEVICES=""` to hide the GPU from Python. The model loading should work both with and without a GPU; you can use `map_location="cpu"` when calling `torch.load` to map CUDA tensors to the CPU.

You can make sure that your class interface works with our test scripts, for example `test_agent.py`.

4 Grading

The grading will be based on the following criteria:

1. Approach, **together with a thorough explanation of the reasoning behind all design choices** - 25%
2. Analysis of the whole system and its performance - 25%
3. Performance against the baseline AI included in the environment (based on the winrate) - 20%
4. Performance in the competitive phase - 20%
5. Report quality - 10%

The expected method for the reinforcement learning algorithm is to process the **pixels of the pong environment frame** as **input**. Thus, the input to the machine learning model that you use should be in the form of a pixel array. The array can be preprocessed e.g. by color transformations, stacking multiple frames together, or by an unsupervised representation

learning model, but extracting the positions of elements (e.g. by color or by a supervised model) is not allowed. Approaches that do not take **directly** the pixels of each frame as the sole input, **will not participate in the competitive phase**. Furthermore, the design choice will be graded up to 15% instead of 25%. The **maximum grade** for the approaches that do not make this consideration is up to **65% of the total course project**. If you're in doubt, please ask the TAs about your approach.

When describing the method, provide the details of the learning algorithm, network structure and training procedure. Make sure to justify each design choice — for example, if you used a DQN, describe why you think the method is suitable for the application, and what its benefits are over other methods you considered. A thorough justification will significantly improve your grade, as this shows how well you understand different reinforcement learning concepts and algorithms.

When describing the performance of the algorithm, include your training plots and report performance against SimpleAI. If one of the approaches failed to learn, include the training plots as well and try to justify why it didn't perform as expected. Report all of your hyperparameter values (the cleanest way is to put them in a table).

During the competitive phase, the agents submitted by all groups will be evaluated against each other. The final score for this part will be based on the number of encounters won by each group.

The best four agents will fight against each other and compete for extra bonus points in the final stage, which will take place after the deadline. The exact time and place will be announced in early December.

5 Hints

Several hints that may help you to keep the ball bouncing :)

5.1 Computational power for training — Google Colab

Step-by-step guide:

1. Log-in into your gmail or Aalto account for using Google Drive and Google Colab.
2. Create a folder in your Google Drive to store the files for the project (the `wimplepong` folder).
3. Go to `https://colab.research.google.com/` and create a new notebook for Python 3.
4. Paste the following code into the notebook you have created:

```
import sys
import gym
from google.colab import drive
drive.mount('/content/drive')
sys.path.append("/content/drive/My Drive/RL_Course")
import wimblepong
env = gym.make("WimblepongSimpleAI-v0")
```

5. If this is your first time using Google Colab you will be asked for going into a Google account URL and enter an authorization code. Allow the permissions and paste the authorization code.

6. To activate the GPU, go to *Runtime* → *Change runtime type* and set *Hardware accelerator* to GPU.

7. That's it! Now you can run the Wimblepong code.

5.2 Computational power for training — Maari

You can train your models on the computers in the Maari building. In order to connect remotely, first use SSH to log in to `kosh.aalto.fi` or `lyta.aalto.fi`, and from there use the `ssh` command to connect to one of the CS computers.

The machine available for students are:

1. Maari-A: albatrossi, broileri, dodo, drontti, emu, fasaani, flamingo, iibis, kakadu, kalkkuna, karakara, kasuaari, kiuru, kiwi, kolibri, kondori, kookaburra, koskelo, kuukkeli, lunni, moa, pelikaani, pitohui, pulu, ruokki, siira, strutsi, suula, tavi, tukaani, undulaatti

2. Maari-C: akaatti, akvamariini, ametisti, baryytti, berylli, fluoriitti, granaatti, hypersteeni, jade, jaspis, karneoli, korundi, kuukivi, malakiitti, meripihka, opaali, peridootti, rubiini, safiiri, sitriini, smaragdi, spektroliitti, spinelli, timantti, topaasi, turkoosi, turmaliini, vuorikide, zirkoni

3. Paniikki: brainfuck, deadfish, bogo, befunge, bit, smurf, piet, emo, entropy, false, fractran, fugue, glass, haifu, headache, intercal, malbolge, numberwang, haifu, ook, regexpl, remorse, rename, shakespeare, smith, spaghetti, thue, unlambda, wake, whenever, whitespace, zombie

Remember that there might be other people working on these machines - please check the usage with `htop` and `nvidia-smi` before running heavy training.

Also, don't run your programs directly on Kosh or Lyta - those servers were not designed to handle heavy workloads, and are meant to be used as gateways to access other Aalto machines.

5.3 Training over multiple days

In order to leave your processes running in the background after you've closed your SSH connection or left the machine, you can use the `screen` command. When you start `screen`, it will open a new terminal, in which you can start your lengthy process. When you want to log out of the machine (or close the SSH session), you can detach your `screen` terminal by pressing `Ctrl+a` and `d` (one after another). Doing so will close the terminal and leave your processes running in the background.

To reconnect to an existing `screen` session, use `screen -Rd`.

5.4 Some techniques to try out

A (very) brief overview of machine learning techniques and neural networks architectures that can help you achieve good results:

- Convolutional neural networks — neural networks with fewer weights, aimed at processing spatially structured data (such as images),
- Variatonal autoencoders — a way of performing unsupervised feature learning (and reducing the dimensionality of your data; example use in RL in [1]),
- Spatial soft-argmax layers — extracting coordinates of points-of-interest from the image [2],

To make your models perform better (or train faster) you can also use one of the "fancier" reinforcement learning algorithms. Things you can check out:

- Deep Q Networks (DQN) — a lot of improvements over the 'basic' Q-learning we had in the exercises [3],
- Trust-region policy optimization (TRPO) — applying additional constraints to policy gradient policy updates [4],
- Proximal policy optimization (PPO) — similar idea to TRPO with simplified mathematical formulation [5],
- Actor-critic with experience replay (ACER) — reuse past experience when doing policy updates [6],

If you use an on-policy algorithm, such as PG, A2C, TRPO or PPO, you will likely find [7] to be quite helpful when implementing your algorithm and choosing hyperparameter values. While the paper focuses on continuous control tasks, quite a lot of findings are also applicable for discrete action spaces and image inputs.

Additionally, the parallel environment wrapper from Exercise 6 will stabilize and significantly speed up your training.

6 Submission

The submission **deadline** for the project work is the **5th of December 2020, 23:55**. The project should be **submitted through MyCourses** (via TurnItIn) by one of the group members. During the project show on 8.12.2020 (during the usual lecture time), the best four agents will compete for extra bonus points.

Your submission should include:

1. The code used for training the agent.
2. The agent itself (in a separate `.py` file, meeting the interface specifications listed in section 3).
3. A trained model of the agent, named `model.mdl` in the same directory as `agent.py`.
4. Any other Python files **must be prefixed with the name of your agent**, e.g., if you agent's name is Pippo, and it's using some functions from `utils.py`, the file should be named `pippo_utils.py`.
5. The PDF report (details in Section 2).

References

[1] D. Ha and J. Schmidhuber, "World models," *CoRR*, vol. abs/1803.10122, 2018. [Online]. Available: http://arxiv.org/abs/1803.10122

[2] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *CoRR*, vol. abs/1504.00702, 2015. [Online]. Available: http://arxiv.org/abs/1504.00702

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[4] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[6] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *CoRR*, vol. abs/1611.01224, 2016. [Online]. Available: http://arxiv.org/abs/1611.01224

[7] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem, "What matters in on-policy reinforcement learning? a large-scale empirical study," 2020.