README.md

My Journey into Reinforcement Learning and Deep Q-Learning

This repository contains projects that I have choosen to learn **Reinforcement Learning** (RL) with **Deep Q-Network** (DQN). I had 2 main goals:

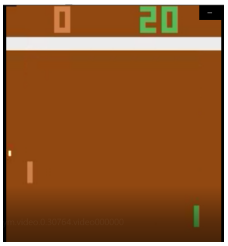
1. Learn how DQN can be implemented in Python
2. Learn a RL framework that makes this implementation easier. I have choosen **TF-Agents** as I have worked mainly with Tensorflow up to now.

You will find 4 projects:

1. The **PONG** game, a first project to understand RL and DQN without a framework;
2. The **Cartpole** environment, a simple RL problem I have selected to approach TF-Agents;
3. The **Atari Breakout** game, a more complex problem to understand the advantages of using a RL framwork like TF-Agents;
4. The **Pacman** game, that allowed me to reuse 98% of the code developed for training Breakout

First DQN: the PONG game

For my first RL DQN project, I have decided to implement the Deep Q-Learning algorithm, without using a RL framework, to train an agent to play the PONG game.

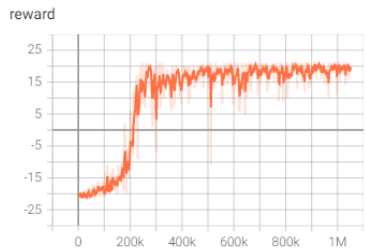


My idea was to understand how DQN can be implemented, particularly the replay buffer and the target model. I read different tutorials and found that Jordi Tores' series "Deep Reinforcement Learning Explained" was the best to get a solid training on RL DQN.

References:

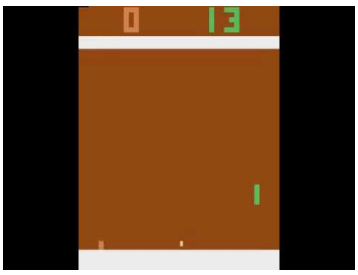
- https://torres.ai/deep-reinforcement-learning-explained-series
- https://github.com/jorditorresBCN/Deep-Reinforcement-Learning-Explained/blob/master/DRL_15_16_17_DQN_Pong.ipynb

I trained the agent network with 3 convolutional layers over about 1M steps to reach the maximun reward of 20:




```
2020-11-28 12:03:37 PST 1052056: 573 games, mean reward 19.020, (epsilon 0.02)
2020-11-28 12:03:37 PST Best mean reward updated 19.020
2020-11-28 12:03:37 PST Sloved in 1052056 frames!
2020-11-28 12:03:37 PST >>>Training ends at  2020-11-28 20:03:37.633613
```

Here is a video showing that the DQN agent was well trained (he is on the right side):



Train a Cartpole DQN with TF-Agents

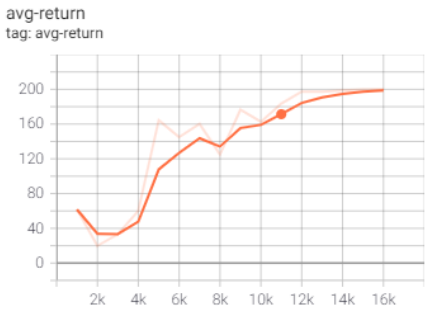
The second example shows how to train a DQN (Deep Q Networks) agent on the cartpole environment using the TF-Agents library. It is the "Hello World" project for TF-Agents. I choose this example because it is simple to train, and so I could focus on the TF-Agents architecture.

References:

- https://www.tensorflow.org/agents/tutorials/1_dqn_tutorial
- https://rubikscode.net/2019/12/23/ultimate-guide-to-deep-q-learning-with-tf-agents/

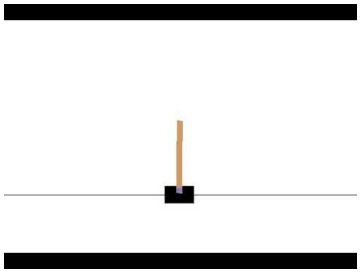
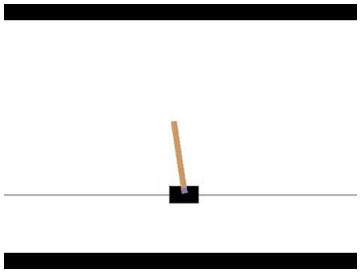
Training this environment using a simple network with one hidden layer with 100 neurons yields the maximum average return of 200 after 16000 iterations:

```
Iteration 1000 - Average Return = 61.70000076293945, Loss = 7.607800006866455.
Iteration 2000 - Average Return = 19.799999237060547, Loss = 22.84531593322754.
Iteration 3000 - Average Return = 33.20000076293945, Loss = 39.23565673828125.
Iteration 4000 - Average Return = 60.099998474121094, Loss = 32.033485412597656.
Iteration 5000 - Average Return = 164.3000030517578, Loss = 44.023162841796875.
Iteration 6000 - Average Return = 145.0, Loss = 54.88655471801758.
Iteration 7000 - Average Return = 160.1999969482422, Loss = 13.389152526855469.
Iteration 8000 - Average Return = 124.9000015258789, Loss = 19.894540786743164.
Iteration 9000 - Average Return = 176.60000610351562, Loss = 177.75894165039062.
Iteration 10000 - Average Return = 162.6999969482422, Loss = 44.767189025878906.
Iteration 11000 - Average Return = 183.6999969482422, Loss = 964.6004638671875.
Iteration 12000 - Average Return = 197.3000030517578, Loss = 154.2781372070125.
Iteration 13000 - Average Return = 197.10000610351562, Loss = 1284.9034423828125.
Iteration 14000 - Average Return = 198.8000030517578, Loss = 71.5666732788086.
Iteration 15000 - Average Return = 199.8000030517578, Loss = 47.69195556640625.
Iteration 16000 - Average Return = 200.0, Loss = 63.23650360107422.
Solved in 16000 iterations!
```



The training took approximatly 25 min. on my computer (I7, 1 GPU).

A video showing the trained agent over 5 episodes was created. You can compare with an agent trained with a random policy (the videos open in YouTube):

trained cartpole (5 episodes)	random policy (5 episodes)
	

Conclusions:

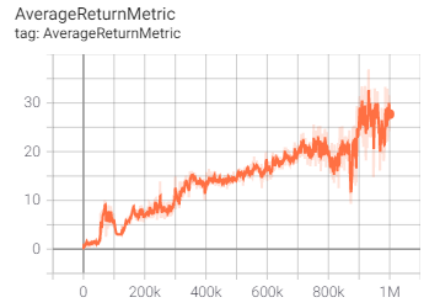
1. A DQN can be trained successfully for the cartpole environment
2. Using a RL framework like TF-Agents simplifies greatly the code. Of course, there is a learning curve...

Solving Atari Breakout with TF-Agents DQN

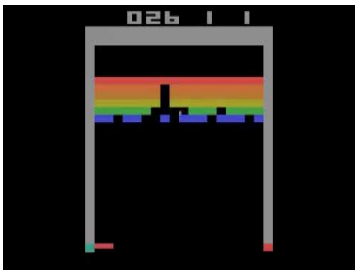
This example uses TF-Agents to train an agent to play Breakout, the famous Atari game, using the DQN algorithm. I used the OpenAI Gym Breakout-v0 environment (https://gym.openai.com/envs/Breakout-v0/). The code is based on the cartpole example above. But this time, I used a CNN Network to train the agent as the observations of the environment are screenshots of the Atari screen.

The Atari Breakout environment and its DQN training with TF-Agents are described in great details in Aurélien Géron's Book: "Hands-On Machine Learning with Scikit-Learn, Keras and tensorflow, 2nd Edition". Its github repository is https://github.com/ageron/handson-ml2/blob/master/18_reinforcement_learning.ipynb

Training an Atari breakout agent requires much more ressources thant the cartpole environment. A. Géron suggests 10E7 steps. On my machine, it would require 8 days... So I have limited the training to 10E6 steps. After 16 hours of calculation, the average return approaches 30 which is still far from 200 that can be obtained with more ressources:



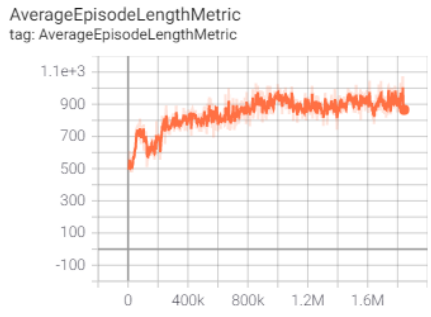
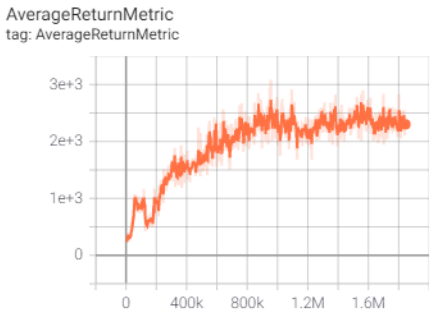
But I consider it is not bad because it validates the algorithm and we can see already a big improvement over a random policy:



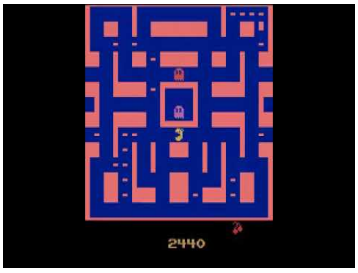
Solving Pacman with TF-Agents DQN

Finally: Pacman. Well, it was much easier than I though thanks to the Breakout project I have done before. I have reused the code and changed the name of the OpenAI Gym environment to MsPacman-v0 (actually to MsPacmanNoFrameskip-v0 because the default Atari environment applies random frame skipping and max pooling and we must train on the raw, nonskipping variant).

Again, I was limited by the resources and trained Pacman over 1.8M steps on Google Cloud (see Appendix 2). Note that A. Geron recommends 10M steps. An average return per episode of ~ 2300 is obtained.

Average Episode Length	Average Return
	

It's time for the video. Not perfect, but already better than what I could perform as a human agent:



The Next Journey

The 4 examples I have described above gave me a good understanding of how RL DQN can be implemented using the TF-Agents framework. There is of course still a lot to discover as the model is only part of what it takes to succeed with a machine learning project. As a software enginneer, I'd like to experiment more how the scripts that I have run on my personal latptop or on Google Cloud can be executed in an environment with multiple GPUs. Indeed, running training models during days is not viable outside the personal experimention or academic domain. In production, parallel processing distributed among multiple CPUs/GPUs are necessary to reduce the time from lab to market. For that, I believe that Dask and RAPIDS are the next things to learn.

Appendix 1 - Run a tf-agents script on Ubuntu 18

1. See https://www.tensorflow.org/install/pip#ubuntu-macos
2. pip3 install --upgrade numpy tf-agents
3. sudo apt-get install python3-matplotlib python-opencv
4. pip3 install gym>=0.17.3 atari-py

Appendix 2 - Run a tf-agents script on Google Cloud

1. Choose a machine with GPU and enough memory e.g. n1-highmem-8 (8 vCPUs, 52 GB memory) with 1 x NVIDIA Tesla P4
2. SSH
3. conda activate base
4. cd cd rl-dqn/
5. python3 ./breakout/main.py

A PDF of this readme page can be generated using grip