Extra Credit Project: Jupyter Experimentation

Steven Wirsz[1]

[1]California State University, Northridge

Due Date:

April 16, 2018 23:59 hours

Breakout: https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/16_Reinforcement_Learning.ipynb

Video: https://www.youtube.com/watch?v=Vz5l886eptw&t=0s

Exercises and Research Ideas

the breakout guide above in a Jupyter notebook. You will need to perform the training (which may require extensive CPU or GPU time) and test out (1) of the suggested exercises at the end of this document.

Unless you have a modern nVidia video card, training the model requires running your computer 24 hours a day for several days. There is an option to download pre-existing checkpoint files, but this requires finding and loading old copies of gym and atari-py. Record your results and describe your observations. **Print to PDF your Jupyter notebook before and after making changes**. Several of these exercises need retraining.

Last Revised: March 30, 2018

Installation recommended within an Ubuntu/Mint VM with 6-8 GB of memory:

1. Anaconda https://www.anaconda.com/download/
2. Download Github Tensorflow tutorials: https://github.com/Hvass-Labs/TensorFlow-Tutorials
3. From the folder where you extracted these files, load a bash shell and install the python packages:

```
conda create --name tensorflow pip python=3

sudo apt-get install zlib1g-dev cmake

pip install -r requirements.txt

pip install gym[atari]
```

To activate/deactivate the environment:

- source activate tensorflow
- jupyter notebook
- click on "http://localhost:8888/?token=..." link to load Jupyter notebook in a web browser. This may start automatically in certain environments.
- Browse to the folder where you extracted the github files and open "16_Reinforcement_Learning.ipynb"
- Shutdown procedure: ctrl-c followed by "source deactivate"

Training:

```
source activate tensorflow

python reinforcement_learning.py --env Breakout-v0 --training
```

First run through the Jupyter notebook step-by-step with the run command with the default values and ensure that the environment is working correctly. Training with Tensorflow/CPU will require **~150 hours of computations**. If you have a nVidia Geforce series 10 (1060/1070/1080) you can reduce this to only a few hours. (edit requirements.txt to install the GPU version of Tensorflow)

Note: The guide references the file "reinforcement-learning.py" in several places. This should be "reinforcement_learning.py"

Suggested exercise list: (identical to the list at the end of the breakout project)

1. Change the epsilon-probability during testing to e.g. 0.001 or 0.05. Which gives the best results? Could you use this value during training? Why/not?
2. Continue training the agent for the Breakout game using the downloaded checkpoint. Does the agent get better or worse the more you train it? Why? (You should run it in a terminal window as described above.)
3. Change the game-environment to Space Invaders and re-run this Notebook. The checkpoint can be downloaded automatically. It was trained for about 150 hours, which is roughly the same as for Breakout, but note that it has processed far fewer states. The reason is that the hyper-parameters such as the learning-rate were tuned for Breakout. Can you make some kind of adaptive learning-rate that would work better for both Breakout and Space Invaders? What about the other hyper-parameters? What about other games?
4. Use different architectures for the Neural Network. You will need to restart the training because the checkpoints cannot be reused for other architectures. You will need to train the agent for several days with each new architecture so as to properly assess its performance.
5. The replay-memory throws away all data after optimization of the Neural Network. Can you make it reuse the data somehow? The ReplayMemory-class has the function `estimate_all_q_values()` which may be helpful.
6. The reward is limited to -1 and 1 in the function `ReplayMemory.add()` so as to stabilize the training. This means the agent cannot distinguish between small and large rewards. Can you use batch normalization to fix this problem, so you can use the actual reward values?
7. Can you improve the training by adding L2-regularization or dropout?
8. Try using other optimizers for the Neural Network. Does it help with the training speed or stability?
9. Let the agent take up to 30 random actions at the beginning of each new episode. This is used in some research papers to further randomize the game-environment, so the agent cannot memorize the first sequence of actions.
10. Try and save the game at regular intervals. If the agent dies, then you can reload the last saved game. Would this help training the agent faster and better, because it does not need to play the game from the beginning?
11. There are some invalid actions available to the agent in OpenAI Gym. Does it improve the training if you only allow the valid actions from the game-environment?
12. Does the MotionTracer work for other games? Can you improve on the MotionTracer?
13. Try the last 4 image-frames from the game instead of the MotionTracer.
14. Try larger and smaller sizes for the replay memory.
15. Try larger and smaller discount rates for updating the Q-values.
16. If you look closely in the states and actions that are displayed above (in the breakout document), you will note that the agent has sometimes taken actions that do not correspond to the movement of the paddle. For example, the action might be LEFT but the paddle has either not moved at all, or it has moved right instead. Is this a bug in the source-code for this tutorial, or is it a bug in OpenAI Gym, or is it a bug in the underlying Atari Learning Environment? Does it matter?