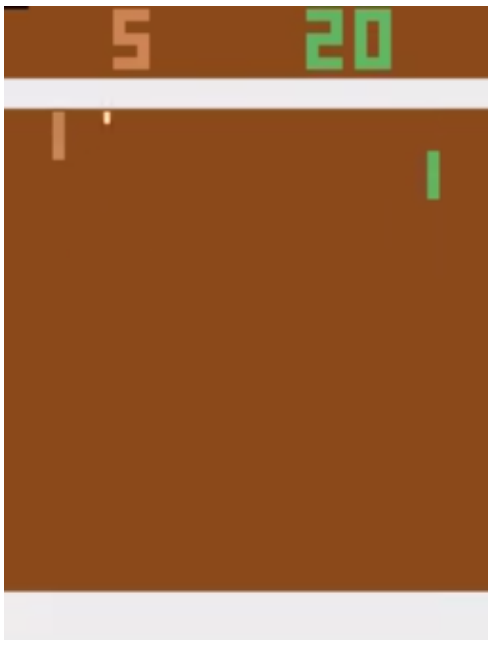
Week 4 - Policy Gradients on Atari Pong and Mujoco

Submitted by hollygrimm on Sat, 06/30/2018 - 09:50



The first part of my week was spent working on the 2nd homework for CS294, Policy Gradients[1]. Source code: https://github.com/hollygrimm/cs294-homework/tree/master/hw2

The Policy Gradients algorithm determines the optimal policy using a parameterized Neural Network (NN) instead of a value function or action function.

Policy Gradient training was done on OpenAI's Gym Environments: CartPole-v0 and InvertedPendulum-v0.

For training, a simple neural network with one fully connected layer of 32 units with a ReLU activation was created.

Discounted Rewards

Discounted rewards were calculated in two different ways.

Reward-to-go, the more state/action-centric method, calculates discounted rewards starting at time step t to the end of the trajectory:

```
rewards = [1, 10, 100]
gamma = .9
discounted_rewards_last_value = 100 * .9^0  = 100
discounted_rewards_2nd_value = 10 * .9^0  + 100 * .9^1  = 100
discounted_rewards_1st_value = 1 * .9^0  + 10 * .9^1  + 100 * .9^2  = 91
discounted_rewards = [91, 100, 100]
```

The second method, the trajectory-centric method, calculates discounted rewards for the entire trajectory and uses that value for all the timesteps:

```
rewards = [1, 10, 100]
gamma = .9
sum_discounted_rewards = 1 * .9^0  + 10 * .9^1  + 100 * .9^2  = 91
discounted_rewards = [91, 91, 91]
```

Sampling Actions

Although the neural network for both the discrete and the continuous policy networks were the same, the outputs were handled differently. In the discrete case, logits were returned. From the logits, a single action was then sampled from a multinomial distribution.

The continuous case considered the output of the NN as the mean of a distribution and with the addition of a second trainable variable, the log of the standard deviation (logstd), the actions could be sampled from a normal distribution.

Computing Loss

The discrete policy network used sparse_softmax_cross_entropy_with_logits to compute the log probability between the logits and the actions that were actually taken during the trajectory.

The continuous policy network created a Multivariate Normal Distribution using the mean and scaled by the logstd. The actions taken during the trajectory were then passed into the log probability density/mass function.

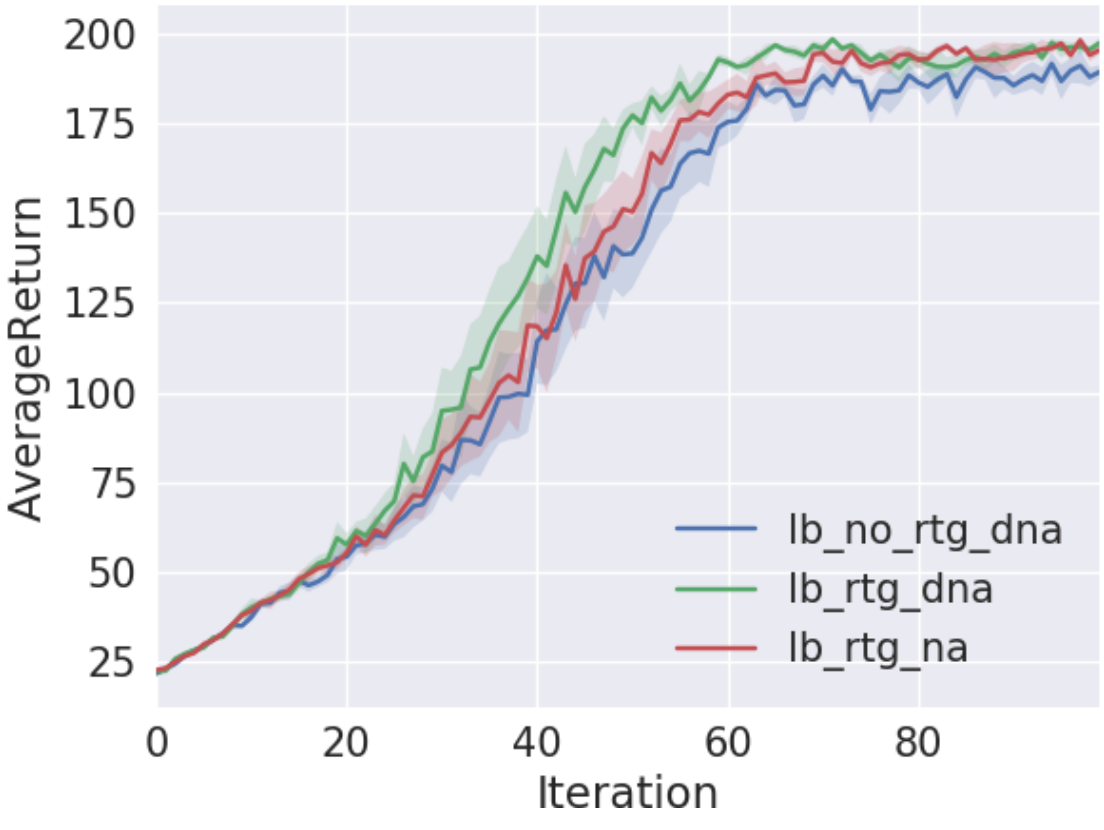
The resulting log probability is then multiplied by the advantages and reduced by the mean to calculate the loss.

Training

The Adam Optimizer was used for training with a learning rate of .005.

CartPole Results

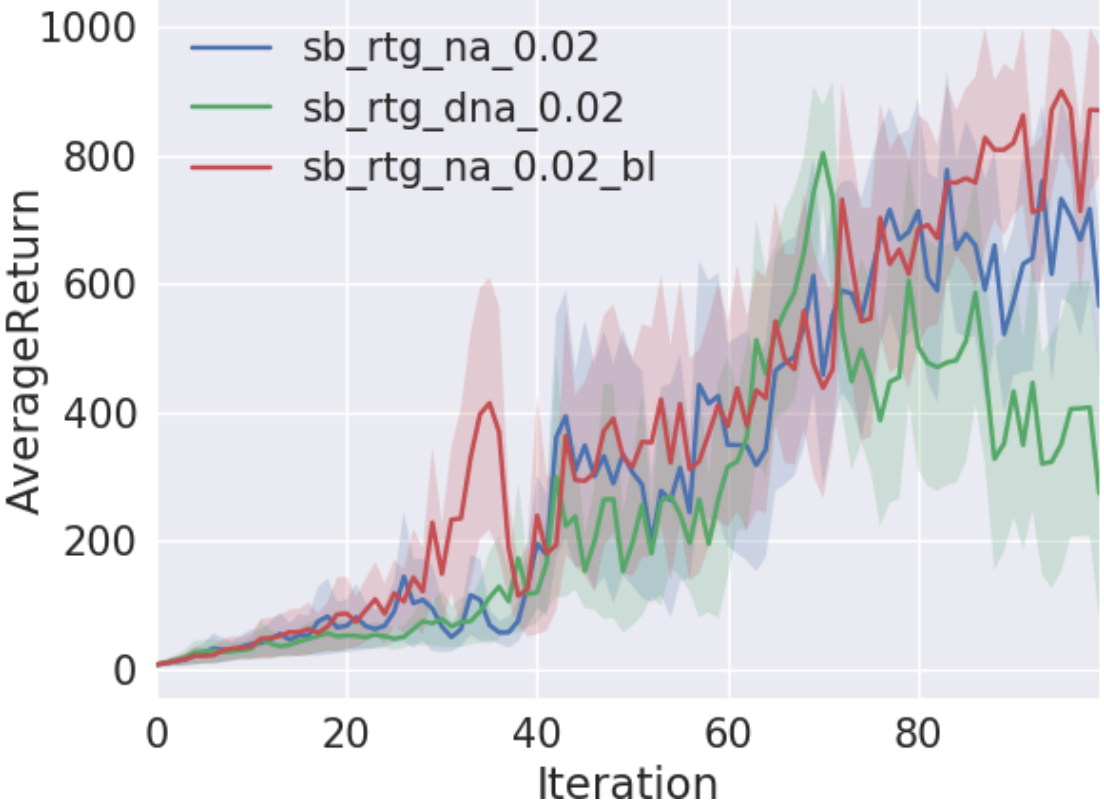
For CartPole, the average return reached close to 200 after about 60 iterations with a large batch size of 5000. The runs done without normalizing advantages (dna) did slightly better than the ones with normalized advantages (na). Reward-to-go (rtg) converged faster than the no-rtg training.



Here is a video of CartPole, trained large batch, reward-to-go, and without normalizing advantages:

0:00 / 0:04

InvertedPendulum Results



Inverted Pendulum did better with small batches of 1000 episodes and by changing the learning rate from the default of .005 to .02. The best results came from reward-to-go, normalized advantages, and a baseline computation.

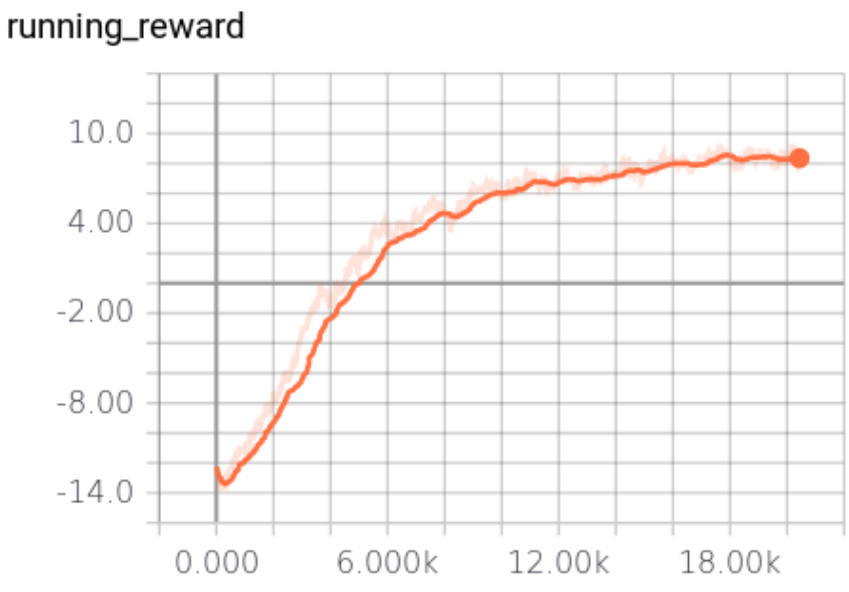
0:00 / 0:40

Pong From Pixels

After completing the homework, I thought I would apply the same algorithm to OpenAI's implementation of Atari 2600 Pong. Andrej Karpathy has written a blog post on how he used the Policy Gradients algorithm to learn how to play Pong [3] [4]. His implementation had computations that were entirely Numpy-based [5]. My version used Tensorflow.

Training for two days on an AWS EC2 p2.xlarge instance resulted in the following rewards. The first chart is the first 300 epochs:

The next chart shows the rewards from epoch number 300 to 2300:

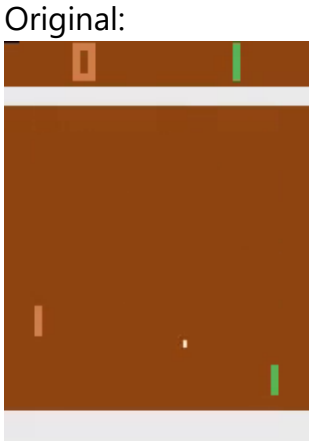


The orange paddle is the hard-coded AI opponent. The green is the agent trained using policy gradients. When the ball goes past the opponent, a reward of +1 is obtained, and if the ball goes past the agent it results in a reward of -1. A score of -21 happens when the opponent wins every game in an episode, +21 is the result when the trained agent wins all the games.

Average human performance is -3 [6], and super human performance is considered +5. My training maxed out with an average reward of +8.

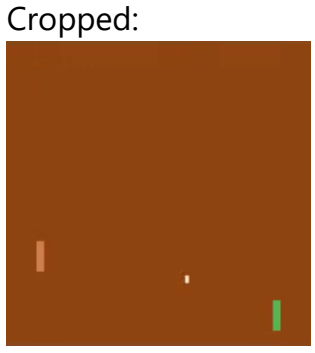
The network is one fully connected layer of 200 units with a ReLU activation. One frame from the Pong Env looks like this:

Original:



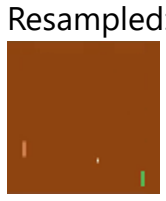
->

Cropped:



->

Resampled:



->

B&W:

The frame is first cropped to only the playing area (160x160). Then the frame is resized by half (80x80). Finally, the background is set to black, and the ball and paddles are set to white. The pixels are flattened into a (6400 x 1) line of pixels. The previous frame is then subtracted from this frame to train the network on motion between two frames. This difference frame is what is input to the network.

I trained the network to have three output actions: NO-OP (No Operation), UP, and DOWN. The environment has a certain amount of jitter built in, where a requested action is repeated between 2 to 4 frames [7]. I thought by adding NO-OP, that the agent could pause in a location if needed.

Here is an example game after training reward-to-go, normalizing advantages, gamma as .99, batch size as 10, and learning rate .001, for 2,300 epochs. The agent received an average reward of +8:

Pong (Atari) Gym Environment on AWS

When installing the OpenAI Gym environment for Atari on an EC2 instance, when running either:

```
pip install -e '.[atari]'  (from gym source)
```

or

```
pip install 'gym[atari]'
```

I had the following error:

```
/tmp/pip-install-gjxnkivl/atari-py/atari_py/ale_interface/src/common/ScreenExporter.cpp:18:10: fatal error:
zlib.h: No such file or directory
#include <zlib.h>
```

The AWS instance had all the required packages installed, such as cmake and zlib1g-dev, but was still unable to find the zlib.h file. I found this issue: https://github.com/openai/atari-py/issues/24 which shed light on the problem. The default ubuntu user on an EC2 instance can't find the zlib.h file due to permissions. Running the command as root worked:

```
sudo pip install -e '.[atari]'
```

As a workaround, I had to set up my OpenAI Gym Atari environment as root.

References

1. Sergey Levine. “Policy Gradients Introduction”. CS294 Fall 2017 Course at Berkeley. Video
2. Pieter Abbeel. “Lecture 4a Policy Gradients and Actor Critic”. Deep RL Bootcamp. Video
3. Andrej Karpathy. “Lecture 4b Pong from Pixels”. Deep RL Bootcamp. Video
4. Andrej Karpathy. “Deep Reinforcement Learning: Pong from Pixels”. Blog post
5. Andrej Karpathy. “Training a Neural Network ATARI Pong agent with Policy Gradients from raw pixels”. Github Gist
6. Volodymyr Mnih et al. “Playing Atari With Deep Reinforcement Learning”. In: NIPS Deep Learning Workshop. 2013
7. OpenAI Gym. “Pong-v0”. Web page

Tags
Reinforcement Learning OpenAI Policy Gradients Mujoco Atari Pong