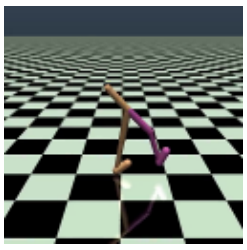
Week 3 - Imitiation Learning and Mujoco

Submitted by hollygrimm on Fri, 06/22/2018 - 16:16



This week I worked on *Homework 1: Imitation Learning* from the Fall 2017 CS294 course at Berkeley. Professor Levine is an amazing lecturer and the information he covers in one lecture is quite dense.

Imitation Learning is a form of Supervised machine learning for behavior. For this exercise, we were supplied with expert policies for six different OpenAI Gym Mujoco environments. Each environment has different observation and action spaces:

Ant: 11 observations and 8 actions

HalfCheetah: 17 observations and 6 actions

Hopper: 11 observations and 3 actions

Humanoid: 376 observations and 17 actions

Reacher: 11 observations and 2 actions

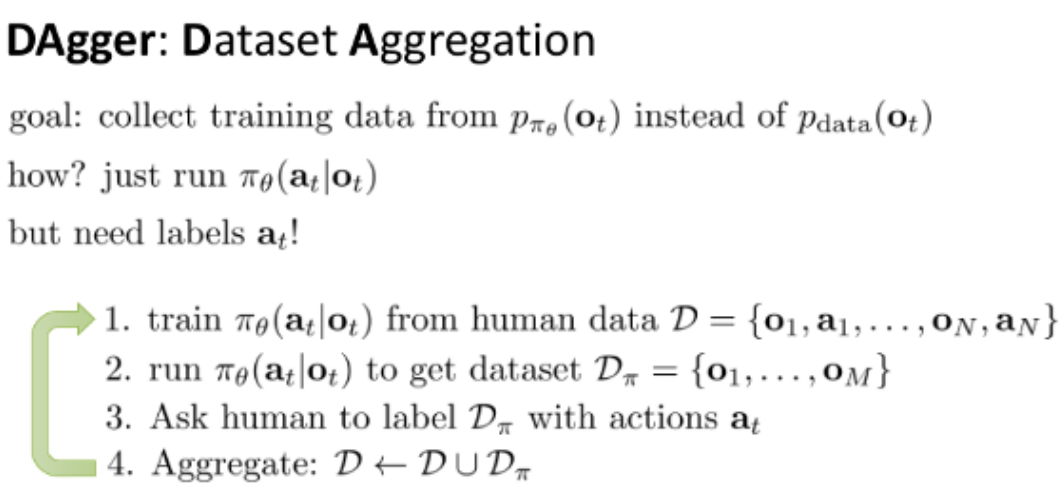
Walker2d: 11 observations and 2 actions

The task was to train a Neural Network on these expert policies (Behavioral Cloning), compare it to the expert results, and, lastly, enhance the Neural Network with an additional aggregation step (DAgger).

A simple Neural Network with non-linear activations is typically used, although a RNN can be deployed for non-Markovian tasks where behavior is dependent on all previous observations, instead of just the current observation.

The input to the network is an observation and the output is an action. Loss is calculated using the mean squared error between the predicted actions and expert actions.

The DAgger algorithm adds an additional step, where observations are generated from the trained policy, then passed to the expert policy for labeling with actions. This new experience is then aggregated into the dataset.



Above diagram from Sergey Levine's CS294 Lecture 2: Supervised Learning of Behaviors

Code

My code can be found here:

https://github.com/hollygrimm/cs294-homework/tree/master/hw1

Here are the dependencies:

- Mujoco version 1.3.1
- mujoco-py version 0.5.7
- OpenAI Gym commit 5f8d1fc1c6ea8a7dab44fcdab8a4ac1c24ba6759

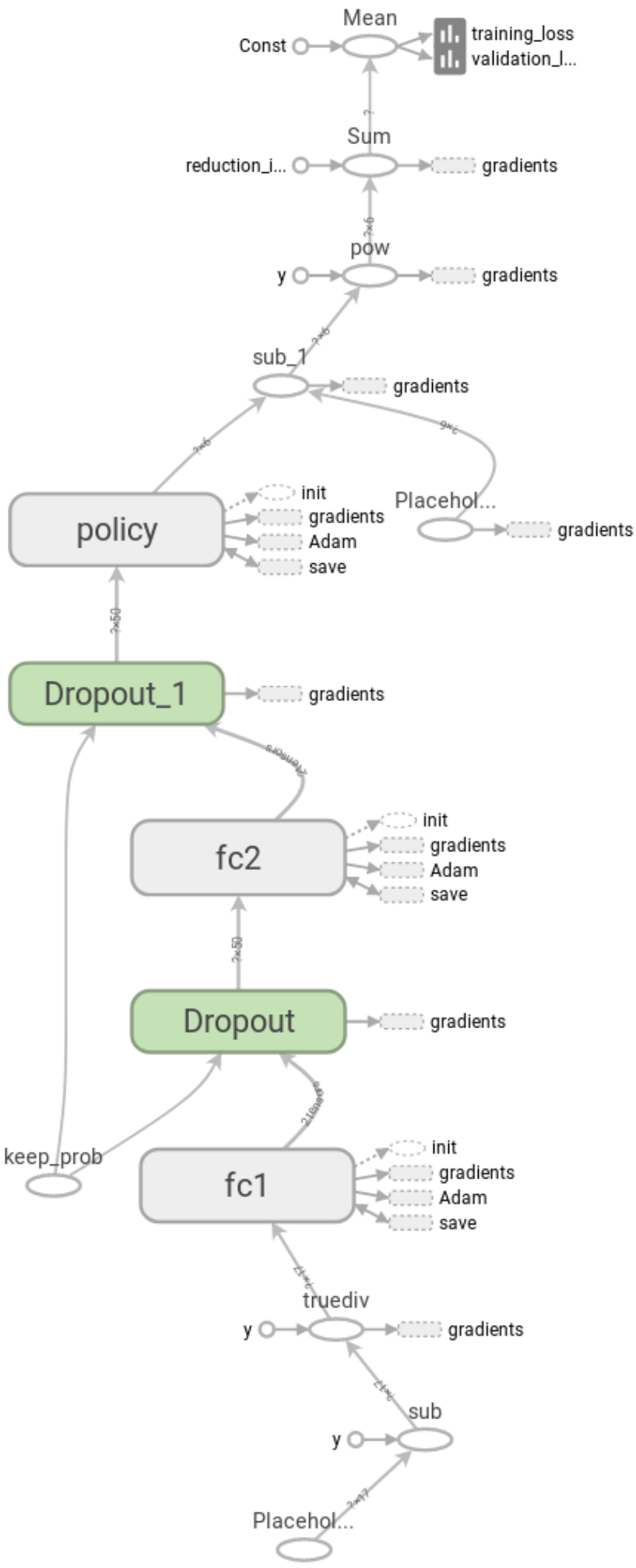
Generate Rollout Data

Before training, roll-out data must be generated from the expert policy files. A single roll-out is the result of a single episode executed until done or maximum timesteps are reached.

```
python
import run_expert
run_expert.generate_all_rollout_data()
```

Model

The network has two fully-connected layers with 50 units per layer, followed by a ReLU non-linearity. The observation data is normalized before training. I used a batch size of 32 and learning rate of .001. For behavioral cloning, I trained for 100 epochs and for DAgger, 40 epochs.



To train the model, run:

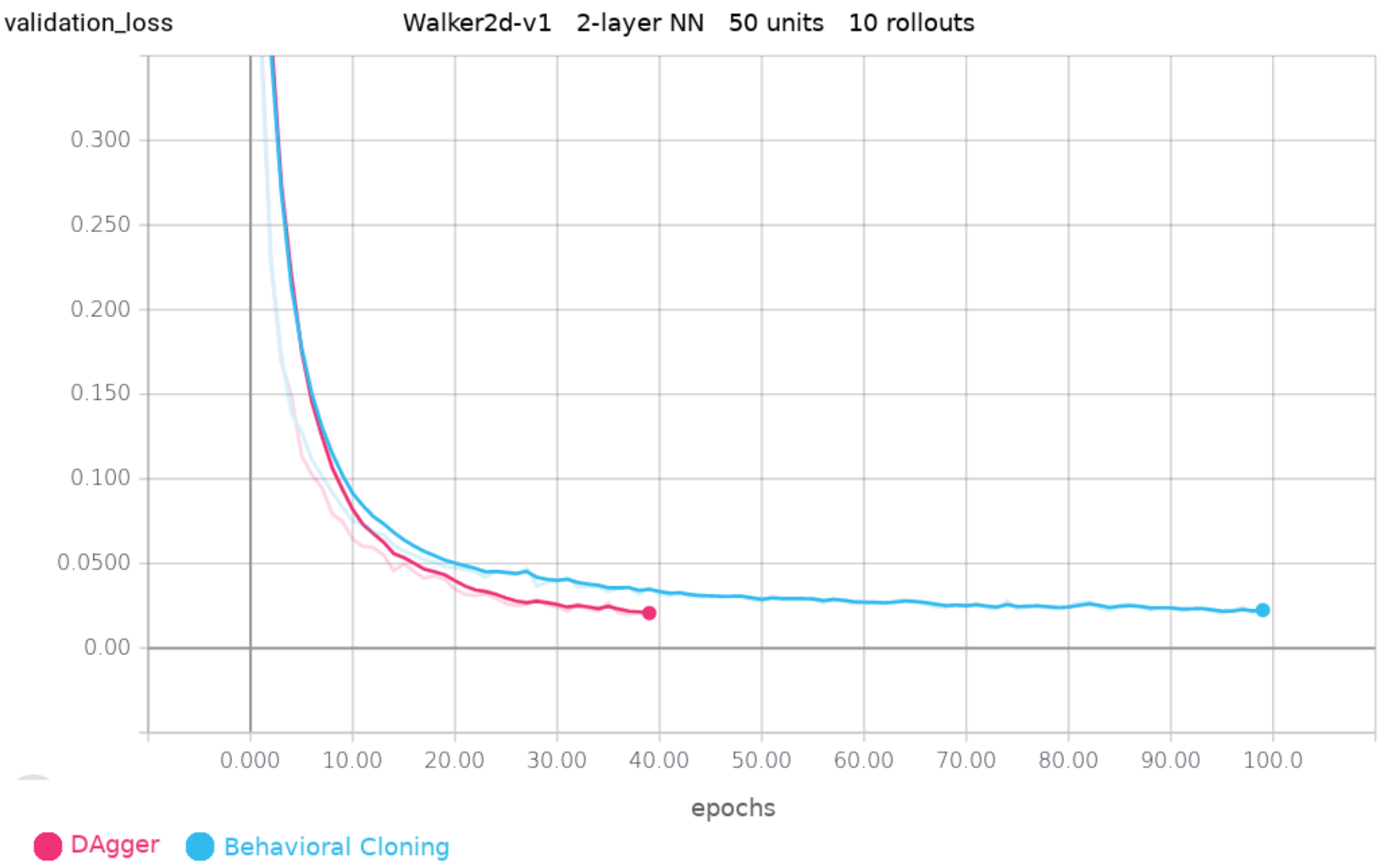
```
python bc.py
```

Training Results

Environment	Roll-outs	Expert Rewards	BC Rewards	DAgger Rewards
Ant-v1	250	4747 (459)	905 (1)	896 (2)
HalfCheetah-v1	10	4161 (69)	4197 (76)	4139 (57)
Hopper-v1	10	3780 (1)	3581 (598)	3775 (2)
Humanoid-v1	250	10402 (107)	354 (7)	385 (24)

Environment	Roll-outs	Expert Rewards	BC Rewards	DAgger Rewards
Reacher-v1	10	-3.8 (1)	-14 (5)	-12 (3)
Walker2d-v1	250	5513 (49)	4993 (1058)	5460 (132)

The standard deviation of the rewards is in parenthesis. I was able to get good results on HalfCheetah, Hopper, and Walker2d. Below shows the validation loss comparison between DAgger and Behavioral Cloning (BC). DAgger was able to train faster and better in 40 epochs than BC in 100 epochs on Walker2d.



0:00 / 0:05

Behavioral Cloning:

0:00 / 0:05

DAgger:

The Walker2d video of the DAgger version looks a little smoother than the BC version.

Tensorboard

To view loss charts after training execution, run:

```
tensorboard --logdir=results
```

Next Week:

Policy Gradients!

Tags

Reinforcement Learning OpenAI