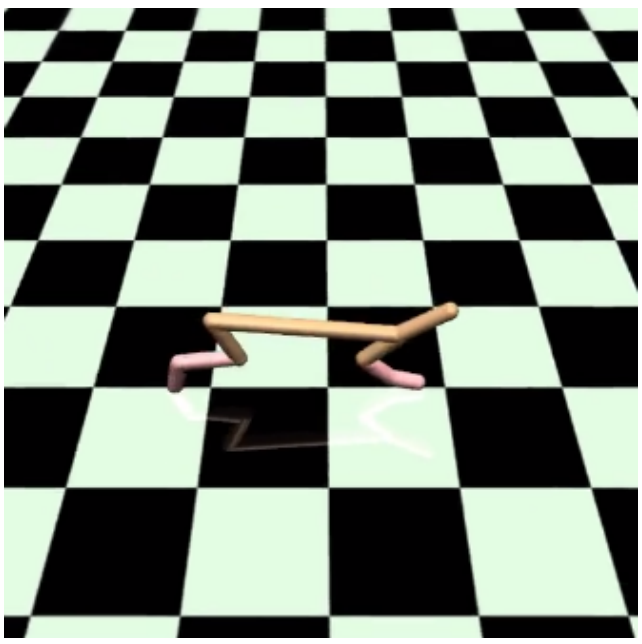
Week 6 - Model-based RL

Submitted by hollygrimm on Fri, 07/13/2018 - 14:06



Model Predictive Control and HalfCheetah

This week I learned about Model-based RL where a model of the dynamics of the environment is used to make predictions. Previous algorithms that I've studied have been model-free where a policy or value function is being optimized. Instead, Model-based RL predicts what the environment looks like, and it can create a model that is independent of the task you are trying to achieve. The dynamics model can be implemented using a Gaussian Process, a Neural Network, or other methods.

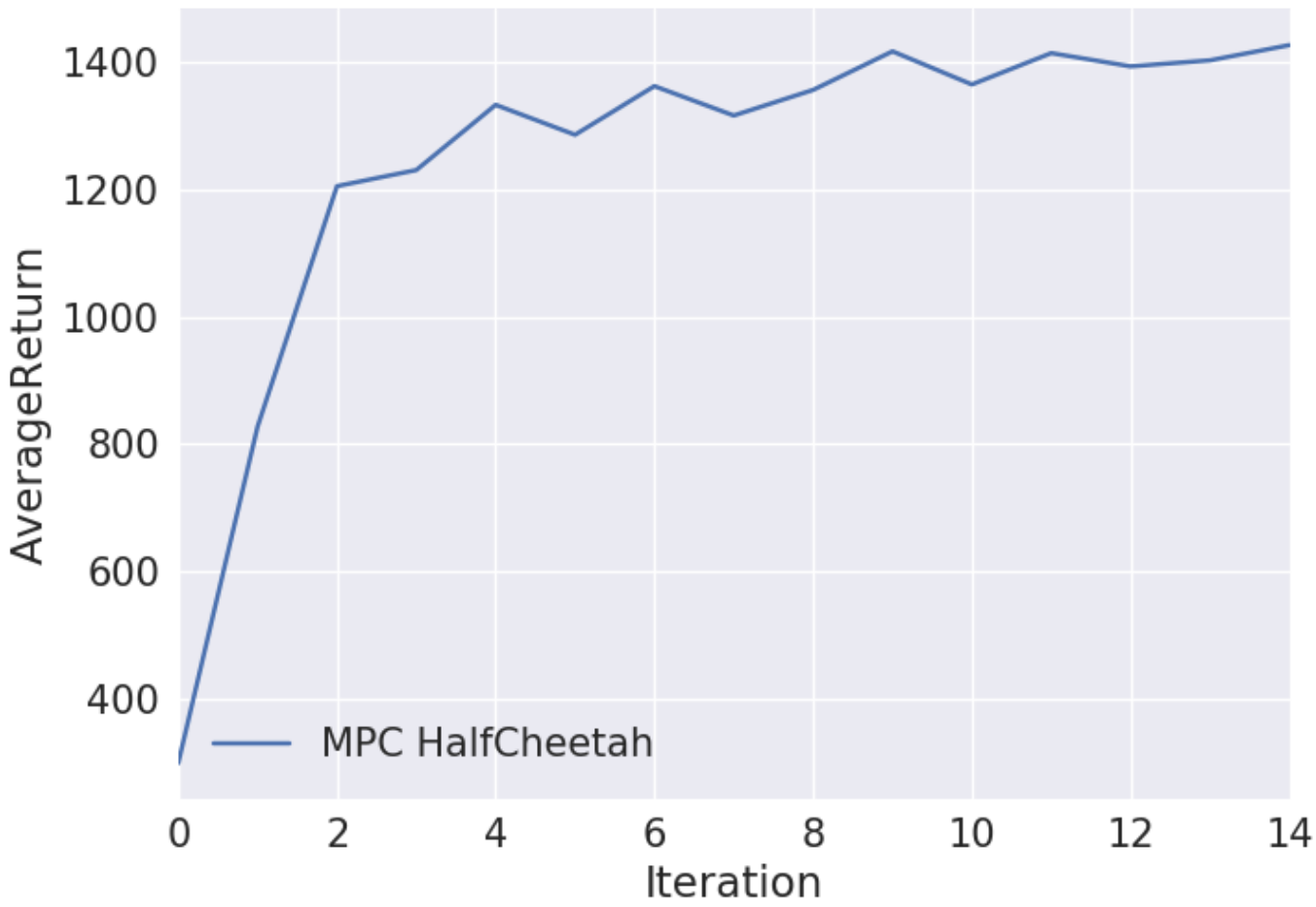
One advantage of model-based RL is that they require fewer samples to train compared to model-free. According to Sergey Levine [4], model-based RL requires 1000 times fewer samples than a model-free learner like A3C.

Code

I implemented the CS294 Homework 4 [7] to understand the interaction between the Neural Network, Model Predictive Control, and data aggregation. The environment was a MuJoCo HalfCheetah simulation in OpenAI Gym.

GitHub Code

Results



An average return of 1430 was reached after 15 iterations.

Final Video:

0:00 / 0:10

Modified HalfCheetahEnv

The homework code included a modified version of HalfCheetah, but it wasn't compatible with OpenAI Gym's wrappers. I wanted to use wrappers.Monitor to record video, so I had to rewrite the code to define a custom environment. I extended the HalfCheetahEnv, modified the frame skip from 5 to 1, and added an additional observation, self.get_body_com("torso").flat, to the _get_obs function:

```
class HalfCheetahTorsoEnv(HalfCheetahEnv, utils.EzPickle):
    def __init__(self, **kwargs):
        mujoco_env.MujocoEnv.__init__(self, kwargs["model_path"], 1)
        utils.EzPickle.__init__(self)

    def _get_obs(self):
        obs = np.concatenate([
            HalfCheetahEnv._get_obs(self),
            self.get_body_com("torso").flat,
        ])
        return obs
```

The new class adds the position of the HalfCheetah's torso to the observation data.

I registered the class inline, so I could adjust the max_episode_steps with a command line argument:

```
    register(
        id='HalfCheetahTorso-v1',
        entry_point='cheetah_env2:HalfCheetahTorsoEnv',
        reward_threshold=4800.0,
        max_episode_steps=args.ep_len,
        kwargs= dict(model_path=os.path.dirname(gym.envs.mujoco.__file__) + "/assets/half_cheetah.xml")
    )
```

Collect Base Data

The first step is to initialize a dataset of trajectories by running a random policy. Here is a video of the HalfCheetah while the data is being collected:

Video:

0:00 / 0:10

Neural Network Dynamics Model

The paper [1] uses a neural network for the dynamics model with two fully-connected layers of 500 units each and a ReLU activation. When fitting the dynamics model, normalized state and action pairs are input, and the state differences (or deltas) between the input state and next state are output. By predicting a change in state, rather than just the next state, the dynamics model can predict over several timesteps instead of just one timestep.

The mean squared error between the predicted and expected state deltas is minimized during training with the Adam optimizer.

Reward Function

The reward function was provided in the homework code, and was a combination of the location of the HalfCheetah's leg, shin, and foot position.

Model Predictive Controller

The Model Predictive Controller(MPC) selects an action for a particular state by first generating ten simulated paths each with a horizon of five actions into the future. The next state is predicted using the dynamics model. The candidate paths are evaluated using the reward function and the best performing trajectory is selected. The first action of that trajectory is then performed.

When ten samples are completed with the MPC, the new data is then aggregated into the dataset. This completes the first iteration. For the next iteration, the dynamics model is refitted to the new data, and new samples are again generated using the MPC.

References

1. Anusha Nagabandi et al. "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning". PDF
2. Chelsea Finn. "Deep RL Bootcamp Core Lecture 9 Model-based RL". Video | Slides
3. Sergey Levine. "CS294 Learning dynamical systems from data". Video | Slides
4. Sergey Levine. "CS294 Learning policies by imitating optimal controllers". Video | Slides
5. Chelsea Finn. "CS294 Advanced model learning and images" Video | Slides
6. Sergey Levine. "CS294 Connection between inference and control" Video | Slides
7. CS294 Model Based RL Project

Tags
Reinforcement Learning OpenAI Model-based RL