HW4

April 16, 2017

1 Homework 4

In `main.py` you will find an implementation of a "vanilla" policy gradient method, applied to an MDP with a discrete action space: an episodic version of the classic "cartpole" task. First, make sure the provided code works on your computer by running `python main.py`. We recommend reading through all of the code and comments in the function `main_cartpole`, starting at the top of the function.

The code computes some useful diagnostics, which you may find helpful to look at while tuning hyperparameters:

- **KL[policy before update || policy after update]**. Large spikes in KL divergence mean that the optimization took a large step, and sometimes these spikes cause a collapse in performance.
- **Entropy of the policy**. If entropy goes down too fast, then you may not explore enough, but if it goes down too slowly, you'll probably not reach optimal performance.
- **Explained variance of the value function**. If the value function perfectly explains the returns, then it will be 1; if you get a negative result, then it's worse than predicting a constant.

Software dependencies:

- tensorflow
- numpy + scipy (Anaconda recommended)
- gym (I'm using 0.8.0, `pip install gym==0.8.0`, but old versions should work just as well)

1.1 Problem 1

Here you will modify the `main_cartpole` policy gradient implementation to work on a continuous action space, specifically, the gym environment `Pendulum-v`. Note that in `main_cartpole`, note that the neural network outputs "logits" (i.e., log-probabilities plus-or-minus a constant) that specify a categorical distribution. On the other hand, for the pendulum task, your neural network should output the mean of a Gaussian distribution, a separate parameter vector to parameterize the log standard deviation. For example, you could use the following code:

```
mean_na = dense(h2, ac_dim, weight_init=normc_initializer(0.1)) # Mean control
logstd_a = tf.get_variable("logstdev", [ac_dim], initializer=tf.zeros_initializ
```

```
sy_sampled_ac = YOUR_CODE_HERE
sy_logprob_n = YOUR_CODE_HERE
```

You should also compute differential entropy (replacing `sy_ent`) and KL-divergence (`sy_kl`) for the Gaussian distribution.

The pendulum problem is slightly harder, and using a fixed stepsize does not work reliably—thus, we instead recommend using an adaptive stepsize, where you adjust it based on the KL divergence between the new and old policy. Code for this stepsize adaptation is provided.

You can plot your results using the script `plot_learning_curves.py` or your own plotting code.

Deliverables

- Show a plot with the pendulum converging to EpRewMean of at least `-300`. Include EpRewMean, KL, Entropy in your plots.

- Describe the hyperparameters used and how many timesteps your algorithm took to learn.

1.2 Problem 2

1. Implement a neural network value function with the same interface as `LinearVF`. Add it to the provided cartpole solver, and compare the performance of the linear and neural network value function (i.e., baseline).
2. Perform the same comparison–linear vs neural network–for your pendulum solver from Problem 1. You should be able to obtain faster learning using the neural network.

Deliverables

- A comparison of linear vs neural network value function on the cartpole. Show the value function's explained variance (EVBefore) and mean episode reward (EpRewMean).
- A comparison of linear vs neural network value function on the pendulum. Show the value function's explained variance (EVBefore) and mean episode reward (EpRewMean).

In both cases, list the hyperparameters used for neural network training.

1.3 Problem 3 (bonus)

Implement a more advanced policy gradient method from lecture (such as TRPO, or the advantage function estimator used in A3C or generalized advantage estimation), and apply it to the gym environment `Hopper-v1`. See if you can learn a good gait in less than 500,000 timesteps. Hint: it may help to standardize your inputs using a running estimate of mean and standard deviation.

```
ob_rescaled = (ob_raw - mean) / (stdev + epsilon)
```

Deliverables

A description of what you implemented, and learning curves on the Hopper-v1 environment.