Policy Gradient Demo

Rishikesh Vaishnav

June 30, 2018

Monte Carlo Implementation

Code

- The code for this project is available at: `https://github.com/rish987/Reinforcement-Learning/blob/master/demos/policy_gradient/code/policy_gradient.py`.

Implementation Details

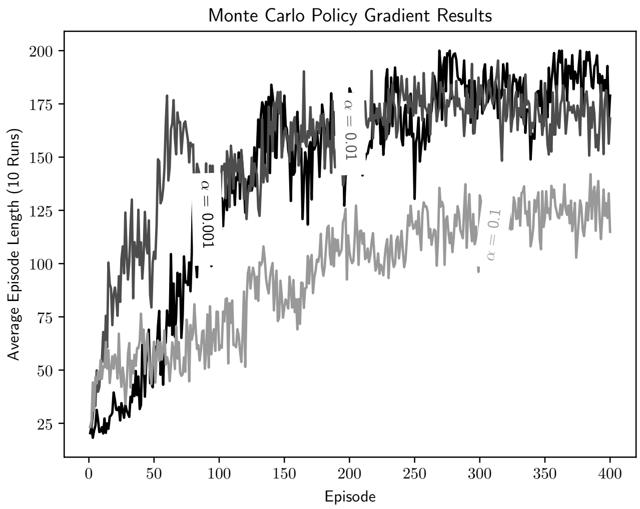
- Each state-action pair is converted to the feature vector $x(s, a)$. Letting S_{obs} and S_{act} be the size of the observation and action spaces, respectively, the size of the vector is $S_{obs} \times S_{act}$, where all features are 0 except for the S_{obs} features starting at index $S_{obs} \times a$, which are set to the environment's parameterization of s.
  - In this case, $S_{obs} = 4$ and $S_{act} = 2$.
- The policy function $\pi(a|s, \theta)$ performs the softmax on a parameterized linear mapping of feature vectors:

$$\pi(a|s,\theta) = \frac{e^{\theta^T x(s,a)}}{\sum_b e^{\theta^T x(s,b)}}$$

- The gradient of this policy function, used in the parameter update, was found to be:

$$\nabla\pi(a|s,\theta) = \frac{e^{\theta^T x(s,a)}}{(\sum_b e^{\theta^T x(s,b)})^2} \left(x(s,a) \sum_b e^{\theta^T x(s,b)} - \left(\sum_b e^{\theta^T x(s,b)} x(s,b) \right) \right)$$

Results



- The results can be summarized as follows:
  - The largest learning rate initiates learning quickly but fails to converge to an optimal policy, likely because it overshoots the mark at each parameter update.
  - The smallest learning rate learns the policy slowly because of its smaller updates but does reach a near-optimal policy.
  - The middle learning rate finds a near-optimal policy relatively quickly.