HW7
RL for Continuous-control

1 Gaussian Policies

For environments where the action-space is continuous, we use Gaussian policies instead of categorical policies (used in HW6). For example, in Pong, the possible actions are paddle-up/paddle-down/no-op, therefore a categorical policy of dimension 3 suffices. For a self-driving car, where the action is the angle of the steering wheel given the input state from a camera, infinite number of action values are possible in a range $[-B, B]$. In such scenarios, we use Gaussian policies where the mean of the Gaussian is parameterized a neural network conditioned on the input state. The standard-deviation could be parameterized similarly, but in this assignment, we'll learn it as a parameter independent of the input state. Therefore,

$$\pi_{\theta,\sigma}(a_t|s_t) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2\sigma^2}(a_t - f_\theta(s_t))\right) \tag{1}$$

where f_θ is a neural network. In general, the action-space is D dimensional (e.g. D degrees of freedom for a robotics control task), leading to a Multivariate Gaussian Distribution with a D dimensional mean vector ($\bar{f_\theta}$) as the output of the neural network, and a DxD dimensional co-variance matrix (Σ). For the special case where $\Sigma = \text{diag}(\sigma_1^2, \sigma_2^2 \dots)$, the PDF of the Multivariate Gaussian can be written as product of independent terms [1].

The environment for this assignment has a 2 dimensional continuous action space [2]. Therefore, for the policy-mean, we'll use a neural network with a 2 dimensional output. The independent variances (σ_1^2, σ_2^2) would be learnt as parameters. The PDF is

$$\pi(\bar{a_t}|s_t) = \frac{1}{\sqrt{2\pi}\sigma_1}\exp\left(-\frac{1}{2\sigma_1^2}(\bar{a_t}^{(1)} - \bar{f_\theta}^{(1)})\right) . \frac{1}{\sqrt{2\pi}\sigma_2}\exp\left(-\frac{1}{2\sigma_2^2}(\bar{a_t}^{(2)} - \bar{f_\theta}^{(2)})\right) \tag{2}$$

where $\bar{x}^{(i)}$ is the i^{th} entry of the vector $\bar{x}$. Recall from the previous assignment the gradient used in the A3C algorithm:

$$\nabla_\theta \eta(\pi_\theta) \equiv E_\tau\left[\sum_{t=0}^{T-1}\nabla_\theta \log \pi(\bar{a_t}|s_t, \theta)A(s_t, \bar{a_t}) + \beta\sum_{t=0}^{T-1}\nabla_\theta \mathrm{H}(s_t|\theta)\right] \tag{3}$$

where the expectation is over the trajectory roll-outs (τ) using policy π, A is the advantage function, H is the policy entropy, T is the time horizon. β is the weight on the entropy term, and therefor controls the amount of exploration the policy performs.

2 Problems

We'll use the same GA3C code, but with minor modifications to make it work on a continuous control task - *LunarLanderContinuous*. Clone/Fork the code from `https://github.com/tgangwani/IE598_RL`. Proceed with the following steps:

- Code in a **policy mean network**. Note that unlike HW6, we don't have visual frames as inputs, but low-dimensional (8-D) input. Therefore, we don't need convolution layers. Build a two-hidden layer (64 hidden units each) network which outputs a 2 dimensional mean for the action vector. You can use helper functions in

"NetworkVP.py". The non-linearity for the hidden layers should be tanh; read the interpretation of the actions from [2] and choose an appropriate non-linearity for the output layer.

The standard-deviation (σ) parameters have been provided for you. Since σ is always positive, instead of σ, the parameters represent $\log \sigma$, and therefore can take any real value. Use an exponent to recover σ.

- Code in a **value network** to be used for advantage estimation. Build a two-hidden layer (64 hidden units each, tanh non-linearity) network which outputs a single value function for the input state. This network is trained using regression.
- Complete the function "def loglikelihood" which is the **policy log-likelihood** ($\log \pi(\bar{a_t}|s_t, \theta)$). You'll find Equation (2) helpful for this.
- Complete the function "def entropy" which is the **policy entropy**. Hint: derive the formula for entropy of a uni-variate Gaussian. Then add the entropy for the two independent action dimensions. This provides us with H in Equation (3).
- Run the code for 12 hours, and submit the learning curve (RScore vs. time).

3 Extra Credit

Complete the function "def kl" which is the **KL-divergence** between the old and new Gaussian policies. Hint: derive the formula for KL between two uni-variate Gaussians. Then add the KL for the two independent action dimensions. Calculate the KL-divergence between old and new policies after each policy update step. Make a plot with KL on y-axis and the update step number on x-axis, and submit.

References

[1] DO, C. B. http://cs229.stanford.edu/section/gaussians.pdf.

[2] OPENAI-GYM. https://gym.openai.com/envs/lunarlandercontinuous-v2/.