Week 2 Briefing

Making robots learn by trial and error

Jack Cashman

July 5, 2024

Plan for presentation

1. Entropy Regularisation in Diagonal Gaussian Policies
   - Motivation
   - Results
2. Beta distribution as a prior for continuous PPO policies
   - Overview
   - Results
   - Improvements

An Introduction to Entropy

- **Entropy** is a topic from statistical physics, and more recently information theory.
- For a random variable X with associated distribution $p(X)$, the entropy of X is defined as:

$$\mathbb{H}(X) := \mathbb{E}\left[-\log(p(X))\right]$$

- Maintaining higher entropy prevents convergence to a deterministic policy.
- This in turn (hopefully) prevents premature convergence to sub-optimal policies.

An Introduction to Entropy

- **Entropy** is a topic from statistical physics, and more recently information theory.
- For a random variable X with associated distribution $p(X)$, the entropy of X is defined as:

$$\mathbb{H}(X) := \mathbb{E}\left[-\log(p(X))\right]$$

- Maintaining higher entropy prevents convergence to a deterministic policy.
- This in turn (hopefully) prevents premature convergence to sub-optimal policies.

An Introduction to Entropy

- **Entropy** is a topic from statistical physics, and more recently information theory.
- For a random variable X with associated distribution $p(X)$, the entropy of X is defined as:

$$\mathbb{H}(X) := \mathbb{E}\left[-\log(p(X))\right]$$

- Maintaining higher entropy prevents convergence to a deterministic policy.
- This in turn (hopefully) prevents premature convergence to sub-optimal policies.

An Introduction to Entropy

- **Entropy** is a topic from statistical physics, and more recently information theory.
- For a random variable X with associated distribution $p(X)$, the entropy of X is defined as:

$$\mathbb{H}(X) := \mathbb{E}\left[-\log(p(X))\right]$$

- Maintaining higher entropy prevents convergence to a deterministic policy.
- This in turn (hopefully) prevents premature convergence to sub-optimal policies.

Entropy Based PPO

- In the vast majority of implementations, PPO assumes a d-dimensional diagonal Gaussian prior for the policy, $\pi_{\boldsymbol{\theta}}$.
- In this case, the entropy of the policy has closed form representation:

$$\mathbb{H}(\pi_{\boldsymbol{\theta}}) = \frac{d}{2}\left(1 + \log(2\pi)\right) + \frac{1}{2}\log\left(\prod_{i=1}^{d} \Sigma_{i,i}\right)$$

- So, we can just include this in our 'benefit' function, $J'(\pi_{\boldsymbol{\theta}}) = J(\pi_{\boldsymbol{\theta}}) + \kappa\mathbb{H}(\pi_{\boldsymbol{\theta}})$ for an appropriate weighting κ.
- Not all distributions have an analytical solution for the entropy, but PyTorch has a library to compute this.

Results

Tested with $\kappa = 0.05$ vs $\kappa = 0$ on 3 `mujoco` environments over $250,000$ time steps. Percentage improvement from transitioning from $\kappa = 0 \to \kappa = 0.05$ is shown on each.

	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5
`Cheetah`	27.22	-3.59	2.16	-38.20	24.50
`Walker`	33.16	-14.77	-17.27	-40.43	88.12
`Hopper`	71.39	37.70	47.32	4.07	-29.43

Whether or not entropy is helpful is probably problem dependent. This agrees with literature.

Overview - Beta Distribution

- A policy is meant to represent a distribution over the *actions*
- Normally, a (d-dimensional) Gaussian distribution is used as the prior for the policy, but this has support $\mathbb{R}^d$
- This is problematic, as the agents actions are often bounds e.g. A rotation would be an element of $[0, 2\pi)$
- This discrepancy can introduce an estimation bias on the end-points of the actions.
- Instead, we propose that we can use a d-dimensional Beta distribution (with support $[0, 1]^d$), and then use an affine mapping to map $[0, 1]^d$ into a bounded action space.

Affine mapping to bounded action space

Suppose that in a general problem, the agent has d decisions to make, where for $1 \leq i \leq d$, the action, a_i, takes values in $[a_{L_i}, a_{H_i}]$. Then, the action space, can be defined as:

$$\boldsymbol{\mathcal{A}} = \left\{ \begin{pmatrix} a_1 & \dots & a_d \end{pmatrix}^\intercal : a_{L_i} \leq a_i \leq a_{H_i}, \quad 1 \leq i \leq d \right\}$$

The affine mapping from $[0,1]^d$ to $\boldsymbol{\mathcal{A}}$, denoted $\mathbf{T}$ is:

$$\mathbf{T}(\mathbf{p}) = \left(\operatorname{diag} \begin{pmatrix} a_{H_1} - a_{L_1} & \cdots & a_{H_d} - a_{L_d} \end{pmatrix} \right) \mathbf{p} + \begin{pmatrix} a_{L_1} & \cdots & a_{H_d} \end{pmatrix}^\intercal$$

Provided that $a_{L_i} \neq a_{H_i}$ for every i, we have an invertible way of mapping samples from the probability distribution to actions, and vice-versa.

Results - Beta Distribution

- Originally, I used two NNs to predict $\log(\alpha)$ and $\log(\beta)$ for the $\text{Beta}(\alpha, \beta)$ distribution (as $\alpha, \beta > 0$).
- This was very numerically unstable, so instead I had to impose an upper-bound of 50 on these parameters.
- Percentage improvement compared to traditional Gaussian policy on 3 `mujoco` environments are shown below:

	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5
`Cheetah`	162.05	169.82	196.69	74.35	230.86
`Walker`	-54.97	-65.52	-77.58	-67.96	-74.64
`Hopper`	2.27	-27.77	-55.62	-69.90	-27.44

Analysis and Improvement

- Despite performing well on `Cheetah`, the agent often converges to prematurely to a sub optimal policy.
- Despite still learning a stochastic policy, the NN chooses large β and α so that the samples from the distribution are extremely close to the mean.
- To discourage this, we can implement entropy regularisation (like we did for the Gaussian policy)
- Furthermore, the algorithms performance increased when reducing the cap on the α and β parameters to 25.

Analysis and Improvement

- Despite performing well on `Cheetah`, the agent often converges to prematurely to a sub optimal policy.
- Despite still learning a stochastic policy, the NN chooses large β and α so that the samples from the distribution are extremely close to the mean.
- To discourage this, we can implement entropy regularisation (like we did for the Gaussian policy)
- Furthermore, the algorithms performance increased when reducing the cap on the α and β parameters to 25.

Analysis and Improvement

- Despite performing well on `Cheetah`, the agent often converges to prematurely to a sub optimal policy.
- Despite still learning a stochastic policy, the NN chooses large β and α so that the samples from the distribution are extremely close to the mean.
- To discourage this, we can implement entropy regularisation (like we did for the Gaussian policy)
- Furthermore, the algorithms performance increased when reducing the cap on the α and β parameters to 25.

Analysis and Improvement

- Despite performing well on `Cheetah`, the agent often converges to prematurely to a sub optimal policy.
- Despite still learning a stochastic policy, the NN chooses large β and α so that the samples from the distribution are extremely close to the mean.
- To discourage this, we can implement entropy regularisation (like we did for the Gaussian policy)
- Furthermore, the algorithms performance increased when reducing the cap on the α and β parameters to 25.

Results - Revisited

Performance on the `walker` environment was still poor, more hyper-parameter tuning is likely required.

Again, we run $250,000$ time-steps, and report the percentage improvement from transitioning from a Gaussian prior, to a Beta prior for the `Hopper` environment:

Beta Prior	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5
No Entropy Reg.	2.27	-27.77	-55.62	-69.90	-27.44
Entropy Reg.	-7.53	56.06	-34.75	-47.23	-26.67

Conclusion

- I've found that using a Beta prior for the policy is beneficial in certain environments, and facilitates faster learning.
- I've also found that reinforcement learning is very volatile to hyper-parameters, and noise so it is difficult to produce consistent results.
- Additionally, incorporating entropy into the objective function can boost algorithmic performance.

Conclusion

- I've found that using a Beta prior for the policy is beneficial in certain environments, and facilitates faster learning.
- I've also found that reinforcement learning is very volatile to hyper-parameters, and noise so it is difficult to produce consistent results.
- Additionally, incorporating entropy into the objective function can boost algorithmic performance.

Conclusion

- I've found that using a Beta prior for the policy is beneficial in certain environments, and facilitates faster learning.
- I've also found that reinforcement learning is very volatile to hyper-parameters, and noise so it is difficult to produce consistent results.
- Additionally, incorporating entropy into the objective function can boost algorithmic performance.