Reinforcement Learning Framed as Probabilistic Inference in Locomotion

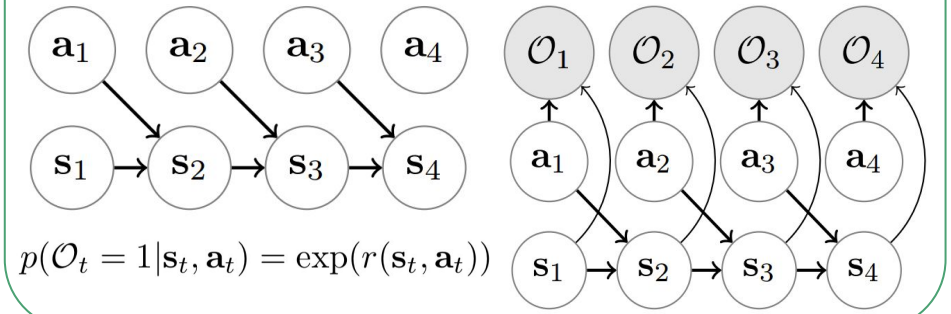
F. Abdolhosseini, A. Hajimoradlou, J. W. Lavington

Outline

- Background
- Maximum Entropy Policy Gradients
- SVI in Pyro
  - Implementing RL in Pyro
  - PPO in Pyro
- Results
- Discussion

RL as Probabilistic Inference

Graphical Model for Reinforcement Learning



$$p(\mathcal{O}_t = 1|\mathbf{s}_t, \mathbf{a}_t) = \exp(r(\mathbf{s}_t, \mathbf{a}_t))$$

Derivation of the Evidence Lower Bound

$$\begin{aligned}
\log p(\mathcal{O}_{1:T}) &= \log \int\int p(\mathcal{O}_{1:T}, \mathbf{s}_{1:T}, \mathbf{a}_{1:T}) d\mathbf{s}_{1:T} d\mathbf{a}_{1:T} \\
&= \log \int\int p(\mathcal{O}_{1:T}, \mathbf{s}_{1:T}, \mathbf{a}_{1:T}) \frac{q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T})}{q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T})} d\mathbf{s}_{1:T} d\mathbf{a}_{1:T} \\
&= \log E_{(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}) \sim q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T})} \left[\frac{p(\mathcal{O}_{1:T}, \mathbf{s}_{1:T}, \mathbf{a}_{1:T})}{q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T})} \right] \\
&\geq E_{(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}) \sim q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T})} \left[\log p(\mathcal{O}_{1:T}, \mathbf{s}_{1:T}, \mathbf{a}_{1:T}) - \log q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}) \right] \\
&\geq E_{(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}) \sim q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T})} \left[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) - \log q(\mathbf{a}_t|\mathbf{s}_t) \right]
\end{aligned}$$

Why would we want to cast this problem as a graphical model?

Because then we can utilize probabilistic programming, and amortized inference!

Why is variational inference necessary?

Because sampling from a hoppl under this framework does not recover a policy!

Max Entropy Reinforcement Learning

What does the maximum entropy objective represent?

The optimal agent under maximum entropy RL, is rewarded for exploration.

What is the point of maximum entropy?

Exploration is a key aspect of reinforcement learning, and is necessary for the agent to guarantee that its solution is near optimal.

Optimization Objective

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim q(s_t,a_t)} \left[(r(s_t,a_t) + \mathcal{H}(q_\theta(\cdot|s_t))) \right]$$

Gradient Derivation

$$\begin{aligned}
\nabla_\theta J(\theta) &= \sum_{t=1}^{T} \nabla_\theta E_{(\mathbf{s}_t,\mathbf{a}_t)\sim q(\mathbf{s}_t,\mathbf{a}_t)} \left[r(\mathbf{s}_t,\mathbf{a}_t) + \mathcal{H}(q_\theta(\mathbf{a}_t|\mathbf{s}_t)) \right] \\
&= \sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t)\sim q(\mathbf{s}_t,\mathbf{a}_t)} \left[\nabla_\theta \log q_\theta(\mathbf{a}_t|\mathbf{s}_t) \left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'},\mathbf{a}_{t'}) - \log q_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'}) - 1 \right) \right] \\
&= \sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t)\sim q(\mathbf{s}_t,\mathbf{a}_t)} \left[\nabla_\theta \log q_\theta(\mathbf{a}_t|\mathbf{s}_t) \left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'},\mathbf{a}_{t'}) - \log q_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'}) - b(\mathbf{s}_{t'}) \right) \right] \\
&= \sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t)\sim q(\mathbf{s}_t,\mathbf{a}_t)} \left[\nabla_\theta \log q_\theta(\mathbf{a}_t|\mathbf{s}_t) \hat{A}(\mathbf{s}_t,\mathbf{a}_t) \right]
\end{aligned}$$

A More Practical Perspective

Addition of a Temperature Coefficient

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot|\mathbf{s}_t)) \right]$$

We are not removing randomness, just reducing the incentive for performing actions that are unlikely under our current policy!

Critic Optimization

We want,

$$b(s_t) \approx \sum_{t'=t}^{T} r(a_{t'}, s_{t'}) - log(q(r(a_{t'}|s_{t'})))$$

So solve,

$$\min_{\theta} \quad \frac{1}{M * T} \sum_{i=1}^{M} \sum_{t=1}^{T} (b_\theta(s_{t,i}) - \sum_{t'=t}^{T} r(a_{t',i}, s_{t',i}) - \log q(a_{t',i}|s_{t',i}))^2$$

Addition of a Baseline

$$\hat{A}(s_t, a_t) = \left(\sum_{t'=t}^{T} r(s'_t, a'_t) - \log q_\theta(a_{t'}|s_{t'}) - b(s_t) \right)$$

Policy gradients has notoriously high variance, adding a clever baseline can mitigate this issue.

Discount factors

$$\hat{A}(s_t, a_t) = \left(\sum_{t'=t}^{T} \gamma^{t'-t} (r(s'_t, a'_t) - \log q_\theta(a_{t'}|s_{t'})) - b(s_t) \right)$$

The discount factor incentivises short term rewards, and clips the large swings in cumulative rewards that often happen for large trajectories.

Variations of of Maximum Entropy Policy Gradients

Natural Gradients (Similar to Newton's Method for SoEs)

$$\begin{array}{rcl} f_1(\theta_1,...,\theta_n) &=& 0 \\ f_2(\theta_1,...,\theta_n) &=& 0 \\ f_2(\theta_1,...,\theta_n) &=& 0 \\ ... &=& ... \\ f_n(\theta_1,...,\theta_n) &=& 0 \end{array} \qquad \begin{array}{rcl} \frac{\partial}{\partial\theta_1} f(\theta_1,...,\theta_n) &=& 0 \\ \frac{\partial}{\partial\theta_2} f(\theta_1,...,\theta_n) &=& 0 \\ \frac{\partial}{\partial\theta_3} f(\theta_1,...,\theta_n) &=& 0 \\ ... &=& ... \\ \frac{\partial}{\partial\theta_n} f(\theta_1,...,\theta_n) &=& 0 \end{array}$$

Say we want to solve for the roots of this system of equations

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \qquad \begin{bmatrix} \frac{\partial}{\partial x_1}\frac{\partial f}{\partial x_1} & \cdots & \frac{\partial}{\partial x_1}\frac{\partial f}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_n}\frac{\partial f}{\partial x_1} & \cdots & \frac{\partial}{\partial x_n}\frac{\partial f}{\partial x_n} \end{bmatrix}$$

Using first order taylor expansion we get the Hessian!

Derivation

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\mathbf{x}_0) + D\mathbf{f}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0).$$

$$\mathbf{f}(\mathbf{x}_0) + D\mathbf{f}(\mathbf{x}_0)(\mathbf{x}_1 - \mathbf{x}_0) = \mathbf{0}.$$

$$\mathbf{x}_1 = \mathbf{x}_0 - (D\mathbf{f}(\mathbf{x}_0))^{-1}\mathbf{f}(\mathbf{x}_0),$$

So let f be our objective, then we can perform optimization with near quadratic convergence using this method.

In our case we don't use the Hessian matrix, but the fisher information!

Variations of of Maximum Entropy Policy Gradients

Natural Policy Gradients

$$F_s(\theta) = \left[\frac{\partial log q(a|s,\theta)}{\partial \theta_i}\frac{log q(a|s,\theta)}{\partial \theta_j}\right]_{i,j\leq N}$$

$$F = \frac{1}{M*T}\sum_{i=1}^{M}\sum_{t=1}^{T}\nabla log(q_\theta)\nabla log(q_\theta)^T$$

$$\theta_{i+1} = \theta_i - \alpha F^{-1}\nabla\theta_i$$

Unfortunately, This method is:

- Computationally expensive
- Numerically unstable.
- Relies on smoothness assumptions of the function
- Only approximates (possibly very poorly) the Fisher Information.

Let's assume the gradient is a distributed as a MVN!

$$\nabla_\theta q_\theta(a_t|s_t) \sim MVN(\mu, \Sigma)$$

This means we can analytically determine the posterior given the likelihood:

$$\nabla_\theta q_\theta(a_t|s_t)|\tau \sim MVN(\hat{\mu}, \hat{\Sigma})$$

All the same caveats remain, but there are some advantages

- We can continuously add trajectory samples while updating the posterior gradient
- We recover a covariance matrix which might be useful for inferring stability of the gradient.

SVI in Pyro

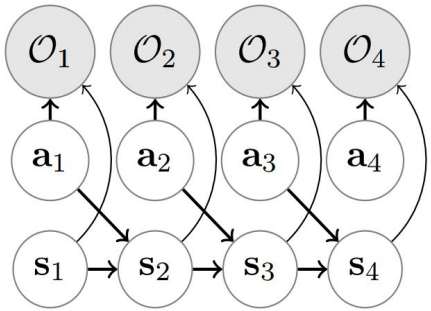
- Performs direct gradient descent on the ELBO
- Requires **`model`** and **`guide`** programs
- Handles mix of Pathwise Derivative and Score Function estimators
- Can specify conditional independencies (limited)
- Still a bit unstable (version 0.3)

SVI in Pyro - Requirements of Model and Guide

- Define parameters θ and ϕ
- Sample from the distribution $q_\phi(\boldsymbol{z})$
- Calculate $\log p_\theta(\boldsymbol{x}, \boldsymbol{z})$ and $\log q_\phi(\boldsymbol{z})$
- (Optional) Calculate $\nabla_\theta \log p_\theta(\boldsymbol{x}, \boldsymbol{z})$ and $\nabla_\phi \log q_\phi(\boldsymbol{z})$

$$\mathrm{ELBO}(\phi, \theta) = \mathbb{E}_{q_\phi(\boldsymbol{z})}\left[\log p_\theta(\boldsymbol{x}, \boldsymbol{z}) - \log q_\phi(\boldsymbol{z})\right]$$

```
def model(env, _):
    s = pyro.sample("s_1", env.InitDistribution)
    for i in range(1, env.T):
        a  = pyro.sample("a_%d" % i, Uniform)
        sp = pyro.sample("s_%d" % i+1, env.Transition(s, a))
        r  = env.Reward(s, a, sp)
        O_dist = pyro.dist.Bernoulli(exp(r))
        pyro.sample("O_%d" % i, O_dist, obs=1)
        s = sp

def guide(env, policy):
    pyro.module("policy", policy)
    s = pyro.sample("s_1", env.InitDistribution)
    for i in range(1, env.T):
        a  = pyro.sample("a_%d" % i, policy(s))
        sp = pyro.sample("s_%d" % i+1, env.Transition(s, a))
        s  = sp
```



Generalizing the Original Model

- Improvements:
  - Positive rewards
  - Variable episode length
  - Introducing randomness through the seed
  - “Infinite Uniform” distribution for actions
- Limits:
  - Discount factor
  - Sophisticated use of critic

Baselines in Pyro

- High Variance due to score function estimators
  - $\log q_\phi(\mathbf{z}_i)\overline{f_\phi(\mathbf{z})} \rightarrow \log q_\phi(\mathbf{z}_i)\left(\overline{f_\phi(\mathbf{z})} - b\right)$

$$\mathbb{E}_{q_\phi(\mathbf{z})}\left[\nabla_\phi(\log q_\phi(\mathbf{z}) \times b)\right] = 0$$

$$\mathbb{E}_{q_\phi(\mathbf{z})}\left[\nabla_\phi \log q_\phi(\mathbf{z})\right] = \int d\mathbf{z}\; q_\phi(\mathbf{z})\nabla_\phi \log q_\phi(\mathbf{z}) = \int d\mathbf{z}\; \nabla_\phi q_\phi(\mathbf{z}) = \nabla_\phi \int d\mathbf{z}\; q_\phi(\mathbf{z}) = \nabla_\phi 1 = 0$$

- Decaying Average Baseline
- Neural Network Baseline
  - Introducing baseline loss to adapt the parameters of the neural network $\left(\overline{f_\phi(\mathbf{z})} - b\right)^2$

Trust Region Methods in Pyro

- Limitations of Policy Gradients
  - Hard to choose the right step size
    - Non-stationary input data due to changing policy
    - bad step size leading to performance collapse
  - Sample Inefficient: one gradient update per a batch of trajectories

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim q_{\theta_{old}}(s_t,a_t)} \left[\frac{q_\theta(a_t|s_t)}{q_{\theta_{old}}(a_t|s_t)} \hat{A}(s_t,a_t) \right]$$

Trust Region Methods in Pyro

- Proximal Policy Optimization (PPO)
  - Adaptive KL penalty

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim q(s_t,a_t)} \left[\frac{q_\theta(a_t|s_t)}{q_{\theta_{old}}(a_t|s_t)} \hat{A}(s_t,a_t) \right] + \beta \mathbb{E}_{(s_t,a_t)\sim q(s_t,a_t)} \left[KL\left[q_\theta(.|s_t) || q_{\theta_{old}}(.|s_t) \right] \right]$$

  - Clipping

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim q(s_t,a_t)} \left[\min \left(\frac{q_\theta(a_t|s_t)}{q_{\theta_{old}}(a_t|s_t)} \hat{A}(s_t,a_t), clip(1-\epsilon, 1+\epsilon, \frac{q_\theta(a_t|s_t)}{q_{\theta_{old}}(a_t|s_t)}) \hat{A}(s_t,a_t) \right) \right]$$

Extending Pyro - PPO

- ELBO

$$\mathrm{E}_{q_\phi(z)}\Big[\log p_\theta(x,z) - \log q_\phi(z)\Big]$$

$$\hat{S} = \log p_\theta(x,z) - \log q_\phi(z)$$

- Surrogate Loss

$$\mathbb{E}_{q_\phi(z)}\Big[\log q_\phi(z)\hat{S}\Big]$$

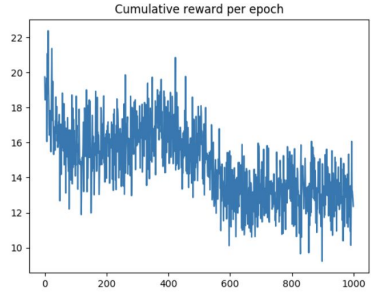
- Applying PPO

$$\min\left(\frac{q_\phi(z)}{q_{\phi_{old}}(z)}\hat{S}, clip\Big(1-\epsilon, 1+\epsilon, \frac{q_\phi(z)}{q_{\phi_{old}}(z)}\Big)\hat{S}\right)$$

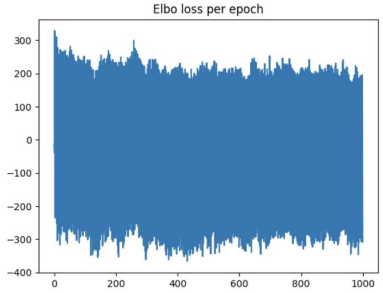
Extending Pyro - PPO

- Limited flexibility in structure of TraceGraph ELBO
- Not having access to the optimizer from TraceGraph ELBO
- Adding a custom loss function
- Difficulties of using a costum loss
  - Taking care of the gradients
  - Taking care Optimization steps

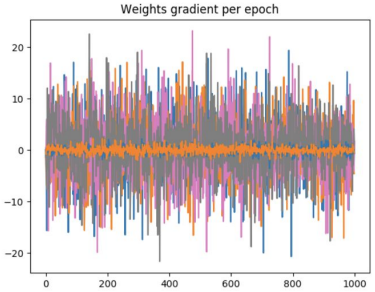
Results - Direct Optimization of Max Entropy



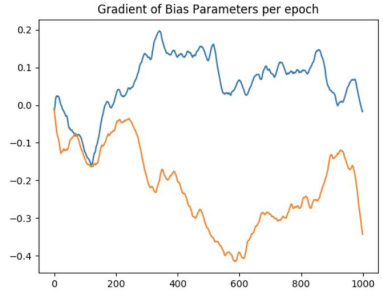
(a) Cumulative Reward Per Epoch



(b) Negative ELBO Loss Per Epoch



(c) Gradient of Weight Parameters Per Epoch



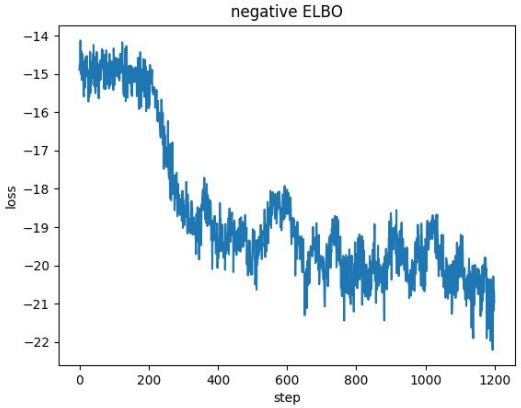
(d) Gradient of Bias Parameters Per Epoch

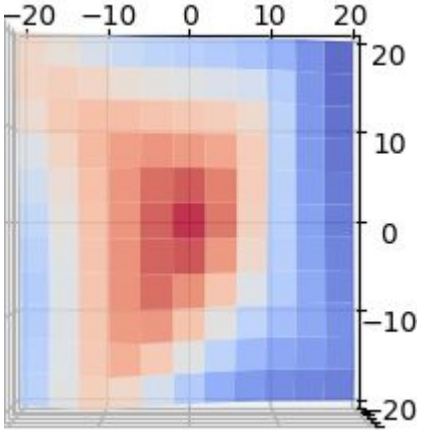
Figure 2: Maximum entropy Policy gradients with Critic applied to parameterization of multivariate normal distribution; in this case, variance is held constant.

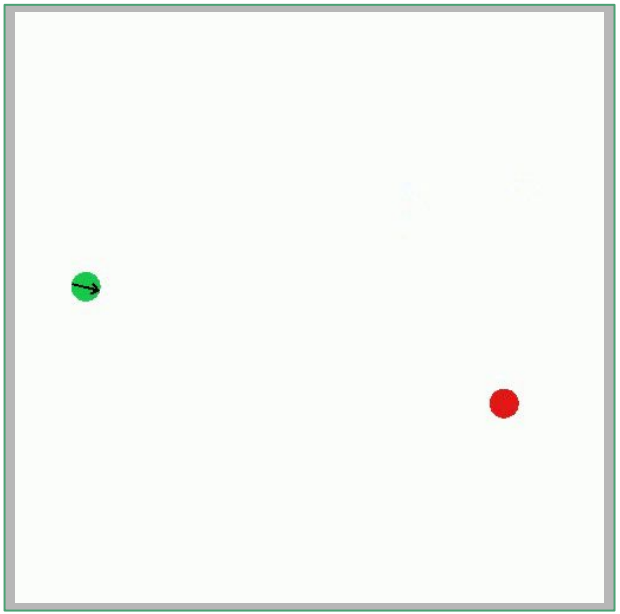
Results - SVI in Pyro

- Takes 20K steps (!) to solve supervised learning in 2D
- Takes 3M steps to solve a simplified environment

$$r = -\left(\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \begin{bmatrix} 1 \\ -1 \end{bmatrix}\right)^2$$







Discussion

- Ease of use
  - Lines of code
  - vs existing libraries
  - Mental burden
- Performance
  - Needs more work
- Incorporating Data/Knowledge
- Future advances
- Better conceptual understanding

Thanks!

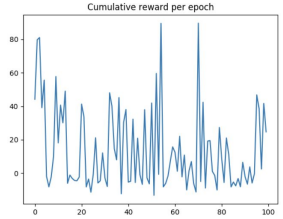
Results - SVI Tests

- Finding fixed direction with MSE (linear)
  - 15K samples, 1 particle
- State-dependent direction with MSE (linear)
  - 30K samples, 10 particles
- State-dependent direction with MSE (neural network)
  - 50K samples, 100 particles
- Multiple Actions with MSE (neural network)
  - 2M samples, 100-200 particles (requires TraceGraph_ELBO)
- Multiple Actions with MSE (NN + baseline)
  - 500K samples, 100-200 particles

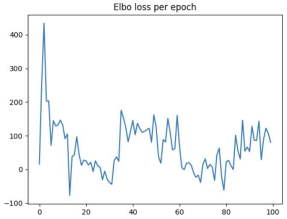
```
direction = [1, -1]

def rl_model():
    a = pyro.sample("a", dist.Uniform(-100 * th.ones(2), 100 * th.ones(2)))
    # reward is the distance to the correct direction
    r = -1 * (a - direction).pow(2).sum()
    O_dist = FlexibleBernoulli(exp(r.detach()))
    pyro.sample("O", O_dist, obs=1)

def rl_linear_guide():
    W = pyro.param("W", th.randn(2)) # linear model
    a = pyro.sample("a", dist.Normal(W, .5))
```

Parameterizing Variance

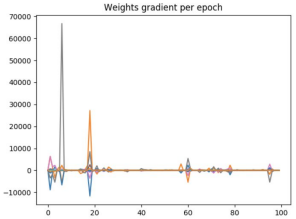
Including variance in the parameterization leads to extremely unstable swings in the gradient, and makes any optimization without an unreasonable number of samples impossible.



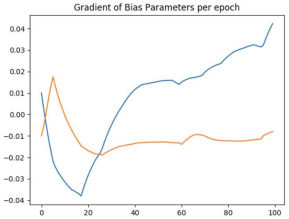
(a) Cumulative Reward Per Epoch



(b) Negative ELBO Loss Per Epoch



(c) Gradient of Weight Parameters Per Epoch



(d) Gradient of Bias Parameters Per Epoch

Figure 3: Maximum entropy Policy gradients with Critic applied to parameterization of multivariate normal distribution; in this case, variance is allowed to vary as part of the parameterization. In this case, a different reward was used to ensure better convergence, and a smaller set of epochs was due to computational constraints.