From Scratch Implementation of Deep Reinforcement Learning Algorithms for Continuous Control

Ruturaj Sambhus

Virginia Tech

December 1, 2022

Overview

1. Theory
   - Policy Gradient Algorithms - REINFORCE
   - Actor Critic Algorithms - A2C
2. Implementation Details
   - Framework
   - Neural Networks
3. Results
   - Gym and PyBullet Results
   - Sensitivity to Hyper-parameters

Class of Reinforcement Learning Methods

Value Based Methods

- The value function (expected return) of the state space is learned
- This is used for discrete state and action spaces
- Algorithms - Deep Q Learning (DQN), Dueling DQN

Policy Gradients

- The policy (action given state) for state space is learned
- Can be used for continuous state and action spaces
- Algorithms - REINFORCE or Vanilla Policy Gradient, TRPO, PPO

Actor Critic

- Combination of both value-based and policy gradient methods
- Can be used for continuous state and action spaces
- Algorithms - Advantage Actor Critic (A2C), A3C, DDPG, SAC

REINFORCE - Vanilla Policy Gradient

REINFORCE with Baseline (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

Figure: REINFORCE with Baseline [Sutton and Barto, 2018]

A2C - Advantage Actor Critic

One-step Actor–Critic (episodic), for estimating $\pi_{\theta} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize S (first state of episode)
    $I \leftarrow 1$
    Loop while S is not terminal (for each time step):
        $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
        Take action A, observe S', R
        $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\,\delta\nabla\hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I\,\delta\nabla\ln\pi(A|S,\boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$

Figure: One Step A2C [Sutton and Barto, 2018]

Way Things Move

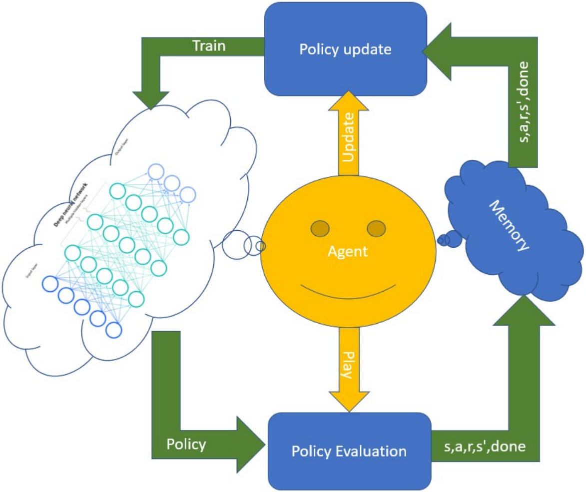


Figure: Framework

Class Based Structure

- Many moving parts and hence divided into following classes
  - Networks - policy and value neural networks
  - Memory - to store batch data generated during policy evaluation
  - Agent - performs policy evaluation and policy update
- Agent has memory, actor, and critic networks
- Agent plays a given policy for a specified number of time steps and collects $\{s, a, r, s', terminal\}$ in the memory
- Reinforce - rewards are converted into returns and stored
- Agent performs policy update
- Advantage (TD-error) is calculated for the batch number of states and backward pass is performed

Neural Network

- Choosing PyTorch - easy to install, less learning curve to use
- Policy outputs mean and variance of Normal distribution for each action dimension
- Multiple ways to do this - some work and some do not
- I found a working implementation with the way shown below

Example (Definition)

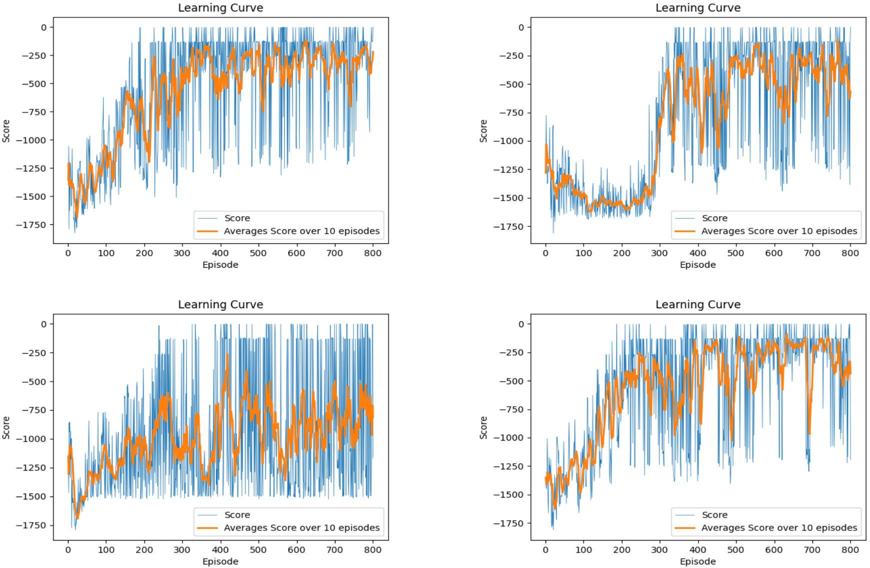
```
self.fc1 = nn.Linear(*self.input_dim, self.fc1_size)
self.fc2 = nn.Linear(self.fc1_size, self.fc2_size)
self.mu = nn.Linear(self.fc2_size, self.action_dim)
logstds_param = nn.Parameter \\
         (torch.full((self.action_dim,),0.1))
self.register_parameter('logstds',logstds_param)
sigma = torch.clamp(self.logstds.exp(), min=1e-3, max=50)
```

Hyperparameters

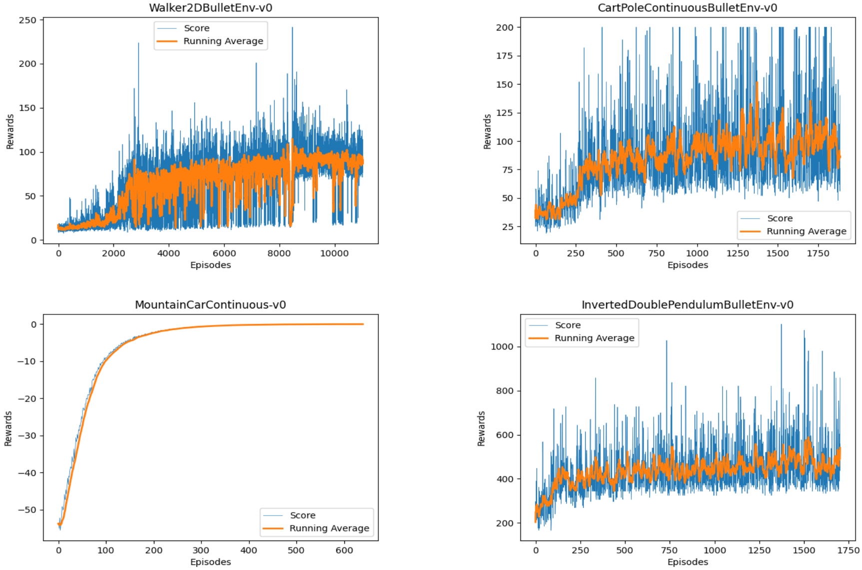
Parameters	Values
Policy-network	{n(state),64,64,n(actions)}
Value-network	{n(state),64,64,1}
Activation	ReLu
Actor-lr	0.0005
Critic-lr	0.005
Discount γ	0.95
Evaluation Batch (timesteps)	64 (REINFORCE), 32 (A2C)
Gradient clip norm	0.5
Policy Standard Deviation Limits	min=1e-3, max = 50

Table: Parameters for Gym Pendulum-v1 Environment

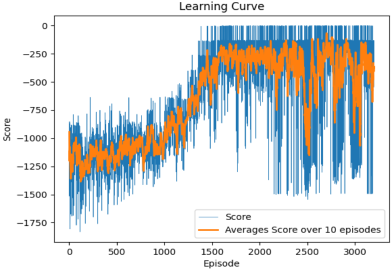
A2C Results - Pendulum-v1

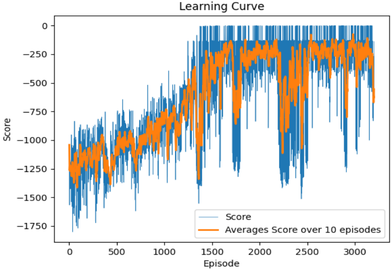


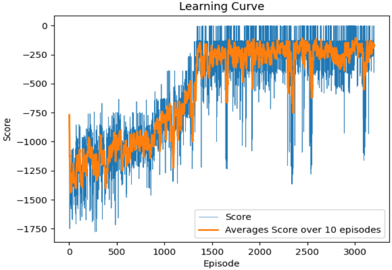
A2C Results - Other Environments

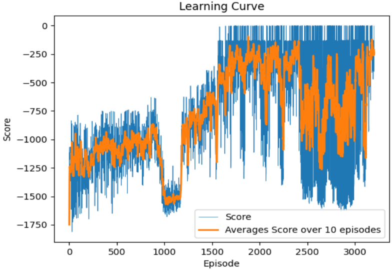


REINFORCE Results - Pendulum-v1

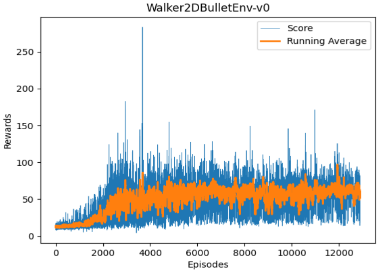


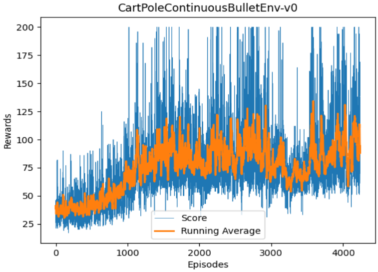


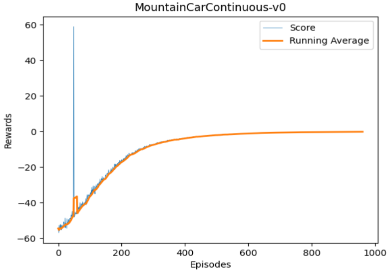


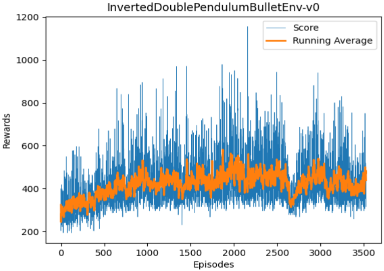


REINFORCE Results - Other Environments









How Important Is Gradient Clipping?

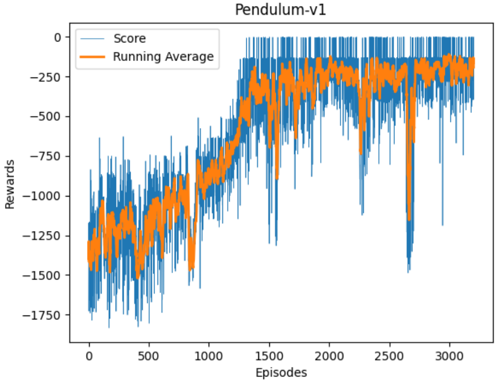


Figure: REINFORCE Gradient Clipping = 1e12 for Pendulum-v1

Gradient Clipping does not affect performance!

How Important Is Gradient Clipping?

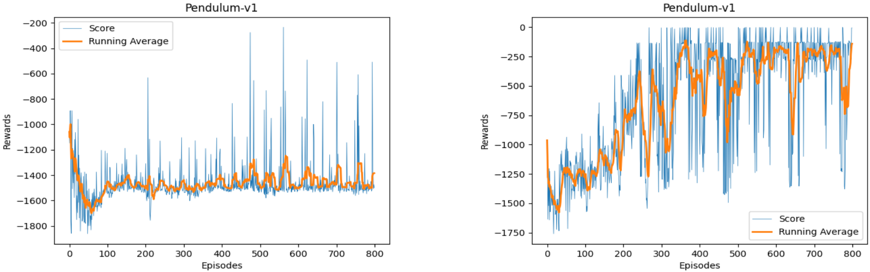


Figure: A2C Gradient Clipping = 100 and 10 for Pendulum-v1

Gradient Clipping must!

Effect of Evaluation Batch Size

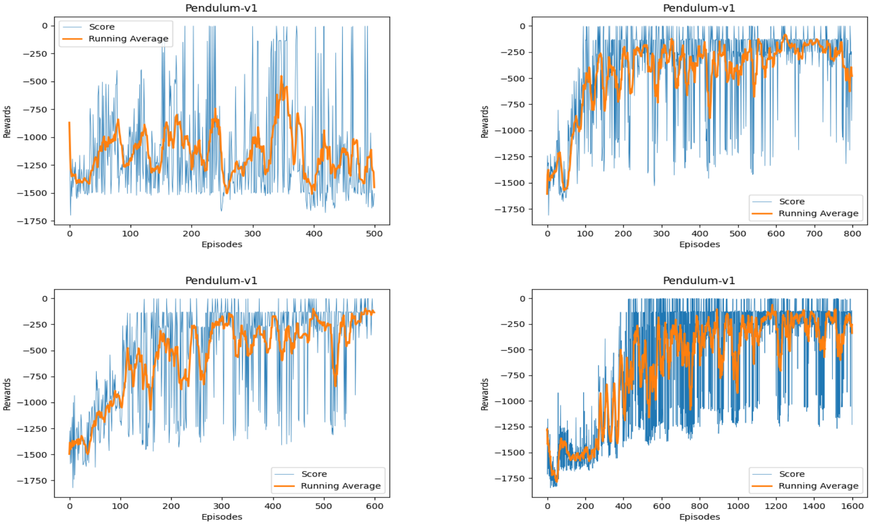


Figure: A2C evaluation batch = 1, 8, 16 and 64 for Pendulum-v1

Effect of Evaluation Batch Size

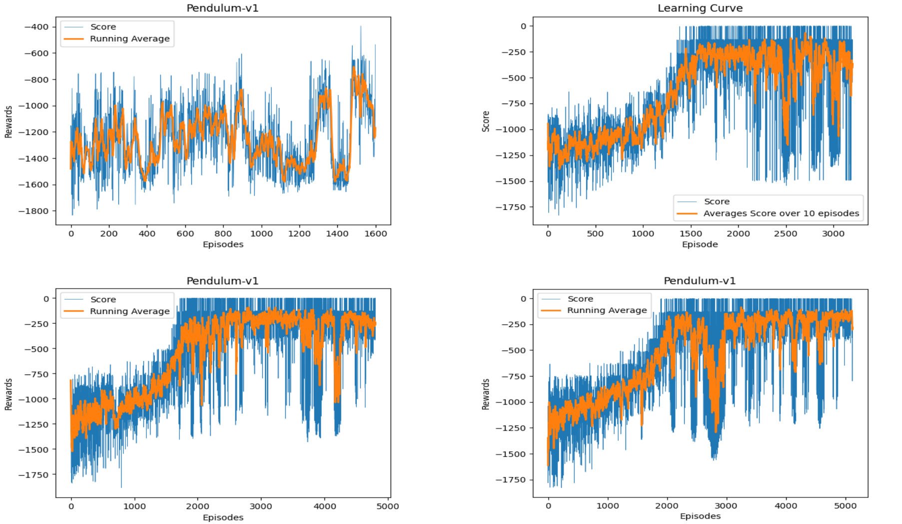


Figure: REINFORCE evaluation batch = 32, 64, 128 and 256 for Pendulum-v1

Effect of Neural Net Sizes

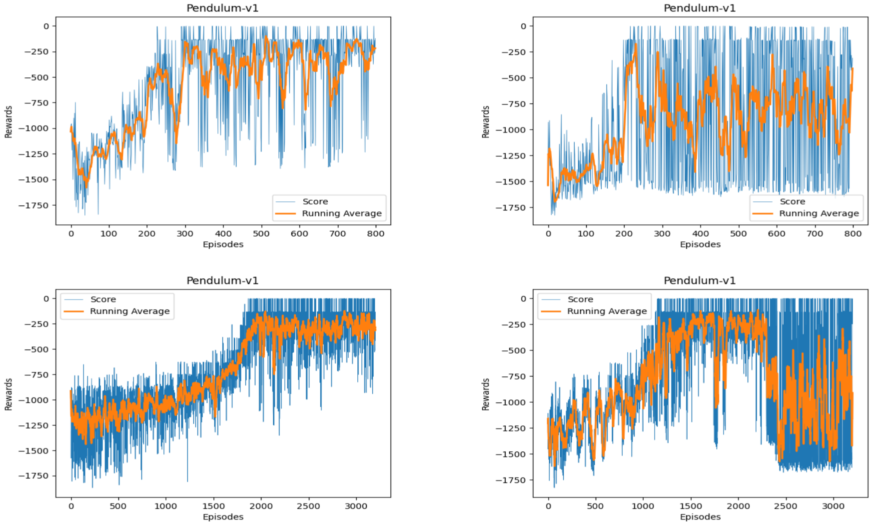


Figure: Neurons per layer 32, 128 (A2C and REINFORCE) Pendulum-v1

Observations and Discussions

- REINFORCE needs more samples than A2C agreeing with the theory
- Both the algorithms fail to learn Mountain Car Continuous, a sparse reward environment
- Need advanced algorithms for a high degree of freedom control tasks
- A2C doesn't work without gradient clipping
- A2C has an optimal sampling batch size, single step A2C did not work; REINFORCE needs a minimum batch size after which it does not affect performance much
- Increased sizes of Neural Nets may not give better performance
- Deep RL needs a lot of hyper-parameter tuning!

References

Sutton, Richard S., and Andrew G. Barto.
Reinforcement learning: An introduction
Journal Name MIT press, 2018

Phil Tabor
https://www.youtube.com/@MachineLearningwithPhil
Github https://github.com/philtabor/Youtube-Code-Repository

hermesdt https://github.com/hermesdt
Github https://github.com/hermesdt/reinforcement-learning/tree/master/a2c

Thank you!