Northeastern University
College of Engineering

IE 7374 - REINFORCEMENT LEARNING

FINAL PROJECT REPORT

Prof. M. DEHGHANI

Solving Inventory Management Problem with Proximal Policy Optimization (PPO)

GROUP

WENCHAO ZHU

KANISHKA PARGANIHA

1. Introduction

1.1 Reinforcement Learning

Reinforcement learning is the study of agents that act in an environment with the goal of maximizing cumulative reward signals. The agent is not told which actions to take but discovers which actions yield the most rewards by trying them. Its action may affect not only the immediate rewards but rewards for the next situations. There are several methods to solve the reinforcement learning problems namely **Armed Bandit Problem, Markov Decision Process, Dynamic Programming, Temporal Differencing,** etc**.** One of the challenges in the RL problem is the exploration versus exploitation problem, which is the trade-off between obtaining rewards from perceived safe options against exploring other possibilities which may be advantageous.

An RL problem can be divided into four sub elements: policy, reward function, value function, and a model. A **policy** specifies how an agent behaves in a given scenario, a **reward** function defines the goal in RL problem, a **value function** maps a state to an estimate of the total reward an agent can expect to accumulate over the future starting from that state and a model is an optional element of the RL system.

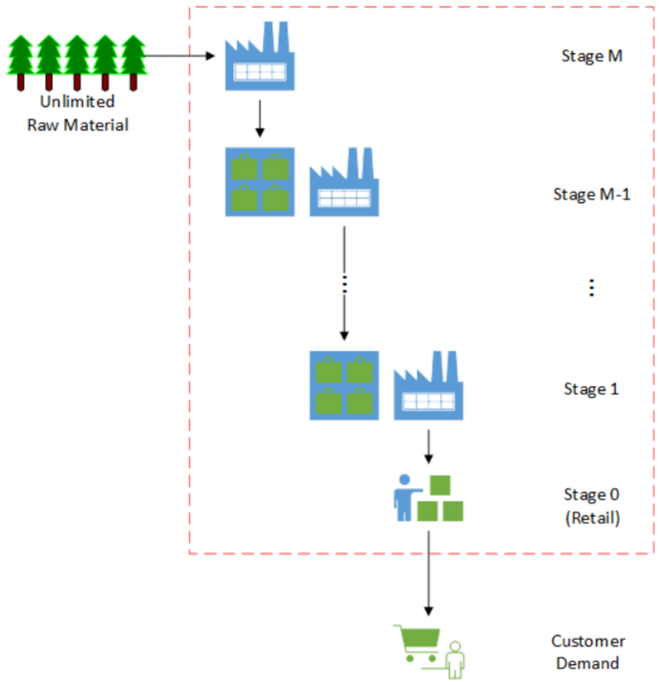
1.2 OR-GYM

OR-GYM is a toolkit for reinforcement learning research problems namely in the field of Operation Research that contains a growing collection of Operation Research problems like **knapsack-v0, knapsack-v1, Binpacking, Vehicle Routing, Inventory Management, Network Management,** etc. It is an open-source python library for developing the applications of reinforcement learning algorithms on OR problems. This library provides RL benchmark using the Ray package for a selection of these problems,as well as heuristic and optimal solutions. Each of the environments that are available in the OR-gym package is easily customizable via configuration dictionaries that can be passed to the environments upon initialization. All the mathematical programming models are solved with Gurobi 8.2 and Pyomo 5.6.2 to optimality on a 2.9 GHz Intel CPU.

1.3 Inventory Management

Modern supply chains are intricate networks that span the world. Supply networks that are efficient are able to control costs and deliver goods to clients with little delays and interruptions. Inventory management is an important part in achieving these goals, based on Glasserman and Tayur's work [3], which depicts a single-product, multi-period, serial capacitated supply chain with production and inventory holding locations at each echelon.

To be specific, A shop encounters erratic consumer demand on a daily basis and must maintain inventory at a cost to accommodate that demand. If the retailer fails to meet that demand, it will either be recognized as a backlog order, which can be filled at a later date with a lesser profit margin (InvManagement-v0), or it will simply be written off as a lost sale with no profit margin (InvManagement-v1). Every day, the retailer must decide how much inventory to order from its distributor, who will manufacture the goods and distribute it to the store within a certain time frame. In a multi-echelon supply chain, the distributor will have a supplier, who may have a supplier above them, and so on, until the supply chain reaches the original party who consumes the raw materials.



2. Methods

In this section we first describe the problem, then two techniques we first select to solve, including Q learning,Proximal Policy Optimization (PPO) and Asynchronous Advantage Actor-Critic (A3C) methods .

2.1 Problem description

In our case, we have M stages going back to the producer of our raw materials all the way to our customers. Each stage along the way has a different lead time, or time it takes for the output of one stage to arrive and become the input for the next stage in the chain. This may be 5 days, 10 days, whatever. The longer these lead times become, the earlier you need to anticipate customer orders and demand to ensure you don't stock out or lose sales.

This is a four-echelon supply chain by default. The actions determine how much material to order from the echelon above at each time step. The orders quantities are limited by the capacity of the supplier and their current inventory. Each echelon has its own costs structure, pricing, and lead times. The last echelon (3 in this case) provides raw materials, and we don't have any inventory constraints on this stage.

As this problem contains both continuous action values and continuous state environment values, we cannot apply a simple Q-learning method or Deep Q Method as these methods require discrete action value or discrete state environment values.

Even though binning is possible for continuous state environment values or continuous action values, that would increase the complexity of the problem exponentially.

2.2 Problem formulation

At each time period in the IMP, the following sequence of events occurs:

- Stages 0 through M − 1 place replenishment orders to their respective suppliers. Replenishment orders are filled according to available production capacity and available inventory at the respective suppliers. Lead times between stages include both production times and transportation times.
- Stages 0 through M − 1 receive incoming inventory replenishment shipments that have made it down the product pipeline after the associated lead times have passed.

- Customer demand occurs at stage 0 (the retailer) and is filled according to the available inventory at that stage.
- One of the following occurs at each stage, (a) Unfulfilled sales and replenishment orders are backlogged at a penalty. Note: Backlogged sales take priority in the following period. (b) Unfulfilled sales and replenishment orders are lost with a goodwill loss penalty.
- Surplus inventory is held at each stage at a holding cost.

2.3 Q learning

Q-Learning is based on the notion of a Q-function. The Q-function (a.k.a the state-action value function) of a policy π, measures the expected return or discounted sum of rewards obtained from state s by taking action a first and following policy π thereafter. Q-learning is a values-based learning algorithm. Value based algorithms update the value function based on an equation (particularly Bellman equation). Whereas the other type, policy-based estimates the value function with a greedy policy obtained from the last policy improvement. The algorithm is defined by:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big] \tag{1}$$

2.4 Proximal policy optimization (PPO)

OpenAI is releasing a new class of reinforcement learning algorithms, Proximal Policy Optimization (PPO), which perform comparably or better than state-of-the-art approaches while being much simpler to implement and tune. PPO has become the default reinforcement learning algorithm at OpenAI because of its ease of use and good performance.

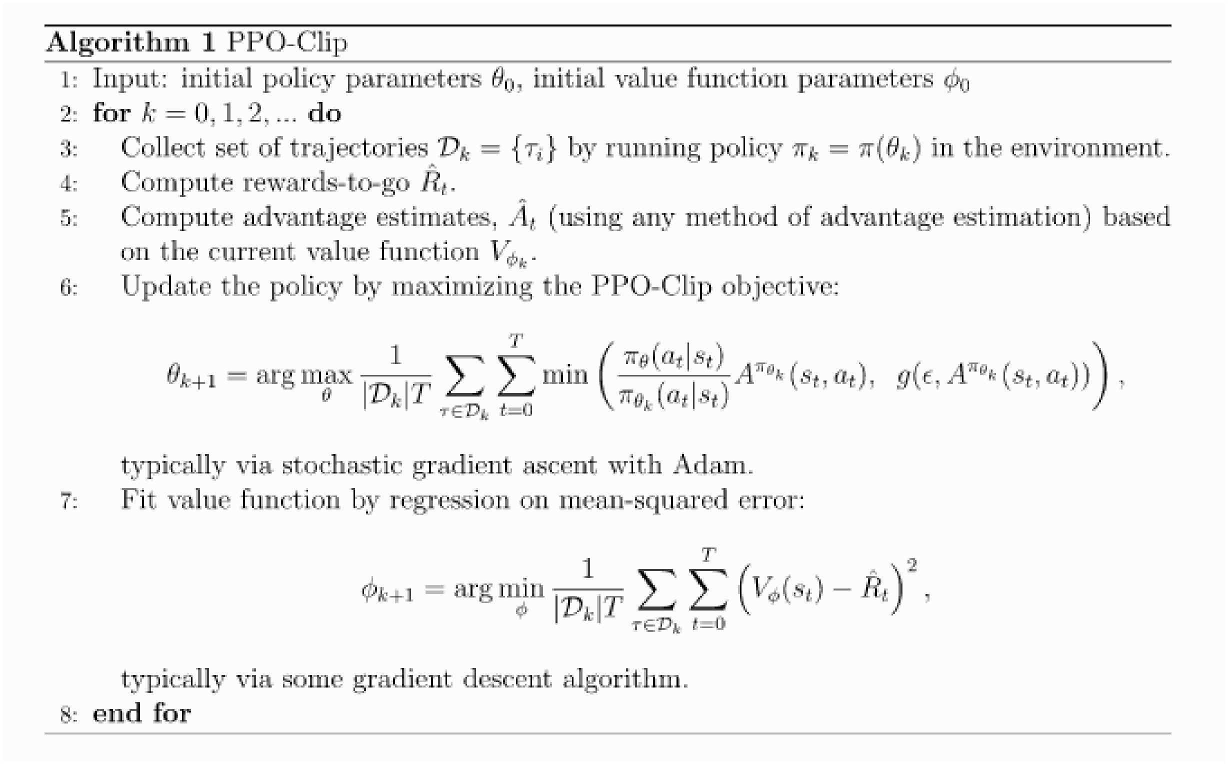
$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon)\hat{A}_t)\ ] \tag{2}$$

Where:

- θ is the policy parameter

- Êt denotes the empirical expectation over timesteps
- Rt is the ratio of the probability under the new and old policies, respectively
- Ât is the estimated advantage at time
- ε is a hyperparameter, usually 0.1 or 0.2

Pseudo Code



```
Algorithm 1 PPO-Clip
1: Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2: for $k = 0, 1, 2, ...$ do
3:     Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4:     Compute rewards-to-go $\hat{R}_t$.
5:     Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based
       on the current value function $V_{\phi_k}$.
6:     Update the policy by maximizing the PPO-Clip objective:
       $\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$
       typically via stochastic gradient ascent with Adam.
7:     Fit value function by regression on mean-squared error:
       $\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$
       typically via some gradient descent algorithm.
8: end for
```

<u>2.5 Asynchronous Advantage Actor Critic</u>

A3C stands for Asynchronous Advantage Actor-Critic. Asynchronous means running multiple agents instead of one, updating the shared network periodically and asynchronously. Agents update independently of the execution of other agents when they want to update their shared network.

The algorithm, which we call asynchronous advantage actor-critic (A3C), maintains a policy π(at|st; θ) and an estimate of the value function V (st; θv). Like our variant of n-step Q-learning, our variant of actor-critic also operates in the forward view and uses the same mix of n-step returns to update both the policy and the value-function. The policy and the value function are updated after every tmax actions or when a terminal state is reached. The update performed by the algorithm can be seen as

$$\nabla_{\theta'} \log \pi(a_t|s_t;\theta') A(s_t, a_t; \theta, \theta_v)$$

where ***A(s$_t$, a$_t$; θ, θ$_v$)*** is an estimate of the advantage function given by where k can vary from state to state and is upper-bounded

Pseudo Code

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

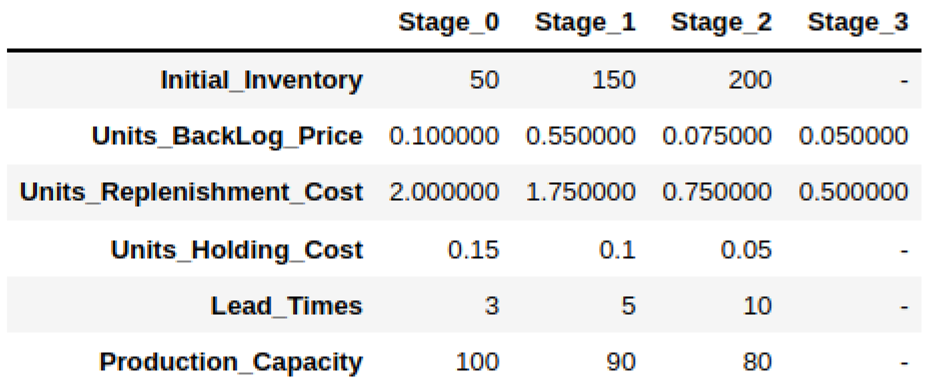
```
// Assume global shared parameter vectors $\theta$ and $\theta_v$ and global shared counter $T = 0$
// Assume thread-specific parameter vectors $\theta'$ and $\theta'_v$
Initialize thread step counter $t \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
    $t_{start} = t$
    Get state $s_t$
    repeat
        Perform $a_t$ according to policy $\pi(a_t|s_t;\theta')$
        Receive reward $r_t$ and new state $s_{t+1}$
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until terminal $s_t$ or $t - t_{start} == t_{max}$
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
    for $i \in \{t-1, \ldots, t_{start}\}$ do
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt $\theta'$: $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
        Accumulate gradients wrt $\theta'_v$: $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i;\theta'_v)\right)^2 / \partial\theta'_v$
    end for
    Perform asynchronous update of $\theta$ using $d\theta$ and of $\theta_v$ using $d\theta_v$.
until $T > T_{max}$
```

3. Implementation and Experimental Setup

The OR-Gym library has a multi-echelon supply chain model that can simulate the structure shown in the Introduction above. For this we are the "**InvManagement-v1**" environment, but results in lost sales if you don't have sufficient inventory to meet the customer's demand.

Each echelon has its own cost structure, pricing and lead time. The last echelon provides raw materials and we don't have any inventory constraints on this stage, assuming that the producer which provides you with the raw materials is large enough that isn't a constraint we need to concern ourselves with.

For this problem we customized our parameters of **Initial Inventory, Units sales Prices, Units Replenishment Cost, Unit backlog Cost, Unit Holding Cost, Production Capacity and Lead times**.



	Stage_0	Stage_1	Stage_2	Stage_3
Initial_Inventory	50	150	200	-
Units_BackLog_Price	0.100000	0.550000	0.075000	0.050000
Units_Replenishment_Cost	2.000000	1.750000	0.750000	0.500000
Units_Holding_Cost	0.15	0.1	0.05	-
Lead_Times	3	5	10	-
Production_Capacity	100	90	80	-

For training the environment with different Reinforcement learning methods we leveraged **Ray** library, a multi-processing library that enables us to scale training to large-scale distributed servers or just take advantage of the parallelization properties to more efficiently train using any local device.

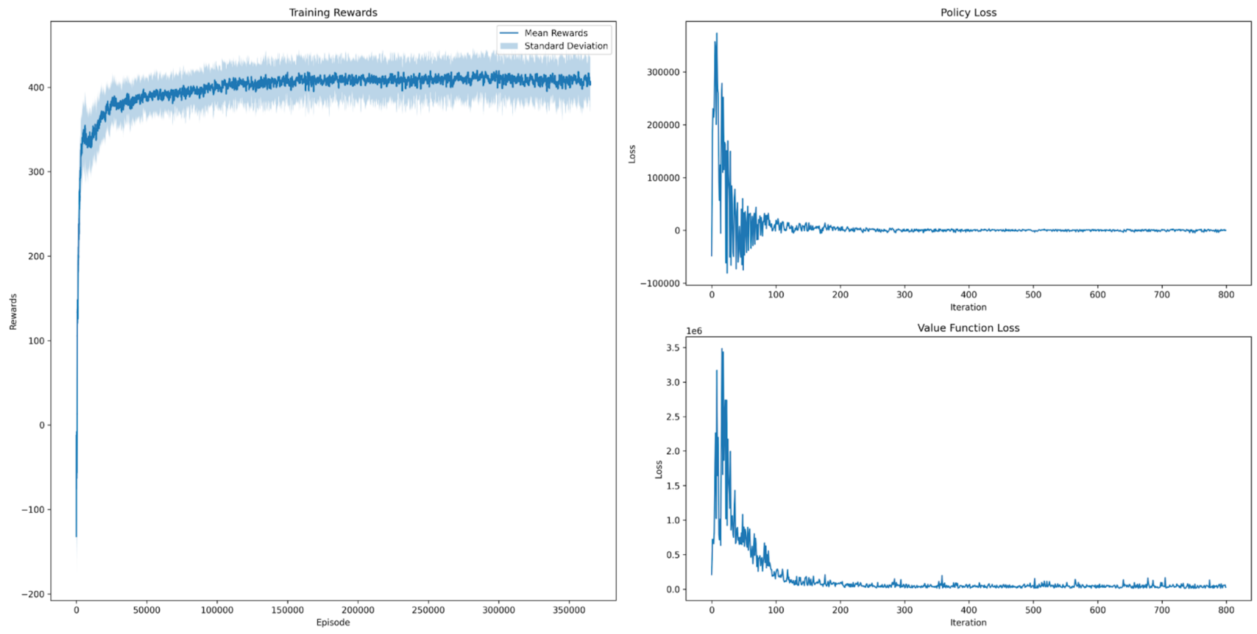
The various algorithms are available through **ray.rllib.agents.** All the algorithms contain specific hyperparameters which can be fine tuned to optimise our model. For **PPO**, we performed the fine tuning for the changing the **Clip Gradient** parameter to different values **0.1, 0.2 and 0.3** and keeping the **learning rate** as **$1x10^{-5}$.**

We then compared the performance of PPO with these different parameter with the **Asynchronous Actor-Critic Method (A3C)** method using **Episode vs Rewards plot, Policy Loss plot and Loss function plot.**

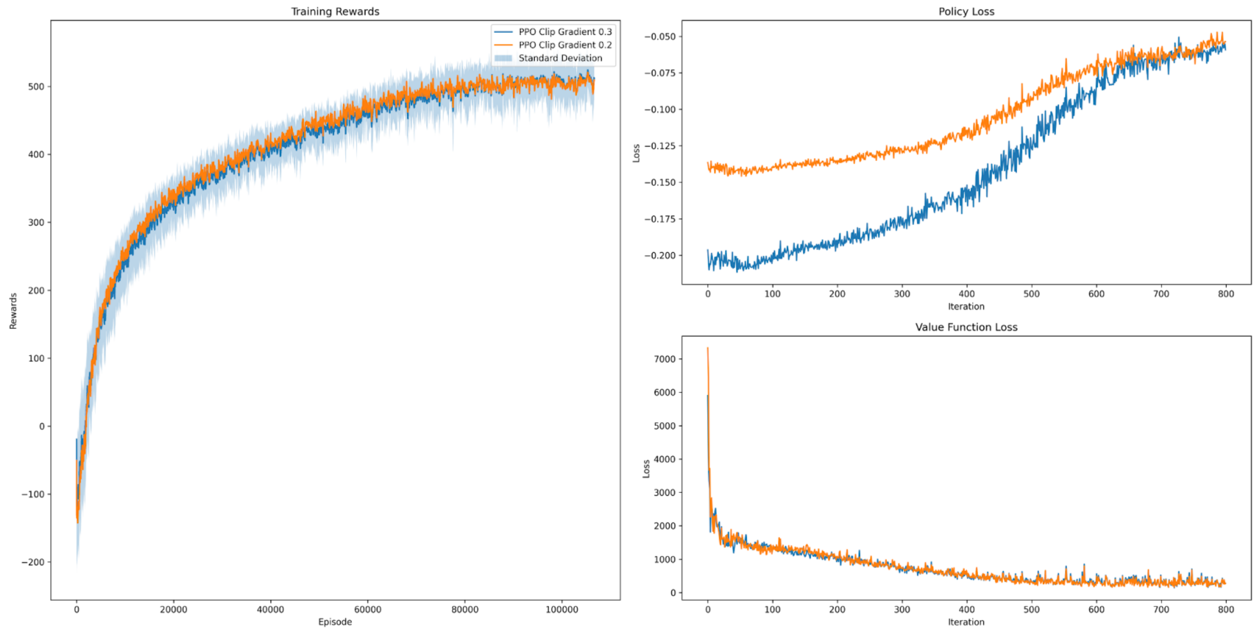
4. Results

We compare PPO with A3C method which are considered to be effective for continuous problems. We compared against tuned implementation of the following algorithms

- **A3C** - Running the model for about 350000+ episodes



- **PPO -** Learning rate : 1x10^{-5} and Clip Gradient : [0.2, 0.3]



5. Discussion and Conclusion

From the results obtained from the plots we can infer that **PPO** might not converge faster than **Asynchronous Advantage Actor-Critic** method but have better reward values as the number of episodes increases and thus converges on high reward values compared to the A3C method.

So in conclusion, we have introduced proximal policy optimization method, a family of **policy optimizing methods** that use multiple epochs of **stochastic gradient ascent** to perform each policy update. These methods have stability and reliability of trust-region methods but are much simpler to implement, requiring only a few lines of code change to a vanilla policy gradient implementation, applicable in more general settings.

References:

- Christian D. Hubbs, Hector D. Perez, Owais Sarwar, Nikolaos V.Sahinidis, Ignacio E. Grossmann, John M. Wassick, **OR-Gym: A Reinforcement Learning Library for Operations Research Problems**, last revised 17 Oct 2020

- Perez, H. D., Hubbs, C. D., Li, C., & Grossmann, I. E. (2021). Algorithmic Approaches to Inventory Management Optimization. Processes, 9(1), 102.

- P. Glasserman and S. Tayur. Sensitivity analysis for base-stock levels in multi echelon production-inventory systems. Management Science, 41(2):263{281, 1995.