Inventory Control with Lost Sales and Lead Times

Matthew Ford & Anna Poulton

May 16 2021

1 Introduction

Optimal inventory management is a major concern for many business. Here, we consider an inventory management problem that exhibits two key traits: *positive lead times* and *lost sales*. In a system with positive lead times, after an order for more inventory is placed, there is a delay before it is received [4]. This is a reasonable assumption because, in many real life situations, it takes a non-negligible amount of time for inventory that is ordered to be produced and shipped to a business. On the other hand, lost sales means that if there is more demand than there is inventory on hand, then any demand that is not fulfilled disappears and cannot be met at a later date [4]. In real life, this may occur when competing business(es) exist, as customers with unmet demand may simply take their business elsewhere [4]. This is in contrast to other inventory models with *backlogged demand*, in which case demand that cannot be immediately met can be fulfilled in the future [4].

There are a variety of policies that have been examined in the context of these inventory models. Two examples of simple policies include the constant-order policy and the base-stock policy. For a *constant-order policy*, the same amount of inventory is always ordered (regardless of the amount of inventory on hand, the amount of inventory that has been ordered but has not yet arrived, etc.) [4]. This type of policy can be characterized by a single parameter r, representing the size of the constant-order. On the other hand, for a *base-stock policy*, an order is only placed when the inventory falls below a certain value; the size of the order is chosen to bring the inventory back up to a specific level [2].

The analysis of different types of models faces different challenges. For instance, especially when lead times are long, models with positive lead times and lost sales are generally not possible to solve with dynamic programming [4]. A precise form for the optimal policy has not yet been found for this type of model, although its properties are starting to be better understood [4]. On the other hand, it has been proven that for models with positive lead times and backlogged demand, a base-stock policy is optimal [4]. Because of their tractability, in the past these models have been used as approximations for models with lost sales; however, the resulting optimal base-stock policy can sometimes show very suboptimal performance when applied to the corresponding lost sales model [4].

Much work has gone into understanding the properties of the optimal policy for the lost sales model with positive lead times. A key result is that the best constant-order policy becomes asymptotically optimal for this model as the lead time grows [4]. However, even under short lead times, previous numerical results have shown decent performance for the constant-order policy [4]. Xin & Goldberg (2016) provide improved bounds for the performance of the constant-order policy, showing that as the lead time increases, the optimality gap of the performance of the optimal policy and the best constant-order policy converges to zero exponentially fast. The results of Xin & Goldberg's work will be discussed more thoroughly in section 2.2.

Our ultimate goal will be to, in a wide parameter space, train policies that achieve better performance than the best constant-order policy. In section two, we define the model for our inventory management problem, as described in Xin & Goldberg (2016). Following this, we verify the model's implementation in an OpenAI Gym environment by testing its performance under various simple policies. We then discuss our deep reinforcement learning (RL) training methodology, which uses the Stable Baselines3 package. In section 3, the results of our training process are presented, with a discussion of where our training procedure produced good results and where improvements could still be made. For a few simple cases, we will also present visual representations of our optimal policy and discuss their general characteristics. Finally, we discuss a modified training procedure to address a specific region of parameter space.

2 Methods

2.1 MDP

Here, we define the Markov decision process (MDP), as described by Xin & Goldberg (2016), to represent the lost sales inventory model with positive lead times. Let $L \in \mathbb{N}$ be the lead time (representing the number of time steps for an order that was placed to be delivered) [4]. The state of the system consists of two components: I_t, representing the current inventory level, and $\mathbf{x}_t = (x_{1,t}, \ldots, x_{L,t})$, a vector of inventory orders that will arrive in the future (often referred to as a 'pipeline' vector) [4]. We only require that these quantities be non-negative, and thus, the state space is $\mathcal{S} = \{(I_t, \mathbf{x}_t) : I_t, x_{1,t}, \ldots, x_{L,t} \geq 0\}$ [4]. For initial conditions, as in Xin & Goldberg (2016), we assume $I_1 = 0$ and $x_{i,1} = 0$ for $i = 1, \ldots, L$.

Following the terminology of Xin & Goldberg (2016), we let $f_t^\pi(\mathbf{x}_t, I_t)$ denote the order placed at time t, which is determined by the policy π. Since any non-negative order is a valid possible action, our action space is $\mathcal{A} = [0, \infty)$ [4]. Demand in our system is modeled as exponentially distributed with $\lambda = 1$, as this simple form allowed Xin & Goldberg (2016) to exactly solve for the best constant-order policy and its performance. At the end of each time period, a penalty $h > 0$ is incurred per unit of leftover inventory, and a penalty $p > 0$ is incurred per lost unit of sales [4]. Bringing these pieces together, the MDP described by Xin & Goldberg (2016) proceeds as follows for each time step t:

1. Update inventory on hand: $\tilde{I}_t = I_t + x_{1,t}$
2. Simulate demand: $D_t \sim Exp(\lambda)$ with $\lambda = 1$
3. Determine post-demand inventory: $I_{t+1} = \max(0, \tilde{I}_t - D_t)$
4. Incur costs: $C_t^\pi = h(\tilde{I}_t - D_t)^+ + p(\tilde{I}_t - D_t)^- = h * \max(0, \tilde{I}_t - D_t) + p * \max(0, -(\tilde{I}_t - D_t))$
5. Update pipeline vector: $x_{L,t+1} = f_t^\pi(\mathbf{x}_t, I_t)$ and $x_{i,t+1} = x_{i+1,t}$ for $i = 1, \ldots, L-1$.

2.2 Long-run Average Cost

The performance of a policy π is determined by its long-run average cost $C(\pi)$, which we wish to minimize [4]. In Xin & Goldberg (2016), the long-run average cost of a policy π is calculated exactly as $C(\pi) = \limsup_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[C_t^\pi]$. However, since we are carrying out simulations, we need to be able to approximate the long-run average cost. Let $C_{t,j}^\pi$ be the cost incurred at time t in the j-th episode of simulating policy π, and let the time horizon (T^*) and number of episodes (J) be large. Then, we estimate the long-run average cost as

$$C(\pi) \approx \frac{1}{T^*}\sum_{t=1}^{T^*}\left[\frac{1}{J}\sum_{j=1}^{J} C_{t,j}^\pi\right] = \frac{1}{J}\sum_{j=1}^{J}\left[\frac{1}{T^*}\sum_{t=1}^{T^*} C_{t,j}^\pi\right]$$

The truncation of the limit results in a biased estimate, which depends on the initial state [3]. However, given a large enough T^*, this bias should be sufficiently small for our purposes. We demonstrate this with an example in section 2.3, in which we show that our estimate for the long-run average cost (taken over $J = 100$ episodes) converges quickly to the true long-run average cost as the time horizon T^* increases.

It should be noted that this approximation only works if $C(\pi)$ is finite. For instance, with $\lambda = 1$, the expected demand per time step is $\mathbb{E}(D_t) = 1$. If the chosen policy π placed an order of $r = 10$ every time period, the amount of inventory on hand would tend to pile up over time, causing $\mathbb{E}[C_t^\pi]$ to increase without limit as $t \to \infty$ and $C(\pi)$ to be infinite. (In Xin & Goldberg (2016), the constant-order policy is restricted to $r \in [0, \mathbb{E}(D))$, since larger r would result in $C(\pi_r)$ being infinite.) If $C(\pi)$ is infinite, the approximation of $C(\pi)$ given above will tend to increase as the time horizon increases. We demonstrate this with an example in section 2.3.

The work of Xin & Goldberg (2016) shows that the best constant-order policy makes an order each time period of r_∞, and incurs a long-run average cost of $C(\pi_{r_\infty})$, where

$$r_\infty = \frac{1}{\lambda}\left(1 - \sqrt{h/(2p+h)}\right), \quad C(\pi_{r_\infty}) = \frac{1}{\lambda}\left(\sqrt{h(2p+h)} - h\right)$$

If Π is the set of all viable policies, then the long-run average cost of the truly optimal policy is given by $OPT(L) = \inf_{\pi\in\Pi} C(\pi)$ [4]. While we do not know the truly optimal policy, the results of Xin & Goldberg (2016) give optimality bounds for the best constant-order policy, which are as follows.

$$\frac{C(\pi_{r_\infty})}{OPT(L)} \le 1 + \left(\tau_{p,h} + \left(\frac{1}{\tau_{p,h}} - 1\right)\left(\frac{1}{e(L+1)}\right)\right)\left(\frac{1}{(1-\gamma_{p,h})\log(1+p/h)}\right)\gamma_{p,h}^{L+1}$$

where $\tau_{p,h} = \sqrt{h/(2p+h)}$ and $\gamma_{p,h} = (1-\tau_{p,h})\exp(\tau_{p,h})$ [4]. The optimality bounds are given for several combinations of p and L in Table A.1 from Xin & Goldberg (2016). For easy reference, a copy of this table can be found in the appendix. For small p (and especially when L is large), the best constant-order policy is guaranteed to be close (or very close) to optimal [4]. This case is less interesting to us, as training will not be able to improve much over the best constant-order policy. On the other hand, when p is large (and especially when

L is small), the optimality bounds can be quite large [4]. As such, this is the region that we will focus our training efforts on.

It should be stated for clarity that the entries of Table A.1. represent the 'worst case' scenario for the best constant-order policy. Even if the bound is large, the performance of the best constant-order policy could still be close to that of the optimal policy. Because of this, we may or may not be able to obtain significant performance improvements in our trained policy over the best constant-order policy: it all depends on the how close the actual performance of the (unknown) optimality policy is to that of the best constant-order policy.

2.3 OpenAI Gym Environment Verification

First, we tested the implementation of the MDP in an OpenAI Gym environment by simulating the performance of two different constant-order policies. Let $h = p = \lambda = L = 1$. Using the results from Xin & Goldberg (2016), the best constant-order policy for this set of parameters is $r_\infty \approx .423$, and this policy has a long-run average cost of $C(\pi_{r_\infty}) \approx .73205$. On the other hand, if we let $r = 10$, then (as noted previously) the true long-run average cost will be infinite, and our estimate for the long-run average cost should increase as the time horizon increases.

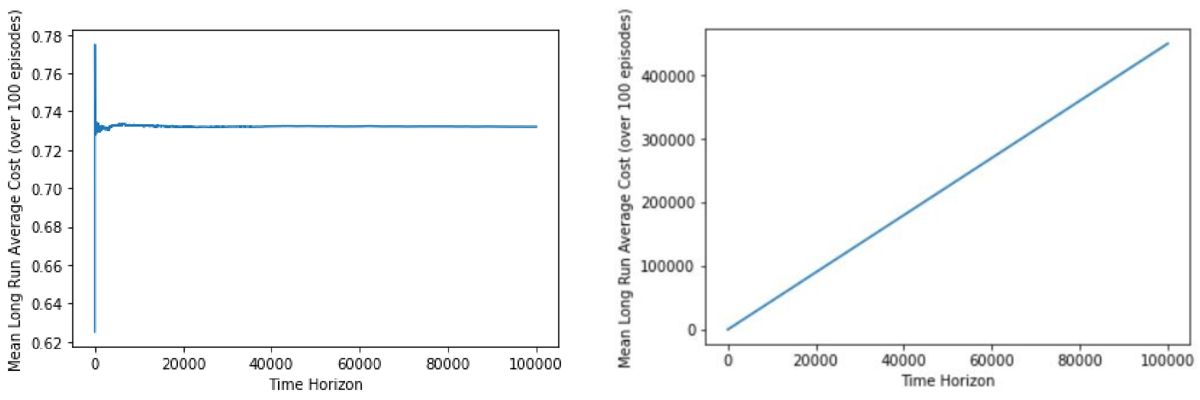


Figure 1: The mean (over 100 episodes) of the average cost for two different policies (left: $r_\infty \approx .423$, right: $r = 10$) as the time horizon increases.

Figure 1 shows our estimate for the long-run average cost for each policy as the time horizon increases. For the best constant-order policy $r_\infty \approx .423$ (left image), the estimate of the long-run average cost quickly converged to approximately $C(\pi_{r_\infty}) \approx .73205$. (At the end of the time horizon, the estimated performance was 0.73195 with a standard deviation of 0.00297.) This example helps to demonstrate the validity of our method for estimating the long-run average cost when $C(\pi)$ is finite. On the other hand, for the $r = 10$ policy (right image), the estimate of the long-run average cost did not converge as the time horizon increased (which was expected, as this policy has an infinite long-run average cost).

2.4 Model Training with PPO

We used Stable Baseline3's implementation of PPO to train our models [1]. We used the default neural network parameterizations. The first network is a feature extractor network with 2 layers each consisting of 64 units. This network learns a useful transformation of the state space to be used as shared input to our policy and value models. The policy network consists of a linear model whose output is then transformed through a squashing function to be non-negative to match our action space. During training we added Gaussian noise to the output of the policy linear model but before the squashing transformation to obtain a stochastic policy which PPO can learn with. We initialized the weights of the policy network to be 0 and the bias to be r_∞, the optimal constant-order amount for the problem. This initialized the policy's median action to approximately (approximation potentially coming from squashing function, though the squashing function did not seem to alter the few cases we manually examined) be the known best constant-order policy. We also set the standard deviation of the noise added at the start of training to be e^{-1} as opposed to the default of 1. Given that the output of our initial policies network were on the order of 1, we did not want to get too many outputs that were very small and would be squashed substantially. We briefly tried using even lower standard deviations but found that they did not yield as good of policies, likely suffering from lack of exploration. The value model was a simple linear model and we did not change its random initialization in any way.

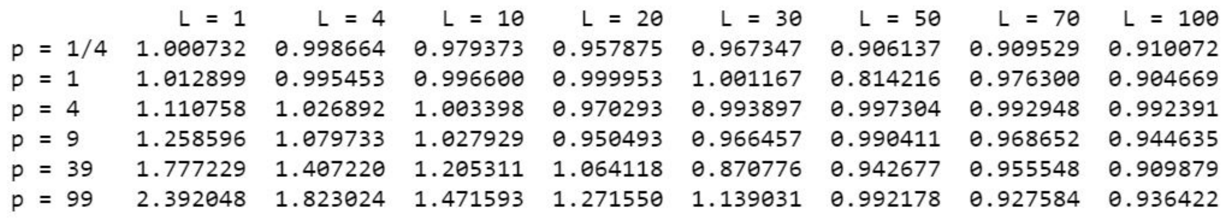
When training for a given problem, we simulated for 500000 total time steps, with model updates happening every 2048 time steps. We meant to simulate multiple environments in parallel during training, but due to a recently found typo in our code we realized we only simulated 1 environment during the training for each problem. We evaluated the current model after intervals of approximately 25000 time steps on 8 environments of episode length 20000. In the end, the policy we selected and retained was the one which had the lowest average cost during this evaluation procedure. To generate our final results, the best trained policy for each problem was evaluated on 50 episodes of length 20000. We carried out several tests (on combinations of small vs large p and small vs large L) to ensure that this choice of a time horizon for policy evaluation was sufficiently good for our purposes. Occasionally, the mean costs of the policies would start increasing and then oscillating around wildly during training producing poorly performing policies. Thus, we repeated this training procedure twice for all the problems, and took the best of the two resulting trained policies. This behavior may have been avoided had we actually simulated multiple environments in parallel during training as we had intended to do for variance reduction.

There were many hyperparameters to tune. We manually experimented with the length of the episodes, how long to train, the standard deviation of our initial policy, the learning rate, and more. Our initial goal was to get good performance on the problems with low values of L and high values of p, as this region had the most potential room for improvement over the best constant-order policy [4]. Other than setting $\gamma = 1$ to tell PPO to solve the average cost problem instead of a discounted one, we found the default parameters values used by Stable Baseline3's implementation of PPO (for learning rate, etc.) to work well. While the episode length of 2048 is long enough to observe multiple regenerative cycles for low to medium values of L, it is likely too short for when L is large (say, when $L = 100$). Additionally, in the regions where the best constant-order policy is close to optimal, any deviation from it is likely a decrease. In such cases, we would likely need longer episode lengths and lower learning rates, clipping parameters, and starting standard deviations for the initial policy to ensure that we do not regress in performance. These parameter settings would have made learning policies for the other problems highly sample inefficient, however. While our initial methodology is focused on the small L and large p region, in section 3.3 we experiment with such changes to improve the performance of our training procedure for the case of large L and large p.

3 Results and Discussion

3.1 Training Results

We trained policies for a total of 48 problems, across which the lost sales penalty (p) and the lead time (L) were varied. The inventory holding penalty (h) and the demand parameter (λ) were both fixed at 1 in all problems. Figure 2 is a summary of our training results when using the procedure described in section 2.4. This table entries are $C(\pi_{r_\infty})/C(\pi_{PPO})$, the ratio of the long-run average cost for the best constant-order policy and our best trained policy. $C(\pi_{PPO})$ was estimated over 50 episodes with time horizons of 20000, while $C(\pi_{r_\infty})$ was calculated using the theoretical results from Xin & Goldberg (2016) discussed in section 2.2. A number greater than 1 indicates that our trained policy performed better on average than the best constant-order policy. 95% confidence intervals are available in Figure 7 in the appendix.



	L = 1	L = 4	L = 10	L = 20	L = 30	L = 50	L = 70	L = 100
p = 1/4	1.000732	0.998664	0.979373	0.957875	0.967347	0.906137	0.909529	0.910072
p = 1	1.012899	0.995453	0.996600	0.999953	1.001167	0.814216	0.976300	0.904669
p = 4	1.110758	1.026892	1.003398	0.970293	0.993897	0.997304	0.992948	0.992391
p = 9	1.258596	1.079733	1.027929	0.950493	0.966457	0.990411	0.968652	0.944635
p = 39	1.777229	1.407220	1.205311	1.064118	0.870776	0.942677	0.955548	0.909879
p = 99	2.392048	1.823024	1.471593	1.271550	1.139031	0.992178	0.927584	0.936422

Figure 2: Using the training methodology from section 2.4, $C(\pi_{r_\infty})/C(\pi_{PPO})$ under different values of L and p. ($h = 1$ and $\lambda = 1$ were fixed.)

Ultimately, we were able to obtain the best results in the case of small L and large p. In such cases, the long-run average cost of our trained policy tended to quickly drop below the long-run average cost of the best constant-order policy, and generally continued to improve throughout the training process. Our best success came in the case of $(p, L) = (99, 1)$, where our trained policy had an estimated long-run average cost of approximately 5.5, versus the best constant-order policy's cost of approximately 13.1. The performance of this policy throughout the training process is shown in the left image of Figure 3.

Our training methodology was not able to achieve as good of results in other regions of parameter space; however, we did not expect to see performance gains in many of these regions. For example, in the case of large L and small p, the best constant-order policy is nearly optimal anyways [4], so we did not expect our trained policies to perform better. Despite starting close to the best constant-order policy, the training process produced a policy that performed somewhat worse, likely due to the hyperparameters being tuned for the case of small L and large p.

For small L and small p, our trained policies performed similarly or slightly better than the best constant-order policy. The training procedure was also less reliable, and the policy would often experience periods of becoming gradually worse (demonstrated by the $(p, L) = (1, 1)$ case in the right image of Figure 3). Additional tuning of our training procedure could have perhaps improved the performance of our policies here, but the optimality bounds in this region only leave slight room for improvement [4]. Interestingly enough, while our trained policy performed similarly to the best constant-order policy, the policies could look very different from each other. An example of this will be discussed in section 3.2.

For large L and large p, our trained policies performed slightly worse than the best constant-order policy. The training procedure would often exhibit large spikes of poor performance. In some cases the best policy produced would even be right after the very first evaluation at 25000 time steps (demonstrated by the $(p, L) = (99, 70)$ case in the left image of Figure 5). Because our current training procedure appeared to not work in this region, and since the optimality bounds were somewhat large (notably when $p = 99$) [4], we decided that additional tuning would be worthwhile to see if it might still be possible to improve upon the best constant-order policy. In section 3.3, we discuss changes to our training methodology to better address this region of parameter space.

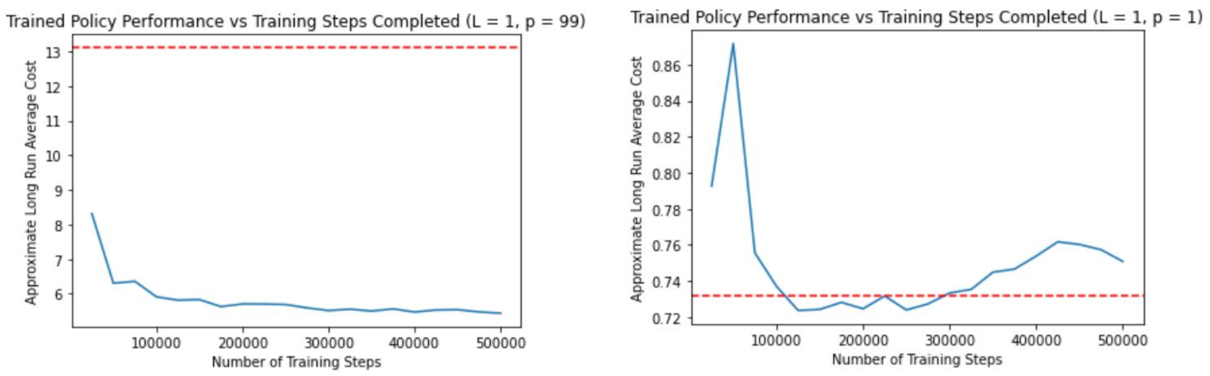


Figure 3: Two runs of our training procedure showing great (left) and ok (right) performance. The blue line indicates the performance of our trained policy (evaluated every 25000 time steps), while the red dashed line shows the performance of the best constant-order policy.

3.2 Visualizations of Trained Policies

For the $L = 1$ case, we were able to produce simple visual representations of our trained policies, three examples of which are given in Figure 4. Across a range of values for p, our trained policies show some similarities. For instance, the order size generally decreases as the amount of on hand inventory (I_t) and the amount of inventory in the pipeline ($x_{1,t}$) increases. Additionally, the order size tends to increase as p increases: this is intuitively reasonable, as increasing p results in larger penalties if demand goes unfulfilled, meaning that it should be desirable to order more inventory to avoid this risk. The trained policy for the $(p, L) = (1, 1)$ case is especially interesting. While this policy shows very similar performance to the best constant-order policy ($r_\infty \approx 0.423$), it looks considerably different. Much larger orders are placed near $(I_t, x_{1,t}) = (0, 0)$, while nothing is ordered when $I_t + x_{1,t}$ is larger than approximately 1.5. In fact, our trained policy for this case shares many similarities with a base-stock policy (not ordering anything when above a certain amount of inventory, etc.).

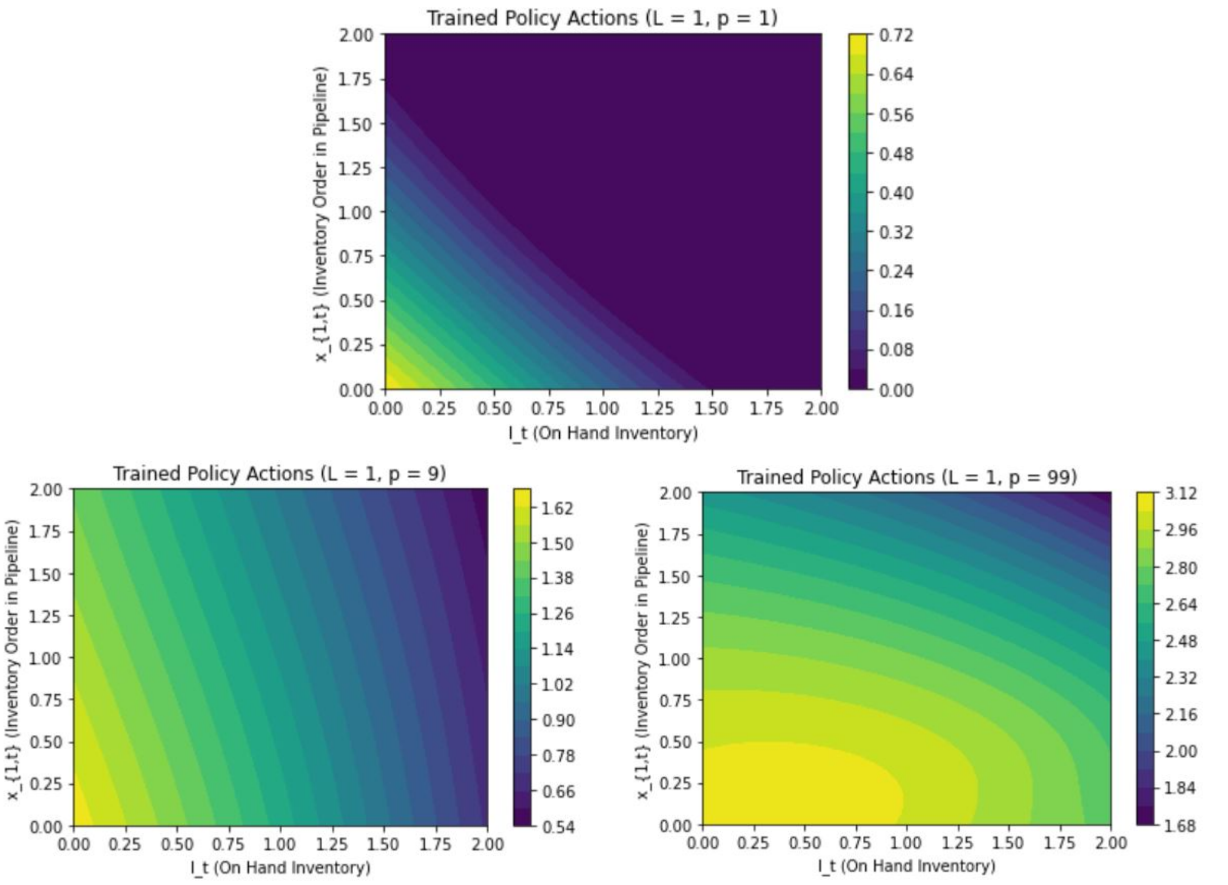


Figure 4: Actions taken by three different trained policies on a subset of the state space. Note that the color scale differs in each image. **Top**: $(p, L) = (1, 1)$, where the best constant-order policy is $r_\infty \approx 0.423$. **Bottom left**: $(p, L) = (9, 1)$, where $r_\infty \approx 0.771$. **Bottom right**: $(p, L) = (99, 1)$, where $r_\infty \approx 0.929$.

3.3 Improving training performance for large L and large p

While our initial training procedure worked great for small L and large p, it did not perform well for the case of large L and large p. Since the optimality bounds are somewhat large in this region [4], we wanted to see if we could further improve our results through additional tuning of our training process. As a test case, we selected the problem $(p, L) = (99, 70)$. To improve the training performance for this problem, we experimented with several of the hyperparameters in Stable Baseline3's implementation of PPO, along with the parameters we used to initialize our starting policy. Ultimately, we were able to achieve improvements by using a significantly larger number of steps between model updates ($24 * 2048 = 49152$), decreasing the learning rate to .0002, and decreasing the standard deviation of actions taken by the initial policy to e^{-2}. We also increased the total number of training time steps to $2 * 10^6$, increased the number of episodes used when evaluating the policy during training to

20, and decreased the evaluation frequency to every 50000 time steps. This updated training procedure was still somewhat unreliable and would occasionally show moderate spikes of poor performance; because of this, we performed 5 total runs of training and took the results from the best run. However, our updated procedure performed significantly better than our old training procedure. Figure 5 shows runs of the old training procedure (as described in section 2.4) versus the updated training procedure for the $(p, L) = (99, 70)$ case.

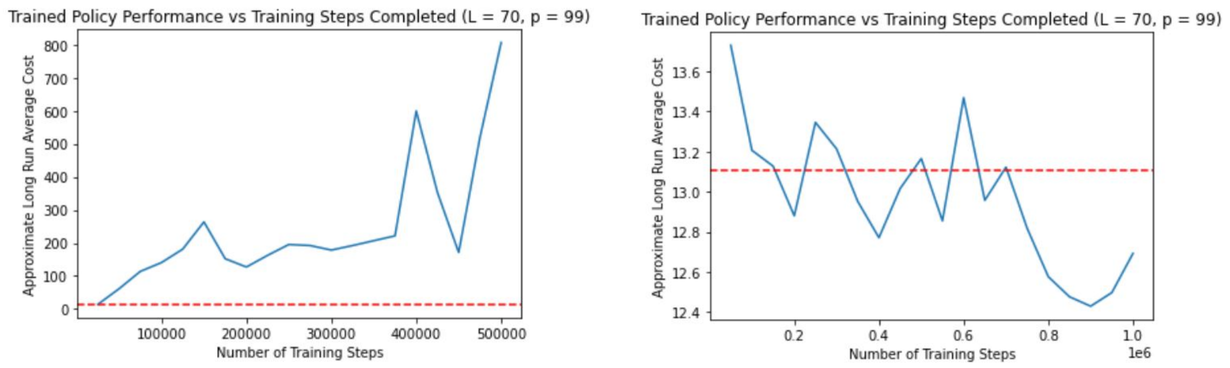


Figure 5: A run of our old (left) vs updated (right) training procedure when $(p, L) = (99, 70)$. The blue line shows the performance of our trained policy, while the red dashed line shows the performance of the best constant-order policy. To focus on where the policy showed the best improvements, only the first million training time steps are shown in the right image.

Using our updated training procedure, the resulting estimate $C(\pi_{r_\infty})/C(\pi_{PPO}) \approx 1.0639$ (95% confidence interval: $[1.0543, 1.0736]$) was obtained. This is a slight, but noticeable improvement over the best constant-order policy. With additional time, we could perform a similar procedure to obtain updated results for other (p, L) combinations in this region. Additional tuning of the other parameters could have also potentially led to even better performance gains. However, recall that even if the optimality bound given by Xin & Goldberg (2016) is large, it's possible that the best constant-order policy performs close to optimally anyways, in which case no amount of tuning would produce significant performance improvements. Improved theoretical bounds could be a huge help in determining whether or not this is the case, demonstrating the importance of theoretical results in reinforcement learning problems.

4 Conclusion

We used PPO to train policies for the inventory management problem with lost sales and positive lead times. We examined this problem across 48 different combinations of the parameters p and L (with a specific focus on the regions with the largest optimality bounds). Our goal was to find policies that performed better than the best constant-order policy, and we succeeded for the values of p and L for which we hoped to do so. Our initial training procedure produced good results for the case of large p and small L, with trained policies in this region performing as much as 2.4x better than the best constant-order policy. While the initial training procedure did not work for large p and large L, a modified training procedure produced a policy in a test case that was slightly better than the best constant-order policy.

In the future, we would extend this work by further tuning of our training procedure to target regions of parameter space where potential improvements might still be obtained. However, we reiterate that the optimality bounds from [4] are upper bounds: even where improvements could be made, it is likely impossible to obtain improvements on the scale of the bounds given in Table A.1. (for instance, a 204x improvement in the $(p, L) = (99, 1)$ case).

5 Appendix

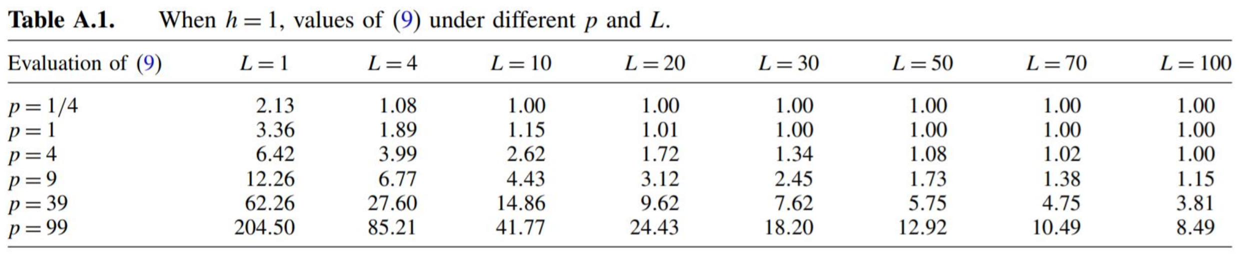
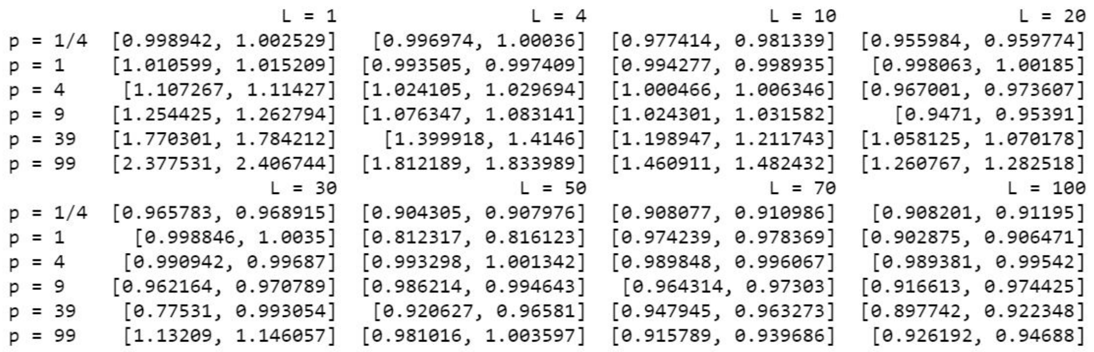


Table A.1. When $h = 1$, values of (9) under different p and L.

Evaluation of (9)	$L = 1$	$L = 4$	$L = 10$	$L = 20$	$L = 30$	$L = 50$	$L = 70$	$L = 100$
$p = 1/4$	2.13	1.08	1.00	1.00	1.00	1.00	1.00	1.00
$p = 1$	3.36	1.89	1.15	1.01	1.00	1.00	1.00	1.00
$p = 4$	6.42	3.99	2.62	1.72	1.34	1.08	1.02	1.00
$p = 9$	12.26	6.77	4.43	3.12	2.45	1.73	1.38	1.15
$p = 39$	62.26	27.60	14.86	9.62	7.62	5.75	4.75	3.81
$p = 99$	204.50	85.21	41.77	24.43	18.20	12.92	10.49	8.49

Figure 6: Table A.1. from Xin & Goldberg (2016). The entries correspond to the upper bound on $C(\pi_{r_\infty})/OPT(L)$, the equation for which was given in section 2.2.



```
                          L = 1                    L = 4                   L = 10                   L = 20
p = 1/4  [0.998942, 1.002529]   [0.996974, 1.00036]  [0.977414, 0.981339]  [0.955984, 0.959774]
p = 1    [1.010599, 1.015209]  [0.993505, 0.997409]  [0.994277, 0.998935]   [0.998063, 1.00185]
p = 4     [1.107267, 1.11427]  [1.024105, 1.029694]  [1.000466, 1.006346]  [0.967001, 0.973607]
p = 9    [1.254425, 1.262794]  [1.076347, 1.083141]  [1.024301, 1.031582]     [0.9471, 0.95391]
p = 39   [1.770301, 1.784212]    [1.399918, 1.4146]  [1.198947, 1.211743]  [1.058125, 1.070178]
p = 99   [2.377531, 2.406744]  [1.812189, 1.833989]  [1.460911, 1.482432]  [1.260767, 1.282518]
                         L = 30                   L = 50                   L = 70                  L = 100
p = 1/4  [0.965783, 0.968915]  [0.904305, 0.907976]  [0.908077, 0.910986]   [0.908201, 0.91195]
p = 1      [0.998846, 1.0035]  [0.812317, 0.816123]  [0.974239, 0.978369]  [0.902875, 0.906471]
p = 4     [0.990942, 0.99687]  [0.993298, 1.001342]  [0.989848, 0.996067]   [0.989381, 0.99542]
p = 9    [0.962164, 0.970789]  [0.986214, 0.994643]   [0.964314, 0.97303]  [0.916613, 0.974425]
p = 39    [0.77531, 0.993054]   [0.920627, 0.96581]  [0.947945, 0.963273]  [0.897742, 0.922348]
p = 99    [1.13209, 1.146057]  [0.981016, 1.003597]  [0.915789, 0.939686]   [0.926192, 0.94688]
```

Figure 7: 95% confidence intervals for $C(\pi_{r_\infty})/C(\pi_{PPO})$ under different lead times (L) and lost sales penalties (p), using our initial training methodology (from section 2.4). $h = 1$ and $\lambda = 1$ were fixed.

GitHub

https://github.com/fordmatt18/gym-inventory

References

[1] Raffin, Antonin & Hill, Ashley & Ernestus, Maximilian & Gleave, Adam & Kanervisto, Anssi & Dormann, Noah. (2019). Stable Baselines3. GitHub, GitHub repository, `https://github.com/DLR-RM/stable-baselines3`

[2] Scarf H. (1960) The optimality of (s, S) policies in the dynamic inventory problem. Mathematical Methods in the Social Sciences (Stanford University Press, Redwood City, CA), 196–202.

[3] Xie, Qiaomin (2021). Dynamic Programming for Average Reward MDP. ORIE 6590 course notes, Canvas. https://canvas.cornell.edu/

[4] Xin, Linwei & Goldberg, David (2016). Optimality Gap of Constant-Order Policies Decays Exponentially in the Lead Time for Lost Sales Models. Operations Research. 64. 10.1287/opre.2016.1514.