Airline Revenue Mangement

Tao Jiang
Laurel Newman

ORIE 6590 Spring 2021

Problem setup

- A central hub
- Flights to and from L different destinations
- Single- and two-leg itineraries
- High- and low-fare tickets
- All flights have same number of available seats
- Epoch of time within which all bookings must be made

Model (DTMC)

Problem inputs:

1. L, locations gives us $2L$ flight legs and $n = 2L(L+1)$ total possible bookings
2. κ, the seat capacity of each flight leg
3. τ, the deadline for all bookings

At each discrete time step, ≤ 1 customer arrives and attempts to make a single booking.

Randomly generated other data for each $L \in \{3, 5\}$ to fully describe the problem instance:

- Probabilities:
  - no customer arrives: 0.2
  - itinerary probabilities come from `numpy.random.uniform` scaled to sum to 0.8
  - low-fare bookings: 0.75 of the itinerary probability
  - high-fare bookings: 0.25 of the itinerary probabilities.
- Revenue: the cost of each low-fare booking comes from `numpy.random.randint` on the interval $[15, 50)$. High-fare booking cost 5 times low-fare ones.

Model (DTMC)

- The state space is the set of all possible available seats for every flight into and out of each location up to the full capacities.
- The action space is all possible binary vectors of length n which tells you whether a customer (with a specific fare and itinerary) is accepted or declined by the airline company.
- The one-step reward is the revenue gained from applying the predetermined action (of this time-step) to a customer who appears during this time-step.

Implementation

Used PPO, part of `stable-baselines3`.

Cloned `stable-baselines3` in order to change aspects of the source code, as well as adjust in-line hyperparameters.

To evaluate the performance of PPO as we made changes, we compared outputs for the small case $L = 3$, $\tau = 20$, $\kappa = 2$. Our criteria for a good RL algorithm were 3-fold:

- the policy/its performance had converged;
- the policy performed comparably to prior approaches to the problem;
- the variance between different random evaluations of the policy was low: its performance was consistent with respect to random arrivals.

Advantage estimation

With the aim of variance reduction, we explored four ways of estimating the advantage function:

- generalized advantage estimator
- simple cumulative moving average
- weighted moving average
- double exponential moving average

Advantage estimation

Recall that the estimate of advantage of the action a_t is defined as

$$\delta_t^V := rt + \gamma V(s_{t+1}) V(s_t),$$

where V is an approximate value function. A k-step estimate of the returns, minus a baseline is defined as [5]

$$\hat{A}_t^{(k)} := \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V.$$

As $k \to \infty$, we get $A_t^{(\infty)}$ which is the empirical returns minus the value function.

Advantage estimation: Generalized advantage estimator (GAE)

GAE is a truncated exponentially-weighted average of k-step estimators defined as follows [5]

$$\hat{A}_t = (1-\lambda)\sum_{l=0}^{k-1}\lambda^l A_t^{(1+l)} = \sum_{l=0}^{k-1}(\gamma\lambda)^l \delta_{t+l}^V$$

Convergence: 300k iterations. Reward: 715.866. Standard deviation: 28.967.

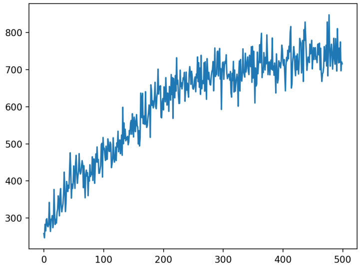


Figure 1: Mean vs. policy iterations

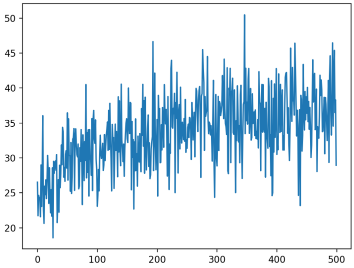


Figure 2: Std vs. policy iterations

Advantage estimation: Double exponential smoothing (DES)

DES puts more weights on the recent values, hoping to remove lag associated with moving average [4]. (Convergence: 200k iterations. Reward: 827.533. Standard deviation: 25.386.)

$$s_0 = 0.0, \qquad b_0 = \delta_t^V$$

$$s_l = \alpha\delta_{t+l}^V + (1-\alpha)\gamma(s_{l-1} + bl - 1), \qquad b_l = \beta(s_l - s) + (1-\beta) * b, \quad l = 1, dots, k$$

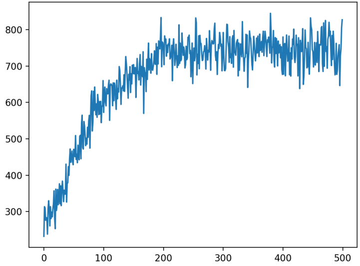


Figure 3: Mean vs. policy iterations

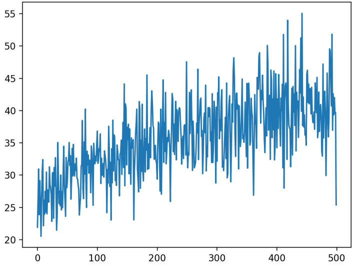


Figure 4: Std vs. policy iterations

Learning Rate

Default 0.0003, we tested larger ones to get faster convergence.

Ultimately used 0.003.

Mean revenue earned vs. number of PPO learn batches

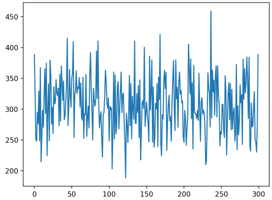


Figure 5: Learning Rate = 0.03

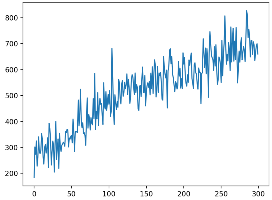


Figure 6: Learning Rate = 0.003

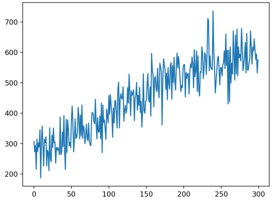


Figure 7: Learning Rate = 0.0003

Optimization Algorithms

Default `Adam` algorithm. `Adamax` is a variant of Adam with infinity norm [3]. `SGD` is a state-of-art algorithm for solving optimization problems [6]. `Adagrad` includes more geometric information from earlier iterations [2].

Mean revenue earned vs. number of PPO learn batches

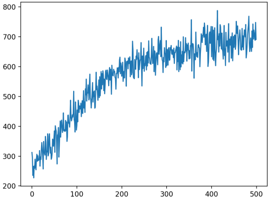


Figure 8: `Adamax`

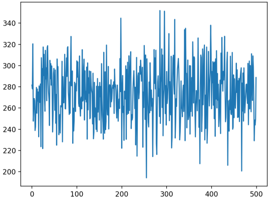


Figure 9: `SGD`

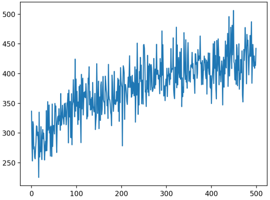


Figure 10: `Adagrad`

Other Hyper-parameters

A few other hyper-parameters also provide interesting insight into our model:

- Buffer length: only changed runtime of algorithm, so we went with a middling value that helped PPO converge quickest
- GAE parameter: large λ (closest to Monte-Carlo, further from Bellman) ensured variance did not grow over time
- Discount factor: values much smaller than normally recommended in the literature led to the fastest convergence to largest solutions

Insights

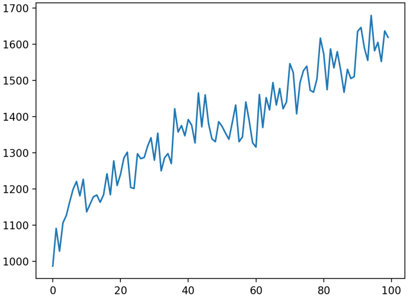
- Discount factor: since the problem works over very small, and perpetually decreasing, future window, found that lower discount factors than commonly suggested in the literature worked best for our problem instance
- Adjusting epoch length: for fixed seat capacities, higher epochs lead to policies that favor high-fare and single-leg seats. Following Adelman, we doubled epoch and seat capacity together, preserving their ratio, which led to a roughly linear growth of the means.
- Punishment: any reasonable number of PPO iterations ensures that we almost always sell all the seats. Therefore a punishment factor to the reward function for finishing an epoch with seats remaining made little change to the final policies.

Numerical results and conclusion

L	τ	κ	PPO		# timesteps	# episodes	DBPC Mean (std. err.)
			Mean	Std			
3	20	2	764.2925	6.989	800,000	500	567.78 (20.54)
	50	6	1790.6675	13.815	800,000	500	1,759.91 (33.76)
	100	12	3744.49875	20.998	800,000	500	3,730.04 (53.87)
	200	24	7551.33625	33.774	800,000	500	7,683.04 (70.09)
	500	61	20884.3325	65.567	800,000	500	19,793.50 (132.23)
5	20	1	112.816	3.322	500,000	500	486.92 (17.86)
	50	4	1233.67	17.022	1,000,000	500	1,874.34 (36.70)
	100	8	3237.696	31.583	1,000,000	500	3,905.97 (50.49)
	200	16	7369.488	50.794	1,000,000	500	8,109.55 (73.00)
	500	42	21938.526	95.407	1,000,000	500	21,189.10 (125.73)

Numerical results and conclusion

- For $L = 3$, our results are competitive with the results in Adelman [1].
- For $L = 5$, our results did not quite reach convergence (and further, that PPO performed very poorly when the seat capacities were 1).
- More iterations would likely improve the performance of the $L = 5$ cases. (e.g. for $L = 5$, $\tau = 50$ and $\kappa = 4$, the result improves from 1233.67 to 1619.12 if we increase the iterations from 1,000k to 5,000k)



References i

Daniel Adelman.
Dynamic bid prices in revenue management.
Operations Research, 55(4):647–661, 2007.

John Duchi, Elad Hazan, and Yoram Singer.
Adaptive subgradient methods for online learning and stochastic optimization.
Journal of machine learning research, 12(7), 2011.

Diederik P Kingma and Jimmy Ba.
Adam: A method for stochastic optimization.
arXiv preprint arXiv:1412.6980, 2014.

Patrick G Mulloy.
Smoothing data with faster moving averages.
Stocks & Commodities, 12(1):11–19, 1994.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel.
High-dimensional continuous control using generalized advantage estimation.
arXiv preprint arXiv:1506.02438, 2015.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton.
On the importance of initialization and momentum in deep learning.
In International conference on machine learning, pages 1139–1147. PMLR, 2013.