Airline Revenue Management using Advantage Actor Critic

Tyler Sam and Samuel Tan

ORIE 6590

May 16, 2021

Please find our code here: `https://github.com/samstan/ORIE6590-Airline`

1 Introduction

In this project, we consider a simplified airline revenue management task using bid-price control. Bid-price control (and revenue management in general) is a common application of operations research, wherein the company must delicately balance accepting arriving bids to earn revenue immediately while conserving resources for future bids. The structure of this problem makes it particularly suitable to model using Markov Decision Processes, as this report will explain.

As a motivating example, consider a scenario involving a fictional airline company called Ithaca Flies. Ithaca Flies only operates three direct flights: Ithaca to NEK, Ithaca to Detroit, and Detroit to LA. Additionally, one may elect to fly from Ithaca to LA via Detroit. Each flight has economy and business seats. Ithaca Flies implements a bid-price control policy to sell tickets on these flights, where the bid-prices are the fixed trip fares. The goal of the controller is to maximize revenue. Previous approaches to the general bid-price control problem for airline revenue management used linear programming [4]. In this report, we provide several deep reinforcement learning (RL) approaches and compare their performances with those of previous approaches.

2 Background

We follow the formulation described in [4], wherein this problem is framed as a finite-horizon Markov Decision Process (MDP). In that context, we consider a scenario in which the relevant airline has a central hub, and L other locations. There are single-leg itineraries between the hub and each location in both directions, as well as two-leg itineraries between different locations. Therefore in total, we have $m = 2L$ distinct flights, also called resources. As we additionally distinguish between low and high class fares for each

itinerary, we therefore have $n = 2(m + L(L-1))$ distinct itineraries, which also characterizes the type of each customer.

As part of this formulation, we specify a matrix A, where the ijth element denotes the amount of resource i a customer of type j uses, e.g. a business seat on the Ithaca to LA via Detroit flight requires a business seat on both flights.

We also specify for each class and time step t, a probability $p_{t,j}$ which is the probability that a customer of type j arrives at time t. Note we restrict that only one customer arrives at each period.

2.1 MDP Formulation

Given these parameters, the MDP is defined with:

- State space: non-negative integer for each resource. Initialized at a specified capacity for each resource c_i (i.e. the number of seats on each flight).
- Action space: (0,1) for each class j, denoting whether controller would accept a class j customer if one arrives. We enforce that there are sufficient resources if $a_j > 0$ (according to the A matrix).
- Rewards: given state s and action a, earn revenue f_j if a class j bid is accepted.
- Transitions: given state s and action a, with probability $p_{t,j}$ the next state will be $s - A^j a_j$, where A^j denotes the jth column of A.

2.2 Existing Approaches

As previously mentioned, the original MDP formulation for bid-price control was given in [4]. However, the authors approached the problem with linear programming due to the curse of dimensionality, using column generation to address the high dimensionality. Specifically, they use an affine functional approximation to the optimal value function and derive bounds on the difference in revenue compared to the optimal policy. Airline revenue management was a specific example of bid-price control that was analyzed.

Similarly, [11] addresses the more specific bid-price problem applied to airline revenue management. It uses Lagrangian relaxation to solve the optimization problem, taking into account the time until departure as well as the remaining leg capacities.

2.3 Advantage Actor Critic

Advantage Actor Critic (A2C) is a popular variant of the actor-critic framework for reinforcement learning [8]. To remind the reader, actor-critic algorithms maintain two sets of parameters, one for the neural network approximating the value function ("critic") and another for the the neural network representing the policy ("actor"), which uses the critic for updates. Specifically, at each iteration, first, the parameters of the policy neural network are updated given the chosen action and the "critic" neural network. Then, using the received reward, the correction for the value function is computed and

used to update the parameters for the "critic" neural network. This process is repeated for a fixed number of iterations or until a convergence criteria is reached. Also, since $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$, variance reduction is inherent in the method as $V(s_t)$ can be thought of as a baseline function.

In the case of A2C, the updates use an estimate for the advantage function (the generalized advantage estimator), given by

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \approx \sum_{i=0}^{k-1} \gamma^i r_{t+1} + \gamma^k V(s_{t+k}) - V(s_t).$$

We further note that some refer to the synchronous version of Asynchronous Actor Critic (A3C) as A2C, in which there are multiple workers but only perform global updates after all workers are done. However, in the Stable Baseline3 A2C GitHub repository and documentation, there is no feature for varying the number of workers, so the implementation of A2C that is used in this report is the one with only one worker [9].

2.4 Recurrent Neural Networks

As we use recurrent neural networks (RNN) in our neural network architectures, we briefly introduce them here. An RNN [10] is a special type of neural network that has temporal connections in the hidden layers. This enables an RNN to exhibit temporal behavior and process sequences of inputs better than a vanilla neural network. This is achieved by having a layer that keeps an internal hidden state.

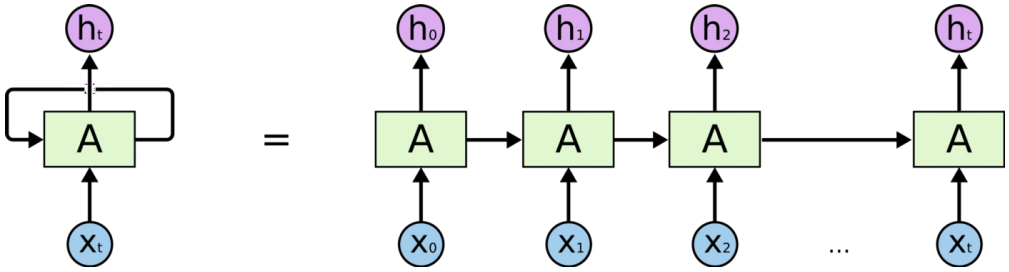

Figure 1: An illustration of an RNN [1].

Here, the x_i are the sequence of inputs, A is a cell, and h_i is the hidden layer at time i. For every x_i that is passed into the RNN, the cell processes it and updates the hidden state h_i.

Because of the RNN's ability to process sequential data, we hypothesize that using it to predict the value function given a state will lead to better performance compared to traditional MLPs. Despite the success of RNNs, vanilla RNNs often run into the vanishing gradient and exploding gradient problems. This is an issue where during gradient descent, the updates can be effectively zero or unbounded since RNNs build up long chains of derivatives during backpropagation. To alleviate this issue, many use a long short-term memory (LSTM) network [10].

2.5 Long Short-Term Memory

An LSTM essentially keeps some memory cells that are more resistant to change. The main components of an LSTM are the cell, input gate, output gate, and forget gate. LSTMs have proved to be superior to vanilla RNNs due to their ability to avoid the vanishing and exploding gradient problems.

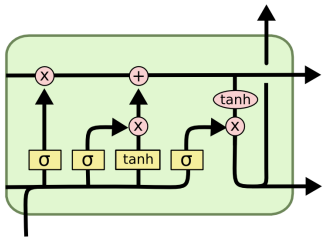

Figure 2: An illustration of an LSTM cell [3].

The main idea behind LSTMs can be found in the horizontal black line near the top of the cell. This serves as the memory of the LSTM and is the cell state. Since it has minimal interactions with the inputs, it can retain its state for a long period of time.

Aside from all of the advantages of LSTMs, others have recently found success in using LSTMs as the neural network in RL algorithms. Specifically, in *The LSTM-Based Advantage Actor-Critic Learning for Resource Management in Network Slicing With User Mobility*, Li et al. utilizes an LSTM in both the neural network approximating the policy and the neural network approximating the value function; both neural networks share the LSTM and then include a couple of feed forward layers each [7].

2.6 Gated Recurrent Unit

One downside of LSTMs is that they are more complex than vanilla RNNs. Due to computational issues with training, we use a newer and simpler modification of RNNs, the gated recurrent unit (GRU) [5]. This architecture has fewer parameters to fit but still achieves similar results compared to LSTMs. A GRU achieves a simpler architecture by reducing the number of gates. It uses an update and reset gate to control its memory cells.

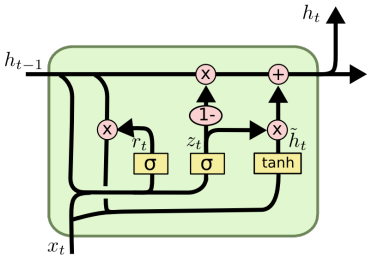

Figure 3: An illustration of a GRU cell [2].

3 Methods

3.1 Model Parameters

The parameters varied for the underlying MDP are the number of locations L, the time horizon τ, and the capacities for each resource c.

Note that the fares f and probability A of demands were randomly generated. According to [4], fare prices for the lower class were generated discrete-uniformly from $\{15, 49\}$, and the higher class fares were five times the corresponding lower class fare for the same itinerary. For demand probabilities, we also enforce that there is always a 0.2 probability that no customer arrives, where as the other probabilities were generated randomly where 75% of demand was for lower class while 25% of demand was for higher class. For the specific values, please see the codebase.

3.2 Neural Network Architecture

We consider three sets of neural network architectures, where the policy and value function each require a (not necessarily separate) network. As our input is a vector, we only focus on multilayer perceptrons and variations thereof. The base model has two fully connected layers of 64 hidden units for both the policy and value function. The shared model adds an additional initial layer of 64 hidden units which are shared between the policy and value networks. The later policy and value units remain two fully connected layers of 64 hidden units. Finally, the GRU model uses a two-layer two-hidden-unit network for the policy function and a two-layer five-hidden-unit GRU network for the value function. Due to computational constraints related to the GRU, we had to (arbitrarily) choose only one of the two networks to include a GRU and chose to drastically reduce the size of the other.

4 Results

The goal of this work is to compare our results with the other Airline Revenue Management team's results as well as the benchmarks given in table 6 of *Dynamic bid prices in revenue management* [4]. However, one major issue with comparing our results to the benchmark values is we likely use different parameters, that is, the profit for each flight and the transition probabilities, since Adelman does not state the parameters used in his report. For example, the author states that the fixed fairs are drawn uniform on the range $\{15, 49\}$ but does not state what the realized values were in the results. Hence, we cannot draw any concrete conclusions from our results compared to his.

Our results, summarized in Table 1, were ran with `n_steps = 5` for 500,000 learning steps with a learning rate of 0.0007 (we experimented with different learning rates, learning steps, and other neural network architectures but found no noticeable improvements). The reported results are averaged over 500 runs of `evaluate`.

L	τ	c	Base MLP	Shared	GRU	DBPC	PPO
3	20	2	364.29(150.89)	317.01(139.27)	188.17(108.42)	567.78(20.54)	764.39(6.98)
	50	6	845.68(247.119)	706.30(235.96)	737.47(228.16)	1759.91(33.76)	1790.66(13.81)
	100	12	1546.01(341.09)	1556.57(342.26)	1543.84(336.87)	3730.04(53.87)	3744.49(20.99)
	200	24	2913.53(488.92)	2901.09(487.35)	3107.65(521.16)	7683.04(70.09)	7551.33(33.77)
	500	61	8189.19(850.59)	7257.78(754.45)	8089.46(832.38)	19793.50(132.23)	20884.33(65.56)
5	20	1	116.18(72.08)	106.38(72.87)	93.72(69.54)	486.92(17.86)	654.81(9.28)
	50	4	585.96(252.78)	608.05 (243.24)	654.85(250.75)	1874.34(36.70)	1233.67(17.02)
	100	8	1489.12(457.99)	1641.90(449.97)	1650.27(459.54)	3905.97(50.49)	3237.69(31.58)
	200	16	3660.86(792.38)	3942.27(775.49)	3625.08(742.68)	8109.55(73.00)	7369.48(50.79)
	500	42	8698.46(1266.06)	8292.50(1140.22)	10985.99(1419.44)	21189.10(125.73)	21938.52(95.40)

Table 1: Our version of Table 6 also including the PPO team's results. The parameters are consistent between our results and those of the other team. DBPC is dynamic bid price control, the result of [4], which uses different parameters.

5 Discussion

Despite having different parameters from Adelman, our results fall quite short from both his and the other team's results. Our best model reaches about half of the values given by the other team/Adelman. Also, the standard deviation of our rewards are significantly larger compared to the other results. However, when we evaluated our policy for 500 runs, the maximum reward was close to Adelman's and the PPO team's results (in the first row, for the Base MLP A2C model, the maximum reward was 792). We hypothesize that this large standard deviation is due to the model not having "learned enough" as the actions the model chooses do not vary much depending on the state. Hence, we believe that these models can still perform competitively against the baselines given in Adelman and the PPO team, but they require significantly more training time in order for our A2C policies to learn the correct strategies.

6 Conclusion and Future Directions

In our deep reinforcement learning approach to bid-price control, we found that A2C does not seem to be able to learn a good policy. This may be due to slow convergence, as mentioned in Section 5, and poor choices of hyperparameters. Although we are disappointed, we also note that this is not a surprising result. Indeed, the A2C library on which the `stable-baselines3` implementation is based mentions that their implementations are very sensitive to hyperparameters and may simply fail to work (see the Github repository for `pytorch-a2c-ppo-acktr-gail`) for RL tasks. On the other hand, PPO seems to perform quite competitively compared with the linear program approaches.

Given these results, we conclude that A2C may not be a suitable approach for the task of bid-price control for airline revenue management. An exception may be made if extensive computation power is available, although the performance for this case was not tested and only extrapolated from our results.

For future work, one thing to try for future work is varying the hyperparameters such as the learning rate and the neural network architectures. In particular, we seem to have been limited by our modest computational power, so using GPUs may help achieve competitive results. Other similar RL approaches such as Soft Actor Critic [6], which aim to maximize entropy in addition to expected reward, are also of interest.

References

[1] Introduction to recurrent neural networks. URL: https://miro.medium.com/max/1838/1*NKhwsOYNUT5xU7Pyf6Znhg.png.

[2] Python deep learning tutorial: Create a gru (rnn) in tensorflow. URL: https://www.data-blogger.com/wp-content/uploads/2017/08/gru.

[3] Understanding lstm networks. URL: https://colah.github.io/images/post-covers/lstm.

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

[5] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL: http://arxiv.org/abs/1412.3555, arXiv:1412.3555.

[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[7] Rongpeng Li, Chujie Wang, Zhifeng Zhao, Rongbin Guo, and Honggang Zhang. The lstm-based advantage actor-critic learning for resource management in network slicing with user mobility. *IEEE Communications Letters*, 24(9):2005–2009, 2020. doi:10.1109/LCOMM.2020.3001227.

[8] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[9] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

[10] Alex Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *CoRR*, abs/1808.03314, 2018. URL: http://arxiv.org/abs/1808.03314, arXiv:1808.03314.

[11] Huseyin Topaloglu. Using lagrangian relaxation to compute capacity-dependent bid prices in network revenue management. *Operations Research*, 57(3):637–649, 2009.