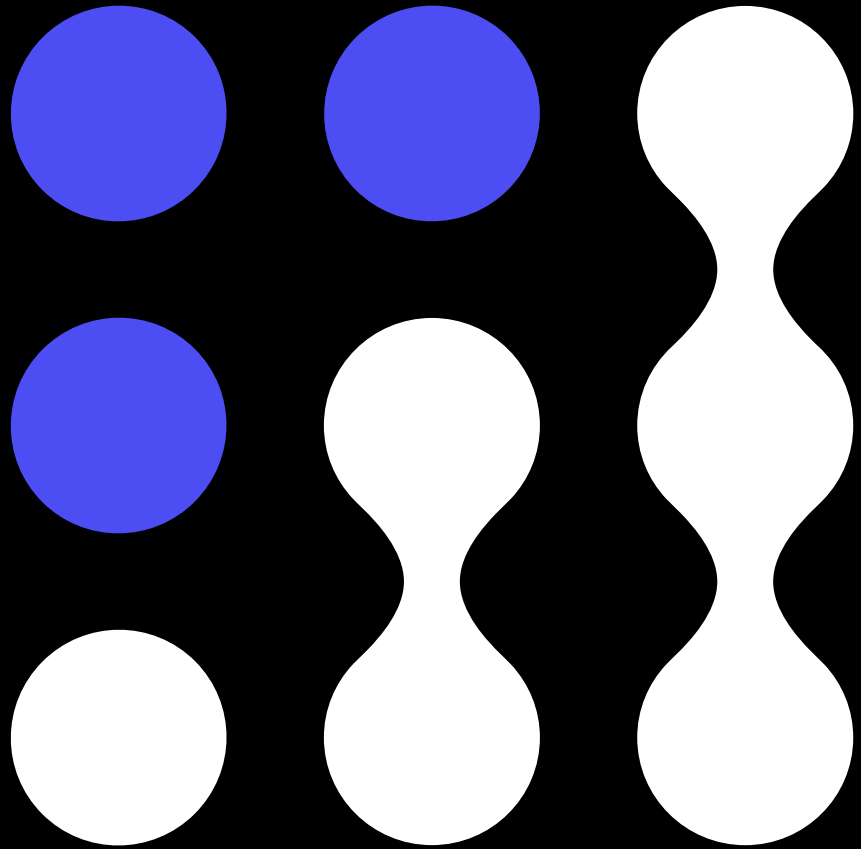
FLYR

Generating optimal bid prices via reinforcement learning with batch and shape constraints

Akhil Gupta
Naman Shukla

AGIFORS 08 May 2024



Akhil Gupta
Senior Applied Scientist

Akhil is a senior applied scientist at FLYR with seven years of experience working across a variety of industries (healthcare, retail, airline) in the data science domain. Akhil specializes in combining the fundamentals of optimization with latest advancements in machine learning to tackle real-world data challenges.

Naman Shukla
Product Manager, Applied Science

Naman is product manager of data science at FLYR, where he leads a team of data scientists and engineers to build products for revenue management on both passenger and cargo domain. Naman has expertise in research and development of end to end machine learning solutions that powers RM systems for airlines and travel providers.

Cameron Young
Senior Applied Scientist

Ryan Shiroma
Senior Applied Scientist

Nick Dulchin
Applied Scientist

Jon Ham
Senior Manager, Applied Science

FLYR

Table of contents

FLYR

Introduction and context

Machine learning in revenue management

Learning the optimal bid price policy using data science

Moving beyond forecast and optimize: The need for **reinforcement learning** approaches

Significance of deep nets

Use a massive amount of context to make a decision or prediction. Any input signal can be linked with any neuron, with any combination of non-linear interaction effects allowed.

Automatically detect relevant features and assign correct weights

Beyond supervised learning

Supervised learning inherently assumes that the labels are the ground truth for machine to learn. Though in pricing it's tricky to define such labels

This approach can only produce policies that previously have been applied.

Bad or missing training data

Generally adopted forecasting techniques or supervised techniques to come up with an optimal bid price requires high quality, regularly spaced time series data.

Models are sensitive to missing data, may even fail to train

FLYR

Deep Q-Networks (DQN)

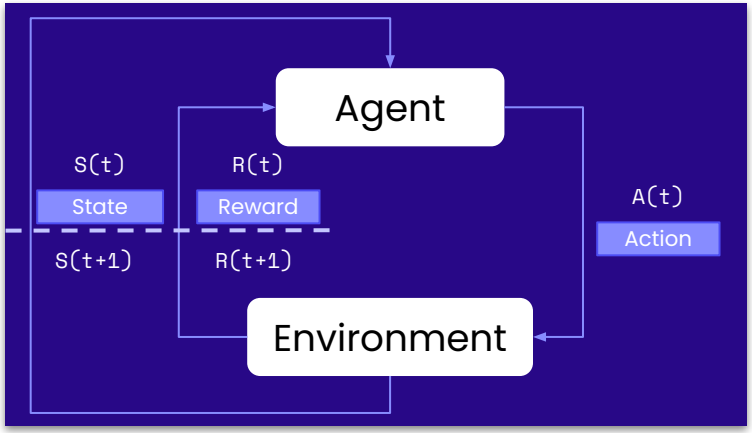
Introduction to deep reinforcement learning

What is Q learning?

Q-Learning is an RL approach that attempts to learn the optimal policy for a task by estimating the **action-value function**. This function maps a state and action to the expected discounted reward that can be earned by following that policy from now until the end of the episode.

About DQN

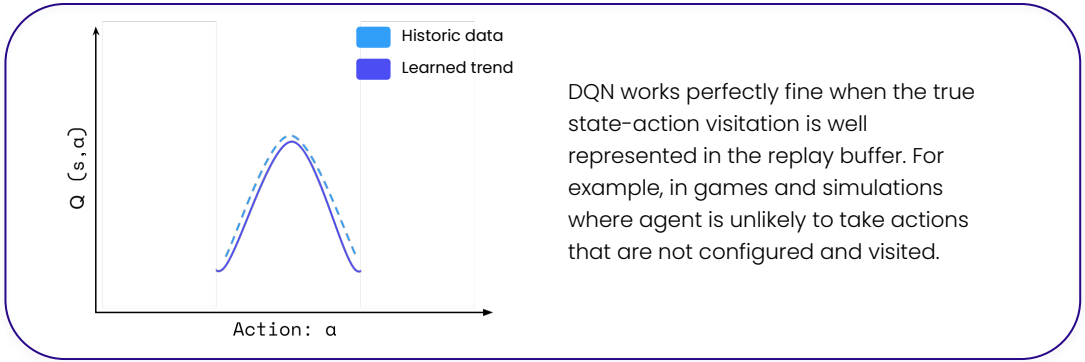
- DQN employs experience replay, storing transitions (state, action, reward, next state) in a replay buffer to break correlations and stabilize training.
- DQN also uses a target network to stabilize training, with periodically updated parameters to provide more stable targets for the Q-value predictions.



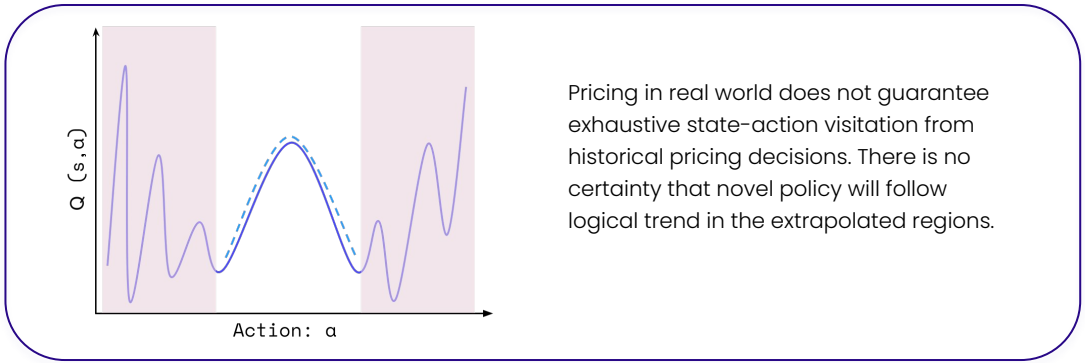
$$Q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k} | s_t = s, a_t = a\right]$$

$$= \mathbb{E}_\pi\left[r_t + \gamma\mathbb{E}_\pi\left[Q_\pi\left(s',a'\right)\right] | s_t = s, a_t = a\right]$$

[1] Human-level control through deep reinforcement learning by Mnih et al (2015)



DQN works perfectly fine when the true state-action visitation is well represented in the replay buffer. For example, in games and simulations where agent is unlikely to take actions that are not configured and visited.



Pricing in real world does not guarantee exhaustive state-action visitation from historical pricing decisions. There is no certainty that novel policy will follow logical trend in the extrapolated regions.

Extrapolation Error

Beyond training batch data

What is extrapolation error?

Extrapolation error is an error in off-policy value learning which is introduced by the mismatch between the dataset and true state-action visitation of the current policy.

Causes of extrapolation error

- Absent data
- Model bias
- Training mismatch

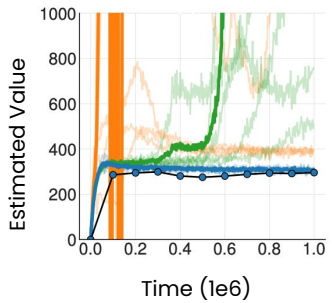
FLYR

Batch-Constrained Q-Learning (BCQ)

BCQ augments the Q-Learning process by restricting the available actions to only those that were likely to occur historically. This stabilizes learning and results in more accurate Q estimates.

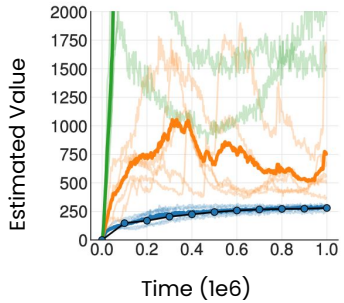
Evaluation of batch constraints on Hopper-v1 environment [2]

Final Buffer



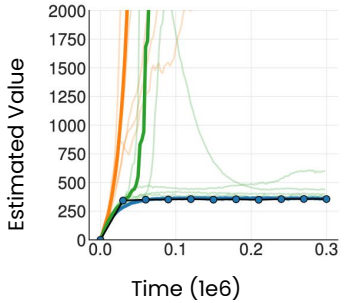
Agent learns from the final replay buffer gathered by training a DDPG agent over a million time steps [3]

Concurrent



Agent learns concurrently, with the same replay buffer, as the behavioral DDPG policy

Imitation



Agent learns from a dataset collected by an expert policy in imitation method

[2] Off-Policy Deep Reinforcement Learning without Exploration by Fujimoto et al (2019)
[3] Continuous control with deep reinforcement learning by Lillicarp (2015)

FLYR

Scientific approach

BCQ for next sellable seat bid price

Technique to set optimal bid prices

BCQ - DQN approach

We use deep reinforcement learning with batch constraints to optimize for the next sellablable seat's bid price, ensuring that the bid price policy is stable and more grounded to historical pricing decisions

Overall construct

- Two fully-connected neural networks are used for learning the Q-values (training and target)
- Bid-price action space is discretized into fixed number of buckets (*design choice*) to stabilize the agent training
- Sub-networks in our framework are jointly trained with a single optimizer to tackle the pricing task

Algorithm 1 Batch constraint bid price

Input: Replay buffer $\mathcal{B}$, horizon T, target network update rate τ , mini-batch size N, number of sampled actions n in action space A and threshold λ.
Initialize Q-network Q_θ with random parameters θ, probability network Φ_ω with random parameters ω and target networks $Q_{\theta'}$ with $\theta' \leftarrow \theta$

for $t \leftarrow 1$ to T **do**
  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  Sample n actions: $\{\ a_n \sim A\ \}$
  Select legal actions: $a_i = \Phi_\omega(s|\lambda)$
  Set value target:
  $y = r + \gamma Q'_\theta(s', argmax_{a_i}(Q_\theta(s, a_i)))$
  $\theta \leftarrow argmin_\theta \sum (y - Q_\theta(s, a))^2$
  $\omega \leftarrow argmin_\omega \sum (a - \Phi_\omega(s|\lambda))^2$
  Update networks:
  $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$
  $\omega' \leftarrow \tau\omega + (1 - \tau)\omega'$
end for

FLYR

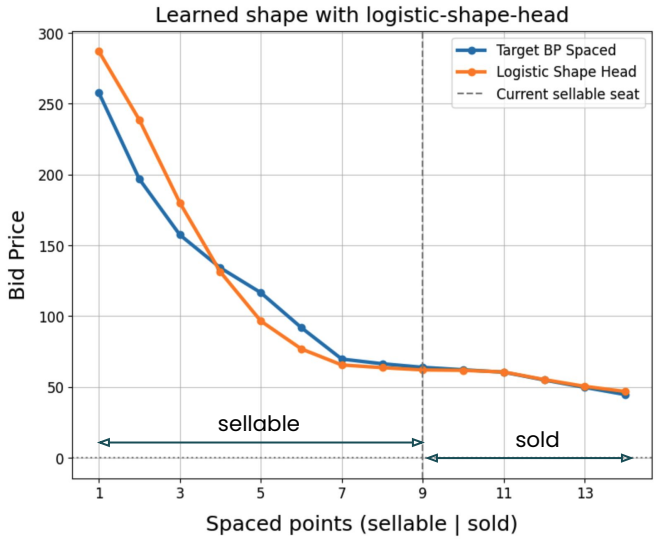
Learning the full bid-price curve

From BCQ to bid-price vector

Shape Constraining (Supervised Learning)

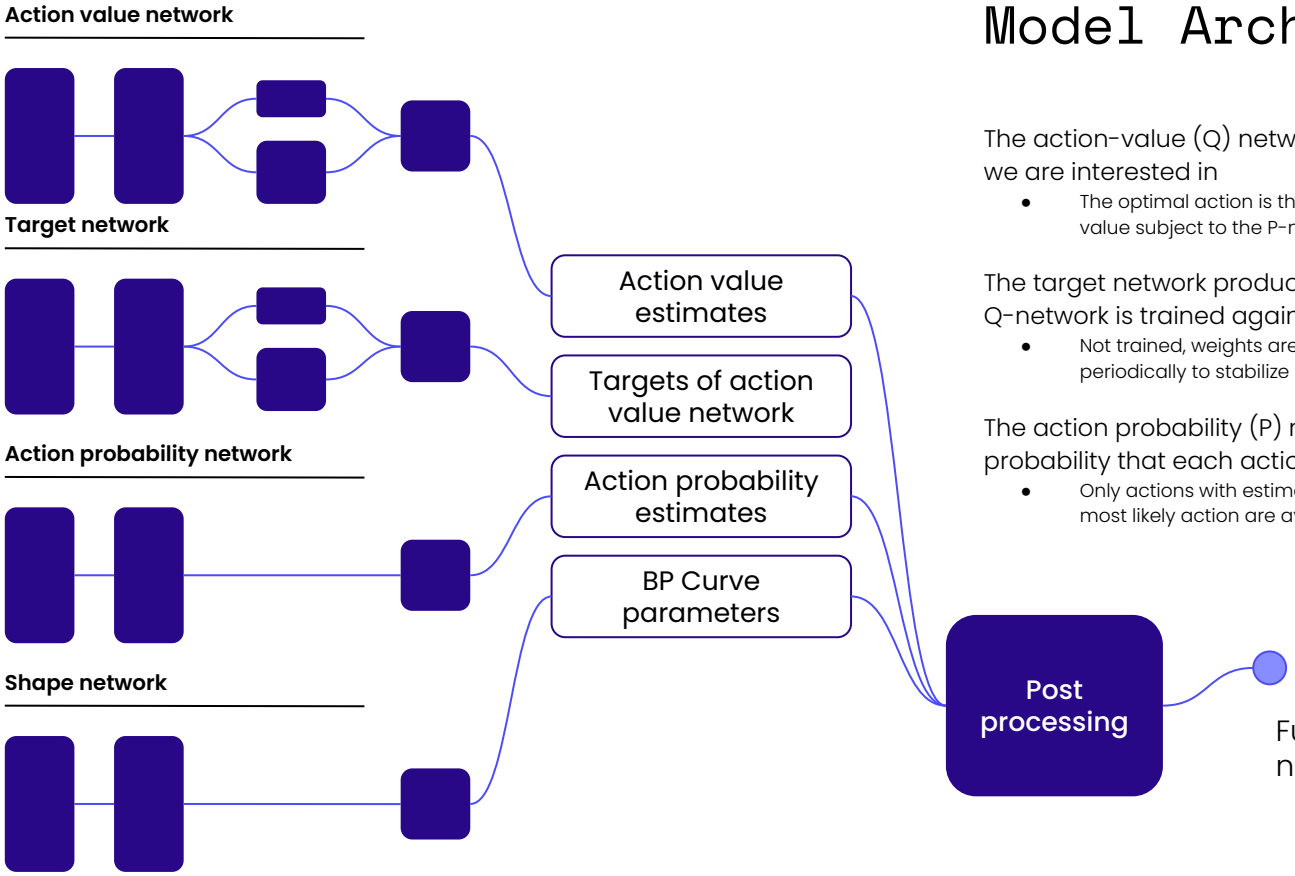
To generate the full bid price vector, we propose to fit two Generalized Logistic Functions (*design choice*) using mean-squared-error (MSE) loss.

- Two logistic functions are desirable in order to accurately model the inflection points that occur both ahead-of and behind the next sellable seat
- During inference, both the shape curves need to be adjusted to the "optimal next-sellable BP", the optimal output from BCQ: shift the entire curves up/down (additively) by the change in next sellable BP (between learned and optimal BP)



$$y = A + \frac{(B - A)}{(1 + Ce^{-1*Dx})^{1/E}}$$

FLYR



Model Architecture

The action-value (Q) network produces the final output we are interested in

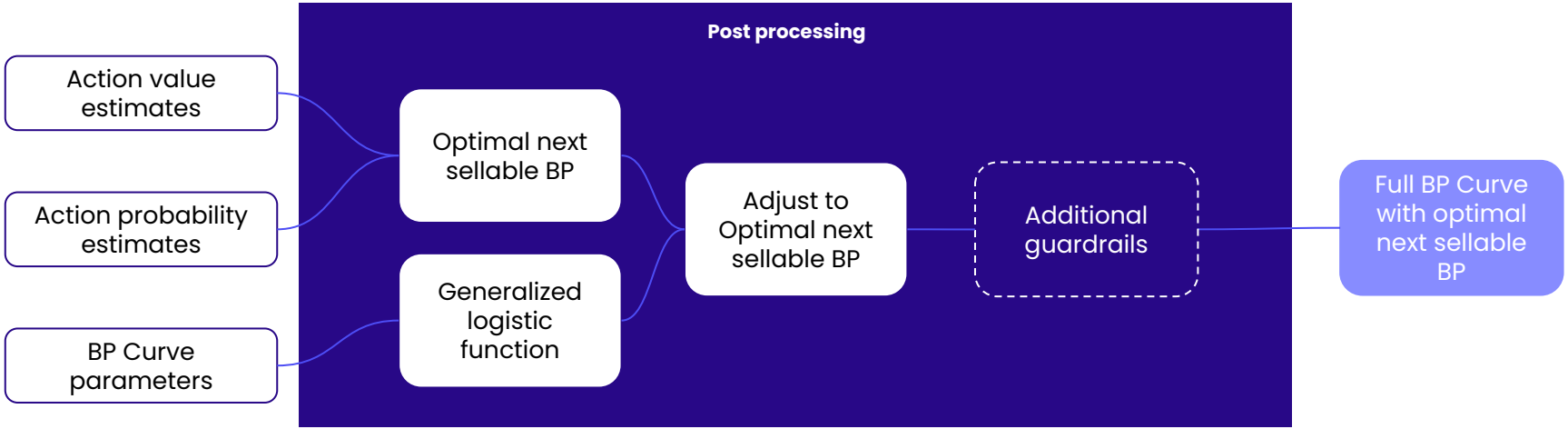
- The optimal action is the one with the highest estimated Q value subject to the P-network constraint

The target network produces the targets that the Q-network is trained against

- Not trained, weights are copied from the Q-network periodically to stabilize learning

The action probability (P) network estimates the probability that each action would be taken historically

- Only actions with estimated probabilities of at least 0.5x the most likely action are available



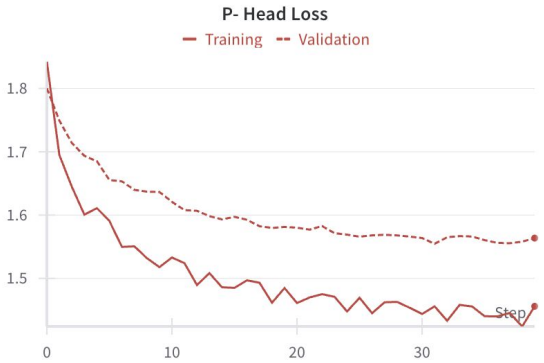
- Going from optimal bid price for next sellable seat to complete bid price vector involves constraining the learned shape of logistic function to match with (a) sellable spaced and (b) sold spaced bid prices.
- Generalized logistic function is a good candidate for parameterizing the bid price curve with few learnable function parameters; it is flexible enough to model the different shapes we'd expect.
- Optional component to apply additional guardrails to make sure the business rules are followed.

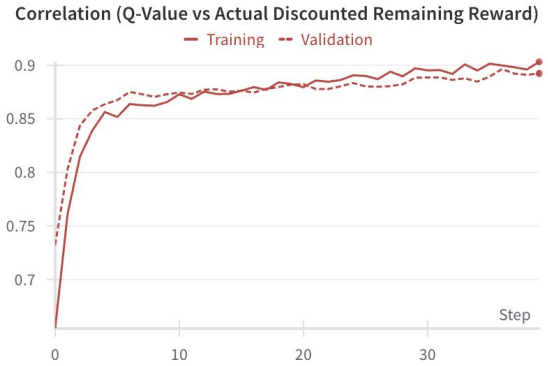
FLYR

Results and quality review

Model stability and loss trends

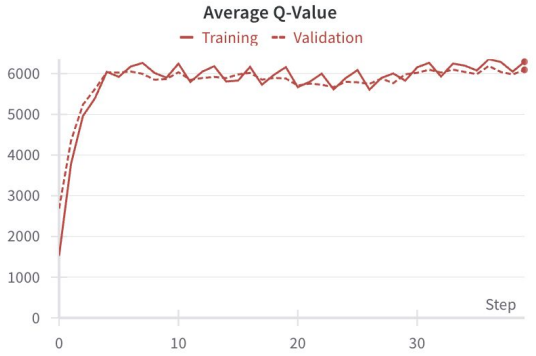
Training and validation plots





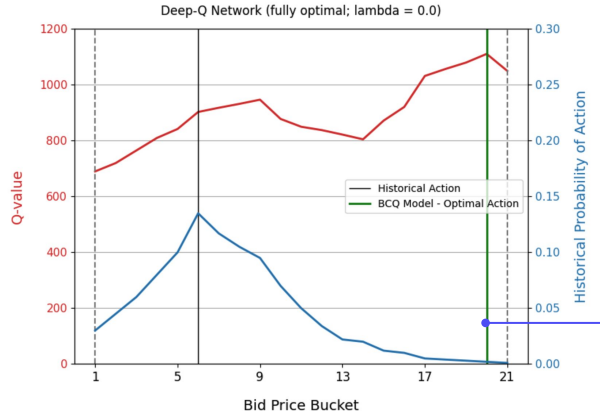
Key observations

- Batch constraining helps the loss value for action probabilities (both training and validation) to reach stability at the end of the training cycle. There is no sign of overfitting.
- The correlation between Q-values and actual discounted remaining rewards converges to roughly 0.9
- Q-value convergence is stable as expected

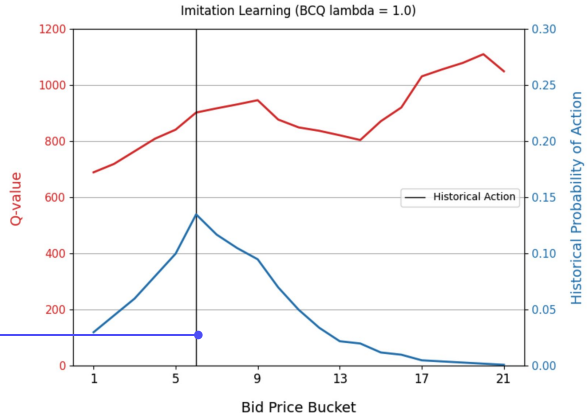


Batch constraining in action

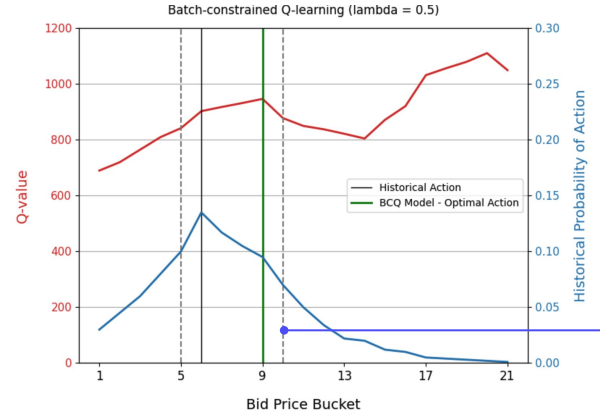
Importance of batch constraint in limiting legal action-space



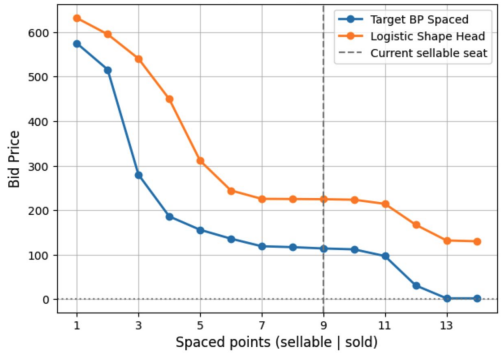
Without constraints, DQN suffers from extrapolation error and can end up picking highly sparse or unseen (historical) price actions.

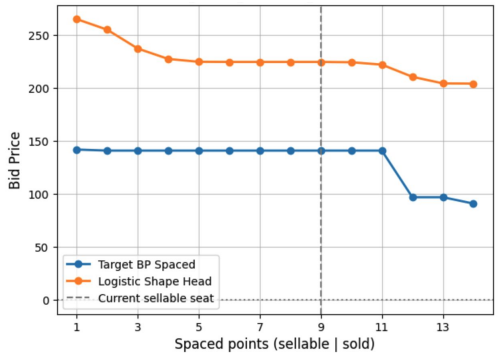


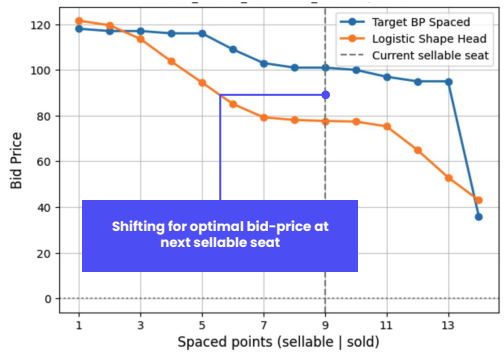
P-network learns the probability mass around the historical action.

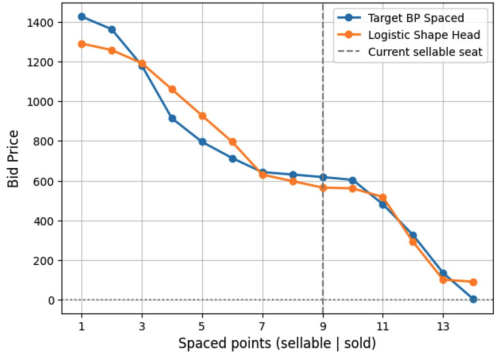


Relaxation of the lambda (probability threshold) with BCQ helps in exploring nearby potentially optimal price actions (bucket 9 with higher Q than historical bucket 6).









Flexibility of shapes

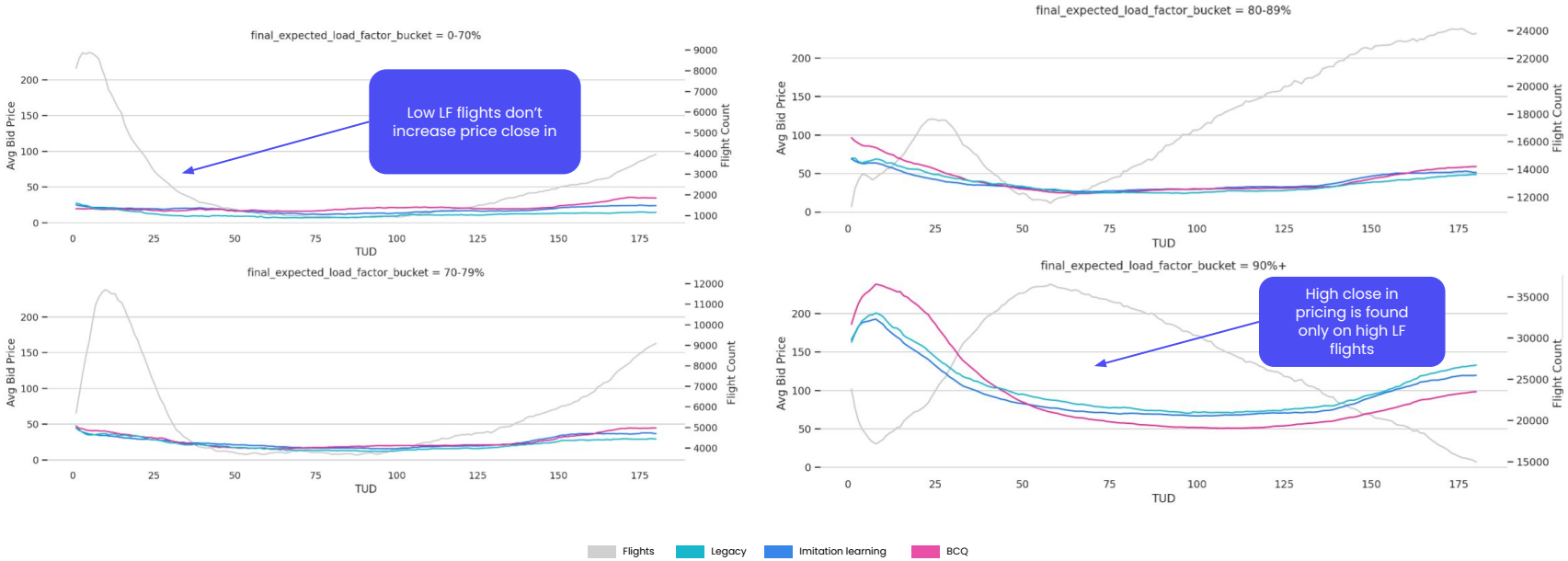
Generalized logistic function

- Chosen function is flexible enough to learn a variety of S-shaped curves, important to model the growth and drop in bid prices on the same day.
- The designed shape-network is invariant to changes in the absolute values of bid prices - able to learn the shape parameters accordingly.
- Need for two logistic functions is clear from the different shape patterns (and the inflection points) on sellable and sold sides.

FLYR

Global policy analysis

Aggregated view of policies across final expected load factor buckets



Heatmap view of the results

The following results are across time until departure and load factor

TUD-LF Price Heatmap

load_factor_bin_by_5 / optimal_bp

tud_days_bin	0-0.09	0.10-0.19	0.20-0.29	0.30-0.39	0.40-0.49	0.50-0.59	0.60-0.69	0.70-0.79	0.80-0.89	0.90+
170-180	50	68	92	117	128	111	104	93	117	108
160-169	47	62	85	112	119	107	95	82	114	112
150-159	41	54	76	104	113	96	90	82	119	117
140-149	37	48	67	95	113	94	103	85	118	126
130-139	33	43	60	88	112	96	117	88	122	129
120-129	31	40	55	80	108	103	119	96	128	135
110-119	30	37	51	74	109	110	131	116	131	150
100-109	29	34	48	68	105	119	138	139	141	157
090-099	27	32	44	64	95	142	157	140	152	168
080-089	25	29	39	60	90	154	184	154	158	177
070-079	23	26	34	53	90	152	202	169	178	175
060-069	21	24	30	45	80	138	223	206	202	179
050-059	19	22	27	41	74	130	230	280	241	202
040-049	17	21	26	37	68	123	210	340	332	233
030-039	15	20	27	35	59	109	182	299	427	309
020-029	13	18	27	34	46	82	150	243	405	402
010-019	11	13	21	30	38	52	90	172	301	388
000-009	10	12	15	18	25	34	46	80	167	280

DQN with BCQ is learning TUD-LF relationship at an aggregate level

Bias Heatmap vs. Legacy

load_factor_bin_by_5 / me

tud_days_bin	0-0.09	0.10-0.19	0.20-0.29	0.30-0.39	0.40-0.49	0.50-0.59	0.60-0.69	0.70-0.79	0.80-0.89	0.90+
170-180	9	-23	-43	-69	-69	-72	-48	-13	-7	-12
160-169	8	-24	-46	-76	-74	-76	-56	-16	-5	-9
150-159	6	-25	-42	-71	-75	-74	-58	-17	-8	-9
140-149	4	-23	-42	-69	-73	-74	-49	-20	-8	-5
130-139	3	-21	-40	-59	-66	-71	-45	-21	-6	-5
120-129	5	-17	-40	-58	-71	-72	-52	-30	-9	-7
110-119	6	-13	-38	-59	-71	-72	-49	-32	-18	1
100-109	7	-8	-33	-57	-66	-70	-50	-37	-20	0
090-099	8	-5	-25	-52	-63	-57	-49	-51	-22	-9
080-089	7	-3	-18	-44	-56	-36	-35	-50	-30	-3
070-079	5	-3	-14	-32	-47	-29	-17	-53	-44	1
060-069	5	-3	-12	-21	-35	-30	-1	-38	-58	-2
050-059	6	-4	-11	-14	-17	-20	19	22	-37	-9
040-049	6	-4	-9	-9	-3	6	27	81	36	-16
030-039	7	-2	-7	-6	3	22	42	78	120	19
020-029	7	1	-3	-4	-1	16	47	77	132	59
010-019	5	1	0	-0	-1	1	15	53	105	41
000-009	-2	-7	-3	-1	3	3	4	15	56	21

DQN with BCQ is cheaper far out with low LF and more expensive close in with high LF

FLYR

Conclusion & next steps

- Introduction of batch-constraints to DQN helps reduce the extrapolation error in real-world pricing scenarios, and produces policies which look reasonable, both at an aggregate level and individual itineraries.
- The framework presented is generic enough to cater to varying domain requirements: (i) ability to choose a different distribution for the action-probability (P) network, and (ii) the option to train the model for any shape function which represents the expected behavior of pricing policy with changes to available inventory.
- Through the concept of replay buffer central to training policies in RL world, we are able to circumvent the potential issues which can arise in model training with missing or bad (zero or intermittent sales). By maintaining the state-action-reward-next state pairs in memory and training on them, BCQ framework focuses on the transitions and not necessarily the entire trajectory.

Key takeaways

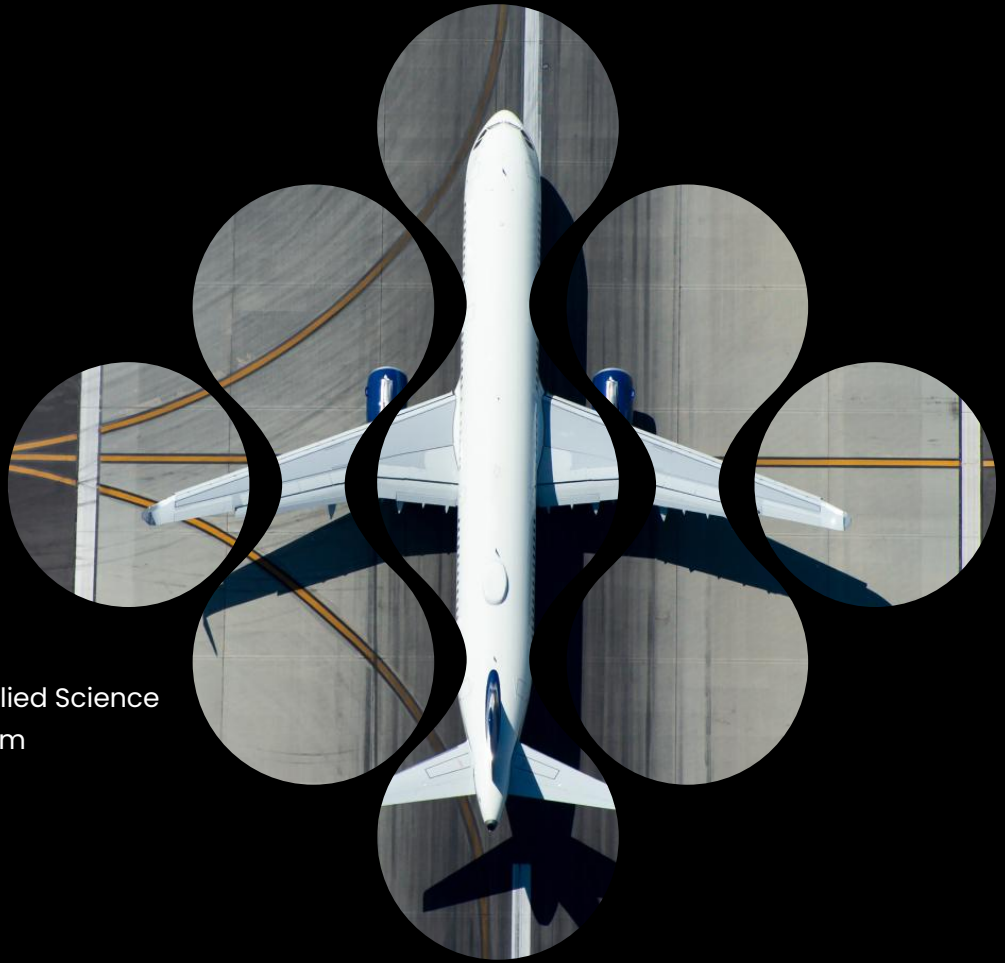
Conclusion and Implications

Next steps

- While the pricing policies are reasonable in offline validation, it is natural to run a *live A/B test* to test out the performance of the BCQ model - especially to track revenue growth in hope for optimal pricing decisions.
- The current framework and implementation is centered around discrete price buckets (discrete BCQ), but we are interested in formulating the *continuous version of BCQ* and stabilizing the training in that direction.

FLYR

Thank you



Akhil Gupta
Senior Applied Scientist
akhil.gupta@flyr.com
+1 217-819-7444

Naman Shukla
Product Manager, Applied Science
naman.shukla@flyr.com
+1 646-267-9093

AGIFORS 08 May 2024