# Automate Page Layout Optimization: An Offline Deep Q-Learning Approach

Zhou Qin
qinzhouhit@hotmail.com
Amazon
Seattle, WA, USA

Wenyang Liu
lwenyang@amazon.com
Amazon
Seattle, WA, USA

## ABSTRACT

The modern e-commerce web pages have brought better customer experience and more profitable services by whole page optimization at different granularity, e.g., page layout optimization, item ranking optimization, etc. Generating the proper page layout per customer's request is one of the vital tasks during the web page rendering process, which can directly impact customers' shopping experience and their decision-making. In this paper, we formulate the request-rendering interactions as a Markov decision process (MDP) and solve it by deep reinforcement learning (RL). Specifically, we present the design and implementation of applying offline Deep Q-Learning (DQN) to the contextual page layout optimization problem. Through the offline evaluation method, we demonstrate the effectiveness of the proposed framework, i.e., the RL agent has the potential to perform better than the baseline ranker by learning from the offline data set, e.g., the RL agent can improve the average cumulative rewards up to 36.69% comparing to the baseline ranker.

## CCS CONCEPTS

• **Information systems** → **Recommender systems**; **Learning to rank**.

## KEYWORDS

deep reinforcement learning, recommendation, offline learning, e-commerce

**ACM Reference Format:**
Zhou Qin and Wenyang Liu. 2022. Automate Page Layout Optimization: An Offline Deep Q-Learning Approach. In *Sixteenth ACM Conference on Recommender Systems (RecSys '22), September 18–23, 2022, Seattle, WA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3523227.3547400

## 1 INTRODUCTION

In a typical request-rendering process in e-commerce, generating the "right" page layout for a specific customer request is important because different page layout brings different overall display, items, ads, etc. Rendering the web page, i.e., a configuration of page layout with certain specifications and limitations, can show different contents to the customer, leading to different customer behaviors, e.g., re-query, clicks, or purchases, as shown in Figure 1. Normally,

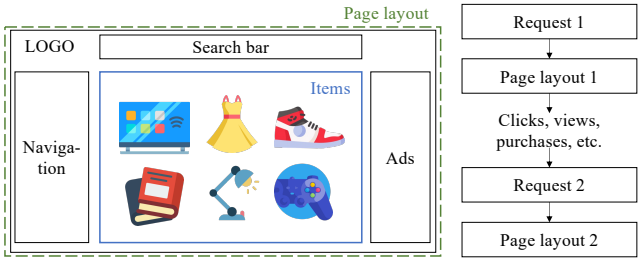


**Figure 1: Illustration of the interaction of customer and search system**

in e-commerce systems, a page layout can be configured by varying the position of item slots and the number of items, which can be optimized for better presentation in terms of customer experience and business value.

In the most common case, a page layout optimization system can be implemented via a ranking fashion [8], e.g., using a ranking module (aka ranker) to choose from a few well designed layouts. Regression methods such as Bayesian Linear Regression (BLR) are often used as the first steps to build such ranking systems due to low implementation efforts, decent performance, and good explainability. Then the ranker could be utilized to learn from contextual features and optimize towards one or a few common metrics, e.g., revenue, profit, click-through rate (CTR), etc. Online service deployed with a trained ranker then can predict the metrics through BLR-based models and choose the page layout with the highest predicted metric value. But such BLR model often suffers from cumbersome manual feature combination configuration and extra training for additional metrics. To overcome these weaknesses, we approach the page layout optimization by deep reinforcement learning [7] for three reasons: 1) The interaction of the customer and the e-commerce system is essentially a dynamic sequential decision-making process (as shown in Figure 1), which fits in the setting of reinforcement learning. 2) We aim to not achieve the maximum immediate rewards for the moment but to ultimately achieve the maximum cumulative long-term rewards across the session, which can be achieved by reinforcement learning. 3) For future iterations, a lifelong RL agent in an online setting can continuously learn and make decisions based on real-time interactions with the customers, providing a more curated shopping experience for the customers.

For the page layout optimization (PLO), generally, we aim to train an RL agent which can learn from the historical log data of customer requests and corresponding events in shopping sessions, including purchases, membership sign-up, sponsored product clicks, to make a proper decision (i.e., page layout selection) when encountering new requests. Specifically, we train the agent by offline Deep Q-Learning (DQN) method [1] due to two reasons: 1) We


*RecSys '22, September 18–23, 2022, Seattle, WA, USA*

ACM ISBN 978-1-4503-9278-5/22/09.
https://doi.org/10.1145/3523227.3547400

can only effectively train the model by offline approaches due to the limitation of offline data sets, e.g., there is no ground truth for reward when a new combination of context request and page layout appears, which can only be obtained by real interactions with customers. 2) The combinations of contextual requests and actions are huge due to the variations of context features, thus we refer to DQN rather than traditional Q-learning. The main contributions of this work are summarized below.

- We design and implement an offline DQN-based framework for page layout optimization to select page layout given a contextual request from the customer in order to maximize cumulative rewards.
- We evaluate our framework through real-world data and present comprehensive evaluation results showing the advantage of our proposed framework, e.g., the average rewards have increased 36.69% at most.

# 2 METHODOLOGY

## 2.1 Data Pipelines and Data Sets

To prepare for training and evaluating agent, for input, we extract 12 different features as context features of a query request, i.e., *layoutId*, *queryAlias*, *location*, *isMember*, *maxNumResults*, *device*, *queryType*, *deviceOs*, *deviceEnv*, *month*, *weekday*, *hour*. All these features are quantified by one-hot encoding during the data processing for modeling. For metrics, we investigate a mixture of a few major metrics, which are proprietary business metrics such as profit, revenue, clicks, etc. For output, the ground truth is from the page layout chosen by the existing policy. We also keep a list of candidate page layouts for the RL agent to choose from.

## 2.2 Problem Definition

To formulate the PLO problem in a reinforcement learning fashion, we first define the basics of a Markov Decision Process (MDP). In the PLO setting, we define a shopping session as one episode, which includes a sequence of requests from the customers. The basics for the MDP can then be denoted by a five-element tuple $\mathcal{G} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$, where $\mathcal{S}$ is the set of states; $\mathcal{A}$ is the action space; $\mathcal{R}$ is the reward function; $\mathcal{P}$ is the transition probability function; $\gamma$ is a discount factor. With the definition, our problem is to train an agent who can maximize the cumulative rewards by learning from the offline data set. The detailed definition of the MDP $\mathcal{G}$ in our problem is shown below.

- **Agent**: In PLO, the agent is the page layout ranker that selects over the page layout candidates, e.g., the ranker can be set to always choose the page layout with the maximum expected rewards.
- **State** $\mathcal{S}$: We define the state by in-session historical context features and selected page layouts, i.e., for request record $i$ the state is defined as $(\frac{1}{i+1}\sum_{j=0}^{i}(encoded(\mathbf{cft}_j)), \frac{1}{i+1}\sum_{j=0}^{i}(encoded(pl_j)))$, where $encoded(\mathbf{cft}_j)$ denotes a concatenated vector of encoded context features, and $encoded(pl_j)$ denotes the selected page layout.
- **Action** $\mathcal{A}$: The action space here refers to the set of all candidate page layouts and action means selecting a specific page layout given the current state.
- **Reward** $\mathcal{R}$: The reward is defined as $f((metric_i|i \in \{1, 2, ..., N\})$, where $f$ can be set as summation operation or index operation based on real-world scenarios, e.g., optimizing over a compound metric or a single metric.
- **Transition probability** $\mathcal{P}$: The transition probability function determines the transition between the states by taking action $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$, e.g., $p(\mathbf{s}_{i+1}|\mathbf{s}_i, \mathbf{a}_i)$ denotes the probability of transition to the next state $\mathbf{s}_{i+1}$ given the current state $\mathbf{s}_i$ and action $\mathbf{a}_i$.
- **Discount factor** $\gamma$: The discount factor essentially determines how much the RL agent cares about the rewards in the distant future relative to those in the immediate future.

## 2.3 DQN Framework

In this section, we introduce the design and implementation of our DQN-based framework, which includes three main components: 1) A dedicated environment "simulating" the interactions as in the online environment to provide feedback (i.e., reward and new state) for our agent; 2) A neural network-based agent which maps the state to action and Q-values; 3) An offline training methodology enabling the agent to learn from the offline data effectively.

*2.3.1 Environment Building.* As the first step, we build an environment that simulates the setting where the agent can interact with the real-time feedback from the environment. For example, we need to generate corresponding action $a_{i+1}$, reward $r_{i+1}$, and new state $s_{i+1}$ given the current state $s_i$. We build the customized environment (*env*) based on a commonly used toolkit *Gym* [6]. There are three main functions of the offline environment, i.e., 1) an initialization function for state, feature dimension, etc.; 2) a step function to return the next state, action, corresponding rewards, and a session end indicator; 3) a reset function to start a new session.

*2.3.2 Agent.* Different from vanilla Q-learning, in DQN, we use a neural network (NN) to map the states to action and Q-values due to the large context feature and action space. Our agent is designed as a neural network with three fully connected layers, with the state as input and Q-values as output. The dimension of the input is the dimension of concatenated features dimensions and the output dimension is the dimension of action space $\mathcal{A}$, i.e., the number of page layout candidates. The agent neural network works as a regression model with output values directly used as Q-values. We use Huber loss function [2] to quantify the difference between the target Q-values and the current Q-values.

*2.3.3 Offline DQN Training.* The training process of the DQN framework is summarized in Figure 2. The whole training methodology can be described in a few steps [1, 4]: 1) We call the customized *env* to obtain a transition tuple including the current state, action, reward, and next state. Then we update this tuple in the replay memory which provides mini-batches for agent NN optimization. 2) We initialize the agent NN twice as the main model and the target model [5]. 3) We optimize the main model by experience replay based on Bellman Equation. 4) We update the target model

**Table 1: Performance of agent under different platform and treatment settings. (GT means Ground Truth)**

| Platform | Desktop | | | | Mobile | | | |
|---|---|---|---|---|---|---|---|---|
| Ranker | Ranker1 | | Ranker2 | | Ranker1 | | Ranker2 | |
| Policy | GT | Agent | GT | Agent | GT | Agent | GT | Agent |
| Rewards | 1.76 | **3.41** | 2.24 | **3.25** | 1.38 | **1.79** | 2.34 | **2.43** |
| Improvement (%) | NA | **93.06** | NA | **44.85** | NA | **30.12** | NA | **3.64** |

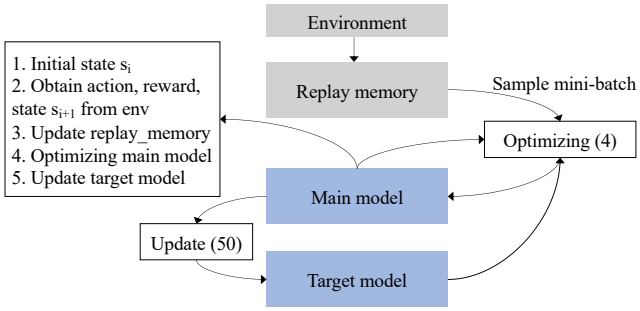


**Figure 2: Overview of offline DQN training pipeline**

occasionally, e.g., every 50 steps. The whole training methodology can be found in [4]. Notice that for offline training, we do not choose an action based on the inferred Q-values from the agent NN. Instead, we strictly replay the episodes from the offline data, which guarantees we always have the corresponding rewards and avoids efforts to design a dedicated yet biased reward function.

## 3 EVALUATION

### 3.1 Evaluation Methodology

**Methods**: In this part, we introduce the evaluation methodology for our offline DQN framework. Specifically, we evaluate our framework by cumulative rewards: we take the reward from the ground truth data if the agent acts the same as the test ranker, and skip the transition tuple if the agent acts differently. In the end, we calculate the average cumulative rewards per request as the final result. The evaluator details can be found in Policy Evaluator of [3].

**Training and Test Split**: We evaluate the agent by cross-validation, i.e., randomly sampling 80% of episodes for training and the rest 20% for test.

**Metrics**: For the evaluator, we use average cumulative rewards in the test data to show the performance, i.e., $R^{agent} = \frac{1}{N}\sum_{i=0}^{N} R_i$. Notice that $N$ is the total number of requests where the agent acts the same as the ground truth.

**Data set**: We leverage the data from a large e-commerce company which were generated based on two rankers. We use Ranker1 to denote the BLR ranker and Ranker2 as the random ranker. We also use data from both the desktop and mobile platforms to demonstrate the agent's performance under different settings.

### 3.2 Evaluation Results

In this part, we present our evaluation of the DQN agent under different scenarios, e.g., overall performance by average cumulative rewards under different platforms and rankers.

**Impact of Platform and Ranker.** To comprehensively investigate the performance of our DQN agent, we show the average cumulative rewards under different combinations of platforms and rankers in Table 1. Specifically, training and test are done on the data from the same platform and ranker setting. From the table, we can see that the agent can generally perform better than BLR ranker and random ranker on both platforms with proper tuning. We also find out that the agent can achieve more reward improvement by learning from the BLR ranker than the random ranker for both platforms, which is expected, i.e., the agent can learn less from the data generated by a random ranker.

## 4 CONCLUSION

In this paper, we perform a comprehensive investigation of applying offline deep reinforcement learning on page layout optimization. Specifically, we build an offline data processing pipeline to generate session-based data with rewards assigned. We then apply Deep Q-Learning in an offline fashion to the historical data and conduct a comprehensive evaluation in terms of platforms and rankers. In general, we show that our DQN agent can learn from the offline data generated by the existing ranker and achieve a higher average cumulative reward.

## REFERENCES

[1] Hado Hasselt. 2010. Double Q-learning. *Advances in neural information processing systems* 23 (2010), 2613–2621.
[2] Peter J Huber. 1992. Robust estimation of a location parameter. In *Breakthroughs in statistics*. Springer, 492–518.
[3] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. 2011. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*. 297–306.
[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).
[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.
[6] OpenAI. 2021. Gym. https://gym.openai.com/docs/.
[7] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
[8] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep reinforcement learning for page-wise recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*. 95–103.