Tile Networks: Learning Optimal Geometric Layout for Whole-page Recommendation

Shuai Xiao
Alibaba Group

Zaifan Jiang
Alibaba Group

Shuang Yang
Alibaba Group

Abstract

Finding optimal configurations in a geometric space is a key challenge in many technological disciplines. Current approaches either rely heavily on human domain expertise and are difficult to scale. In this paper we show it is possible to solve configuration optimization problems for whole-page recommendation using reinforcement learning. The proposed *Tile Networks* is a neural architecture that optimizes 2D geometric configurations by arranging items on proper positions. Empirical results on real dataset demonstrate its superior performance compared to traditional learning to rank approaches and recent deep models.

1 Introduction

Finding optimal configurations in a geometric space is a key challenge in many technological disciplines, for example, *drug discovery* (i.e., optimal configuration of molecules), *material engineering* (i.e., optimal nanostructure), *circuit design* (i.e., optimal component layout), *logistics* and *transportation* (e.g., optimal plan of routes). For these scenarios, sequential selections of members of *inputs* and *positions* are required to generate optimal configurations. Taking chip design for example, we need to iteratively select chip cells from a pool of cells and place them in appropriate positions onto a chip canvas to obtain a chip of high quality in the end as described by Mirhoseini et al. (2020). Current practises in these areas have relied heavily on human domain expertise, as a result, these approaches could be *expensive* (i.e., requiring highly specialized skills), *inefficient* (i.e., throughput bottlenecked by the

availability of human experts as well as their latency), *ineffective* (i.e., quality depends entirely on the expertise level of human designers and is often hard to control or to further optimize), and *difficult to scale* (i.e., solutions are often case specific, not easily transferable or automatable) as done in works(Liwo et al., 1999; Azami and Omidkhah, 2019; Kalinli, 2004; Kool et al., 2019). The recent work of Vinyals et al. (2015a) suggests a promising new direction of using data-driven approach to tackle this challenging problem automatically using machine learning. In particular, for problems like Travelling Salesman Problem (TSP), Convex Hull, the output for the algorithm is the permutation of inputs.

Now we consider a more complex problem: How to place items and arrange those sequential selections of members of inputs over proper positions? The *Pointer Networks* can be modified to solve geometric configuration by sequentially selecting a member of inputs and placing it following a predefined layout, such as top to bottom. We argue that this is not the most efficient way as we can relax from the predefined layout and propose the layout automatically using a data-driven manner which leads to better generalization over the human-defined layout. To this end, we present *Tile Networks*, a neural architecture for generating optimal configurations in geometric space by proposing elements from inputs and their positions simultaneously. The model follows an encoder-decoder architecture: the encoder is input-invariant, which aggregates input information via message passing; while the decoder iteratively selects elements from inputs and generate layout by keeping memory of the current state of placement and modeling its influence to future placement.

While our method easily applies to high-dimensional scenarios, we focus on a 2D instantiation of the problem under the application of personalized *whole-page optimization* (WPO), a problem of optimization nature that is prevalent in news feeds, social networks, etc. Specifically, the goal is, for each given user, to recommend a set of items and at the same time design their layout as displayed on a 2-D geometric space (e.g., the screen of a smartphone or tablet, a webpage of an

e-commerce site) in such a way that an objective of interest (e.g., user clicks, purchases, dwell time, revisits) is optimized. To simplify things, we consider the setting that is typical in today's internet industry, where the layout follows a predefined grid template (usually defined by user-interface designers) and items are to be arranged into the grid-bounded cells called "*tiles*". Even for this simplified case, the configurational space is combinatorial, i.e., for n total items (i.e., inventory size) and k tiles (i.e., display-space size), the total number of possible layouts is $O(\binom{k}{n} \times k!)$, which is daunting for any practical setting.

Conventional approaches to recommender system address the problem by estimating the utility of each item, *either* individually (Agarwal et al., 2019) *or* as a pair (Rendle et al., 2009) or k-tuple (Yang et al., 2011), by learning a ranking function. Items are then ranked in a descending order according to the estimated utility, and the top-k list is arranged into *tiles* following a predefined order, such as top to bottom. These methods solve the WPO problem approximately, and they often work but only to the extent where (1) items can be considered independent such that the utility of the bundle *as a whole* are decomposable as the sum (or some linear function) of the individual utilities; and (2) the attraction of the whole page depends entirely on the content of these k items, regardless of their layout. In any realistic scenario, these two assumptions are regarded too restrictive and cannot reflect the truthful reality as shown in works (Ding et al., 2019; Wang et al., 2016; Zhao et al., 2018; Wang et al., 2017). For example, numerous user studies, e.g., Lee and Ahn (2012) have found that there exist complex interaction patterns not only between items but also among the content of the items, their visual appearances and user's attention distribution over the screen. Our work is similar to recent work of Wang et al. (2016) in the sense that the model generates elements from inputs and their layout simultaneously. Wang et al. (2016) proposed using deconvolutional neural networks to generate embedding in each *tile*. The outputs of the deconv in *tiles* are mapped to discrete items through computing the distance between output embedding and item embedding.

We design *Tile Networks*, a neural architecture for generating optimal configurations in geometric, and apply it in the context of personalized whole-page layout in recommender systems. The *Tile Networks* can be trained using reinforcement learning to optimize the overall quality of the page as a whole (including the selection of items as well as the layout). Experiments in a number of diverse settings show that *Tile Networks* are able to find configurations that are better than alternative methods with a notable margin. The

contributions of this paper include:

- We propose *Tile Networks*, a neural architecture for optimizing 2D geometric configurations such as personalized whole-page layout in recommender systems.

- We design a learning framework that solves geometric configurations optimization based on reinforcement learning.

- We testify the *Tile Networks* over different settings of datasets and the results demonstrate *Tile Networks* performs better than baselines including *Pointer Networks* and has a good generalization ability.

2 Problem Formulation

In this section, we formulate the problem of geometric configuration as a combinatorial optimization problem, and propose machine-learning based solutions for it.

2.1 Geometric Configuration as Combinatorial Optimization

Given a d-dimensional configuration space and n elements, the goal of geometric configuration is to find the optimal placement of those n elements within the configuration space subject to placement constraints. In this paper, we restrict our attention to the special scenario where the configuration is further constrained onto a grid of size $k = \ell^d$ within the configuration space, as is common in many applications. For instance, in the WPO problem, the recommended commodities are placed onto an ℓ^2 grid each page, where ℓ is the number of rows or columns.

Denoting a set of elements as $I := [n] := \{1, \ldots, n\}$, $\Omega := [\ell]^d$, a configuration space as C can be mathematically defined as a mapping from I to Ω. Under a given reward function $R(C|I)$ that evaluates a configuration's performance under I, the geometric configuration can be posed as an optimization problem

$$\max_{\theta} R(C|I). \tag{1}$$

The nature of geometric configuration optimization is combinatorial, and is NP-hard, for which a common solution is the greedy algorithm. In the WPO problem, this boils down to a repeated procedure where the learner greedily selects the next element to place on the grid in a greedy fashion. Furthermore, the learner can re-arrange the elements already selected at each iteration to improve upon the performance of the vanilla-greedy selection.

2.2 Combinatorial Optimization with Reinforcement Learning

Recently, Bello et al. (2016) proposed to solve combinatorial optimization problems via reinforcement learning, modeling the greedy selection and rearrangement procedures as outcomes of an action policy. More specifically, denote $C(I_t, \Omega_t)$ as the selected element and the arrangement at the t-th iteration, a policy refers to a θ-parameterized probability distribution $p_\theta(C(I_t, \Omega_t)|C_{t-1})$ over all possible $C(I_t, \Omega_t)$ that assigns a probability to each possible configuration C as follows:

$$P(C|I, \Omega) = \prod_{t=1}^{k} p_\theta\big(C(I_t, \Omega_t)|C_{t-1}\big). \qquad (2)$$

With a slight abuse of notations, denote $C(I_t)$ as the selected element from I at the t-th iteration and $C(\Omega_t)$ as the rearrangement, we further have

$$p_\theta\big(C(I_t, \Omega_t)|C_{t-1}\big) = p_\theta\big(C(I_t)|C_{t-1}\big)p_\theta\big(C(\Omega_t)|C_{t-1}\big). \qquad (3)$$

Under this setting, the goal reduces to finding the best policy parameter θ by solving

$$\max_\theta J(\theta) = E_{C \sim p} R(C|I). \qquad (4)$$

In the following, we propose to efficiently solve this formulation using *Tile Networks*, based on the design of *Pointer Networks* which we introduce below.

2.3 Pointer Networks

Vinyals et al. (2015b) proposed a novel seq2seq framework called Pointer Networks which uses attention as a pointer to select a member of inputs which are decoded in a sequence as the output. The architecture can be used to tackle problems that the number of target classes in each step of output depends on the variable input sequence, which can't be well handled by existing sequence-to-sequence networks used by Sutskever et al. (2014). Pointer Networks trained using supervised learning show promising performance on three one-dimensional geometric configuration problems: finding *planar convex hulls*, computing *Delaunay triangulations*, and solving *travelling salesman problems* (TSPs). Nazari et al. (2018) improved the seq2seq architecture by replacing RNN encoder with feed-forward networks. They are among the first to use reinforcement learning to train the model for geometric configuration optimization. Finally, latest work by Kool et al. (2019) further enhanced the architecture by using attention based encoders instead of RNNs. Both Nazari et al. (2018) and Kool et al. (2019) applied the algorithms to solve *vehicle routing problems* (VRPs). These works have by

now focused primarily on the setting where the solution is a sequence (e.g., a route in TSPs/VRPs,) and it's no need to further consider the placement positions in geometric space. This work attempts to extend this line of work for solving geometric configuration optimization in high-dimensional space where the predefined layout is incapable to handle this complexity.

3 Neural Network Architecture

In this section, we present the policy parameterization *Tile Networks* for geometric configuration and the motivation behind it.

Tile Networks follows an encoder-decoder network architecture where the main purpose of encoder is to project discrete values $I = \{1, \ldots, n\}$ and grid-bounded *tiles* Ω into a hidden representation. Inspired by previous work of Kool et al. (2019) which lets each member of inputs aware the existence of other members during encoding process, message passing is utilized in our encoder and is implemented through self-attention mechanism.

Different from *Pointer Networks*, our decoder needs to select a discrete value from $I = \{1, \ldots, n\}$ and place it into a *tile* in space Ω. At the same time, the decoder should memorize previously selected values and their layout to explicitly model their influence to future decisions.

The network architecture is depicted in Figure 1. The parameters of the decoder remain constant regrading to the number of discrete values $I = \{1, \ldots, n\}$ and keep relatively compact.

Encoder Network The discrete values $I = \{1, \ldots, n\}$ (e.g., the inventory of items in recommender systems) and *tiles* in geometric space Ω are encoded separately using self-attention mechanism. For discrete value i of I, its feature is represented by a feature vector X_i (which may depend on users or contexts). For WPO, feature vector X_i contains user features and item features. Thus, the generated pages are conditioned on the personalized user features. The X_i acts as query and features $X_1, \ldots, X_n$ of all other values serve as the keys and values simultaneously. Then the hidden state $\mathbf{H}_i^I$ of discrete value i becomes:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}}, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}}, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}} \qquad (5)$$

$$\mathbf{H} = \frac{\text{Softmax}\big(\mathbf{Q} * \mathbf{K}^T\big)}{\sqrt{\mathbf{D}_{\mathbf{K}}}}\mathbf{V} \qquad (6)$$

$$\mathbf{H}^I = \text{LayerNorm}(\mathbf{H} * \mathbf{W}^{\mathbf{O}} + \mathbf{Q}) \qquad (7)$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are query, key, and value matrix after affine transformation by separately multiplying $\mathbf{X}$ with trainable parameters $\mathbf{W}^{\mathbf{Q}} \in \mathcal{R}^{O \times D_K}, \mathbf{W}^{\mathbf{K}} \in \mathcal{R}^{O \times D_K}, \mathbf{W}^{\mathbf{V}} \in \mathcal{R}^{O \times D_V}$. $\mathbf{W}^{\mathbf{O}}$ is the linear projection
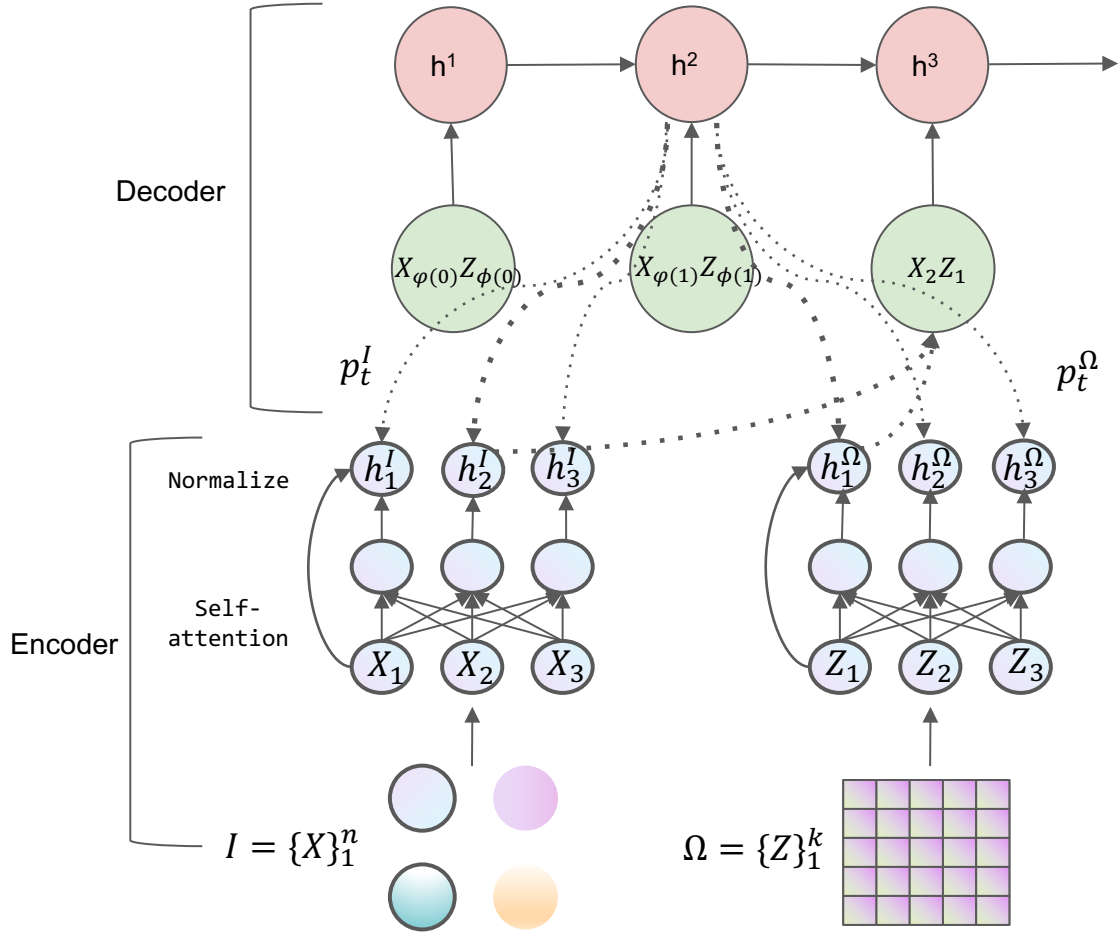
Figure 1: The architecture of *Tile Networks*. The dashed arrows pointed to the encoder represent attention over inputs. The dashed arrows pointed to the decoder indicate selecting decision at each decoding step.

parameter.

For *tiles* in geometric space λ, we use Z_j to denote coordinate representation of *tile* j. Then the hidden state of *tile* j denoted by $\mathbf{H}_j^{\Omega}$ can be obtained using Equation 5 and 7 with a separate set of parameters.

Decoder Network The decoder needs to do two things: one is to select member of inputs and determine their layout; the other is to keep memory of past actions to make future decisions. To this end, we use LSTM to keep track of the current state of configuration C_t. At step t, the hidden state of LSTM decoder is updated as:

$$h^t = \psi(W * h^{t-1} + V * [X_{\varphi(t-1)}, Z_{\phi(t-1)}]) \quad (8)$$

where h^t can be seen as the representation of current configuration C_t. $\varphi(t-1)$ is the index of selected discrete value from I at step $t-1$ and $\phi(t-1)$ is the index of selected *tile* at step $t-1$. Therefore,

$C(I_t, \Omega_t) = (X_{\varphi(t-1)}, Z_{\phi(t-1)})$ in Equation 2. ψ is a nonlinear activation function applied element-wise. [] in Equation 8 is the vector concatenation operator. Note for the starting step $t=1$, $X_{\varphi(0)}, Z_{\phi(0)}$ are not from inputs but trainable parameters. At step t, the conditional probability of selecting member i from discrete values I and placing it into *tile* j is:

$$u_i^t = v^I * \tanh(W_1^I * h^t + W_2^I * H_i^I) \quad (9)$$
$$s_j^t = v^{\Omega} * \tanh(W_1^{\Omega} * h^t + W_2^{\Omega} * H_j^{\Omega}) \quad (10)$$
$$p_t^I = \text{softmax}(u^t) \quad (11)$$
$$p_t^{\Omega} = \text{softmax}(s^t) \quad (12)$$

where $\{v^I, W_1^I, W_2^I$ are the trainable parameters of attention networks for selecting members of inputs and $v^{\Omega}, W_1^{\Omega}, W_2^{\Omega}\}$ are parameters of attention networks for designing layout. p_t^I is the probability distribution over inputs I and p_t^{Ω} is the probability distribution over *tiles*. Note items and *tiles* that have been selected pre-

viously will be masked when computing the probability distributions.

The *Tile Networks* selects a member of inputs and find its placement position sampled from distributions in Equation 11 and 12 rather than a deterministic way. This stochastic policy allows for exploration of configurations with high rewards.

Remark Although we use whole-page optimization as the testbed, the proposed method *Tile Networks* is not only applicable to the 2D setting and can be directly applied to more general settings, such as genome generation and high-dimensional space, such as 3D, etc. The items and positions can be encoded and decoded in a similar way as in 2D. For the self-attention encoder, the positions are encoded by considering all other positions, similar to text encoders in the natural language process. Therefore, for more general settings, like genome generation or irregular space, each individual position can be encoded only if we know its features, such as coordinates. Our method is not only applicable in regular 2D scenarios. For scenarios with too many possible "tiles", one possible approach is to aggregate positions into blocks and use the proposed method in a hierarchical manner by iteratively placing items in space.

4 Learning Algorithms

The stochastic policy p_θ parameterized by *Tile Networks* can be optimized through stochastic gradient descent reinforce algorithm proposed by Williams (1992).

$$\nabla_\theta J(\theta) = E_{C \sim p_\theta} R(C|I) \nabla_\theta \log p_\theta(C|I) \qquad (13)$$

The reward $R(C|I)$ can be any value with interest that needs to optimized, such as dwell time, the number of clicks, *NDCG*. The learning procedure of *Tile Networks* is given in Figure 2. A batch of input samples $\{I\}_1^B$ is draw and their configurations are proposed through the *Tile Networks* and presented to users. The users will decide to click the items based on their preference to the items and viewing order. The reward $R(C|I)$ can be estimated via a critic. Then the policy p_θ is updated using Equation 13. The critic is updated by minimizing $\sum_1^B ||R(C'|I) - V(C'|I)||^2$ where $V(C'|I)$ is the reward for C' collected from user's feedback. The learning algorithm is summarized in Algorithm 1.

Critic Architecture. The critic has three components: a self-attention encoder, a position-wise fully-connected networks and an aggregate layer. The self-attention encoder is similar to the encoder of *Tile Networks* but the input contains the feature concatenation of X and position embedding Z. The position-wise fully-connected networks will apply fully-connected networks to the each hidden states of the encoder. The

dimensions for queries, keys and values in self-attention is 128. The position-wise fully-connected layer has an output dimension 16 with ReLU activation function. Then the outputs are concatenated and passed through a fully-connected layer to produce the estimation of expected rewards.

Algorithm 1 Stochastic policy training of *Tile Networks*

Require: : Training dataset S, batch size B.
Require: : w_0, initial critic parameters. θ_0, initial *Tile Networks*' parameters.
1: **while** θ has not converged **do**
2:     Sample $\{I\}_1^B$ from training dataset S.
3:     Sample configuration $\{C\}_1^B$ from policy $p_\theta(C|I)$.
4:     Estimate reward $R(C|I)$ from critic networks.
5:     Update *Tile Networks*' parameters θ using Equation 13.
6:     Update critic parameters w by minimizing $\sum_1^B ||R(C'|I) - V(C'|I)||^2$.
7: **end while**

---

5 Experiments

We conduct experiments on real-world commercial recommendation system datasets. The results show improvement over several core performance metrics.

Datasets and Experimental Setup The dataset collected from one of the largest e-commercial platforms in the world includes a large-scale records of whole-page recommendations. It contains more than 14 million records with more than 743 thousands users and 7 million items. Both users and items have dense features and categorical features, indicating users' basic profiles and items' features. Users also have personalized features which show users' preferences over different items. Users' click-through data is also recorded where the items in tiles are presented to them. This dataset is publicly available [1]. We follow the settings in real application scenarios of the whole-page optimization where items are arranged into grid-spaced *tiles* and the tiles' size is 5 rows and 6 columns. 70% of the entire dataset are randomly sampled for training, 15% for validation and 15% for testing.

We adopt the procedure of Joachims et al. (2017) to generate the click data. First of all, we use the recorded click-through data to train a user-item preference model which estimates the click probability of each item when observed by users. Three popular layouts indicating users' scanning behaviors are used, namely 1) top-down row-wise order (Row-Env), 2) left-to-right column-wise

---

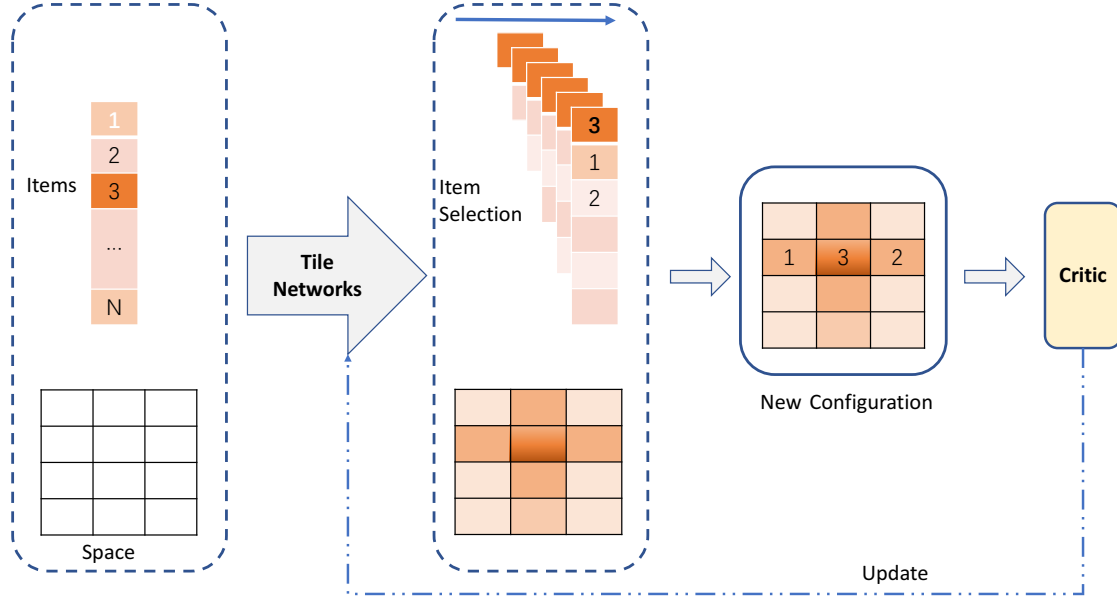[1]https://github.com/rank2rec/rerank

Figure 2: The pipeline of learning algorithm. The configurations are sampled from *Tile Networks*, which can be updated through stochastic gradient descent.

order (Col-Env), 3) central area to edges (Z-Env). A real layout (Real-Env) computed from the real dataset is also used. By simulating the user cascade model, $\frac{1}{i^\eta}$ represent the observation probability of items observed by users, where i is the item order scanned by users and $\eta = 0.05$ is the decay parameter. The click label is generated based on the product of click probability and observation probability.

The model for estimating user-item preference is a feed-forward neural networks with 3 hidden layers and the hidden dimensions and activation functions are (32, 16, 8) and (relu, relu, tanh). The input is the concatenation of embedding features of users and items. The final layer is a sigmoid function which outputs the click probability. The real scanning order is computed following the method proposed by Guo et al. (2019) by factorizing the click probability into positions' attractiveness and items' attractiveness. The learned positions' attractiveness can be used as the real scanning order.

To testify the generalization capability of the proposed model, we simulate two item interaction dynamics following the procedure proposed by Bello et al. (2018), namely diverse clicks and similar clicks. For diverse clicks, one observed item won't be clicked if it's too similar to previously clicked items. For similar clicks, even an irrelevant item will be clicked when observed if they are very similar to previously clicked items. The similarity is defined as being the smallest 0.5 percentiles of Euclidean distances between items' features pairs

within one page.

In summary, an environment consists of users' scanning pattern, user-item preference and item interaction dynamic. An environment will simulate user click based on user' viewing priority, items' attractiveness to users and item interaction dynamics. The generated clicks in training dataset are used to train all models. An environment generates clicks for each configuration (layout) output by learned models to evaluate the performance of those models.

Evaluation Metrics The widely-used metrics in recommendation systems, NDCG and Pre@K are employed to evaluate the model performance. NDCG is normalized discounted cumulative gain which measures the ranking quality. Pre@K calculates the percentage of clicked items in the top K tiles viewed by users. The NDCG/Prec@K are computed following the order of user view behavior, e.g, NDCG is computed in row-wise in Row-Env and in predicted order in Real-Env. The NDCG/Pre@10 computed this way is very similar to the grid metrics 11 used in Xie et al. (2019) where they first predict user view behavior and then compute grid metrics based on that. The definitions of those metrics are as follows:

- **NDCG** is a measure of ranking quality. NDCG = DCG/IDCG, where DCG is calculated as the

following equation:

$$DCG = \sum_{i=1}^{n} \frac{2^{rel_i} - 1}{\log_2(i+1)} \qquad (14)$$

where i is the number indicating the item is viewed in the i-th order and rel_i is the click label of this item. IDCG is ideal discounted cumulative gain and is equal to DCG after transforming rel_i by sorting rel_i decreasingly.

- **Pre@K** is defined as the percentage of clicked items in the top-k viewed items.

$$Pre@K = \frac{\sum_i^K 1(i)}{K} \qquad (15)$$

where $1(i)$ is the indicator function about whether the item in the t-th viewed *tile* is clicked.

Baselines Many sophisticated learning to ranking methods have been developed for personalized recommendation, such as DeepFM (Guo et al., 2017), NFM (He and Chua, 2017) and Wide&Deep model (W&D) (Cheng et al., 2016). Those methods focus on CTR estimation and don't generate item layout directly. As Wide&Deep is the most effective recommendation algorithm used in industrial settings, we use WideDeep to represent those methods for CTR estimation. For whole-page recommendation, W&D model can be used to rank items and then ranked items are placed into *tiles* following a specific layout. The default layout used in practice is top-down row-wise order, which we denote as Row-W&D model. We also use other two layouts which are widely used in practice which are called Rol-W&D and Z-W&D. *Pointer Networks* proposed by Bello et al. (2018) can be used to determine the output sequence of items. Like W&D model, we also use three default layouts to place items ranked by *Pointer Networks*, leading to Row-Pointer, Col-Pointer, Z-Pointer model. Wang et al. (2016) propose using deconvolutional neural networks (Deconv) to generate optimal configurations, which simultaneously output items and their layout.

For personalized whole-page optimization, the inputs of items' features are the the concatenation of both items' and users' features. For simplicity, the user queries are not explicitly denoted in computation. The items usually contains dense features and categorical features.

For Wide&Deep model (W&D) (Cheng et al., 2016), a four-layer fully connected network is used with hidden state size of 128, 64 and activation functions of *ReLU*, *ReLU*. The model is trained using mini-batches of 128 training examples and Adam optimizer with an initial learning rate of 0.001.

For *Pointer Networks*, we follow the implementation as in Bello et al. (2018) where mini-batches of 128 training examples and LSTM cells with 128 hidden units are used. The optimizer used is Adam with an initial learning rate of 0.001.

For *Tile Networks*, the self-attention module also contains skip connections (He et al., 2016) and layer normalization (LN) (Ba et al., 2016). The dimensions of parameters $\mathbf{W^Q} \in \mathcal{R}^{O \times D_K}, \mathbf{W^K} \in \mathcal{R}^{O \times D_K}, \mathbf{W^V} \in \mathcal{R}^{O \times D_V}$ in self-attention network and $\mathbf{W^O}$ are 128. For decoder, the dimensions of LSTM hidden state and weight matrix $\{v^I, W_1^I, W_2^I, v^\Omega, W_1^\Omega, W_2^\Omega\}$ are all 64. The policy is learned using ADAM optimizer with an initial learning rate of 0.001. Lastly, our method is implemented in Tensorflow, and the experiments were run with two Nvidia Tesla K80 GPUs.

Results Table 1 shows the learning ability of *Tile Networks* when the diversity of items is preferred by users. We can observe that even though *Tile Networks* has to learn the layout, it can still estimate the layout better except for *Pointer Networks* whose default layout coincides with the environment's ground-truth layout. For these cases, *Tile Networks* has a lower but close performance with *Pointer Networks*. This is quite expected: When learning *Pointer Networks* and we know its optimal layout, the problems reduces to selecting appropriate items. However, for complex layout as in Real-Env, *Tile Networks* outperform *Pointer Networks* with a notable margin. Like *Tile Networks*, Deconv can also select items from inputs and output the layout directly. We can observe that Deconv performs stably across environments but has a lower performance than *Tile Networks*. We speculate this may be due to that Deconv models the interaction of items using local dependency, which can't capture the complex interaction of items. The results also show the robustness of *Tile Networks* in various settings.

To demonstrate the generalization ability and flexibility of *Tile Networks*, we also run models on environments where the concentration of items is preferred by users. Table 2 shows this empirical results. *Tile Networks* performs better than baselines except for *Pointer Networks* whose default users' viewing priority is the same with that of the environment. This demonstrates that our model can learn from data and is adaptive to various types of interactions in the data.

The learned users' viewing priority (corresponding layouts) are visualized in Figure 3 after transformation from priority to pixel intensity. The lightness of tiles indicates the *tile* priority of users' viewing of each tiles. The learned priority recover environment ground-truth layout well except small deviations that exist as shown in the Column-wise pattern in Figure 3. This uncovered
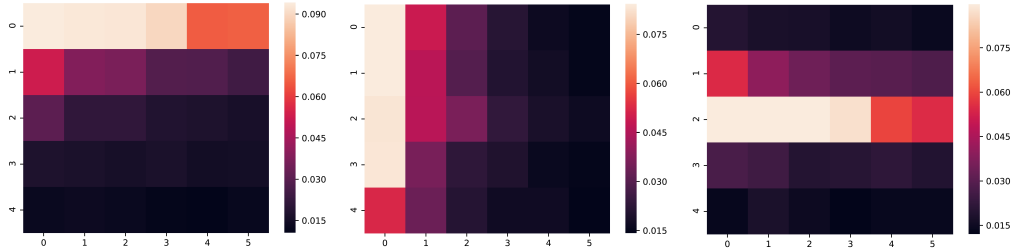
Figure 3: The learned layout for three typical patterns. The left figure shows the learned layout for Row-Env. The middle figure shows the learned layout for Col-Env. The right figure shows the learned layout for Z-Env.

Table 1: Models' performance on various environments for diverse clicks.

Algorithm	Row-Env		Col-Env		Z-Env		Real-Env	
	NDCG	Pre@10	NDCG	Pre@10	NDCG	Pre@10	NDCG	Pre@10
Row-W&D	0.801	0.172	0.615	0.158	0.697	0.136	0.711	0.152
Col-W&D	0.728	0.159	0.793	0.172	0.708	0.141	0.713	0.154
Z-W&D	0.781	0.160	0.628	0.162	0.732	0.157	0.701	0.141
Deconv	0.790	0.165	0.658	0.167	0.776	0.171	0.765	0.163
Row-Pointer	**0.846**	**0.191**	0.632	0.173	0.613	0.158	0.745	0.161
Col-Pointer	0.652	0.173	**0.876**	**0.186**	0.696	0.150	0.736	0.158
Z-Pointer	0.684	0.163	0.631	0.165	**0.827**	**0.182**	0.721	0.156
Tile Networks	**0.845**	**0.193**	**0.875**	**0.186**	**0.826**	**0.182**	**0.794**	**0.169**

Table 2: Different models' performance on various environments for similar clicks.

Algorithm	Row-Env		Col-Env		Z-Env		Real-Env	
	NDCG	Pre@10	NDCG	Pre@10	NDCG	Pre@10	NDCG	Pre@10
Row-W&D	0.803	0.154	0.595	0.129	0.603	0.109	0.672	0.132
Col-W&D	0.751	0.127	0.648	0.150	0.608	0.118	0.683	0.135
Z-W&D	0.728	0.131	0.602	0.137	0.632	0.130	0.670	0.120
Deconv	0.767	0.143	0.636	0.142	0.629	0.137	0.723	0.143
Row-Pointer	**0.835**	**0.162**	0.603	0.130	0.613	0.132	0.712	0.139
Col-Pointer	0.745	0.145	**0.672**	**0.159**	0.606	0.127	0.708	0.136
Z-Pointer	0.764	0.141	0.618	0.141	**0.698**	**0.152**	0.709	0.137
Tile Networks	**0.831**	**0.160**	**0.670**	**0.158**	**0.695**	**0.152**	**0.796**	**0.148**

viewing priority can serve as interpretability map for regions that users pay more attention to. Those learned patterns are extremely useful for human–computer interaction study. For instance, promoters can show what they want to sell most in the region where users focus.

The variances of the evaluation metric performance are given in Table 3 to testify the robustness of methods. Tile Networks has comparable variance with Pointer Network, but smaller variance than that of Deconv.

6 Conclusion

The paper addresses an under-explored problem: optimal configuration in high-dimensional geometric space. We show a promising direction and propose a neural architecture called *Tile Networks* by extending *Pointer*

Networks to solve this problem. To verify the proposed idea, we use 2d personalized whole page optimization as testbed to carry out a series of experiments. Empirical results show that *Tile Networks* which can capture complex item interactions and user's viewing priority of *tiles* outperforms traditional learning-to-rank methods and recent data-driven approaches. Stable performances on variable environments show its generalization ability. For the future work, designing hierarchical decoding mechanism is a promising direction as the hierarchical structure is more efficient for capturing the interactions both in whole and local scale. Another promising direction is to make the selection of tile dependent on the content which can further improve the model's expressive power. Introducing interpretability into the model is beneficial for improving robustness and easiness for

Table 3: Different models' performance variances on various environments for similar clicks.

Algorithm	Row-Env		Col-Env		Z-Env		Real-Env	
	NDCG	Pre@10	NDCG	Pre@10	NDCG	Pre@10	NDCG	Pre@10
Row-W&D	2.9e-3	5.3e-3	8.3e-3	7.1e-3	5.9e-3	4.8e-3	9.2e-3	8.5e-3
Col-W&D	5.1e-3	4.6e-3	8.1e-3	4.9e-3	7.3e-3	6.8e-3	8.4e-3	7.9e-3
Z-W&D	5.6e-3	3.2e-3	5.6e-3	5.1e-3	4.1e-3	3.9e-3	7.2e-3	5.7e-3
Deconv	4.6e-2	3.1e-2	4.5e-2	4.8e-2	4.4e-2	2.7e-2	5.4e-2	4.6e-2
Row-Pointer	3.6e-2	2.9e-2	3.1e-2	2.8e-2	3.8e-2	2.6e-2	4.2e-2	3.7e-2
Col-Pointer	3.9e-2	3.1e-2	3.7e-2	2.7e-2	2.9e-2	2.2e-2	3.7e-2	3.2e-2
Z-Pointer	3.4e-2	2.7e-2	2.7e-2	2.1e-2	3.4e-2	2.3e-2	4.2e-2	3.4e-2
Tile Networks	4.2e-2	3.1e-2	3.8e-2	2.8e-2	3.6e-2	2.9e-2	4.4e-2	3.6e-2

end users.

References

Agarwal, A., Takatsu, K., Zaitsev, I., and Joachims, T. (2019). A general framework for counterfactual learning-to-rank. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*.

Azami, H. and Omidkhah, M. R. (2019). Modeling and optimization of characterization of nanostructure anodized aluminium oxide membranes. *Journal of the Iranian Chemical Society*, 16(5):985–997.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bello, I., Kulkarni, S., Jain, S., Boutilier, C., Chi, E., Eban, E., Luo, X., Mackey, A., and Meshi, O. (2018). Seq2slate: Re-ranking and slate optimization with rnns. *arXiv preprint arXiv:1810.02019*.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., et al. (2016). Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10.

Ding, W., Govindaraj, D., and Vishwanathan, S. (2019). Whole page optimization with global constraints. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3153–3161. ACM.

Guo, H., Tang, R., Ye, Y., Li, Z., and He, X. (2017). Deepfm: a factorization-machine based neural network for ctr prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 1725–1731.

Guo, H., Yu, J., Liu, Q., Tang, R., and Zhang, Y. (2019). Pal: A position-bias aware learning framework for ctr prediction in live recommender systems. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, page 452–456, New York, NY, USA. Association for Computing Machinery.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

He, X. and Chua, T.-S. (2017). Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 355–364.

Joachims, T., Swaminathan, A., and Schnabel, T. (2017). Unbiased learning-to-rank with biased feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 781–789. ACM.

Kalinli, A. (2004). Optimal circuit design using immune algorithm. In Nicosia, G., Cutello, V., Bentley, P. J., and Timmis, J., editors, *Artificial Immune Systems*, pages 42–52, Berlin, Heidelberg. Springer Berlin Heidelberg.

Kool, W., van Hoof, H., and Welling, M. (2019). Attention, learn to solve routing problems! In *International Conference on Learning Representations*.

Lee, J. and Ahn, J.-H. (2012). Attention to banner ads and their effectiveness: An eye-tracking approach. *International Journal of Electronic Commerce*, 17(1):119–137.

Liwo, A., Lee, J., Ripoll, D. R., Pillardy, J., and Scheraga, H. A. (1999). Protein structure prediction by global optimization of a potential energy function. *Proceedings of the National Academy of Sciences*, 96(10):5482–5485.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Bae, S., et al. (2020). Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746*.

Nazari, M., Oroojlooy, A., Snyder, L., and Takác, M. (2018). Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pages 9839–9849.

Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. (2009). Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, page 452–461, Arlington, Virginia, USA. AUAI Press.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Vinyals, O., Fortunato, M., and Jaitly, N. (2015a). Pointer networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc.

Vinyals, O., Fortunato, M., and Jaitly, N. (2015b). Pointer networks. In *Advances in neural information processing systems*, pages 2692–2700.

Wang, Y., Ouyang, H., Wang, C., Chen, J., Asamov, T., and Chang, Y. (2017). Efficient ordered combinatorial semi-bandits for whole-page recommendation. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Wang, Y., Yin, D., Jie, L., Wang, P., Yamada, M., Chang, Y., and Mei, Q. (2016). Beyond ranking: Optimizing whole-page presentation. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 103–112. ACM.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Xie, X., Mao, J., Liu, Y., de Rijke, M., Shao, Y., Ye, Z., Zhang, M., and Ma, S. (2019). Grid-based evaluation metrics for web image search. In *The world wide web conference*, pages 2103–2114.

Yang, S.-H., Long, B., Smola, A. J., Zha, H., and Zheng, Z. (2011). Collaborative competitive filtering: Learning recommender using context of user choice. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, page 295–304, New York, NY, USA. Association for Computing Machinery.

Zhao, X., Xia, L., Zhang, L., Ding, Z., Yin, D., and Tang, J. (2018). Deep reinforcement learning for page-wise recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 95–103. ACM.