

APG: Adaptive Parameter Generation Network for Click-Through Rate Prediction

Bencheng Yan*, Pengjie Wang*, Kai Zhang, Feng Li, Jian Xu and Bo Zheng[†]

Alibaba Group

{bencheng.ybc,pengjie.wpj,victorlanger.zk,adam.lf,xiyu.xj,bozheng}@alibaba-inc.com

ABSTRACT

In many web applications, deep learning-based CTR prediction models (deep CTR models for short) are widely adopted. Traditional deep CTR models learn patterns in a static manner, i.e., the network parameters are the same across all the instances. However, such a manner can hardly characterize each of the instances which may have different underlying distribution. It actually limits the representation power of deep CTR models, leading to sub-optimal results. In this paper, we propose an efficient, effective, and universal module, Adaptive Parameter Generation network (APG), where the parameters of deep CTR models are dynamically generated on-the-fly based on different instances. Extensive experimental evaluation results show that APG can be applied to a variety of deep CTR models and significantly improve their performance. We have deployed APG in the Taobao sponsored search system and achieved 3% CTR gain and 1% RPM gain respectively¹.

KEYWORDS

CTR Prediction, Pattern Learning, Adaptive Parameters

1 INTRODUCTION

Recently, deep CTR models have achieved great success in various web applications such as recommender systems, web search, and online advertising [5, 11, 15, 30]. Formally, a traditional deep CTR model can be expressed as $y_i = \mathcal{F}_\Theta(x_i)$ where x_i, y_i are the input features and the predicted CTR of the instance i respectively, Θ is the parameter, and \mathcal{F} is usually implemented as a neural network in deep CTR models.

Improving the performance of deep CTR models has been a heated topic in the research and industrial areas. Existing works can be broadly divided into two categories: (1) Focusing on x_i , more and more elaborate information (e.g., user behavior features [22, 37], multimodal information [4, 12], knowledge graph [29, 36], etc) is

¹Note this is a significant improvement in the sponsored search system of Taobao, the largest e-commerce platform in China.

* These authors contributed equally to this work and are co-first authors.

[†] Corresponding author.

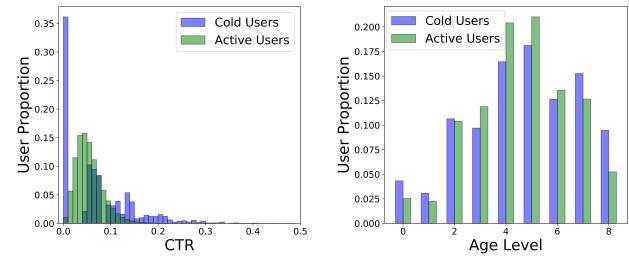
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, July 2017, Washington, DC, USA

© 2022 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>



(a) The distribution of CTR from different user groups
(b) The distribution of age level from different user groups

Figure 1: An example of data distribution from different users (i.e., active vs. cold users). (a) A custom pattern should be designed to capture the CTR distribution of different user groups. (b) A common pattern is welcomed to contribute each groups on age distribution modeling.

introduced to enrich feature space (i.e., x_i) and helps model better understand instances; (2) Focusing on \mathcal{F} , advanced architectures (including feature interaction modeling [5, 11, 30], automated architecture search [15, 26] and so on) are designed to improve the model performance.

However, few works focusing on the improvement of the model parameters Θ , especially for the weight matrix $W \in \mathbb{R}^{N \times M}$ used in hidden layers of deep CTR models². It is another orthogonal aspect for the performance improvement. Actually, most of the existing works simply adopt a static manner, i.e., all the instances share the same parameters W . We argue that such a manner is sub-optimal for pattern learning and limits the representation power of deep CTR models. On the one hand, although the common patterns among instances can be captured by the shared parameters W , it is not friendly to custom pattern modeling. Specifically, taking the Taobao sponsored search system as an example, the data distribution can be varied from different users (e.g., active vs. cold users), different categories (e.g., clothing vs. medicine), and so on (see Figure 1 (a) as an example). Simply applying the same parameters across all the instances can hardly capture the characteristic of each instance from different distributions. On the other hand, the learned common pattern may not be suitable for each of the instances. For example, the shared parameters tend to be dominated by high-frequency features and may give a misleading decision for the long-tailed instances. This leads us to the following question: *Do we really need the same and shared parameters for all instances?*

²For simplicity, in this paper, we mainly discuss the weight matrix W . Our method can also be easily applied to the parameters of other modules (e.g., transformer, attention network, etc) in deep CTR models since most of them can be regarded as a variety of MLP with a set of weight matrices [35].

Ideally, besides modeling the common pattern, the parameters should be more adaptive and can be dynamically changed for different instances to capture custom patterns at the same time. Then, the representation power (or model capacity) can be enhanced by the dynamically changed parameter space even if the weight matrix size is unchanged (e.g., $N \times M$). To achieve this goal, we propose a new paradigm for CTR prediction. The key insight is to propose an Adaptive Parameter Generation network (APG) to dynamically generate parameters depending on different instances. Formally, it can be expressed as $y_i = \mathcal{F}_{\mathcal{G}(z_i)}(x_i)$ where \mathcal{G} refers to APG which generates the adaptive parameters $W_i \in \mathbb{R}^{N \times M}$ by the input-aware condition $z_i \in \mathbb{R}^D$ (see Section 3.1). Naturally, there are two challenges for APG: (1) **Time- and memory- efficiency**. Directly generating the weight matrix W_i of a deep CTR model needs $O(NMD)$ cost in computation and storage, which grows linearly with NM and is D times larger than the cost in a regular deep CTR model (see Section 3.2). Such a large extra cost is often prohibitive, especially for a web-scale application where N, M are usually set as a large value (e.g., $N = M = 1,000$) Thus, a time- and memory- efficient parameter generation process is required; (2) **Pattern-effectiveness**. If we directly generate the weight matrix W_i totally depending on the condition z_i , the generated matrix may only capture the custom patterns and ignore the common patterns which do contribute to better understanding the click behaviors (see Figure 1 (b)), which is sub-effective for the pattern learning. Thus how to simultaneously consider the custom and common patterns is a big challenge.

In this paper, we propose an efficient and effective APG to address the above challenges: (1) **For the efficiency**, motivated by the low-rank methods [1, 17] which show that the weight matrix resides on a low intrinsic dimension, we parameterize the target weight matrix W_i as the production of three low-rank matrices $U_i S_i V_i$ where $U_i \in \mathbb{R}^{N \times K}, V_i \in \mathbb{R}^{K \times M}, S_i \in \mathbb{R}^{K \times K}$ to significantly reduce the computation and storage cost from $O(NMD)$ to $O((NK + MK)D)$ and keep high performance at the same time (see Section 3.2.1). Then, to further reduce the complexity of the generation process and efficiently model the common patterns, we take the center matrix S_i as the specific parameters which are dynamically generated to capture the custom patterns and the rest two matrices U, V as the shared parameters which are shared across instances to capture common patterns. In this way, the complexity of generating the specific parameters can be easily controlled by setting a small K (i.e., $K \ll \min(M, N)$), resulting in the computation and storage complexity of $O(KKD)$ which is free from the large value N, M (see Section 3.2.3). Besides, with the further proposed Decomposed Feed-Forwarding, the total time cost of a deep CTR model with APG per layer is reduced to $O((N + M)K + KKD)$ compared with the cost $O(NM)$ in a regular deep CTR model (see Section 3.3); (2) **For the effectiveness**, besides introducing two kinds of parameters to model the custom and common patterns (mentioned above), we further propose some strategies to improve these two pattern learning respectively. (a) For the shared parameters, we extend U and V to an over parameterization version which enriches the model capacity without any additional memory and time cost during inference (see Section 3.2.4). (b) For the specific parameters, we borrow the idea of conditional computation [2, 6] and propose

three kinds of condition strategies for z_i to meet various requirements in real applications. Furthermore, we also find the semantic relationships among different generated S_i which implicitly model the common information (see Section 4.6).

All in all, APG firstly designs input-aware condition z_i , and then, according to the condition, combines the shared and specific parameters to generate adaptive parameters through Re-Parameterization which consists of low-rank parameterization, decomposed feed-forwarding, parameter sharing, and over parameterization. Here, we present the nice and desirable properties of APG: (1) **Efficiency**. The generation process of the adaptive parameters is time- and memory- efficient by re-parameterization. Furthermore, since the generation process is computed on-the-fly, there is no need to store the specific parameters for different instances to save memory cost; (2) **Effectiveness**. On the one hand, both the custom and common patterns are captured effectively. On the other hand, due to the input-aware adaptive parameters, APG has significantly enlarged the parameter space and improved the representation power; (3) **Compatibility**. It is orthogonal to many prior methods and is a universal module that can be easily applied in most existing deep CTR models.

It is worth noting that, there are some research areas that bring a coarse-grained parameter modeling strategy that may be related to our goals, including multi-domain learning [16, 25] and multi-task learning [20, 21]. Both of them maintain and allocate different parameters to different domains or tasks manually. However, such a coarse-grained parameter allocation can hardly be extended to a fine-grained (e.g., user, category, or instances sensitive) manner. It not only cost too much memory to maintain and store a large number of parameters when considering the fine-grained modeling but also is lack flexibility and extensibility since the parameter allocation is manually pre-defined.

In summary, the contributions of this paper are presented as follows:

- We propose a new learning paradigm in deep CTR models where the model parameters are input-aware and dynamically generated to boost the representation power.
- We present APG which generates the adaptive parameters in an efficient and effective way. On the one hand, low-rank parameterization decomposed feed-forwarding is proposed to significantly reduce the memory and computation cost. On the other hand, the design of condition, shared parameters, and over-parameterization is presented to capture the custom and common patterns effectively.
- Extensive experimental evaluation results demonstrate that the proposed method is a universal module and can improve the performance of most of the existing deep CTR models. Besides, we have developed APG in the Taobao sponsored search system and achieved 3% CTR gain and 1% RPM gain respectively.

2 RELATED WORK

2.1 Deep CTR Models

A traditional CTR prediction method usually adopts a deep neural network to capture the complex relations between users and items. Most of them focus on (1) introducing abundant useful informations

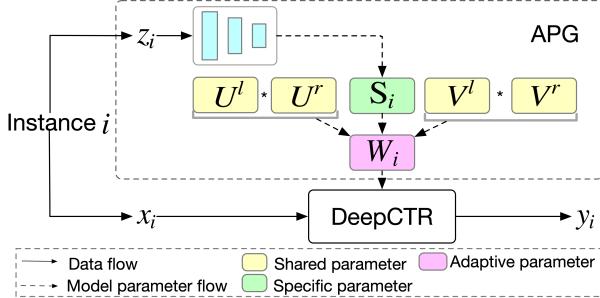


Figure 2: The framework of APG. For simplicity, the process of decomposed feed-forwarding (described in Section 3.2.2) is omitted in this figure.

(e.g., user behavior features [22, 37], multimodal information [4, 12], knowledge graph [29, 36], etc) to improve the model understanding, (2) designing advanced architecture (including feature interaction modeling [5, 11, 30], automated architecture search [15, 26], etc) to achieve better performance. Although these methods can learn some useful patterns and show competitive performance, the static parameters manner limit the model capacity of these methods, leading to sub-optimal performance.

2.2 Coarse-grained Parameter Modeling

There are some research areas that try to develop a coarse-grained parameter modeling strategy that may be related to our goals, including multi-domain learning and multi-task learning.

Multi-domain learning [16, 25]. For example, Star [25] uses a central parameter to learn the common knowledge among different domains, and allocates different parameters to different scenes (i.e., domains) to capture the domain-specific knowledge. However, such a coarse-grained parameter allocation is not friendly to extend to a fine-grained (e.g., user, category, or instances sensitive) version. On the one hand, it needs a huge memory cost to store a large number of parameters when adopting a fine-grained allocation. On the other hand, it lacks flexibility and extensibility due to the manually pre-defined parameter allocation.

Multi-task learning [20, 21]. The key idea is sharing the information in the bottom layer and learning the task-aware information in separate task-specific output layers (e.g., MMOE [20]). Similar to multi-domain learning, this kind of method also has drawbacks in parameter storage, flexibility, and extensibility when extending to a fine-grained manner. Besides, although manually separating the parameters of the output layer is helpful for the different task modeling where the label space may be different, such a design cannot sufficiently exploit the common knowledge in the label space for a single task problem, i.e., the CTR prediction.

2.3 Dynamic Deep Neural Networks

Our method is related to recent works of dynamic neural networks used in computer vision and natural language processing, in which the model parameters [9, 14, 23, 31] and architecture [18, 34] can be dynamically changed. In our paper, we take the first step to bring the idea about dynamic networks into deep CTR models and

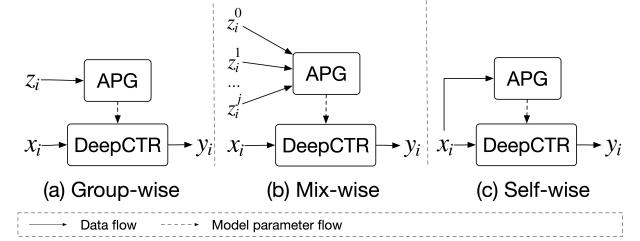


Figure 3: An example of different condition designs.

develop it in real applications where the efficiency in computation and memory is extremely needed and the common and custom pattern learning are also required.

3 APG

In this paper, we denote scalars, vectors and tensors with lower-case (or upper-case), bold lower-case, and bold upper-case letters, e.g., n (or N), x , X , respectively.

The goal of APG is to dynamically generate parameters W_i by different condition z_i , i.e.,

$$W_i = \mathcal{G}(z_i) \quad (1)$$

Then the generated parameters are applied to the deep CTR models, i.e.,

$$y_i = \mathcal{F}_{\mathcal{G}(z_i)}(x_i) \quad (2)$$

Then we need to answer the following two questions:

- Q1: What is the condition z_i for different instance i ?
- Q2: How to generate parameters efficiently and effectively?

Next, we present the design of the condition in Section 3.1 and the parameters generation by re-parameterization in Section 3.2 to answer the aforementioned questions respectively.

3.1 Condition Design

As a matter of fact, z_i takes an important role of APG since different z_i gives different prior knowledge on the parameter space design. In this paper, we propose three kinds of strategies (including group-wise, mix-wise, and self-wise) to design different kinds of z_i .

3.1.1 Group-wise. The group-wise strategy tries to generate different parameters depending on different group instances. The purpose of designing this strategy is that, sometimes, the instances can be divided into different clusters [3] and each cluster may have similar patterns. Thus, designing the adaptive parameters for each group (or cluster) can help the model to better characterize the data and give a more suitable pattern for different groups (see Figure 3 (a)). An example can be found in Appendix A.1.

3.1.2 Mix-wise. To further enrich the expression power and flexibility of the adaptive parameters, a mix-wise strategy is designed to take multiple conditions into consideration. Specifically, given k condition embeddings $\{z_i^j \in \mathbb{R}^d | j \in \{0, 1, \dots, k-1\}\}$ ³ of the instance i , we propose two aggregation policies to consider different

³For simplicity, the embedding dimensions of different conditions are set the same.

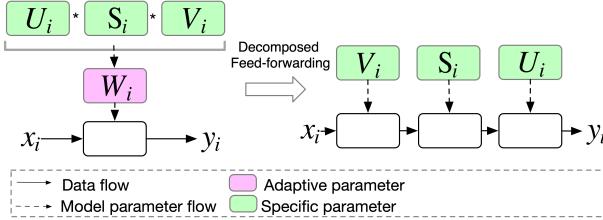


Figure 4: An example of the decomposed feed-forwarding.

conditions at the same time (see Figure 3 (b)). Examples can be found in Appendix A.2.

Input Aggregation. This policy firstly aggregates different condition embeddings and then feeds the aggregated embeddings into the parameters generation network \mathcal{G} to obtain the mix-wise based parameters. The aggregation function can be (but are not limited to): (1) Concatenation function: $z_i = \text{concat}(z_i^j)$; (2) Mean function: $z_i = 1/k \sum_j z_i^j$; (3) Attention function: $z_i = \sum_j \alpha_j z_i^j$ where α_j refers to the attention score calculated by self-attention.

Output Aggregation. This policy firstly feed different z_i^j into different APG and obtain the corresponding parameters W_i^j respectively. Then the aggregation is applied to these adaptive parameters W_i^j . Similarly, the aggregation function includes Concatenation, Mean, and Attention.

3.1.3 Self-wise. The above two strategies including Group-wise and Mix-wise need additional prior knowledge to generate parameters. Self-wise strategy tries to adopt a simple and easily obtained knowledge (i.e., self-knowledge) to guide parameters design, i.e., the network parameters are generated by its own input (see Figure 3 (c)). For example, for the 0-th hidden layer of a deep CTR model, we can set $z_i = x_i$. For l -th hidden layer, $z_i = h^{l-1}$ where h^{l-1} is the input of the l -th hidden layer. In this strategy, $z_i \in \mathbb{R}^M$ where M is the width of the current hidden layer.

3.2 Re-Parameterization

After obtaining the conditions, a basic idea is to adopt a multilayer perceptron⁴ to generate parameters depending on these conditions (see Figure 5 (b)), i.e.,

$$W_i = \text{reshape}(\text{MLP}(z_i)) \quad (3)$$

where $W_i \in \mathbb{R}^{N \times M}$, $z_i \in \mathbb{R}^D$, and the operation *reshape* refers to reshaping the vectors produced by the *MLP* into a matrix form. Then a deep CTR model with APG can be expressed as:

$$y_i = \sigma(W_i x_i) \quad (4)$$

where σ is the activation function. Here we take a deep CTR model with MLP layer as an example, and other deep CTR models can also be easily extended since most of them can be regarded as a variety of MLP with a set of weight matrices [35]

⁴In this paper, we take MLP as an example, and other implementations of \mathcal{G} can also be considered.

However, as introduced in Section 1, directly generating this weight matrix has two challenges: (1) **Time- and memory- efficiency.** Assuming that APG adopts a single perceptron layer in Eq 3, the generation needs $O(NMD)$ computation cost and $O(NMD)$ memory cost. Then the computation cost in a deep CTR model with APG is $O(NMD + NM)$ per layer where NM refers to the cost of the feed-forwarding in Eq 4. It means the computation (or memory costs) are $D + 1$ (or D) times larger than the cost of the corresponding hidden layer in a regular deep CTR model (see Table 1 and 2). Such a huge extra time and memory cost will definitely influence the model serving and may not be tolerated by real applications; (2) **Pattern-effectiveness.** In the current version, W_i is totally dependent on the given condition z_i , which may ignore the common pattern modeling. Actually, such a common pattern usually gives useful information for CTR prediction. To address the above two challenges, we propose Re-parameterization including low-rank parameterization, decomposed feed-forwarding, parameter sharing, and over parameterization to parameterize the weight matrix in an efficient and effective way.

3.2.1 Low-rank parameterization. Inspired the recent success of the low-rank methods [1, 17] which have demonstrated that strong performance can be achieved by optimizing a task in a low-rank subspace, we hypothesize that the adaptive parameters also have a low “intrinsic rank”. To this end, we propose to parameterize the weight matrix $W_i \in \mathbb{R}^{N \times M}$ as a low-rank matrix, which is the product of three sub-matrices $U_i \in \mathbb{R}^{N \times K}$, $S_i \in \mathbb{R}^{K \times K}$, $V_i \in \mathbb{R}^{K \times M}$ and the rank $K \ll \min(N, M)$ (see Figure 5 (c)). Formally, the weight generation process can be expressed as:

$$U_i, S_i, V_i = \text{reshape}(\text{MLP}(z_i)) \quad (5)$$

Intuitively, we can set a small value of K to control the time and memory cost. Meanwhile, ignoring the substantial storage and computation cost, K can also be set to a higher value to enlarge the adaptive parameter space (see detailed discussion in Section 4.5). Overall, through the low-rank parameterization, we can significantly reduce dimensionality in the parameter space, i.e., $K \ll \min(N, M)$, to enable a more compact model. Specifically, replacing the generation of the large weight matrix W_i in Eq 3 with the three sub-matrices in Eq 5 can reduce the computation complex from $O(NMD)$ to $O((NK + MK + KK)D)$ (see Table 1). Since $K \ll \min(N, M)$, the complexity cost is approximated to $O((NK + MK)D)$. The memory cost is also reduced to $O((NK + MK)D)$ (see Table 2).

3.2.2 Decomposed Feed-forwarding. After low-rank parameterization, the Eq 4 can be written as

$$y_i = \sigma(W_i x_i) = \sigma((U_i S_i V_i) x_i) = \sigma(U_i(S_i(V_i x_i))) \quad (6)$$

Here instead of directly reconstructing the weight matrix W_i by the sub-matrix production, we design a decomposed feed-forwarding and apply x_i to each sub-matrix sequentially (see Figure 4). Such a design helps us avoid the heavy computation of the sub-matrix production which costs $O(NKK + NMK)$. Actually, this design benefits from the low-rank parameterization which naturally supports the decomposed feed-forwarding.

Since the computation complexity of the decomposed feed-forwarding is $O(NK + KK + MK)$ which is approximated to $O(NK + MK)$ due to $K \ll \min(N, M)$, the total time cost of a deep CTR model with APG

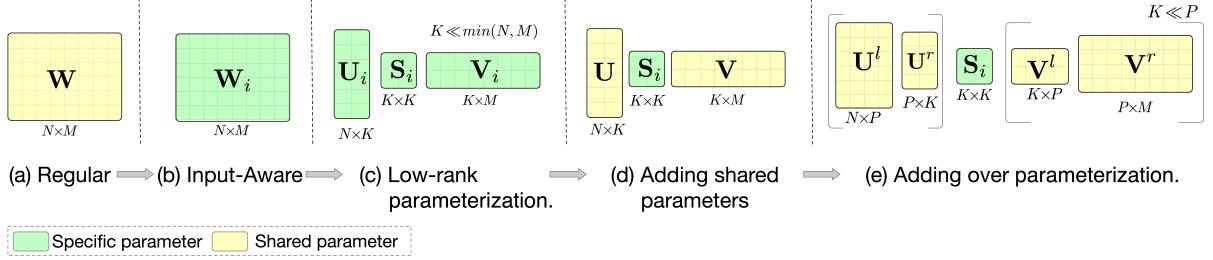


Figure 5: An example of Re-Parameterization.

Table 1: The computation complexity of different versions of APG. Since over parameterization dose not introduce any additional time cost, it is not depicted here.

	specific param generation	reconstructing W_i	feed-forwarding in a deep CTR model	Total
Wx_i	-	-	$O(NM)$	$O(NM)$
$W_i x_i$	$O(NMD)$	-	$O(NM)$	$O(NMD + NM)$
$(U_i S_i V_i) x_i$	$O((NK + MK)D)$	$O(NKK + NMK)$	$O(NM)$	$O((NK + MK)D + NKK + NMK + NM)$
$U_i(S_i(V_i x_i))$	$O((NK + MK)D)$	-	$O(NK + MK)$	$O((NK + MK)(D + 1))$
$U(S_i(V x_i))$	$O(KKD)$	-	$O(NK + MK)$	$O(KKD + NK + MK)$

Table 2: The memory complexity of different versions of APG. The memory cost of APG comes from two parts (1) the MLP used in Eq 3,5,7 to generate the specific parameters (2) the added shared parameters. Since over parameterization dose not introduce any additional memory cost, it is not depicted here.

	specific parameter generation	added shared parameters	Total
Wx_i	-	$O(NM)$	$O(NM)$
$W_i x_i$	$O(NMD)$	-	$O(NMD)$
$(U_i S_i V_i) x_i$	$O((NK + MK)D)$	-	$O((NK + MK)D)$
$U_i(S_i(V_i x_i))$	$O((NK + MK)D)$	-	$O((NK + MK)D)$
$U(S_i(V x_i))$	$O(KKD)$	$O(NK + MK)$	$O(KKD + NK + MK)$

per layer is reduced from $O(NMD + NM)$ to $O((NK + MK)(D + 1))$ (see Section 3.3 and Table 1 for detailed analysis).

3.2.3 Parameter sharing. As described in Section 1, the common patterns also take an important role in CTR prediction. Here, we present our common pattern modeling strategy. Thanks to the decomposition the weight matrix into U_i, S_i , and V_i , it allows us to be more flexible. Consequently, we divide these three matrices into (1) *specific parameters* that capture custom patterns from different instances; (2) *shared parameters* that are shared across instances to characterize the common patterns. Specifically, we define S_i as the specific parameters since the matrix scale is totally controlled by K and is more efficient for the generation process. U and V are regarded as the shared parameters (see Figure 5 (d)). Then we can rewrite Eq 5 and 6 as:

$$S_i = \text{reshape}(\text{MLP}(z_i)) \quad (7)$$

$$y_i = \sigma(U(S_i(V x_i))) \quad (8)$$

Since the size of the generated specific parameter is reduced to $K \times K$, the computation complexity of the specific parameter generation can be further reduced from $O((NK + MK)D)$ in Eq 5 to $O(KKD)$ in Eq 7, and the total time cost of a deep CTR model with APG

is reduced to $O(KKD + NK + MK)$ (see Table 1). Meanwhile the memory cost is also reduced to $O(KKD + NK + MK)$ where NK and MK refers to the storage of the shared parameters U and V respectively (see Table 2).

3.2.4 Over Parameterization. Compared with the shared weight matrix $W \in \mathbb{R}^{N \times M}$ in a regular deep CTR model, in APG, U and V can be hardly scaled to large matrices due to the efficiency constraint with $K \ll \min(N, M)$, leading to a possible performance drop. To address this problem and further enlarge the model capacity, we follow the idea about the over parameterization [1, 7] to enrich the number of the shared parameters (see Figure 5 (e)). Specifically, we replace shared parameters in Eq 8 with two large matrices, i.e.,

$$U = U^l U^r \quad (9)$$

$$V = V^l V^r \quad (10)$$

where $U^l \in \mathbb{R}^{N \times P}, U^r \in \mathbb{R}^{P \times K}, V^l \in \mathbb{R}^{K \times P}, V^r \in \mathbb{R}^{P \times M}$ and $P \gg K$.

Although U (or V) is exactly equal to $U^l U^r$ (or $V^l V^r$) at the mathematical perspective, replacing V with $V^l V^r$ can contribute into two folds: (1) Since $P \gg K$, more shared parameters are introduced

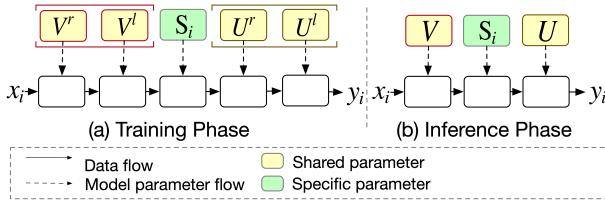


Figure 6: An example of over parameterization.

to enlarge the model capacity [7, 8]; (2) The form as the multiple matrix production can result in an implicit regularization and thus enhance generalization [1].

No Additional Inference Latency and Memory Cost. During the training phase, we can set P to a large value to enlarge the model representation power. Remarkably, when $P > \max(N, M)$, the shared parameter space can be large than that in a regular deep CTR model. During the inference phase, we explicitly pre-compute and store V, U , and use these two matrices for inference (see Figure 6). Critically, this guarantees that we can introduce abundant shared parameters without any additional latency and memory cost during the inference phase. Note we do not apply over parameterization to S_i since it can not benefit the efficiency where the dynamically generated specific parameters cannot be pre-computed and pre-stored during inference. Besides replacing S_i with two large matrices also need heavy computation to generate these two matrices.

3.3 Complexity

In this section, we detailed analyze the proposed model complexity including the memory and computation complexity during the inference phase. For analysis, the parameters generation network is implemented as a single perceptron layer, and the per layer costs in a regular deep CTR model and an adaptive deep CTR model are compared. A summarization can be found in Table 1 and 2.

Memory Complexity. For a regular deep CTR model, the memory cost is $O(NM)$ per layer, i.e., storing the shared weight matrix. For APG, the memory cost comes from two parts: (1) The memory cost of generating S_i in Eq 7 is $O(KKD)$; (2) The shared parameters U, V in Eq 9,10 cost $O((N + M)K)$ during the inference phase. Then the total memory complexity of APG is $O(NK + MK + KKD)$ per layer.

Computation Complexity. A regular deep CTR model needs $O(NM)$ per layer for the feed-forward computation. APG needs $O(KKD)$ in Eq 7 to calculate the specific parameters. Meanwhile, the corresponding feed-forward computation of a deep CTR model is $O(NK + KK + MK)$ by the decomposed feed-forwarding in Eq 8. In total, the computation complexity is $O(KKD + NK + KK + MK)$. Since $K \ll \min(N, M)$, the computation cost is nearly $O(KKD + NK + MK)$.

In summary, APG has $O(KKD + NK + MK)$ in bost memory cost and computation cost. Since $K \ll \min(N, M)$ and D is usually smaller than N, M ⁵, the memory and computation cost of APG can

⁵The width (i.e., M and N) of the hidden layer in a deep CTR model is usually scaled to the sum of dimensionality (e.g., nd) of all (e.g., n) feature embeddings. Thus when adopting the group-wise or mix-wise strategy, the condition embedding dimensionality

even be smaller than that (i.e., $O(NM)$) in a regular deep CTR model. The experimental results in Section 4.4 also show the efficiency of APG.

4 EXPERIMENTS

4.1 Experimental Settings

The detailed experimental settings including datasets, baselines, training details are presented in Appendix B.

Datasets. Four real-world datasets are used including **Amazon**, **MovieLens**, **IAAC**, and **Alibaba**. The first three are public datasets and the last is an industrial dataset.

Baselines. Here, we compare our method with two kinds of methods (1) *Existing CTR prediction methods* include WDL[5], PNN[24], FIBINET[13], DIFM[19], DeepFM[11], DCN[30], and AutoInt[27]; (2) *Coarse-grained parameter modeling methods*. multi-task learning: MMoE [20] and multi-domain learning: Star [25]. Note due to the limited space, we present the results with the coarse-grained parameter modeling methods in Appendix E.

Training Details. See Appendix B.3 for the detailed introduction.

4.2 Performance Evaluation with Existing Deep CTR Models

We first apply APG on CTR prediction tasks on public datasets. Since APG is a universal module and can be applied to most of the existing deep CTR models. Hence, to evaluate the performance of APG, we apply APG on various existing deep CTR models, and report the results of the original model (denoted as Base) and the model with APG. Here, AUC (%) [10] score is reported as the metric⁶.

The results are shown in Table 3. We can find that with the help of APG, all of the methods achieve a significant improvement on all datasets. For example, the gains of DCN is 0.33% 1.61% (other methods also can obtain similar improvement). It demonstrates that (1) giving adaptive parameters for models can enrich the parameter space and learn more useful patterns for different instances; (2) the proposed APG is a universal framework that can boost the performance of many other methods. Such nice property encourages APG to be applied to various scenarios and various methods.

4.3 Evaluation of Condition Design

In this section, we analyze the influence of the condition design. Specifically, we evaluate different condition strategies on CTR prediction tasks and report the AUC results. We take WDL as the backbone of APG. For the mix-wise strategy, the input aggregation with attention function is used and the effect of the aggregation method of the mix-wise strategy is detailed analyzed in Appendix C. Besides we also provide the results of WDL (defined as Base) for comparison.

The effect of different condition strategies. We first analyze the effect of different condition strategies, including the group-wise,

⁶ $D = d$ is usually much smaller than N and M . When adopting a self-wise strategy, $D = M$.

⁶Note 0.1% absolute AUC gain is regarded as significant for the CTR task [5, 28, 37].

Table 3: The AUC (%) results of Click-Through Rate (CTR) prediction on different datasets. Note Base refers to the original results of the corresponding methods and Base+APG refers to the results with the help of APG.

Data	Method	WDL	PNN	FIBINET	DIFM	DeepFM	DCN	AutoInt
MovieLens	Base	79.21	79.5	79.78	79.84	79.3	79.29	79.36
	Base+APG	79.60	79.67	79.82	79.94	79.60	79.62	79.64
Amazon	Base	69.15	69.16	68.88	69.17	69.1	68.98	68.96
	Base+APG	69.37	69.37	69.19	69.23	69.43	69.42	69.38
IAAC	Base	65.17	65.3	65.15	65.76	65.64	64.78	64.99
	Base+APG	65.86	65.87	66.15	66.42	66.17	66.39	66.21

Table 4: The AUC (%) results of APG with different kinds of condition strategy, including Group-wise, Mix-wise, and Self-wise. Note Base refers to the results of the method without APG. U, I, and C refers to the embedding of users, items, and contexts respectively.

Strategy	z_i	MovieLens	Amazon	IAAC
Base		79.21	69.15	65.17
Group-wise	U	79.58	69.28	65.80
	I	79.58	69.35	65.76
Mix-wise	U,I,C	79.45	69.31	65.90
Self-wise	x_i	79.60	69.37	65.86

the mix-wise, and the self-wise strategies. From Table 4, we can find that (1) Compared with Base, all of these condition strategies can obtain better performance. It indicates the effectiveness of APG which can dynamically generate the parameters for better pattern learning; (2) The self-wise strategy achieves the best performance among other strategies in most cases. One of the possible reasons is that the prior knowledge of the self-wise strategy is directly from the hidden layers' input which is a more immediate signal for the current layer and may lead to better parameter generation.

The effect of different prior knowledge in the same strategy. We also evaluate the effect of different prior knowledge in the same strategy. Specifically, considering the group-wise strategy, we compare the performance between the prior knowledge about the user embedding and the item embedding. The results are reported in Table 4. We can conclude that (1) Different prior knowledge in the same strategy can get competitive performance; (2) Designing a proper prior knowledge may achieve better performance. For example, for group-wise strategy, taking the item embedding as the prior knowledge obtains higher AUC on Amazon. While the prior knowledge about the user embedding performs better on IAAC.

4.4 Evaluation of Re-Parameterization

In this section, we implement various versions of APG and conduct experiments to evaluate the effectiveness and efficiency of the re-parameterization

4.4.1 The impact of the low-rank parameterization. The purpose of low-rank parameterization is to reduce the computation and

memory cost and keep high performance at the same time. Thus, we conduct two versions of PGNnet including directly generating adaptive parameters in Eq 3 (denoted as v1), adopting low-rank parameterization in Eq 5 (denoted as v4)⁷. From Table 5, we can observe that (1) Compared with the base, the computation cost of v1 is 9.55~39.48× larger than base, and v1 also need 1.25~2.62× more storage⁸. It indicates that directly generating the specific parameters needs an amount of extra time and memory cost (as discussed in Section 3.2) which definitely influences the model severing; (2) Compared with v1, with the help of low-rank parameterization, we only need 1.09~1.42× time cost and 1.00~1.05× memory cost and achieves high performance at the same time. It verifies the efficiency of the proposed low-rank parameterization.

4.4.2 The impact of the decomposed feed-forwarding. As described in Section 3.2.2, we can leverage the factorized form of the weight matrix to further reduce the computation cost by the decomposed feed-forwarding. Here, we compare the time cost of version v2 and v3 in Table 5 to evaluate the efficiency of the decomposed feed-forwarding. It can be found that (1) Due to the heavy computation of the weight matrix production, although the adaptive parameter generation process can be reduced by low-rank parameterization, the total time cost of v2 is still 10.66~45.29× larger than Base; (2) Compared to v2, making use of the decomposed low-rank matrix during the feed-forwarding of deep CTR models, v3 can significantly save computation time and can even be faster than Base (e.g., 0.84× in MovieLens and 0.91× in Amazon) which also verifies the analysis in Section 3.3.

4.4.3 The impact of the parameter sharing. As introduced in Section 1, common patterns play an important role in CTR prediction. In APG, we introduce the shared parameters to characterize such patterns. Here, we detailedly analyze the influence of the shared parameters. Specifically, we firstly compare the performance of versions v3 and v4 in Table 5. We can find that compared with v4, v3 can further improve the performance by introducing the shared parameters, and meanwhile, the time cost of v3 can also be reduced due to the smaller size of the generated specific parameters. Then, to further fairly evaluate the shared parameters, we implement a version v5 where the shape of the specific parameters is set the same

⁷By default, v4 adopt DFF since low-rank parameterization also contributes the efficiency of DFF (see Section 3.2.2)

⁸Because in this paper the embedding layer, which is storage costly in deep CTR models [33], is parameterized with share parameters, the storage cost does not increase deeply as the computation cost in v1.

Table 5: The results of the evaluation about different versions of APG. Note LP refers to the low-rank parameterization. $|\Theta_i|$ and $|\Theta|$ refer to the specific and shared parameter size in a deep CTR model respectively. DFF refers to the decomposed feed-forwarding. We report AUC (%) as the performance metric, second/epoch as the time cost, and megabytes as the memory cost, and the corresponding ratio compared with Base is also presented. Besides, in this part, to avoid the influence of over parameterization, all of the versions of APG do not adopt over parameterization.

Version	Setting				MovieLens				Amazon				IAAC							
	LP	$ \Theta_i $	$ \Theta $	DFF	AUC	Time		Memory		AUC	Time		Memory		AUC	Time		Memory		
						Cost	Ratio	Cost	Ratio		Cost	Ratio	Cost	Ratio		Cost	Ratio	Cost	Ratio	
Base	Wx_i	-	X	NM	-	79.21	36.01	1.00	61.5	1.00	69.15	21.72	1.00	150.6	1.00	65.17	5.05	1.00	27.7	1.00
v1	$W_i x_i$	X	NM	X	-	79.51	800.17	22.22	94.1	1.53	69.33	857.49	39.48	187.5	1.25	65.52	48.23	9.55	72.5	2.62
v2	$(US_iV)x_i$	✓	KK	$(N+M)K$	X	79.56	857.12	23.80	60.6	0.99	69.34	983.68	45.29	149.6	0.99	65.67	53.85	10.66	27.6	1.00
v3	$U(S_i(Vx_i))$	✓	KK	$(N+M)K$	✓	79.56	30.41	0.84	60.6	0.99	69.34	19.66	0.91	149.6	0.99	65.67	5.47	1.08	27.6	1.00
v4	$U_i(S_i(Vx_i))$	✓	$(N+M+K)K$	X	✓	79.49	39.2	1.09	62.0	1.01	69.24	33.1	1.52	151.1	1.00	65.61	7.15	1.42	29.1	1.05
v5	$U(W_i(Vx_i))$	✓	NM	NN+MM	✓	79.58	955.38	26.53	97.0	1.58	69.40	1217.41	56.05	190.5	1.26	66.04	64.37	12.75	75.8	2.74

Table 6: The evaluation results of the over parameterization.

AUC(%)	Version	MovieLens	Amazon	IAAC
Base	W	79.21	69.15	65.17
v3	US_iV	79.56	69.34	65.67
v6	$U^l U^r S_i V^l V^r$	79.60	69.37	65.86

as v1 (ignoring the heavy memory and time cost). We can observe that v5 makes a significant improvement on AUC, compared with v1, especially in IAAC. It demonstrates the effectiveness of APG in common pattern modeling.

4.4.4 The effect of the over parameterization. In this part, we evaluate the effect of the proposed over parameterization. Specifically, we firstly compare the performance with or without over parameterization (i.e., v3 vs. v6 in Table 6). It shows that v6 performs better than v3 in all cases, which indicates adding more shared parameters can enrich the model capacity and lead to better performance. Besides, since there is no additional time and memory cost through over parameterization during inference (see Section 3.2.4), such a nice property gives us more flexibility in the parameter space design. Furthermore, we extend over parameterization to a stack version which further improves the performance (see Appendix F).

4.5 Evaluation of Hyper-parameters

In this section, we conduct experiments to analyze the effect of the hyper-parameters including P , K , and the number of MLP layers in APG. Due to limited space, the results on Amazon are reported here and more results on other datasets can be found in Appendix D.

4.5.1 The effect of different P . As presented in Section 3.2.4, the hyper-parameter P is introduced to add more shared parameters. Thus P is required to be much larger than K (i.e., $P \gg K$). Here we keep $K = 8$ for all cases, and set P to $\{32, 64, 128, 256, 512\}$ respectively and evaluate the performance of APG. The results are plotted in the right part (i.e., Large P) in Figure 7 (a). Note we also

provide the results of the Base and APG w/o over parameterization (denoted as APG (US_iV)) for comparison. We can find that setting different large values of P can give a similar better performance compared with APG (US_iV) in most cases. It indicates that when adding more parameters by setting a large P , the model can be stably improved.

Does a small P works? We further manually set P as a small value (e.g., $\{2, 4, 6, 8, 16\}$). From the left part (i.e., Small P) in Figure 7 (a), we can find, although APG still performs better than Base, it fails to improve the performance of APG (US_iV) due to fewer shared parameters used in APG than that in APG (US_iV).

4.5.2 The effect of different K . In this part, we evaluate the effect of different K . Specifically, we set $P = 32$ and the hidden layers of the deep CTR model as $\{256, 128, 64\}$ for all cases. Since K is required to be much smaller than $\min(N, M)$ (see Section 3.2.1), we firstly set K to $\{2, 4, 6, 8\}$ respectively. The results are reported in the left part (i.e., Small K) in Figure 7 (b). Note we also provide the results of the Base and APG w/o re-parameterization (denoted as APG (W_i)) for comparison. We can find that (1) APG performs better than Base in all cases which shows the effectiveness of APG; (2) With the increase of K , APG can also achieve better performance. It indicates that it is important to give sufficient specific parameters to characterize the custom patterns of different instances. (3) We also notice that when K is set to an extremely small value (e.g., 2 or 4), APG performs worse than APG (W_i). One of the possible reasons is that adopting too small values of K makes it difficult to capture the custom patterns.

Does a large K helps? Similar to the purpose of over parameterization, ignoring the heavy cost in storage and computation, we can also set K to a large value (e.g., $\{16, 32, 64, 128, 256, 512\}$) to see whether the model can obtain further improvements. The results are presented in the right part (i.e., Large K) in Figure 7 (b). Some observations are summarized as follows: (1) When we set a large value of K , compared with the small one, APG can further improve the model performance in most cases. It indicates increasing K to a large value does help to model the custom patterns; (2) When K is set to extremely large values (e.g., 256 or 512), APG only achieves

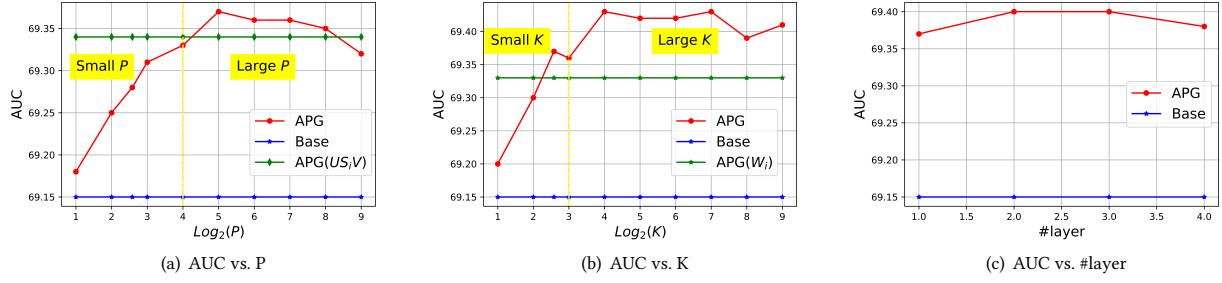


Figure 7: Evaluation of the hyper-parameters on Amazon.

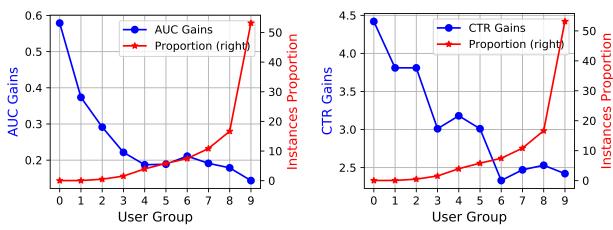


Figure 8: The gains in different user groups. Note the frequency of user group is increased from group 0 to group 9.

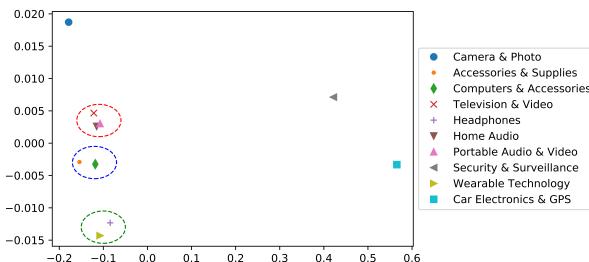


Figure 9: The visualization of the generated specific parameters.

similar performance with the cases where $K \in \{16, 32, 64, 128\}$; It shows enlarge the specific parameters does not always give a positive contribution and it is wiser to set a suitable value of K .

4.5.3 The effect of the different number of layers. Here, we increase the number of the MLP layers in APG to evaluate the performance. We also report the results of Base for comparison. The results are depicted in Figure 7 (c). Some observations are summarized as follows (1) Compared with Base, APG with the different number of layers can always achieve significant improvement; (2) Compared with the performance by a single layer, increasing the number of layers can make a slight improvement.

4.6 Visualization

In this section, we try to visualize the generated specific parameters. Specifically, we take the group-wise strategy on Amazon as an

Table 7: The results of the severing efficiency. Note RP refers to the Re-Parameterization.

	Base	APG	APG w/o RP
RT(ms)	15.4	17.2	58.6
PVR(%)	0	-0.01	-30

example. The item category is set as the condition, and there is a total of 10 different categories. Then we plot the generated specific parameters (i.e., S_i) into a 2-D space by PCA [32]. The visualization is presented in Figure 9. Each point refers to the specific parameters generated by taking one of the categories as the condition. Interestingly, the observed groupings (i.e., in the same dashed circle) correspond to similar categories. This shows that the learned specific parameters by APG are meaningful and can capture the relations among different specific parameters implicitly.

5 PERFORMANCE IN TAOBAO SPONSORED SEARCH SYSTEM

Here, we show the performance of APG on industrial applications. Specifically, we firstly train APG with the dataset Alibaba and then develop it in the sponsored search system in Taobao (the biggest e-commerce platform in China). Since December 2021, APG is developed and served as the main traffic of our system.

Overall Gains. Compared with the online model in Taobao, it achieves 0.2% gains in AUC. During the online A/B test, we observe a 3% improvement in CTR and a 1% gain on RPM (Revenue Per Mile) respectively, which shows the great effectiveness of APG.

Severing Efficiency. We further evaluate the severing efficiency of APG. The average Response Time (RT) and Page View Rate (PVR) of the model inference online are evaluated when a user search queries in Taobao. Not PVR is influenced by the request timeout. We also provide the performance of the base model and the version of APG w/o Re-Parameterization. The results are reported in Table 7. Some observations can be summarized as follows: (1) Compared with Base, APG achieves similar performance in RT and PVR. It indicates APG is a lightweight paradigm that does not need too much time cost to allow the model parameters sensitive to different instances; (2) APG w/o RP needs 3× times in RT and the

PVR also drops 30%. It shows the importance of introducing Re-Parameterization. Otherwise, such a model with heavy computation can never be tolerated in real applications.

The effect on low-frequency users. As described in Section 1, introducing the specific parameters can help the model better learn the patterns for different instances, especially for the long-tailed instances, since without the specific parameters the model can be easily dominated by the hot instances. In this part, we detailedly analyze the impact on the different instances when considering the specific parameters. Specifically, we divide the users into 10 groups by their frequency. The frequency of user group is increased from group 0 to group 9 and the number of users is set the same in different groups. Then we evaluate the improvement (including AUC and CTR) of different groups respectively. The results are reported in Figure 8. We can find that (1) Since group 9 refers to the users with the highest frequency, although it has only 10% users, this group produces more than 50% instances (see the red line in Figure 8.); (2) The specific parameters contribute more for low-frequency users since it achieves more gains of AUC and CTR in low-frequency users (e.g., group 0). It demonstrates that specific parameters do allow low-frequency instances to better represent their features, leading to better performance.

6 CONCLUSION

In this paper, we propose an efficient, effective, and universal module to adaptive generate parameters for different instances. In this way, the model can carefully characterize the patterns for different instances by adopting different parameters. Experimental results show that with the help of APG, all of the existing deep CTR models can make great improvements, which also encourages a wide application for APG.

REFERENCES

- [1] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*, 2020.
- [2] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.
- [3] Liangliang Cao, Jiebo Luo, Andrew Gallagher, Xin Jin, Jiawei Han, and Thomas S Huang. Aworldwide tourism recommendation system based on geotaggedweb photos. In *ICASSP*, pages 2274–2277. IEEE, 2010.
- [4] Xu Chen, Hanxiong Chen, Hongteng Xu, Yongfeng Zhang, Yixin Cao, Zheng Qin, and Hongyuan Zha. Personalized fashion recommendation with visual explanations based on multimodal attention network: Towards visually explainable recommendation. In *SIGIR*, pages 765–774, 2019.
- [5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.
- [6] Kyunghyun Cho and Yoshua Bengio. Exponentially increasing the capacity-to-computation ratio for conditional computation in deep learning. *arXiv preprint arXiv:1406.7362*, 2014.
- [7] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *ICCV*, pages 1911–1920, 2019.
- [8] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Diverse branch block: Building a convolution as an inception-like unit. In *ICCV*, pages 10886–10895, 2021.
- [9] Qingnan Fan, Dongdong Chen, Lu Yuan, Gang Hua, Nenghai Yu, and Baoquan Chen. Decouple learning for parameterized image operators. In *ECCV*, pages 442–458, 2018.
- [10] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.
- [11] Hufeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.
- [12] Li He, Hongxu Chen, Dingxian Wang, Shoaib Jameel, Philip Yu, and Guandong Xu. Click-through rate prediction with multi-modal hypergraphs. In *CIKM*, pages 690–699, 2021.
- [13] Tongwen Huang, Zhiqi Zhang, and Junlin Zhang. Fibinet: combining feature importance and bilinear feature interaction for click-through rate prediction. In *RecSys*, pages 169–177, 2019.
- [14] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. *NIPS*, 29, 2016.
- [15] Manas R Joglekar, Cong Li, Mei Chen, Taibai Xu, Xiaoming Wang, Jay K Adams, Pranav Khaitan, Jiahui Liu, and Quoc V Le. Neural input search for large scale recommendation models. In *KDD*, pages 2387–2397, 2020.
- [16] Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. Melu: Meta-learned user preference estimator for cold-start recommendation. In *KDD*, pages 1073–1082, 2019.
- [17] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. *arXiv preprint arXiv:1804.08838*, 2018.
- [18] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, pages 2736–2744, 2017.
- [19] Wantong Lu, Yantao Yu, Yongzhe Chang, Zhen Wang, Chenhui Li, and Bo Yuan. A dual input-aware factorization machine for ctr prediction. In *IJCAI*, pages 3139–3145, 2020.
- [20] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *KDD*, pages 1930–1939, 2018.
- [21] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *CVPR*, pages 3994–4003, 2016.
- [22] Qi Pi, Guorui Zhou, Yujing Zhang, Zhe Wang, Lejian Ren, Ying Fan, Xiaoqiang Zhu, and Kun Gai. Search-based user interest modeling with lifelong sequential behavior data for click-through rate prediction. In *CIKM*, pages 2685–2692, 2020.
- [23] Emmanouil Antonios Plataniotis, Mrinmaya Sachan, Graham Neubig, and Tom Mitchell. Contextual parameter generation for universal neural machine translation. In *EMNLP*, pages 425–435, 2018.
- [24] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. Product-based neural networks for user response prediction. In *ICDM*, pages 1149–1154. IEEE, 2016.
- [25] Xiang-Rong Cheng, Liqin Zhao, Guorui Zhou, Xinyao Ding, Qiang Luo, Siran Yang, Jingshan Lv, Chi Zhang, and Xiaoqiang Zhu. One model to serve all: Star topology adaptive recommender for multi-domain ctr prediction. *CIKM 2021*, 2021.
- [26] Qingquan Song, Dehua Cheng, Hanning Zhou, Jiyan Yang, Yuandong Tian, and Xia Hu. Towards automated neural interaction discovery for click-through rate prediction. In *KDD*, pages 945–955, 2020.
- [27] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *CIKM*, pages 1161–1170, 2019.
- [28] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *CIKM*, pages 1161–1170, 2019.
- [29] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. Knowledge graph convolutional networks for recommender systems. In *WWW*, pages 3307–3313, 2019.
- [30] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *ADKDD’17*, pages 1–7. 2017.
- [31] Ze Wang, Zichen Miao, Jun Hu, and Qiang Qiu. Adaptive convolutions with per-pixel dynamic filter atom. In *ICCV*, pages 12302–12311, 2021.
- [32] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1–3):37–52, 1987.
- [33] Bencheng Yan, Pengjie Wang, Jinquan Liu, Wei Lin, Kuang-Chih Lee, Jian Xu, and Bo Zheng. Binary code based hash embedding for web-scale applications. In *CIKM*, pages 3563–3567, 2021.
- [34] Zhihang Yuan, Bingzhe Wu, Guangyu Sun, Zheng Liang, Shiwan Zhao, and Weichen Bi. S2dnas: Transforming static cnn model for dynamic inference via neural architecture search. In *ECCV*, pages 175–192. Springer, 2020.
- [35] Weinan Zhang, Jiarui Qin, Wei Guo, Ruiming Tang, and Xiuqiang He. Deep learning for click-through rate estimation. *IJCAI 2021 (Survey Track)*, 2021.
- [36] Jun Zhao, Zhou Zhou, Ziyu Guan, Wei Zhao, Wei Ning, Guang Qiu, and Xiaofei He. Intentgc: a scalable graph convolution framework fusing heterogeneous information for recommendation. In *KDD*, pages 2347–2357, 2019.
- [37] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *KDD*, pages 1059–1068. ACM, 2018.

A EXAMPLES FOR DIFFERENT CONDITION DESIGNS

A.1 Group-wise

Example 1 (Thousand users with Thousand Models (a.k.a Personalized Parameters)). In a traditional recommendation model, although some personalized signals (e.g., user-item interactions) are introduced, the shared parameters cannot significantly characterize the personality of each user especially for long-tailed users. In our method, with the help of the group-wise strategy, we can simply give the prior knowledge about users to the model parameters to capture personalized patterns and achieve "Thousand users with Thousand Models". Specifically, given the embedding of a user m as $\mathbf{u}_i^m \in \mathbb{R}^d$ and an involved instance i of this user, we can directly set $\mathbf{z}_i = \mathbf{u}_i^m$. In other words, by the prior knowledge \mathbf{u}_i^m , we explicitly group the instances by users and allow different users to enjoy different parameters.

A.2 Mix-wise

Example 2 (Real-time Personalized Parameters). In practice, user interests may be dynamically changed. For a mix-wise strategy, we can add prior knowledge about user latest behaviors to allow the parameters sensitive with the real-time interests. Specifically, given an instance i associated with a user m , we denote $\mathbf{z}_i^{0,m} \in \mathbb{R}^d$ as the user m embedding and $\mathbf{z}_i^{1,m} \in \mathbb{R}^d$ as the latest behaviors embedding of user m ⁹. Then these two conditions $\mathbf{z}_i^{0,m}, \mathbf{z}_i^{1,m}$ are both considered to generate parameters. In this way, the generated parameters are specific for different users and can be adjusted in real-time by different user interests.

Example 3 (Thousand instances with Thousand Models (a.k.a Instance-aware Parameters)). We can consider more conditions to allow the model parameters sensitive in instances-level. To achieve this goal, the conditions are required to identify each instance. Specifically, assuming in a recommendation application, for each instance, we can take the embedding of the associated user, item, and context as \mathbf{z}_i^j .

B THE DETAILED EXPERIMENTAL SETTING

B.1 Datasets

Four real-world datasets are used: (1) **Amazon**¹⁰ is collected from the electronics category on Amazon. There are total 1,292,954 instances, 1,157,633 users. (2) **MovieLens**¹¹ is a review dataset and is collected from the MovieLens web site. There are total 1,000,209 instances, 6,040 users. (3) **IJCAI2018 Advertising Algorithm Competition (IAAC)**¹² is a dataset collected from a sponsored search in E-commerce. Each record refers to whether a user purchases the displayed item after clicking this item. There is a total of 478,138 records, 197,694 users, and 10,075 items. (3) **Alibaba** is an industrial dataset which is obtained from Taobao used for industrial evaluation (see Section 5). Each instance refers to a user who searches

a query in this platform, and the platform returns an item to this user. The label is defined as whether the user clicks this item. There are a total of 4 billion instances, 100 million users. The statistics of the data sets are summarized in Table 8.

Table 8: The statistic of datasets.

	#Data	#User ID	#Item ID
Amazon	1,292,954	1,157,633	9,560
MovieLen	1,000,209	6,040	3,706
IAAC	478,138	197,694	10,075
Alibaba	4 billion	100 million	80 million

B.2 Baselines.

Here, we compare our method with two kinds of methods

Existing CTR prediction methods: To show the effectiveness of the proposed APG, we apply it to various existing deep CTR models (1) WDL[5] adopts wide and deep parts to memorize and generalize patterns of instances. (2) PNN[24] explicitly introduces product operation to explore the interactions of categorical data in multiple fields. (3) FIBINET[13] designs a squeeze-excitation network to dynamically learn the feature importance and use a bilinear-interaction layer to learn the interactions among features. (4) DIFM[19] brings the idea of the transformer and learns vector-wise and bit-wise interactions among features. (5) DeepFM[11] takes the linear part of WDL with an FM network to better represent low-order features. (6) DCN[30] learns low-order and high-order features simultaneously and needs low computation cost. (7) AutoInt[27] learns the feature interactions automatically via self-attention neural networks.

Coarse-grained parameter modeling methods: We also try to compare the proposed APG with the coarse-grained parameter modeling methods: (1) Multi-task learning: MMoE [20] keeps multiple parameters by adopting multiple network branches for different tasks; (2) Multi-domain learning: Star [25] allocates multiple parameters for different scenarios.

B.3 Training Details

The embedding dimension is set the same for all methods. The number and units of hidden layers are set $\{256, 128, 64\}$ for all methods by default. Other hyper-parameters of different methods are set by the suggestion from original papers. The backbone of the deep CTR models is set as WDL by default in all experiments except Section 4.2. For APG, we set the self-wise strategy as the default condition strategy. \mathcal{G} is implemented as an MLP with a single layer by default. We set the hyper-parameters $K \in \{2, 4, 6, 8\}$ and $P \in \{32, 64, 128, 256, 512\}$ and perform grid search over K and P . We include the results for different values of K and P in Section 4.5. We use the Adam optimizer with a learning rate of 0.005 for all methods. The batch size is 1024 for all datasets.

⁹The latest behaviors embedding can be obtained by averaging the embedding of the latest clicked items of this user

¹⁰<https://www.amazon.com/>

¹¹<https://grouplens.org/datasets/movielens/>

¹²<https://tianchi.aliyun.com/competition/entrance/231647/introduction>

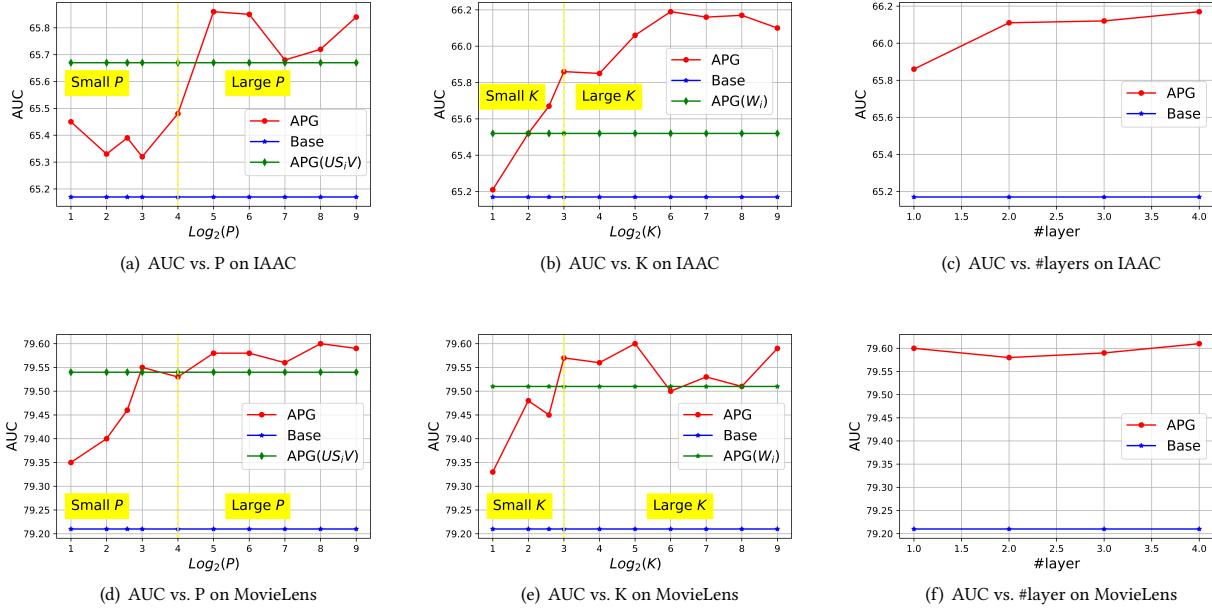


Figure 10: Evaluation of the hyper-parameters on IAAC and MovieLens.

C EVALUATION OF THE DIFFERENT AGGREGATION FUNCTIONS

In this section, we conduct experiments to evaluate the performance when using different aggregation functions for the mix-wise strategy. All of the cases use the embedding of users, items, and the context as the condition. The results are reported in Table 9. We can observe that (1) Compared with other functions, the Attention function achieves the best performance in most cases due to the high representation power of the attention function; (2) Compared with Output Aggregation, Input Aggregation performs better. One of the possible reasons is that the relations among different conditions can be implicitly modeled through \mathcal{G} while output aggregation simply summarizes different specific parameters.

Table 9: The results of using different aggratetion function for the mix-wise strategy.

AUC (%)	Function	MovieLens	Amazon	IAAC
Base		79.21	69.15	65.17
Input Aggregation	Mean	79.33	69.28	65.44
	Concat	79.38	69.36	65.73
	Attention	79.45	69.31	65.90
Output Aggregation	Mean	79.25	69.34	65.44
	Concat	79.25	69.36	65.48
	Attention	79.33	69.26	65.70

D ADDITIONAL RESULTS OF THE EVALUATION OF THE HYPER-PARAMETERS

Here, we present additional results about the performance with different P and K on IAAC and MovieLens in Figure 10. Detailed analysis can be found in Section 4.5.

E COMPARISON THE PERFORMANCE WITH COARSE-GRAINED PARAMETER MODELING METHODS

In this section, we try to apply coarse-grained parameter modeling methods (including Star and MMoE) to fine-grained versions. Since the specific parameters of Star and MMoE should be pre-defined manually, we allocate the instances (containing different items) with different parameters. To have a fair comparison, we also set APG with the group-wise condition and generate different parameters for the instance with different items. The AUC and Memory results are reported in Table 10. It can be found that compared with base, Star and MMoE can only achieve slight improvement but have huge memory costs. It indicates that such pre-defined and coarse-grained modeling methods are not suitable for fine-grained parameters modeling.

F AN EXTENSION OF THE OVER PARAMETERIZATION

Here, we extend the over parameterization into a stacking version to seek further improvement. Formally, the shared parameters U

Table 10: The comparison with coarse-grained parameter modeling methods. Mem refers to the memory cost (M).

	MovieLens		Amazon		IAAC	
	AUC	Mem	AUC	Mem	AUC	Mem
Base	79.21	61.5	69.15	150.6	65.17	27.7
Base+MMoE	79.31	186.5	69.17	465.4	65.22	361.1
Base+Star	79.28	3848.6	69.20	11195.0	65.36	13815.2
Base+APG	79.58	60.6	69.35	149.5	65.76	27.4

Table 11: The extension results of the over parameterization.

AUC(%)	Version	MovieLens	Amazon	IAAC
Base	W	79.21	69.15	65.17
v6	$U^l U^r S_i V^l V^r$	79.60	69.37	65.86
v7	$(\prod_j^J U^j) S_i (\prod_j^J V^j), J = 3$	79.59	69.39	66.13
v8	$(\prod_j^J U^j) S_i (\prod_j^J V^j), J = 4$	79.61	69.39	65.96

and V can be written as:

$$U = \prod_j^J U^j, \quad V = \prod_j^J V^j \quad (11)$$

where U and V are implemented as the production of a series of weight matrices. Actually, such multiple productions enlarge the model capacity and bring the regularization to the deep CTR models [1, 7, 8]. From the results among v6, v7, and v8 in Table 11, some observations can be found (1) v7 and v8 can make some improvement in most cases, compared with v6. It demonstrates that stacking more shared parameters can contribute to feature learning, leading to better performance; (2) Although v8 stacks more parameters, it only achieves similar results compared with v7. It indicates that stacking parameters do not always give a positive influence. Overall, it is a good choice to consider stacking more shared parameters since such stacking can still be converted into a single weight matrix during inference to avoid any extra time and memory cost.