AdaSparse: Learning Adaptively Sparse Structures for Multi-Domain Click-Through Rate Prediction

Xuanhua Yang, Xiaoyu Peng, Penghui Wei, Shaoguo Liu, Liang Wang and Bo Zheng
Alibaba Group
Beijing, China
{xuanhua.yxh,pengxiaoyu.pxy,wph242967,shaoguo.lsg,liangbo.wl,bozheng}@alibaba-inc.com

ABSTRACT

Click-through rate (CTR) prediction is a fundamental technique in recommendation and advertising systems. Recent studies have proved that learning a unified model to serve multiple domains is effective to improve the overall performance. However, it is still challenging to improve generalization across domains under limited training data, and hard to deploy current solutions due to their computational complexity. In this paper, we propose a simple yet effective framework AdaSparse for multi-domain CTR prediction, which learns adaptively sparse structure for each domain, achieving better generalization across domains with lower computational cost. In AdaSparse, we introduce domain-aware neuron-level weighting factors to measure the importance of neurons, with that for each domain our model can prune redundant neurons to improve generalization. We further add flexible sparsity regularizations to control the sparsity ratio of learned structures. Offline and online experiments show that AdaSparse outperforms previous multi-domain CTR models significantly.

1 INTRODUCTION

In online advertising and recommendation systems, click-through rate (CTR) prediction is a fundamental technique. There is a great need to capture differentiated user preferences for *multiple domains*. On one hand, there are always more than one *scenario* (e.g., search, feeds and others) that require the ability of prediction models, however their data are usually not homologous. Fig. 1 (a) shows the CTR distribution of 10 business scenarios from our advertising platform, and we can see that the averaged CTR from large/small scenarios (e.g., B5/B2) are quite different. On the other hand, if we partition the data with *representative features* such as the position of displayed ads, from Fig. 1 (b) we can also observe that the subsets with different feature values have different natures.

As in Fig. 1 (c), recent studies pay more attention to multi-domain learning paradigm in CTR prediction, aiming to improve the overall performance on all domains. There are two crucial issues: (1) **Generalization** across domains. To achieve better effectiveness, the model need to possess the ability of capturing the user interest specific to each domain. Besides, advertisers often change delivery strategies (such as delivery scenarios, targeted audiences and ad positions), leading to emerging domains having insufficient training data. (2) **Computational cost**. Considering the online deployment procedure, the model should be parameter-efficient, which contributes to memory space and response time in online service.

Recent studies in multi-domain CTR prediction can be divided into two categories. (1) Models with **individual parameters** [5, 9], in which each domain has its separate learnable parameters. Yu et al. [9] regard each domain as a meta-task and learn task-specific parameters via meta-learning. Sheng et al. [5] decompose all parameters to shared and domain-specific part, where each domain's parameters are trained using the data from this domain only. They are usually inefficient on computation and memory, because the complexity grows much faster with the increase of domains. Besides, the generalization for domains having limited training data is not ideal. (2) Models with **generated parameters** [8]. An auxiliary network is used to generate domain-specific parameters, taking domain-aware features as input. Due to the large amount of parameters to be generated in these solutions, it makes the models hard to converge and does harm to generalization. These models are also faced with computational cost issue.

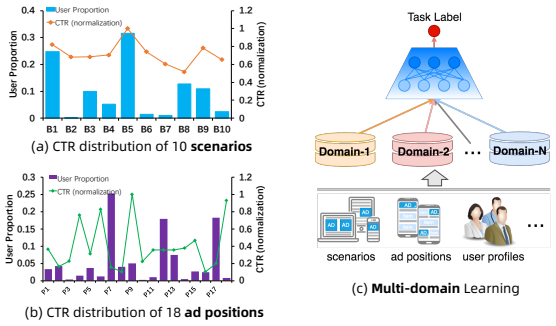


Figure 1: Two cases of domain differences are shown in (a) and (b). Multi-domain learning paradigm is shown in (c).

We propose a simple yet effective framework AdaSparse, which learns adaptively sparse structure for each domain to take both the generalization across domains and computational efficiency into account. We introduce domain-aware neuron-level weighting factors to measure the importance of neurons w.r.t. different domains. With that our model can prune redundant neurons to improve generalization, especially for domains with limited training data. We perform neuron-level rather than connection-level pruning to guarantee lower computation complexity, because the number of neurons is far less than connections. Our contributions are:

- To our knowledge, this is the first work that learns sparse structures to take both generalization and computational cost into account for multi-domain CTR prediction.
- We propose AdaSparse to adaptively prune redundant neurons w.r.t. different domains via learned neuron-level weighting factors to improve generalization, which also guarantees lower computation complexity.
- Both offline and online experiments verify that AdaSparse significantly outperforms previous models. We show that the learned sparse structures capture domain distinction.

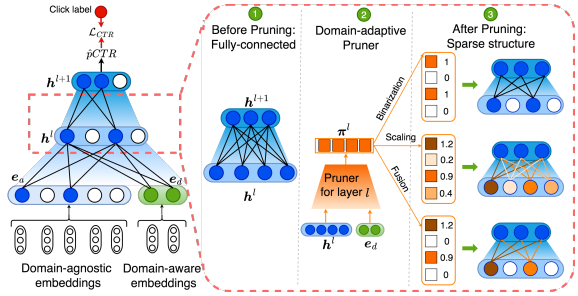


Figure 2: AdaSparse for multi-domain CTR prediction.

2 PROBLEM FORMULATION

Considering a training dataset $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^{|\mathcal{D}|}$, where x_j and y_j represent the feature set and binary click label of the j_{th} sample.

We partition the dataset $\mathcal{D}$ to multiple domain-specific subsets $\mathcal{D}^d$ (that is, $\mathcal{D} = \bigcup_d \mathcal{D}^d$), where domain d's subset $\mathcal{D}^d = \left\{\left(x_i^d, x_i^a, y_i\right)\right\}_{i=1}^{|\mathcal{D}^d|}$ is obtained based on *domain-aware feature set*: here, the whole feature set x_i is divided by domain-aware feature set x_i^d and domain-agnostic feature set x_i^a. Take Fig. 1 (b) as an example, the domain-aware feature set x_i^d contains one feature `ad_position`, resulting in 18 domains in $\mathcal{D}$ and the data sizes of these domains are usually quite different. We can also assign more features to construct the domain-aware feature set, such as {`scenario`, `user_profile`, `ad_position`}, and this may result in thousands of domains. The goal of multi-domain CTR prediction is to learn a model $\hat{pCTR} = f(x)$ that performs well on all domains.

3 PROPOSED APPROACH

We propose AdaSparse that learns adaptively sparse structure for each domain to achieve better generalization. Figure 2 givens an overview. The core is a domain-aware pruner that produces neuron-level weighting factors to prune redundant neurons.

Without loss of generality, we use an L-layered fully-connected neural network [1] as the backbone model architecture $f(\cdot)$ to introduce our AdaSparse. Our model operates on neuron-level, thus it can be easily extended to other common architectures in CTR prediction, such as deep&cross network [6].

After transforming all features into embeddings, we concatenate domain-aware / -agnostic feature embeddings as $\boldsymbol{e}_d$ / $\boldsymbol{e}_a$. The model input is the concatenation of $[\boldsymbol{e}_d, \boldsymbol{e}_a]$. Let $\mathbf{W}^l$ denote learnable matrix of the l_{th} layer. That is, $\boldsymbol{h}^{l+1} = \tanh\left(\mathbf{W}^l \boldsymbol{h}^l\right)$, where $\boldsymbol{h}^l$ is the input (neuron) of the l_{th} layer and we omit the bias for simplification. The model is trained through cross-entropy criterion $\mathcal{L}_{CTR}(y, \hat{pCTR})$ over all domains' samples:

$$\min_{\mathbf{W}} \frac{1}{|\mathcal{D}|} \sum_d \sum_{i=1}^{|\mathcal{D}^d|} \mathcal{L}_{CTR}\left(y_i, f\left(x_i^d, x_i^a\right); \{\mathbf{W}^l\}_{1 \le l \le L}\right). \quad (1)$$

For given domain d, AdaSparse employs the domain-aware pruner to adaptively remove each layer's neurons. Specifically, consider the l_{th} layer's matrix $\mathbf{W}^l \in \mathbb{R}^{N_l \times N_{l+1}}$ that connects from N_l neurons $\boldsymbol{h}^l$ to next layer's N_{l+1} neurons $\boldsymbol{h}^{l+1}$, the domain-aware pruner produces a weighting factor vector $\boldsymbol{\pi}^l(d) \in \mathbb{R}^{N_l}$ to prune the $\boldsymbol{h}^l$. It operates for each layer and finally we obtain a sparse structure.

3.1 Domain-adaptive Pruner

3.1.1 ***Lightweight Pruner.*** As in the right part of Fig. 2, the domain-aware pruner is a lightweight network that takes $\boldsymbol{h}^l$ and domain-aware features $\boldsymbol{e}_d$ as input.

For each layer l, it outputs the neuron-level weighting factors $\boldsymbol{\pi}^l(d)$, which is sparse and used to prune the neurons by

$$\boldsymbol{h}^l(d) \coloneqq \boldsymbol{h}^l \odot \boldsymbol{\pi}^l(d) \quad (2)$$

where $\odot$ is element-wise multiplication. We use sigmoid $\sigma(.)$ as activation function of the pruner, and employ a soft-threshold operator $S_\epsilon(.)$ to enforce that the outputs are less than a predefined value ϵ. Formally, the computation for the l_{th} layer of pruner is:

$$\boldsymbol{\pi}^l(d) = S_\epsilon\left(\sigma\left(\mathbf{W}_p^l \cdot [\boldsymbol{e}_d; \boldsymbol{h}^l]\right)\right), \quad (3)$$

where $\mathbf{W}_p^l$ denotes the learnable parameters of the l_{th} layer.

3.1.2 ***Formulations of Weighting Factors.*** We introduce three formulations of weighting factors $\boldsymbol{\pi}^l$, and detail the subtle difference about their sigmoid function and soft-threshold operator, inspired by model compression techniques [3, 4, 10].

1. **Binarization** method. It requires that all elements of $\boldsymbol{\pi}^l$ should be 0-1 binary value. It is challenging to achieve this goal automatically if we only use original sigmoid function $v_{out} = \sigma(v_{in})$, because the element v_{out} can only be approximate 0 or 1 value when $|v_{in}|$ becomes magnitude. Or if we use an improper threshold (e.g. 0.5) to truncate the output value v_{out}, we may prune the neurons by accident when the model is still under-fitting during early training.

We incorporate a parameter α into the sigmoid function:

$$v_{out} = \sigma(\alpha \cdot v_{in}) \quad (4)$$

and gradually increase α during training. When α is large enough, v_{out} will converge to 0-1 value. Here, we formulate the soft-threshold operator $S_\epsilon(.)$ as:

$$S_\epsilon(v_{out}) = sign(|v_{out}| - \epsilon) \quad (5)$$

where $\epsilon > 0$ is a tiny threshold.[1] In binarization method, the prune operation $\boldsymbol{h}^l(d) \coloneqq \boldsymbol{h}^l \odot \boldsymbol{\pi}^l(d)$ can be implemented using sparse operators to accelerate the computation.[2]

2. **Scaling** method. We regard binarization method as a hard weighting, and here Scaling method is a soft one:

$$v_{out} = \beta \cdot \sigma(v_{in}), \quad \beta \ge 1 \quad (6)$$

which produces factor values ranging from 0 to β. It is expected that the more important neurons, the larger factor values they will have. We design the soft-threshold operator of Scaling method as

$$S_\epsilon(v_{out}) = v_{out} \cdot sign(|v_{out}|). \quad (7)$$

3. **Fusion** method. In Scaling method we give less important neurons very small factor values. To prune the redundant information as much as possible, we combine Binarization and Scaling methods by formulating:

$$\begin{aligned} v_{out} &= \beta \cdot \sigma(\alpha \cdot v_{in}), \quad \beta \ge 1 \\ S_\epsilon(v_{out}) &= v_{out} \cdot sign(|v_{out}| - \epsilon), \quad \epsilon > 0. \end{aligned} \quad (8)$$

[1] $sign(\cdot)$ is short for signum function, $sign(x) = \begin{cases} 1 & x > 0 \\ 0 & x \le 0 \end{cases}$

[2] Such as `tf.sparse.sparse_dense_matmul` in TensorFlow.

3.1.3 ***Discussion on Memory and Computational Costs.*** As we mentioned in § 1, there are two types of approaches for multi-domain CTR prediction, based on individual parameters and generated parameters respectively. AdaSparse belongs to the latter.

AdaSparse only incorporates an additional network as pruner, thus it is parameter-efficient and the memory cost is much smaller than individual parameters. In terms of computational cost, we compare it with individual-based STAR [5] and generated-based APG [8]. Let D denote the number of domains, and take the computation for l_{th} layer for comparison. The complexities of STAR, APG and AdaSparse are $O(D \cdot N_l \cdot N_{l+1})$, $O(D \cdot (N_l K + N_{l+1} K))$ and $O(D \cdot N_l)$ respectively, verifying that our approach is time-efficient.

3.2 Sparsity Controlled Regularization

In Binarization method, a core problem is that we need to control the weighting factors of redundant neurons gradually converge to zero during training. We propose a flexible sparsity regularization to control learning of factors $\boldsymbol{\pi}^l(d)$, and it also meets the various requirements of sparsity ratio for learned structures.

In Binarization method, $\boldsymbol{\pi}^l$ is a 0-1 binary vector, and an usual sparsity regularization way is adding ℓ_1-norm term $||\boldsymbol{\pi}^l||_1$. However, we empirically find that it is hard to achieve our expected sparsity ratio without explicit controlling.

Considering that $\frac{||\boldsymbol{\pi}^l||_1}{N_l}$ represents the percentage of 1 code in $\boldsymbol{\pi}^l$, we denote the sparsity ratio as $r^l = 1 - \frac{||\boldsymbol{\pi}^l||_1}{N_l}$. We then predefine *sparsity ratio boundary* $[r_{min}, r_{max}]$ as our expected goal, and explicit control the actual sparsity ratio during training by adding an sparsity regularization loss term:

$$R_s\left(\{\boldsymbol{\pi}^l\}_{1\le l\le L}\right) = \frac{1}{L}\sum_{l=1}^{L}\lambda^l||r^l - r||_2, \quad r = (r_{min} + r_{max})/2, \tag{9}$$

where λ^l is an dynamic balancing weight based on the current actual sparsity rate r^l in training procedure:

$$\lambda^l = \begin{cases} 0 & r^l \in [r_{min}, r_{max}] \\ \hat{\lambda} \cdot |r^l - r| & r^l \notin [r_{min}, r_{max}] \end{cases}, \tag{10}$$

where $\hat{\lambda}$ is initialized by 0.01 and gradually increase over training step, thus the model pays less attention on sparsity loss term during early training and focuses more on adjusting $\boldsymbol{\pi}^l$. We can see that when the current sparsity ratio r^l falls into our expected boundary $[r_{min}, r_{max}]$, $\lambda^l = 0$ and this loss term will not affect the learning.

4 EXPERIMENTS

4.1 Experimental Setup

4.1.1 ***Datasets.*** We conduct experiments on two datasets. One is a million-scale public dataset **IAAC** [8], containing 10.9 million impressions and *300+ domains*. Another is a billion-scale production dataset collected from our advertising system, named **Production**, containing 2.2 billion impressions and *5,000+ domains*.[3] We split each into training/dev/test sets by timestamp with 4:1:1 proportion.

[3]Selected domain-aware feature set: {user_gender_id, user_age_level, user_star_level} for public dataset, and {scenario, ad_position, user_profile} for production dataset.

Table 1: Results on multi-domain CTR prediction. "*" denotes that AdaSparse significantly outperforms the second-best approach at the level of $p < 0.05$.

Approach	Production Dataset			Public Dataset		
	LogLoss	AUC	GAUC	LogLoss	AUC	GAUC
DNN	0.063630	0.7266	0.6669	0.094886	0.6506	0.6442
+MAML	0.063474	0.7308	0.6715	0.093912	0.6531	0.6463
+STAR	0.063445	0.7313	0.6762	0.093598	0.6541	0.6505
+APG	0.063232	0.7332	0.6826	0.093264	0.6585	0.6553
+AdaSparse	**0.063215**	**0.7359***	**0.6848***	**0.093139**	**0.6607***	**0.6572***
DCNv2	0.063602	0.7282	0.6684	0.095097	0.6497	0.6443
+MAML	0.063619	0.7326	0.6730	0.094024	0.6536	0.6474
+STAR	0.063424	0.7325	0.6794	0.093754	0.6538	0.6501
+APG	0.063192	0.7356	0.6840	0.093375	0.6572	0.6536
+AdaSparse	**0.063187**	**0.7378***	**0.6871***	**0.093292**	**0.6594***	**0.6565***

4.1.2 ***Competitors.*** We use DNN [1] and DCNv2 [6] as two backbones of prediction models. On this basis, we compare AdaSparse with the previous state-of-the-art multi-domain CTR prediction approaches: (1) MAML [9] treats each domain as a meta-task and quickly learns domain-specific parameters from a small amount of domain data. (2) STAR [5] learns predefined domain-specific parameters using the data from this domain only. (3) APG [8] generates domain-specific parameters using additional networks, and uses low-rank decomposition to reduce computation cost.[4]

The metrics of LogLoss, AUC and Group AUC (GAUC [2] for short) of domains are used for evaluation. For fair comparison, all approaches have the same feature embedding size, and the main network contains three layers. The output sizes of hidden layers are set to {512,256,128} (Production) and {128,64,32} (Public). We use Adam optimizer with 1024 (Production) and 256 (Public) batch size and 0.001 learning rate. For the pruner, a fully-connected layer is used to produce weight factors of each layer l. For each comparative model we carefully tune their hyperparameters. The hyper-parameters in AdaSparse are set as the following: $\beta = 2$, $\epsilon = 0.25$, α is initialized to 0.1 and its upper limit is set to 5, $r_{min} = 0.15, r_{max} = 0.25$.

4.2 Results and Discussion

4.2.1 ***Main Results.*** Table 1 shows the results of multi-domain CTR predictive approaches, with DNN and DCNv2 as backbone respectively. On both datasets, all multi-domain methods achieve better performance compared to DNN or DCNv2, demonstrating that capturing the differentiated user preferences for multiple domains is the key for CTR prediction. Compared to DNN, DCNv2 performs better on Production dataset but worse on Public dataset. We empirically explain that because the size of Public dataset is too small, it is hard to train DCNv2 (with more parameters) adequately.

APG and AdaSparse outperform MAML and STAR, verifying that approaches based on generated parameters have advantages over those based on domain individual parameters. Benefiting from parameter generating and pruning redundant neurons, AdaSparse learns domain-adaptive sparse structures, achieving a significant improvement compared to all competitors.

[4]Note that for the approaches based on individual parameters, they cannot be used for thousands of domains due to huge memory cost. Therefore, we retain a most representative feature in the domain-aware feature set.

Table 2: Comparison of different weighting factors.

Methods	AUC(↑)	GAUC(↑)
DNN	0.7266	0.6669
+ AdaSparse (Binarization)	0.7355	0.6843
+ AdaSparse (Scaling)	0.7334	0.6817
+ AdaSparse (Fusion)	**0.7359**	**0.6848**

4.2.2 ***Comparison among Three Formulations of Factors.*** Table 2 shows the performance of three weighting factors based on DNN backbone. The effectiveness of Binarization method verifies that pruning the redundant information specific to each domain can improve the overall generalization. Scaling method considers the importance of neurons and also boosts the performance. Fusion method that combines Binarization and Scaling achieves the best performance. This shows that taking both redundance and importance of neurons into account is indeed an effective strategy for improving multi-domain CTR prediction.

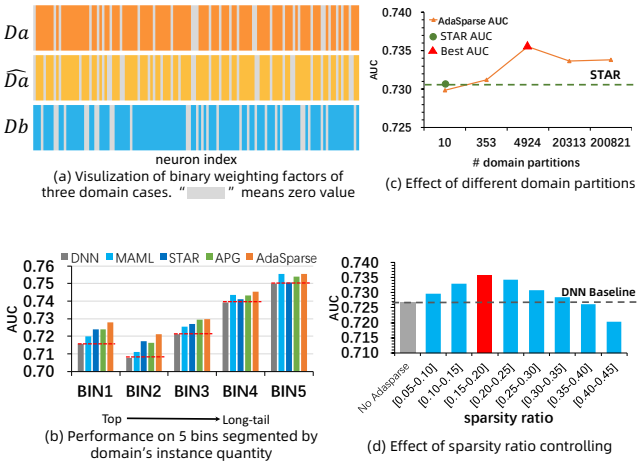
4.3 Further Analysis

4.3.1 ***Generalization across Domains.*** To explicitly understand AdaSparse's ability of capturing the characteristics of domains, we select three domains' subset: denoted as Da, $\widehat{Da}$ and Db, where the data distributions of Da and $\widehat{Da}$ are similar, while Db is quite different from them. Fig. 3 (a) visualizes the learned binary weighting factors for the 128-dimension layer. The gray areas represent 0 value and others mean 1 value. As we can see, Da and $\widehat{Da}$ have a great proportion of overlaps, while Db only has a small proportion. This shows that AdaSparse can capture both the commonalities and characteristics among domains.

To verify the performance boosts from top to long-tail domains, we rank all 5,000+ domains by their sample sizes, and then merge them into 5 bins based on equal-frequency segmentation. Let *BIN1 / BIN5* denote the top/long-tailed domains respectively, which contain *18 / 3000+ domains* with both 20% of samples. As shown in Fig. 3 (b), all approaches improve significant performance on top domains (BIN1, 2 and 3). This verifies that rich training data is helpful for learning domain-specific parameters. However if we consider long-tail domains (BIN4 and 5), only AdaSparse and MAML can still achieve improvement by a large margin. Benefiting from pruning domain-aware redundancy, AdaSparse achieves better performance long-tail domains having limited training data, and has the best generalization across all domains.

4.3.2 ***Effect of Different Domain Partitions.*** We change the domain-aware feature set to analyze the effect of domain partitions. We first use the feature `scenario` to partition dataset into 10 domains. Then, we gradually add new features [5] to partition dataset into more domains. As in Fig. 3 (c), STAR has a slight advantage over AdaSparse under a small number of domains. However, as the number of domains increases, AdaSparse achieves significant AUC boosts. This indicates that fine-grained domain partitions can help to improve the effectiveness. Besides, the results also show that too large amount of partitions is harmful to the effectiveness.

[5] adding { `scenario`, `ad_position`, `user_profile`, `user_behaviors`} sequentially.



(a) Visulization of binary weighting factors of three domain cases. " " means zero value

(c) Effect of different domain partitions

(b) Performance on 5 bins segmented by domain's instance quantity

(d) Effect of sparsity ratio controlling

Figure 3: (a) visualizes the binary weighting factors from two similar domains and one discrepant domain. (b) shows the different performance boosts from top to long-tail domains. (c) verifies that fine-grained domain division improves generalization. (d) shows the effect of sparsity ratio controlling.

4.3.3 ***Sensitiveness of Sparsity Controlling for Pruning.*** We change the hyperparameters *sparsity ratio boundary* $[r_{min}, r_{max}]$ for evaluation. Here we empirically consider the sparsity ratio up to 0.5 in a model is not necessary. Therefore, we test several groups of sparsity ratio totally under 0.5. As in Fig. 3 (d), the performance first increases and then decreases along with the growing of sparsity ratio. This verifies that pruning redundant information can boost the effectiveness. We further notice that even if we use $[0.35, 0.40]$ as the sparsity ratio boundary, the resulting model still achieves almost the same performance as the base DNN model. This indicates that the model structure is usually over-parameterized and have a great proportion of redundant parameters.

4.4 Online A/B Test

We conduct online experiments in our advertising system's CTR module for 10 days. We use metrics including $CTR = \frac{\#\text{click}}{\#\text{impression}}$, $CPC = \frac{\text{cost of advertisers}}{\#\text{click}}$, and define a new metric named uplifted domains ratio ($UPR = \frac{\#\text{ Uplifted domains}}{\#\text{Total domains}}$) to show the proportion of domains having performance boost. AdaSparse brings **4.63**% gain on CTR and **3.82**% reduction on CPC, and improves the CTR on 90%+ of domains, verifying its domain-effectiveness in industry.

5 CONCLUSION

We propose AdaSparse to learn adaptively sparse structures for multi-domain CTR prediction. It prunes redundant neurons w.r.t different domains via learned neuron-level weighting factors to improve generalization. Both offline and online experiments verify that AdaSparse outperforms previous models [5, 8, 9]. In future work we will combine pruning and neural architecture search techniques [7] to further improve generalization across domains.

REFERENCES

[1] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[2] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*. 507–517.

[3] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*. 2736–2744.

[4] Jian-Hao Luo and Jianxin Wu. 2020. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *Pattern Recognition* 107 (2020), 107461.

[5] Xiang-Rong Sheng, Liqin Zhao, Guorui Zhou, Xinyao Ding, Binding Dai, Qiang Luo, Siran Yang, Jingshan Lv, Chi Zhang, Hongbo Deng, et al. 2021. One Model to Serve All: Star Topology Adaptive Recommender for Multi-Domain CTR Prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 4104–4113.

[6] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. 2021. DCN V2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the Web Conference 2021*. 1785–1797.

[7] Penghui Wei, Weimin Zhang, Zixuan Xu, Shaoguo Liu, Kuang-chih Lee, and Bo Zheng. 2021. AutoHERI: Automated Hierarchical Representation Integration for Post-Click Conversion Rate Estimation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 3528–3532.

[8] Bencheng Yan, Pengjie Wang, Kai Zhang, Feng Li, Jian Xu, and Bo Zheng. 2022. APG: Adaptive Parameter Generation Network for Click-Through Rate Prediction. *arXiv preprint arXiv:2203.16218* (2022).

[9] Runsheng Yu, Yu Gong, Xu He, Bo An, Yu Zhu, Qingwen Liu, and Wenwu Ou. 2020. Personalized adaptive meta learning for cold-start user preference prediction. *arXiv preprint arXiv:2012.11842* (2020).

[10] Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878* (2017).