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 computational complexity. In this paper, we propose AdaSparse for multi-domain CTR prediction, which learns adaptively sparse structure for each domain, achieving better generalization across domains with lower computational cost. 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.

CCS CONCEPTS
• Information systems → Probabilistic retrieval models.

KEYWORDS
Multi-domain CTR prediction; Adaptively Sparse structures

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.

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

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 multiple scenarios (e.g., search, feeds and others) that require the ability of prediction models, yet their data are usually not homologous. As in Fig. 1 (a), the CTR distribution of 10 business scenarios reveals that the statistical CTR from large/small scenarios (e.g., B5/B2) is quite different. On the other hand, if we partition the data with representative features such as ad positions, we can observe a similar nature that the CTR also varies significantly by feature values (shown in Fig. 1 (b)).

As in Fig. 1 (c), recent studies pay more attention to multi-domain learning paradigm in CTR prediction, aiming to improve the overall performance. 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, contributing 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] regards each domain as a meta-task and learn task-specific
parameters via meta-learning. Sheng et al. [5] decomposes all parameters into shared part and domain-specific part, where each domain’s parameters are trained using the data from this domain only. They tend to be inefficient on computation and memory, due to the exponential growth in complexity with more domains. Besides, for domains with limited data, the generalization is subpar. (2) Models with generated parameters [8]. An auxiliary network is used to dynamically generate domain-specific parameters, taking domain-aware features as input. Due to the large amount of parameters to be generated in current solutions, it is hard for the models to converge, which hurts generalization. These models are also faced with computational cost issue.

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.

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 \( \mathcal{D}_d = \left\{ \left( x^d_j, y^d_j \right) \right\}_{j=1}^{|\mathcal{D}_d|} \) is obtained based on domain-aware feature set: here, the whole feature set \( x^d_j \) is divided by domain-aware feature set \( x^d_a \) and domain-agnostic feature set \( x^d_i \). Take Fig. 1 (b) as an example, the domain-aware feature set \( x^d_a \) contains one feature ad_position, resulting in 18 domains in \( \mathcal{D} \). We can also assign more features to construct the domain-aware feature set, such as \( \{\text{scenario, user_profile, ad_position}\} \), and this may result in thousands of domains. The goal of multi-domain CTR prediction is to learn a model \( \hat{p}_{\text{CTR}} = f(x) \) that performs well on all domains.

3 PROPOSED APPROACH

Figure 2 gives an overview of our proposed AdaSparse. 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 approach operates on neuron-level, thus it can be easily extended to other architectures in CTR prediction, such as deep&cross network [6].

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

\[
\min \frac{1}{|\mathcal{D}|} \sum_d \sum_{i=1}^{|\mathcal{D}_d|} \mathcal{L}_{\text{CTR}}(y_i, f(x^d_i, x^a_i); \{W^d\}^{1 \leq L}) \quad . \tag{1}
\]

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 \( W^d \in \mathbb{R}^{N_l \times N_{l+1}} \) that connects from \( N_l \) neurons \( h^l \) to next layer’s \( N_{l+1} \) neurons \( h^{l+1} \), the domain-aware pruner produces a weighting factor vector \( \pi^d(l) \in \mathbb{R}^{N_l} \) to prune the \( h^l \). It operates for each layer and finally we obtain a sparse structure.

3.1 Domain-adaptive Pruner

3.1.1 Lightweight Pruner. As in Fig. 2, the domain-aware pruner is a lightweight network that takes \( h^l \) and domain-aware features \( e_d \) as input. For layer \( l \), it outputs the neuron-level weighting factors \( \pi^l(d) \), which is sparse and used to prune the neurons by

\[
H^l(d) := h^l \odot \pi^l(d) \quad . \tag{2}
\]

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

\[
\pi^l(d) = S_\varepsilon \left( \sigma \left( W^d_p \cdot [e_d; h^l] \right) \right) , \quad \tag{3}
\]

where \( W^d_p \) denotes the learnable parameters of the \( l_{th} \) layer.

3.1.2 Formulations of Weighting Factors. Inspired by pruning techniques [3, 4, 10], we introduce three formulations of weighting factors \( \pi^l \), and detail their subtle difference on \( \sigma(\cdot) \) and \( S_\varepsilon(\cdot) \).

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

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

and gradually increase $\alpha$ during training. When $\alpha$ is large enough, $v_{out}$ will converge to 0-1 value. Here, we formulate $S_e(\cdot)$ as:

$$S_e(v_{out}) = \text{sign}(|v_{out}| - \epsilon)$$

where $\epsilon > 0$ is a tiny threshold.\(^1\) We can use sparse operators\(^2\) to accelerate the computation on operation $h^l(d) := h^l \odot \pi^l(d)$.

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

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

which produces factor values ranging from 0 to $\beta$. 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:

$$S_e(v_{out}) = v_{out} \cdot \text{sign}(|v_{out}| - \epsilon), \quad \epsilon > 0.$$ \hspace{1cm} (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:

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

$$S_e(v_{out}) = v_{out} \cdot \text{sign}(|v_{out}| - \epsilon), \quad \epsilon > 0.$$ \hspace{1cm} (8)

3.1.3 **Discussion on Memory and Computational Costs.** AdaSparse, based on generated parameters (mentioned in §1), only incorporates an additional network as pruner, thus it is parameter-efficient and the memory cost is much smaller than models with 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 present a flexible sparsity regularization to control learning of factors $\pi^l(d)$ that also meets the various sparsity ratio requirements for learned structures.

An usual sparsity regularization way is adding $l_1$-norm term $||\pi^l||_1$. However, we empirically find that it is hard to achieve our expected sparsity ratio without explicit controlling.

We denote the sparsity ratio as $r^l = 1 - ||\pi^l||_1 / N$, where $||\pi^l||_1 / N$ represents the proportion of 1 code in $\pi^l$. We predefined sparsity ratio boundary $[r_{min}^l, r_{max}^l]$ as our expected goal, and add a sparsity regularization loss term to explicitly control the sparsity ratio:

$$R_{\lambda} \left( \{\pi^l\}_{1 \leq l \leq L} \right) = \frac{1}{L} \sum_{l=1}^{L} \lambda^l \cdot |r^l - r|^2, \quad r = (r_{min}^l + r_{max}^l)/2.$$ \hspace{1cm} (9)

where $\lambda^l$ is a dynamic balancing weight based on the current actual sparsity ratio $r^l$ in training procedure:

$$\lambda^l = \begin{cases} 0, & r^l \in [r_{min}^l, r_{max}^l] \setminus \left[ \lambda \cdot \left| r^l - r \right| : r^l \notin \left[ r_{min}^l, r_{max}^l \right] \right] \end{cases}.$$ \hspace{1cm} (10)

where $\lambda$ 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 $\pi^l$. We can see that when the current sparsity ratio $r^l$ falls into our expected boundary $[r_{min}^l, r_{max}^l]$, $\lambda^l = 0$ 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 called IAAC \([8]\) that has 10.9 million impressions and more than 300 domains. Another is a billion-scale production dataset from our advertising system called Production that has 2.2 billion impressions and about 5,000 domains.\(^3\) We split each into training/dev/test sets by timestamp with 4:1:1 proportion.

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 following strong baselines: (1) MAML \([9]\) treats each domain as a meta-task and quickly learns domain-specific parameters using little data. (2) STAR \([5]\) learns predefined domain-specific parameters using only data from this domain.\(^4\) (3) APG \([8]\) generates domain-specific parameters using additional networks, and reduce computation cost via low-rank decomposition.

The metrics of LogLoss, AUC and Group AUC (GAUC \([2]\) for short) of domains are used for evaluation. For fair comparison, the main network for each method is a three-layered MLP and the hidden sizes 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. The hyper-parameters in AdaSparse are set as the following: $\beta = 2, \epsilon = 0.25, \alpha$ 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 AdaSparse and other baseline methods for CTR prediction on two datasets. 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. DCNv2 outperforms DNN on Production but not on IAAC. We empirically explain that because IAAC is too small, training DCNv2 (with more parameters) adequately is difficult.

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.

---

\(^1\)\text{sign}() is short for signum function, \text{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}

\(^2\)Such as tf.sparse.sparse_dense_matmul in TensorFlow.

\(^3\)\text{user\_gender\_id, user\_age\_level, user\_star\_level} (IAAC) and \{\text{scenario, ad\_position, user\_profile}\} (Production) selected as domain-aware feature set.

\(^4\)Individual parameters based approaches are too memory-intensive for thousands of domains. So, we only use scenario as the domain-aware feature.
we rank all domains by their sample sizes, and merge them into 5 domains. As a result of pruning domain-aware redundancy, we can see, D\textsubscript{a} has the best generalization across all domains. Table 2 shows the performance of three weighting factors based on DNN backbone. The effectiveness of Binarization proves that pruning domain-specific redundant information improves overall generalization. Scaling method considers the importance of neurons and also boosts the performance. Fusion method which combines Binarization and Scaling achieves the best performance. This shows that considering both redundancy and importance of neurons is indeed beneficial for enhancing multi-domain CTR prediction.

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 proves that pruning domain-specific redundant information improves overall generalization. Scaling method considers the importance of neurons and also boosts the performance. Fusion method which combines Binarization and Scaling achieves the best performance. This shows that considering both redundancy and importance of neurons is indeed beneficial for enhancing multi-domain CTR prediction.

4.3 Further Analysis

4.3.1 Generalization across Domains. To explicitly show AdaSparse’s ability that can capture domain characteristics, we select three domain subsets: D\textsubscript{a}, D\textsubscript{b} and D\textsubscript{c}, where the data distributions of D\textsubscript{a} and D\textsubscript{b} are similar, while D\textsubscript{c} is quite different. Fig. 3 (a) visualizes the learned binary weighting factors for the last 128-dimension layer. The gray areas represent 0 value and others mean 1 value. As we can see, D\textsubscript{a} and D\textsubscript{b} have a great proportion of overlaps, while D\textsubscript{c} only has a small proportion. This shows that AdaSparse can capture both the commonalities and characteristics among domains.

To verify the performance gains from top to long-tail domains, we rank all domains by their sample sizes, and merge them into 5 bins using equal-frequency segmentation. Each BIN makes up 20% of total samples and BIN1 / BIN5 with 18 / 3000+ domains denotes the top / long-tailed domains, respectively. As seen in Fig. 3 (b), all methods improve significantly on top domains (BIN1, 2 and 3), indicating that learning domain-specific parameters benefits from rich training data. However, when it comes to long-tail domains (BIN4 and 5), only AdaSparse and MAML can outperform by a large margin. As a result of pruning domain-aware redundancy, AdaSparse achieves better performance on long-tail domains having limited data, and has the best generalization across all domains.

4.3.2 Effect of Different Domain Partitions. To analyze the impact of domain partitions, we modify the domain-aware feature set. We first partition the dataset into 10 domains using the feature index D\textsubscript{a} and D\textsubscript{b}.

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 | 0.6837 |
| + AdaSparse (Fusion)    | 0.7359 | 0.6848 |

Figure 3: Further analysis for AdaSparse. (a) visualizes the binary weighting factors on three domains. (b), (c) and (d) show the performance evolution w.r.t domain sizes, domain partitions and sparsity ratios.

To analyze the impact of domain partitions, we modify the domain-aware feature set. We first partition the dataset into 10 domains using the feature index D\textsubscript{a} and D\textsubscript{b}.

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.

\[ \text{CTR} = \frac{\# \text{click}}{\# \text{impression}}, \quad \text{CPC} = \frac{\text{cost of advertiser}}{\# \text{click}}, \quad \text{UPR} = \frac{\# \text{Uplifted domains}}{\# \text{Total domains}} \]
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, Zhuqiang Shen, Gao Huang, Shunmeng 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. DCNV2: 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).