Automatic Multi-Task Learning Framework with Neural Architecture Search for Recommender Systems

Anonymous Author(s)

ABSTRACT

Recommender systems usually need to simultaneously predict multiple objectives. Existing multi-task learning methods in recommender systems mainly follow the multi-gate mixture-of-experts (MMoE) framework. However, these models are usually designed for specific scenarios and cannot be generalized to other scenarios. And MMoE-based models also suffer from negative transfer. In this paper, we propose an automatic multi-task learning framework named AutoMTL, which utilizes neural architecture search to design superior architectures for the multi-task recommendation in a data-specific manner. We tailor the search space for recommendation tasks, and the overall framework mainly contains three modules: the feature interaction module for finding effective feature interaction operations, the heterogeneous expert module for generating diverse expert sub-networks, and the expert selection module for selecting appropriate experts for each task. Then, we utilize advanced differentiable architecture search algorithm to explore the search space and propose a novel search strategy with an auxiliary loss to stabilize the search process. Furthermore, we propose a novel operation-level early stopping method to solve the aggregation of skip connections during the architecture search. Extensive experiments demonstrate that AutoMTL can consistently outperform state-of-the-art human-crafted multi-task recommendation models.

CCS CONCEPTS

• **Computing methodologies** → **Multi-task learning**; • **Information systems** → **Retrieval models and ranking**.

KEYWORDS

Multi-task learning, Recommender systems, Neural architecture search

ACM Reference Format:

Anonymous Author(s). 2018. Automatic Multi-Task Learning Framework with Neural Architecture Search for Recommender Systems. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In recent years, recommender systems that aim to recommend favorable items for users have been playing an increasingly important role in people's lives. Deep learning models have been widely used in industrial recommender systems and attracted more and more research efforts [5, 10, 14]. Typically, recommender system needs to predict multiple objectives at the same time. For example, the e-commerce platform may want to simultaneously predict users' multiple behaviors (e.g., click, favorite, and purchase) to enhance the user experience as well as boost revenue. As a result, multi-task learning (MTL) in the recommender system has attracted ever-increasing interest from industry and academia and becomes a hot research topic. Through MTL, recommendation models can not only improve the performance of each task but also reduce the total number of parameters by parameter sharing across multiple tasks.

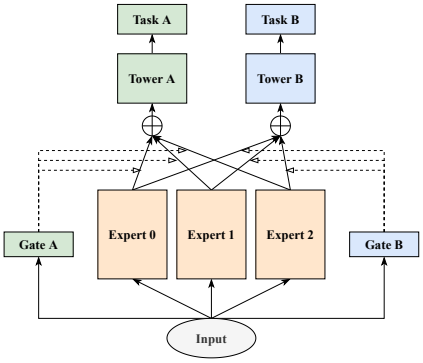


Figure 1: Multi-gate Mixture-of-Experts (MMoE) framework.

Multi-task learning aims to make full use of the knowledge contained in multiple tasks to improve overall performance. Recently, deep neural networks (DNNs) based MTL approaches mainly rely on flexible parameter sharing to transfer knowledge across tasks. The *Multi-gate Mixture-of-Experts (MMoE)* [19], one of the most advanced MTL models, can also be viewed as a paradigm of mutli-task learning. As shown in Figure 1, the MMoE framework shares several expert sub-networks across all tasks and then aggregates the representations from all experts for each task through a gating network. Formally, it can be formalized as:

$$y_k = h^k(f^k(x)),$$

$$\text{where} \quad f^k(x) = \sum_{i=1}^{n} g^k(x)_i f_i(x),$$

$$g^k(x) = \text{softmax}(W_g^k x),$$

where $x \in \mathbb{R}^d$ is the input instance. For a task k, y_k denotes the corresponding prediction, and h^k is the task-specific network (i.e., "tower" in Figure 1) of k. f^k denotes the representation aggregated by the gating network, and $g^k \in \mathbb{R}^n$ is the gating network. n is the number of experts, and f_i denotes the sub-network of expert i. $W_g^k \in \mathbb{R}^{n \times d}$ is the trainable weight matrix in the gating network

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

g^k. The gating network here can also be viewed as an application of the attention mechanism.

Due to the effectiveness of MMoE, many multi-task recommendation models follow the MMoE paradigm to design their model architectures [9, 23, 37, 38]. Specifically, they usually design architectures for expert sub-networks according to application scenarios or tasks. And there are also several other MTL methods that focus on exploring more flexible parameter-sharing patterns on the bias of the MMoE framework [17, 31].

However, in practice, the MMoE framework still has the following limitations: The first is the *negative transfer*, the fundamental problem of MTL. Since multiple tasks may have complex correlations and even conflict with each other, sharing information with a less related task may hurt performance. The MMoE framework is inherently a full-sharing approach that forces all experts to contribute to all tasks [3]. As a result, MMoE-based models also suffer from the negative transfer problem. The second is the *architecture homogeneity* of experts, which makes them difficult to learn diverse representations from the input data. Diverse representations that can reflect different aspects of the data learned by experts are significantly important to the MMoE-based models [19]. However, it is impractical to design diverse architectures for different experts manually. The third is *inefficient model design*. Recommender systems always face a variety of scenarios and tasks. Designing effective multi-task recommendation models is a labor-intensive task, which requires extensive expert experience and domain knowledge.

Inspired by the recent success of neural architecture search (NAS), we propose to utilize NAS to design optimal MTL models based on the MMoE framework. There are also several challenges for applying NAS to multi-task recommendations: 1) How to alleviate the negative transfer problem in multi-task models should be considered during the architecture search process. 2) Designing a general and effective search space that covers both existing and unexplored MTL architectures is challenging. In particular, we should take the characteristics of recommendation tasks into consideration. 3) The MMoE-based models always contain multiple experts. Generating different architectures for different experts can cause a very large search space. We should design an efficient search strategy to handle such a huge search space.

In this paper, to address the above challenges, we propose a NAS-based multi-task learning framework called AutoMTL [1] for automatic architecture search in multi-task recommendations. The overall framework of AutoMTL follows the MMoE paradigm, but the problems inherent in the original MMoE framework can be addressed through our novel search space design and architecture search method. First, to alleviate the negative transfer problem, we design a novel *expert selection module* that exploits expert selection parameters to perform a hard selection of experts for each task. During the retraining phase, the selected experts will be adaptively aggregated for each task by an additional gating network. Second, we design an expressive end-to-end search space that contains an *expert module* to generate heterogeneous architectures for different experts. And we also introduce a *feature interaction module*, which can search for effective feature interaction operations for recommendation tasks. Third, we utilize an efficient differentiable NAS algorithm [16] to explore the search space and propose a novel search strategy with an auxiliary loss to stabilize the search process.

Furthermore, we find that the problem of skip connection aggregation in differentiable NAS also exists in AutoMTL, which leads to performance degenerate. In this work, we observe that the aggregation of skip connections is due to the overfitting of operations in the supernet. Based on the observation, we propose a novel operation-level early stopping method to restrict the number of skip connections in the searched architectures.

In summary, the main contributions are as follows:

- We propose the NAS-based MTL framework to search for optimal architectures for multi-task recommendations. AutoMTL is composed of a feature interaction module, a heterogeneous expert module, and an expert selection module. It can not only enhance representation diversity by generating heterogeneous expert architectures but also alleviate the negative transfer problem by employing task-dependent expert selection.
- We design a general and expressive search space, which can cover both existing and unexplored MTL architectures. Moreover, the search space is tailored for recommendation tasks and can be easily extended.
- We further utilize an efficient differentiable NAS algorithm to explore the huge search space and design a novel search strategy with an auxiliary loss to stabilize the search process. And we propose an operation-level early stopping method to alleviate the aggregation of skip connections.
- Extensive experiments on four public datasets demonstrate the effectiveness of AutoMTL, which can consistently outperform the state-of-the-art human-crafted multi-task recommendation models.

2 RELATED WORK

2.1 Multi-Task Learning

Multi-task learning (MTL) aims to make full use of knowledge contained in multiple tasks to improve overall performance. In fact, MTL can be viewed as a form of transfer learning [27], which transfers knowledge across multiple tasks and introduces inductive bias contained in these tasks. However, the multi-task learning model may not always outperform corresponding single-task models [13, 19]. Since multiple tasks may have complex correlations or even conflict with each other, sharing information with a less-related task may hurt performance, which is known as the negative transfer problem.

Deep multi-task learning models mainly rely on parameter sharing [27] across tasks. There are two typical paradigms of parameter sharing, i.e., hard parameter sharing and soft parameter sharing. Hard parameter sharing is mainly achieved by the Shared-Bottom model [2], which shares several bottom hidden layers across all tasks. It may suffer significant performance degeneration when tasks are less related. In contrast, soft parameter sharing can alleviate negative transfer through more flexible parameter-sharing patterns [7, 22, 28]. Mixture-of-Experts (MoE) model [29] first proposes to share multiple expert sub-networks at the bottom and then aggregates the outputs of experts through a gating network.

[1] Anonymous code is now available at https://github.com/automtl/AutoMTL.

Then MMoE [19] extends the MoE model by using different gating networks for different tasks. MMoE can also be regarded as an effective MTL framework, and many recent MTL models have followed the MMoE framework [31, 38]. However, the negative transfer still exists in these soft parameter-sharing methods due to the full-sharing pattern.

2.2 Multi-Task Learning for Recommender Systems

Typically, recommender systems need to simultaneously predict multiple objectives [39]. There are mainly two forms of multi-task recommendation models, i.e., *Expert-Bottom-based* model and *Probability-Transfer-based* model [35].

The *Expert-Bottom-based* model usually follows the MMoE framework. PLE [31] explicitly separates task-common and task-specific experts to alleviate conflicts caused by complex correlations among tasks. MoSE [25] and SAME [23] are proposed to model sequential user behaviors of the multi-task recommendation. The *Probability-Transfer-based* model utilizes dependency relationships between tasks to transfer probabilities in the output layers of different tasks. ESMM [20] and ESM^2 [34] use the sequential dependencies of user behaviors to boost the prediction of conversion rate (CVR). AITM [35] first models the sequential dependence among multi-step conversions and then learns what and how much information to transfer adaptively.

However, most of these methods are designed for specific scenarios and cannot be easily transferred to other multi-task recommendation scenarios, severely limiting their utility.

2.3 Neural Architecture Search

Neural architecture search (NAS) aims to automatically design optimal neural architectures for given tasks in a data-driven manner. There are basically three existing frameworks for NAS [8]: Evolution-based NAS [11], Reinforcement Learning (RL) based NAS [24, 40], and gradient-based NAS [1, 16, 36]. As a pioneering effort, [40] employs an RL approach that requires training thousands of candidate models to convergence. Considering the search efficiency, the weighting-sharing paradigm has been proposed to design a large model (i.e., a supernet) that connects smaller model components. Different candidate architectures can be generated by selecting a subset of components. ENAS [24] utilizes an RNN controller to sample candidate architectures from the supernet. DARTS [16], ProxylessNAS [1], and NASP [36] take a differentiable approach by relaxing hard selections to continuous weights and then applying bi-level optimization to jointly optimize the architecture parameters and supernet weights.

NAS approaches also have been used to find optimal parameter sharing patterns for MTL. SNR [18], Gumbel-matrix routing [21], and MTNAS [3] efficiently searches for suitable connections between sub-networks or sharing routing for the given MTL problem through differentiable NAS methods. However, these methods are not designed for recommendation tasks and search for parameter sharing routing rather than expressive architectures. Different from the above methods, AutoMTL mainly focuses on utilizing NAS to search for optimal architectures for given multi-task recommendation scenarios.

3 METHODOLOGY

In this section, we first state the problem, then present an overview of AutoMTL, and finally introduce the search space and search algorithm in detail.

3.1 Problem Statement

In this paper, we mainly focus on utilizing NAS to search for superior architectures for the multi-task recommendation. The input features of recommendation tasks usually contain category and numerical features of the user, item, and contextual information. Following a commonly used manner, we use embedding layers to handle these features. Multiple objectives $T_1, \cdots, T_K$ need to be predicted, such as "click", "favorite", "add-to-cart", and "purchase". For generality, we simply model the final objective as a weighted summation of all the tasks' objectives. Consequently, the objective function of multi-task recommendation can be defined as

$$\mathcal{L} = \omega_1 \mathcal{L}_1 + \cdots + \omega_K \mathcal{L}_K, \tag{1}$$

where $\mathcal{L}_i$ is the loss function of task i, ω_i is the weight of $\mathcal{L}_i$ and $\sum_{i=1}^{K} \omega_i = 1$, and $\mathcal{L}$ is the total loss.

Following the formulation of DARTS [16], the goal of AutoMTL can be formalized as a bi-level optimization problem:

$$\begin{aligned} \boldsymbol{\alpha}^* &= \arg\min_{\boldsymbol{\alpha} \in \mathcal{A}} \mathcal{L}_{val}(\boldsymbol{w}^*, \boldsymbol{\alpha}) \\ \text{s. t.} \quad \boldsymbol{w}^* &= \arg\min_{\boldsymbol{w}} \mathcal{L}_{train}(\boldsymbol{w}, \boldsymbol{\alpha}) \end{aligned}$$

where $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ denote losses on the training data and validation data, respectively; $\mathcal{A}$ denotes the architecture search space, $\boldsymbol{w}$ denotes network weights, and $\boldsymbol{\alpha}$ denotes the architecture parameters.

3.2 Overview of the Framework

We design our framework on the basis of MMoE. Figure 2 depicts the overall framework of AutoMTL, which mainly contains three modules: the feature interaction module, the heterogeneous expert module, and the expert selection module. The feature interaction module at the bottom is used to search for effective feature interaction operations for recommendation tasks. Experts in the heterogeneous expert module can obtain different architectures to model different aspects of the data. Then, the expert selection module aggregates representations from the below experts and explicitly learn the importance of each expert for each task. To explore the search space, we propose a differentiable architecture search algorithm with a novel search strategy and an auxiliary loss. In the search stage, the supernet is constructed according to the search space described above. During the retraining stage, gating networks will be introduced to adaptively aggregate representations of relative important experts selected by the expert selection module.

3.3 End-to-End Search Space Design

As Figure 2 illustrates, the overall search space of AutoMTL is end-to-end, which mainly contains three modules: the feature interaction module, the heterogeneous expert module, and the expert selection module. Moreover, the search space is general and can be easily modified and extended by replacing and integrating available operations. Following the weight-sharing paradigm, the search

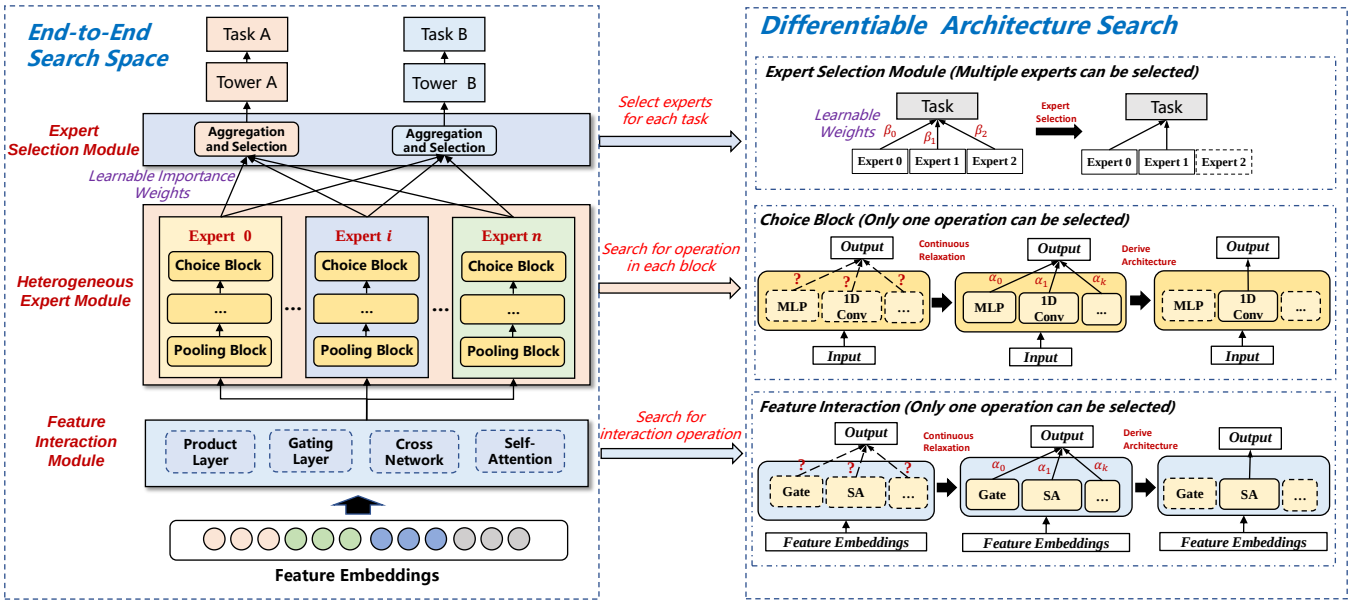


Figure 2: Overall Framework of AutoMTL. Each choice block contains all the available operations. During search, the output of each block is a weighted mixture of candidate operations' outputs. After search, operations with the highest strength will be selected. The expert selection module explicitly selects experts for each task via learned importance weights.

Table 1: Details of the choice blocks.

Operation Type	Operation	Number of Block Types	Hyperparameters of Each Block
Feature Interaction	Product Layer	2	*Product type:* **inner product, outer product**
	Gating Layer	2	*Gating type:* **Bit, Vector**
	Cross Network	4	*The number of cross layers:* **1, 2, 3, 4**
	Self-Attention (SA)	4	*The number of head:* **1, 2, 4, 8**
Pooling	Element-Wise	2	**Sum, Average**
	Attentive	1	/
	Linear Projection	1	/
Expert	Gating Layer	1	*Gating type:* **Hidden**
	Multi-layer Perceptron	7	*Hidden layer size:* **16, 32, 64, 128, 256, 512, 1024**
	1D Convolution	4	*Kernel size:* **1, 3, 5, 7**

space can be constructed as a supernet. The feature interaction module, located at the bottom, takes the output of the embedding layer as input. The embedding representations refined by the feature interaction module are shared by all experts. All the experts in the heterogeneous expert module share a common search space but can obtain heterogeneous architectures. The search space is composed of choice blocks. Each choice block contains all the available operations in the corresponding layer, and only one operation will be finally selected. To adaptively learn the architecture depth, we additionally add a "skip-connection" operation to each choice block. The expert selection module aggregates expert outputs for each task with learnable importance weights. Finally, the aggregated representations are fed to task-specific towers to get predictions. Next, we introduce the three modules in detail.

3.3.1 Feature Interaction Module. Previous works [10, 14, 26, 30, 32] have proven that explicit feature interactions are crucial for the performance of recommendation models, so we also introduce a feature interaction module to provide expressive feature representations for the upper experts. We consider commonly used feature interaction operations from recent deep recommendation models in this module, such as Product Layer [26], Gating Layer [12], CrossNet [33], and Self-Attention [30]. More information is available in Table 1. One operation corresponds to multiple block types with different hyperparameters, e.g., different numbers of heads in Self-Attention represent different block types. For simplicity, we only use one feature interaction block, i.e., the generated architectures contain one feature interaction layer. Note that this module can be easily extended by integrating more advanced feature interaction operations or multiple feature interaction blocks.

3.3.2 Heterogeneous Expert Module. All the experts in the heterogeneous expert module share a common search space but can learn different architectures to model different aspects of the data. In an

expert, the input features are embedding representations refined by the feature interaction module. The input features first go through a pooling block to integrate the embeddings, then are processed by several general blocks. The pooling block contains three types of pooling operations, i.e., Element-Wise, Attentive [15], and Linear Projection. Other blocks in the expert module need to be general enough for different types of tasks. We introduce three types of operations here, i.e., Hidden-level Gating Layer [12], Multi-layer Perceptron (MLP), and 1D Convolution. The Gating Layer and 1D Convolution usually contain much fewer parameters and computations then MLP. As a result, the search space of the expert can not only boost the diversity of experts but also helps to obtain compact architectures. The feature interaction module can also be contained in each expert to further promote performance, but it will significantly swell the search space and the complexity of the resulting architectures.

3.3.3 Expert Selection Module. The expert selection module aggregates expert outputs for each task with learnable importance weights. After the search stage, the expert selection module chooses a set number of experts for each task based on learned importance weights. The hard selection here can significantly alleviate the negative transfer. We regard the learnable importance weights as another type of architecture parameter named expert selection parameters. In the retrain stage, gating networks will be introduced to adaptively aggregate representations from the task-dependent experts selected by the expert selection module. In addition, we propose an auxiliary loss for the expert selection parameters to encourage the expert selection module to select diverse experts for different tasks. The formulation of the auxiliary loss is presented in Section 3.4.2.

3.4 Differentiable Neural Architecture Search

3.4.1 Continuous Relaxation and Search Strategy. To explore the search space, we use an efficient differentiable architecture search algorithm. As shown in Figure 2, for each choice block, the hard selection is relaxed to continuous weights by *Softmax*. The output of each choice block is the weighted mixture of candidate operations' outputs. Formally, let $O = \{o_1, \ldots, o_M\}$ be a collection of M candidate operations in a choice block B, the mixed output is formulated as follows:

$$B(x) = \sum_{o \in O} \frac{\exp(\alpha_o)}{\sum_{o' \in O} \exp(\alpha_{o'})} o(x),$$

where x is the input, and each operation o_i is associated with a coefficient α_{o_i} known as the architecture parameter.

For the expert selection module, we treat the learnable aggregation weights as another type of architecture parameter, named expert selection parameters. The aggregation weights reflect the significance of experts for tasks. Let $\boldsymbol{\beta}^{(k)} = \{\beta_1^{(k)}, \ldots, \beta_N^{(k)}\}$ represent the expert selection parameters for task k, where N denotes the number of experts. The aggregated representation for task k is as follows:

$$f^k = \sum_{i=1}^{N} \frac{\exp(\beta_i^{(k)})}{\sum_{\beta \in \boldsymbol{\beta}^{(k)}} \exp(\beta)} E_i(x)$$

where $E_i(x)$ denotes the output of the i-th expert. The aggregated representation is fed to the task-specific tower to obtain final predictions.

Since the architecture parameters and expert selection parameters are essentially different in purpose, training them simultaneously on the validation data leads to an unstable search process. We propose to train the expert selection parameters on the training data together with supernet weights since we expect them to reflect the importance of learned expert architectures for tasks. During search, we first fix the expert selection parameters for several epochs to make all the experts equally trained, regarded as a "warm-up stage."

3.4.2 Auxiliary Loss and Search Objective. In addition, to encourage diverse selections of experts for different tasks and help all the experts be adequately trained, we propose an auxiliary loss for the expert selection parameters. We hope the distributions of expert selection parameters for different tasks move away when they are close together, and the auxiliary loss has less impact on training when the distributions stay away. To meet the above requirements, we propose the following auxiliary loss:

$$\mathcal{L}_{log} = \sum_{i=1}^{K-1} \sum_{j=i+1}^{K} \log(\|\boldsymbol{p}^{(i)} - \boldsymbol{p}^{(j)}\| + 1), \tag{2}$$

$$\text{where} \quad \boldsymbol{p}^{(k)} = \{p_1^{(k)}, \ldots, p_N^{(k)}\}, \quad p_i^{(k)} = \frac{\exp(\beta_i^{(k)})}{\sum_{\beta \in \boldsymbol{\beta}^{(k)}} \exp(\beta)}.$$

When the distributions are close together, the gradient of the auxiliary loss will increase, and then the expert selection parameters can be pushed apart. To control the strength of this loss, we weight it with a coefficient w_{log}. During search, w_{log} gradually decays to zero, reducing its impact on training. In fact, the auxiliary loss can be regarded as a strong priori regularization, implying that different tasks may prefer diverse experts.

Let $\boldsymbol{\alpha}$ denote the architecture parameters and $\boldsymbol{\beta}$ denote the expert selection parameters. The search objective of AutoMTL can be formalized as:

$$\begin{aligned} \boldsymbol{\alpha}^* &= \arg\min_{\boldsymbol{\alpha}} \mathcal{L}_{val}(\boldsymbol{w}^*, \boldsymbol{\beta}^*, \boldsymbol{\alpha}) \\ \text{s. t.} \quad \boldsymbol{\beta}^*, \boldsymbol{w}^* &= \arg\min_{\{\boldsymbol{\beta}, \boldsymbol{w}\}} \mathcal{L}_{train}(\boldsymbol{w}, \boldsymbol{\beta}, \boldsymbol{\alpha}) + w_{log} \mathcal{L}_{log}, \end{aligned} \tag{3}$$

where $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ denote total losses in the training data and validation data, respectively, as mentioned in Section 3.1. The loss function of each single task can be defined as the commonly used Cross-Entropy loss or Mean-Square loss according to the task.

3.4.3 Operation-Level Early Stopping. In practice, we find that the problem of skip connection aggregation in differentiable NAS [4] still exists in AutoMTL. Considering the characteristics of the recommendation task in this work, we observe that the aggregation of skip connections is mainly due to that the parameters of operations in the supernet overfit the training data and gradually deviate from the validation data, while the architecture parameters are trained on the validation data, so the advantages of parametric operations are increasingly weakened and finally parametric-free operations dominate. Based on this observation, we propose an operation-level early stopping method to alleviate the issue.

Algorithm 1 Search algorithm of AutoMTL

Require: Training dataset and validation dataset, search space $\mathcal{A}$;
Ensure: Resulting searched architectures;
Initialize $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{w}$, w_{log}.
while not converged **do**
  Update architecture parameters $\boldsymbol{\alpha}$ via validation loss $\mathcal{L}_{val}$;
  Verify overfitting of operations through gradient directions.
  Update supernet weights $\boldsymbol{w}$ via training loss $\mathcal{L}_{train}$;
  Record the training gradient of each operation.
  if not in warm-up stage **then**
    Update expert selection parameters $\boldsymbol{\beta}$ via training loss $\mathcal{L}_{train}$ and auxiliary loss $\mathcal{L}_{log}$;
    Decay w_{log};
  end if
end while
Derive chosen operations base on $\boldsymbol{\alpha}$;
Select experts base on $\boldsymbol{\beta}$;
Introduce the gating networks for the selected experts.
return Final architecture.

Specifically, we monitor whether each operation in the supernet tends to overfit the training data and stops training the operation when it overfits. We utilize the difference between the gradient directions of the operation parameters on the training data and the validation data to verify whether the operation is prone to overfitting. If the difference keeps being large in multiple consecutive iterations, the operation will stop training. Taking advantage of the fact that the DARTS-like method trains alternately on the training data and validation data, the operation-level early stopping method can solve the problem of skip connection aggregation without introducing much additional computational overhead.

3.4.4 Search Algorithm. To solve the bi-level optimization problem, we jointly optimize the supernet parameters, architecture parameters, and expert selection parameters like DARTS [16]. We alternately train the supernet parameters and expert selection parameters on the training data and train the architecture parameters on the validation data. And we also use the first-order approximation during architecture parameter training for efficiency. At the beginning of search, we introduce a warm-up stage, which fixes the expert selection parameters for several epochs to stabilize the search process. After searching, operations with the highest strength are selected to form the final architecture. And the expert selection module selects a set number of experts for each task according to the expert selection parameters. Then, a gating network is introduced for each task to aggregate the selected experts, and the final architecture will be retrained to produce results. The detailed search algorithm is described in Algorithm 1.

4 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the effectiveness of AutoMTL. Specifically, we will answer the following research questions:

- **RQ1:** Can the performance of models searched by AutoMTL outperform state-of-the-art human-crafted ones on the benchmark datasets?
- **RQ2:** What are the impacts of design modules in AutoMTL?
- **RQ3:** How do the hyperparameters of AutoMTL influence its performance?
- **RQ4:** What insights can we draw from the searched architectures?

4.1 Experiment Setup

Table 2: Statistics of the datasets.

Dataset	#Samples	#Category Fields	#Numerical Fields	#Tasks
UserBehavior-2017	520, 757	3	0	4
IJCAI-2015	2, 143, 260	7	0	2
NL	17, 717, 195	16	63	2
ES	31, 669, 427	16	63	2
FR	27, 035, 601	16	63	2
US	27, 392, 613	16	63	2

4.1.1 Datasets. Experiments are conducted on the following four public benchmark datasets. The statistics of these datasets are summarized in Table 2. **IJCAI-2015**[2], a public dataset from the IJCAI-15 contest, contains millions of anonymized users' shopping logs from the past six months. We use two tasks: "favorite" and "purchase." **UserBehavior-2017**[3] is a public dataset of anonymized user behaviors from Alibaba. We use four tasks: "click," "buy," "cart," and "favorite." **AliExpress**[4] is a public dataset gathered from traffic logs on the AliExpress. This dataset is collected from five countries, and we use the data from the Netherlands (NL), Spain (ES), French (FR), and America (US) here. We use two tasks: "click" and "purchase."

For IJCAI-2015 and UserBehavior-2017, we randomly split the data with $7:1:2$ to generate the training dataset, the validation dataset, and the testing dataset. For AliExpress, we randomly take 20% of the training dataset to generate the validation dataset and use the testing dataset provided in the original dataset.

4.1.2 Evaluation Metrics. We treat each task as a binary classification problem, and the cross-entropy loss is used for it. Thus, we adopt two commonly used metrics to evaluate the performance of different methods:

- **AUC**. The Area Under the ROC Curve (AUC) measures the goodness of ranking order. A higher AUC indicates better performance.
- **MTL-Loss**: The total loss of all K tasks, i.e., the weighted summation of cross-entropy losses of all tasks. We simplely use equivalent weights for all tasks here.

It is worth noting that *an improvement of AUC at the* 0.001 *level is a significant improvement for tasks like CTR prediction*, which has also been pointed out in existing works [5, 10, 32].

4.1.3 Baselines. To our best knowledge, there is no other NAS-based MTL method that can be directly used for the multi-task recommendation. The work [3] utilizes NAS to search for sparse sharing routing for MLT, which is not open-source and hard to

[2]https://tianchi.aliyun.com/dataset/dataDetail?dataId=47
[3]https://tianchi.aliyun.com/dataset/dataDetail?dataId=649
[4]https://tianchi.aliyun.com/dataset/dataDetail?dataId=74690

Table 3: Overall performance comparison. The best results are marked in bold. * denotes statistically significant improvement (measured by t-test with p-value < 0.005) over the best baselines.

Method	UserBehavior-2017				IJCAI-2015		AliExpress-NL		AliExpress-ES		AliExpress-FR		AliExpress-US	
	AUC0	AUC1	AUC2	AUC3	AUC0	AUC1	AUC0	AUC1	AUC0	AUC1	AUC0	AUC1	AUC0	AUC1
SingleTask	0.7184	0.7220	0.7008	0.8425	0.7691	0.8509	0.7260	0.8543	0.7294	0.8870	0.7266	0.8707	0.7010	0.8666
Shared-Bottom	0.7232	0.7289	0.7017	0.8307	0.7700	0.8530	0.7263	0.8478	0.7275	0.8927	0.7219	0.8714	0.7029	0.8682
MMoE	0.7255	0.7276	0.7077	0.8393	0.7692	0.8526	0.7218	0.8565	0.7291	0.8879	0.7224	0.8782	0.7064	0.8803
CGC	0.7248	0.7281	0.7036	0.8418	0.7700	0.8527	0.7262	0.8577	0.7268	0.8881	0.7273	0.8853	0.7107	0.8700
PLE	0.7247	0.7300	0.7041	0.8407	0.7708	0.8516	0.7293	0.8553	0.7312	0.8927	0.7250	0.8784	0.7101	0.8801
AutoML-R	0.7255	0.7283	0.7041	0.844	0.7719	0.8528	0.7295	0.8620	0.7305	0.8926	0.7271	0.8849	0.7124	0.8849
AutoMLT	**0.7275***	**0.7323***	**0.7085***	**0.8465***	**0.7741***	**0.8571***	**0.7345***	**0.8640***	**0.7342***	**0.8963***	**0.7312***	**0.8879***	**0.7152***	**0.8880***

implemente. Therefore, we compare AutoMTL with four widely used human-crafted MTL models:

- **Shared-Bottom.** [2] Several bottom layers following the input layer are shared across all tasks. Each task has an individual "tower" of network on top of the bottom layers.
- **MMoE** [19] shares some expert sub-networks across all tasks and then aggregates expert representations via gating networks. Each task corresponds to a gating network.
- **CGC** [31] explicitly separates task-sharing and task-specific experts for alleviating negative transfer.
- **PLE** [31]introduces a progressive routing manner on the basis of CGC to achieve more efficient information sharing.

4.1.4 Implementation Details. All the baselines and AutoMTL are implemented in PyTorch. The category features are first transformed to dense vectors by embedding lookup tables, and the numerical features linearly project to the same dimension as embedding. Based on experience, the embedding sizes are set to 64 for *IJCAI-2015* and *UserBehavior-2017*, and 128 for *AliExpress*. To make all models have similar numbers of parameters, the number of experts is set to 8 for MMoE, and the numbers of task-shared experts and task-specific experts are all set to 4 for PLE and CGC. We use Adam optimizer for baseline models, and all other hyperparameters are carefully tuned on the validation set, where the early stopping strategy with the patience of 2 is applied. For AutoMTL, we also set the number of experts to 8. The resulting architectures also have similar or fewer numbers of parameters than baselines. The SGD optimizer is used to optimize the supernet weights with an initial learning rate of 0.01 and cosine learning rate decay. The architecture parameters and expert selection parameters are optimized with Adam optimizer with the learning rate of 0.001. The initial weight of auxiliary loss is set to 0.1 and decays linearly to 0 overtime. For the operation-level early stopping, we set the overfitting threshold to 0.5 (an operation with a difference of gradient directions exceeding the threshold is considered to tend to overfit). The patience of iterations (an operation tending to overfit for multiple consecutive iterations will stop training) is tuned for each dataset. All experiments are executed five times with different random seeds, and the average results are reported.

4.2 Overall Comparison (RQ1)

To demonstrate the effectiveness of the proposed AutoMTL, we compare it with state-of-the-art human-crafted models and the single task model. The results are shown in Table 3, where we report the AUC scores for each dataset. *AutoMTL-R* represents directly perform random search on our search space. From the comparison, we can observe that: 1) For baseline models, no model can consistently outperform others on all datasets or tasks, which demonstrates that improving the performance of MTL for all tasks is non-trivial due to complex correlations among tasks. However, AutoMTL can achieve the best performance on all datasets and tasks, even in the four-task scenario as *UserBehavior-2017*. 2) State-of-the-art human-crafted MTL models can not always outperform the single task model. It demonstrates that the negative transfer can affect the performance of MTL models. 3) In most cases, more advanced MTL models exhibit more advantages, which demonstrates the importance of flexible parameter sharing. However, the simple Shared-Bottom model sometimes even obtains better performance, which demonstrates that adequate parameter sharing may also be important for specific datasets. 4) AutoMTL can consistently outperform other human-crafted MTL models and random search on all tasks and datasets, which demonstrates the effectiveness of AutoMTL. And AutoMTL-R performs competitively in most case. It can demonstrate the effectiveness of our search space.

In summary, AutoMTL can alleviate the negative transfer problem and achieve adequate parameter sharing through effective search space and novel architecture search strategy. Also, due to architecture search, AutoMTL can adapt to various tasks and datasets.

Beside performance comparison, we also report the search cost of AutoMTL. As shown in Table 4, we present the runtime required to perform the architecture search and re-training on each dataset. All experiments are run on an NVIDIA 3090 GPU. Although the search space of AutoMTL is huge, the search cost is almost at the same order of magnitude as training a single model, which is acceptable in real-world applications. In the future, we can further reduce the search cost by sampling a subset of the dataset to perform the architecture search.

Table 4: Search cost. GPU hours are reported.

Dataset	UserBehavior	IJCAI	NL	ES	FR	US
Search	0.8	1.8	10	18	16	16.5
Retrain	0.09	0.2	1.1	2.2	1.8	1.9

Table 5: Performance comparison between AutoMTL and its ablation variants.

Method	UserBehavior-2017				IJCAI-2015		AliExpress-NL	
	AUC0	AUC1	AUC2	AUC3	AUC0	AUC1	AUC0	AUC1
AutoMLT	0.7275	0.7323	0.7085	0.8465	0.7741	0.8571	0.7345	0.8640
w/o Int	0.7252	0.7283	0.7054	0.8433	0.7719	0.8533	0.7305	0.8612
hom Exp	0.7241	0.7293	0.7055	0.8441	0.7708	0.8539	0.7313	0.8607
w/o Sel	0.7235	0.7274	0.7034	0.8428	0.7710	0.8523	0.7303	0.8600
w/o Aux	0.7245	0.7277	0.7054	0.8417	0.7697	0.8531	0.7311	0.8592
w/o OLES	0.7220	0.7266	0.7017	0.8413	0.7672	0.8501	0.7298	0.8579

4.3 Ablation Study (RQ2)

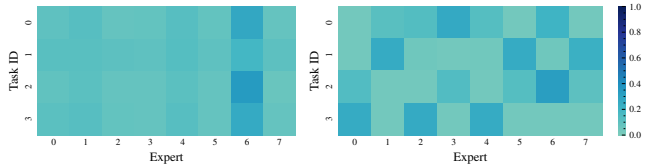
To verify the impacts of different composition modules in AutoMTL, we design several variants: without feature interaction module (w/o Int), homogeneous expert architecture (hom Exp), without expert selection module (w/o Sel), without auxiliary loss (w/o Aux), and without operation-level early stopping (w/o OLES). The performance of these variants is reported in Table 5. Next, we will discuss each variant in detail.

4.3.1 Feature Interaction Module. To verfy the impact of the feature interaction module, we remove the feature interaction module from the framework (w/o Int). The performance drops slightly on most datasets and tasks, which demonstrates the effectiveness of the feature interaction module. However, the interaction module does not have the desired effect on performance. We suppose that the widely-used feature interaction operations in this module may not be general enough for multiple tasks. In addition to the flexible parameter sharing pattern, obtaining more general feature representations that can adapt to multiple tasks is also an important way to boost the performance of multi-task recommendations. In the future, we will design specific feature interaction operations that can adapt to multiple tasks for the multi-task recommendation.

4.3.2 Heterogeneity of Experts. To verify the impact of heterogeneous experts, we force all experts to share a homogeneous architecture (hom Exp). All experts share common architecture parameters during the architecture search, so the resulting architectures have homogeneous experts. The performance also drops slightly on all datasets and tasks. It demonstrates the advantage of heterogeneous experts for MMoE-based MTL models. Heterogeneous experts can easily learn diverse representations that can reflect different aspects of the data.

4.3.3 Expert Selection Module. To verify the impact of the expert selection module, we remove the expert selection module by directly using raw gating networks to aggregate expert representations both in search and retraining stages (w/o Sel). As a result, the expert selection parameters are not contained in the architecture search. All the experts are aggregated for each task. The performance drop is relatively severe on most datasets compared with other variants. This is because the expert selection module not only can alleviate the negative transfer problem by explicitly selecting experts for each task but also plays an important role in stabilizing the architecture search process.

4.3.4 Auxiliary Loss. The auxiliary loss is used to help different tasks select diverse experts and help all the experts be adequately trained. To verify the impact of the auxiliary loss, we remove the auxiliary loss during the architecture search (w/o Aux). The performance also drops obviously on all datasets and tasks, which demonstrates the importance of auxiliary loss for AutoMTL. The auxiliary loss also plays an important role in stabilizing the architecture search process.



(a) Without auxiliary loss **(b) With auxiliary loss**

Figure 3: Distributions of expert selection parameters for different tasks on UserBehavior-2017.

In addition, we present the distributions of expert selection parameters (i.e., importance weights of experts) for different tasks with and without auxiliary loss in Figure 3. We show the result on *UserBehavior-2017* here for clarity, since it contains four tasks. We get the following observations from the heat map: 1) Without adding auxiliary loss, the expert selection parameters for all tasks focus on several experts, and the weights of experts are more uniform. This phenomenon arises from the fact that experts may not be equally trained as a result of unfair initialization in the early stage of search. It causes several experts to be ignored during selection, and the negative transfer can't be alleviated by selecting diverse experts for different tasks. 2) With the auxiliary loss, different tasks can select diverse experts. In conjunction with the expert selection module, the auxiliary loss can stabilize the architecture search process and alleviate the negative transfer.

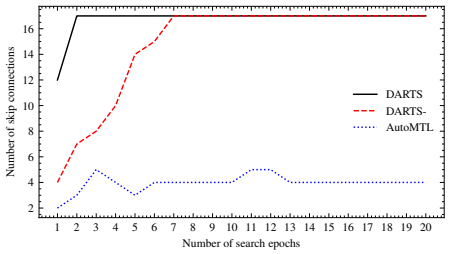
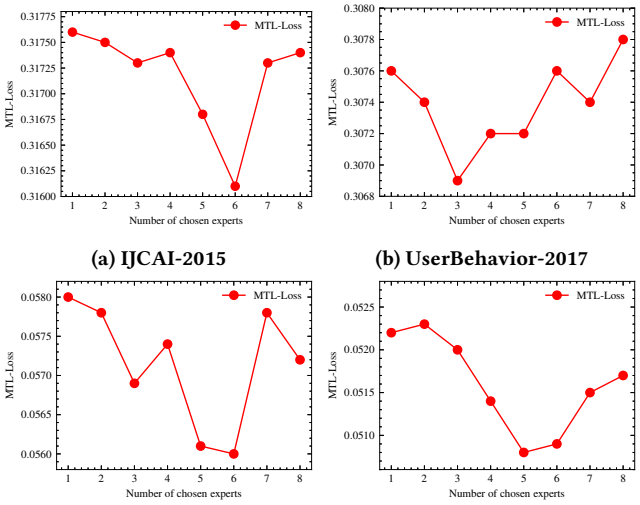


Figure 4: The numbers of skip connections in the architectures searched by different methods.

4.3.5 Operation-Level Early Stopping. To verify the effectiveness of operation-level early stopping, we conduct the architecture search without it (i.e., directly using the well-known differentiable NAS method DARTS [16]) and with the skip connection alleviation method from DARTS− [6]. DARTS− introduces an auxiliary skip connection to separate the unfair advantage of the skip connection and alleviate the aggregation of skip connections. Figure 4 show the numbers of skip connections in architectures searched by these methods on *IJCAI-15*, with the search epochs increasing. Note that the architecture searched by DARTS is full of skip connections

after only 1 epoch. DARTS− also collapse with a few epochs. Therefore, we report the results with only a few epochs of DARTS− in Table 5 (w/o OLES). It can verify the observation that the aggregation of skip connections is caused by overfitting of operations in the supernet and demonstrate the effectiveness of the proposed operation-level early stopping method.

4.4 Hyperparameter Study (RQ3)



(a) IJCAI-2015 (b) UserBehavior-2017

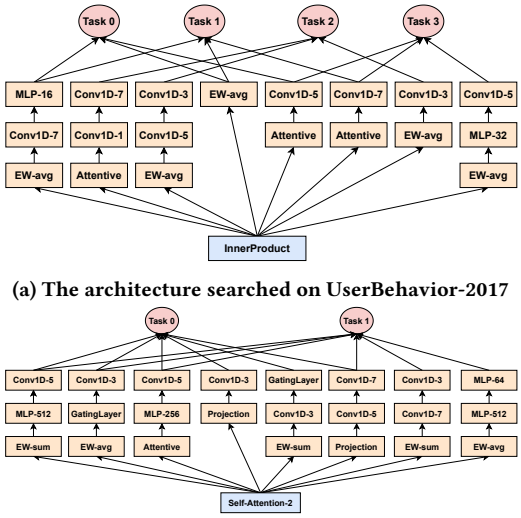
(c) AliExpress-NL (d) AliExpress-US

Figure 5: MTL-Loss with different numbers of chosen experts for each task.

4.4.1 Number of Chosen Experts. One important advantage of AutoMTL is that it can explicitly select a set number of experts for each task through the expert selection module. We conduct experiments to study the impact of the number of chosen experts for each task, and the results are shown in Figure 5, where we report the MTL-Loss. We can observe that: 1) As the number of chosen experts increases, the overall trends of MTL-Losses on all datasets first decrease and then increase. When the number of chosen experts is too small, it is hard to obtain good representations for each task with few experts, and many parameters are wasted due to idle experts. However, when all experts are selected, the negative transfer can easily occur. 2) For datasets with different numbers of tasks, the optimal number of chosen experts is different. For example, when the number of tasks is relative higher, as in *UserBehavior-2017*, it is better to choose fewer experts. AutoMTL can alleviate negative transfer by choosing the appropriate number of experts for each task. Therefore, the number of chosen experts is an important hyperparameter that needs to be carefully set.

4.5 Case Study (RQ4)

In Figure 6, we display the best architectures searched by AutoMTL on *UserBehavior-2017* and *AliExpress-NL*. From the derived architectures, we can observe that: 1) The resulting architectures always contain diverse experts with different operations and layer numbers. 2) The architecture searched on *UserBehavior-2017* is much



(a) The architecture searched on UserBehavior-2017

(b) The architecture searched on AliExperss-NL

Figure 6: The architectures searched by AutoMTL.

simpler than that on *AliExpress-NL*. It is because *AliExpress-NL* has a much larger scale than *UserBehavior-2017*. Actually, AutoMTL is a data-specific method that aims to find optimal architectures for given datasets. 3) Different tasks choose diverse experts in the searched architectures, and several experts can also be selected by multiple tasks. 4) The 1D Convolution and Gating Layer are common selections in the expert module. These operations contain much fewer parameters and computations than fully-connected networks and are also general enough for different types of tasks. AutoMTL tends to generate simple architectures for the expert sub-network.

These observations can inspire human experts to design more powerful MTL models with considerations for experts' diversity, the flexible parameter sharing pattern, and more advanced operations.

5 CONCLUSION AND FUTHER WORK

In this paper, we proposed the AutoMTL framework, which utilizes NAS to design superior architectures for multi-task recommendations in a data-specific manner. We tailored the search space for recommendation tasks, and the overall framework mainly contains three modules: the feature interaction module, the heterogeneous expert module, and the expert selection module. We utilized an efficient differentiable architecture search algorithm to explore the huge search space and propose a novel search strategy with an auxiliary loss to stabilize the search process. To alleviate the aggregation of skip connections in differentiable NAS, we further propose a novel operation-level early stopping method. Extensive experiments demonstrate that AutoMTL can consistently outperform existing state-of-the-art human-crafted MTL models.

In the future, we plan to integrate more operations into the search space and design more general feature interaction operations that can adapt to multiple tasks. And we will further explore how to exploit the correlations among tasks in our framework.

REFERENCES

[1] Han Cai, Ligeng Zhu, and Song Han. 2018. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332* (2018).

[2] Rich Caruana. 1993. Multitask Learning: A Knowledge-Based Source of Inductive Bias. In *Proceedings of the Tenth International Conference on International Conference on Machine Learning* (Amherst, MA, USA) *(ICML'93)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 41–48.

[3] Xiaokai Chen, Xiaoguang Gu, and Libo Fu. 2021. Boosting share routing for multi-task learning. In *Companion Proceedings of the Web Conference 2021*. 372–379.

[4] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. 2021. Progressive darts: Bridging the optimization gap for nas in the wild. *International Journal of Computer Vision* 129 (2021), 638–655.

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[6] Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. 2020. DARTS-: Robustly Stepping out of Performance Collapse Without Indicators. https://doi.org/10.48550/ARXIV.2009.01027

[7] Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. 2015. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Proceedings of the 53rd annual meeting of the Association for Computational Linguistics and the 7th international joint conference on natural language processing (volume 2: short papers)*. 845–850.

[8] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *The Journal of Machine Learning Research* 20, 1 (2019), 1997–2017.

[9] Yulong Gu, Zhuoye Ding, Shuaiqiang Wang, Lixin Zou, Yiding Liu, and Dawei Yin. 2020. Deep multifaceted transformers for multi-objective ranking in large-scale e-commerce recommender systems. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2493–2500.

[10] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. *arXiv preprint arXiv:1703.04247* (2017).

[11] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*. Springer, 544–560.

[12] Tongwen Huang, Qingyun She, Zhiqiang Wang, and Junlin Zhang. 2020. GateNet: gating-enhanced deep network for click-through rate prediction. *arXiv preprint arXiv:2007.03519* (2020).

[13] Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. 2017. One model to learn them all. *arXiv preprint arXiv:1706.05137* (2017).

[14] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1754–1763.

[15] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130* (2017).

[16] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[17] Junning Liu, Zijie Xia, Yu Lei, Xinjian Li, and Xu Wang. 2021. Multi-Faceted Hierarchical Multi-Task Learning for a Large Number of Tasks with Multi-dimensional Relations. *arXiv preprint arXiv:2110.13365* (2021).

[18] Jiaqi Ma, Zhe Zhao, Jilin Chen, Ang Li, Lichan Hong, and Ed H Chi. 2019. SNR: sub-network routing for flexible parameter sharing in multi-task learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 216–223.

[19] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. 2018. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1930–1939.

[20] Xiao Ma, Liqin Zhao, Guan Huang, Zhi Wang, Zelin Hu, Xiaoqiang Zhu, and Kun Gai. 2018. Entire space multi-task model: An effective approach for estimating post-click conversion rate. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 1137–1140.

[21] Krzysztof Maziarz, Efi Kokiopoulou, Andrea Gesmundo, Luciano Sbaiz, Gabor Bartok, and Jesse Berent. 2019. Gumbel-matrix routing for flexible multi-task learning. (2019).

[22] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. 2016. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3994–4003.

[23] Xiaofeng Pan, Yibin Shen, Jing Zhang, Keren Yu, Hong Wen, Shui Liu, Chengjun Mao, and Bo Cao. 2021. SAME: Scenario Adaptive Mixture-of-Experts for Promotion-Aware Click-Through Rate Prediction. *arXiv preprint arXiv:2112.13747* (2021).

[24] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*. PMLR, 4095–4104.

[25] Zhen Qin, Yicheng Cheng, Zhe Zhao, Zhe Chen, Donald Metzler, and Jingzheng Qin. 2020. Multitask mixture of sequential experts for user activity streams. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3083–3091.

[26] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 1149–1154.

[27] Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098* (2017).

[28] Sebastian Ruder, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard. 2017. Sluice networks: Learning what to share between loosely related tasks. *arXiv preprint arXiv:1705.08142* 2 (2017).

[29] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017).

[30] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. 2019. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1161–1170.

[31] Hongyan Tang, Junning Liu, Ming Zhao, and Xudong Gong. 2020. Progressive layered extraction (ple): A novel multi-task learning (mtl) model for personalized recommendations. In *Fourteenth ACM Conference on Recommender Systems*. 269–278.

[32] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*. 1–7.

[33] 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.

[34] Hong Wen, Jing Zhang, Yuan Wang, Fuyu Lv, Wentian Bao, Quan Lin, and Keping Yang. 2020. Entire space multi-task modeling via post-click behavior decomposition for conversion rate prediction. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 2377–2386.

[35] Dongbo Xi, Zhen Chen, Peng Yan, Yinger Zhang, Yongchun Zhu, Fuzhen Zhuang, and Yu Chen. 2021. Modeling the sequential dependence among audience multi-step conversions with multi-task learning in targeted display advertising. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3745–3755.

[36] Quanming Yao, Ju Xu, Wei-Wei Tu, and Zhanxing Zhu. 2020. Efficient neural architecture search via proximal iterations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 6664–6671.

[37] Qianqian Zhang, Xinru Liao, Quan Liu, Jian Xu, and Bo Zheng. 2022. Leaving No One Behind: A Multi-Scenario Multi-Task Meta Learning Approach for Advertiser Modeling. *arXiv preprint arXiv:2201.06814* (2022).

[38] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. 2019. Recommending what video to watch next: a multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 43–51.

[39] Yong Zheng et al. 2021. Multi-Objective Recommendations: A Tutorial. *arXiv preprint arXiv:2108.06367* (2021).

[40] Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).