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

Anonymous

ABSTRACT

Deep learning-based recommendation models have been widely used in industry and attracted more and more research efforts. Typically, recommender systems need to simultaneously optimize multiple objectives. Many multi-task learning (MTL) methods in recommendation follow the multi-gate mixture-of-experts (MMoE) paradigm to design their architectures. However, these models are usually designed for specific scenarios and can not generalize to other scenarios. And the MMoE-based models also suffer from negative transfer problem. Moreover, homogeneous expert architectures in traditional MMoE-based models inherently hinder their performances. In this paper, we propose an effective multi-task learning neural architecture search (NAS) framework named AutoMTL, which utilizes NAS to design superior architectures for multi-task recommendation in a data-specific manner. The overall framework also follows the MMoE paradigm. We tailor the search space for recommendation tasks, and the framework mainly contains three modules: the feature interaction module for finding effective feature interaction operations, the heterogeneous expert module for generating diverse expert subnetworks, and the expert selection module for selecting appropriate experts for each task. 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. Extensive experiments demonstrate that AutoMTL can consistently outperform state-of-the-art human-crafted MTL models. Anonymous code of AutoMTL is now available at https://github.com/automtl/AutoMTL.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

Multi-task learning, Recommender systems, Neural architecture search

ACM Reference Format:
Anonymous. 2018. AutoMTL: 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 have been playing an increasing important role in people's lives. Deep learning models have been widely used in industrial recommender systems and attracted more and more research efforts [4, 9, 15]. Typically, recommender systems often need to optimize multiple objectives at the same time. For example, the e-commercial platform may want to simultaneously predict user's multiple behaviors (such as click, favorite, and purchase.) to enhance the user experience, as well as increase income. As a result, multi-task learning (MTL) in recommender system has attracted ever-increasing interest from industry and academia, and becomes a hot research topic. Through MLT, 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.

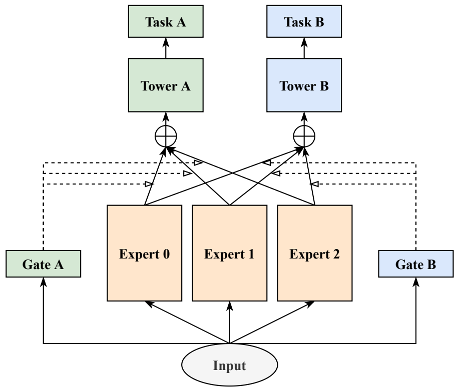


Figure 1: Multi-gate Mixture-of-Experts (MMoE) model

Multi-task learning aims to make full use of the knowledge contained in multiple tasks to improve the overall performance. Recently, deep neural networks (DNNs) based MTL approaches mainly rely on flexible parameter sharing to transfer knowledge across tasks. One of the most effective deep MTL methods is Multi-gate Mixture-of-Experts (MMoE) [19], which can also be viewed as a paradigm of the deep MTL model. As shown in Figure 1, it shares several expert subnetworks across all tasks and then aggregates the results from all experts by the gating networks. 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, y_k denotes the output prediction for task k, h^k is the task-specific network (i.e., "tower"), f^k denotes the aggregated representation for task k, g^k denotes the gating network, n is the number of experts, and f_i denotes the subnetwork of expert i. $W_g^k \in \mathbb{R}^{n \times d}$ is a trainable weight matrix in the gating network. The gating network that can also be viewed as a type of attention mechanism is crucial for MMoE, since it can adaptively aggregate representations for each instance. Due to its effectiveness, many MTL methods in multi-task recommendation follow the MMoE paradigm to design the model architectures [8, 23, 37, 38]. Specifically, they usually design specific expert architectures according to the scenarios and tasks.

However, in practice, the MMoE paradigm also exists the following limitations. The first is the *negative transfer* problem. Since multiple tasks may have complex correlations and even conflict with each other, sharing information with a less related task may hurt performance. MMoE is inherently a full-sharing approach which forces all experts to contribute to all tasks [3]. As a result, the MMoE-based models also suffer from negative transfer problem. The second is the *architecture homogeneity* problem of all the experts in the MMoE-based models, making it difficult to learn different aspects from the input data. The diversity of representations learned by experts is significantly important to the MMoE-based models [19]. It is impractical to design diverse architectures for different experts manually. The third is the issue of *model design efficiency*. Recommender systems always contain various scenarios and tasks. Designing effective multi-task recommendation models is a labor-intensive task, which also requires extensive expert experience and domain knowledge.

Inspired by recent success of neural architecture search (NAS), we propose to utilize NAS to design optimal MTL models based on the MMoE paradigm. There are also several challenges for applying NAS to multi-task recommendation: 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 which 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 large search space. We should design an efficient search strategy to handle such a huge search space.

In this paper, to address above challenges, we propose an effective NAS-based multi-task learning framework called AutoMTL for automatic architecture search in multi-task recommendation. The overall framework of AutoMTL also follows the MMoE paradigm. First, to alleviate the negative transfer problem in MMoE-based models, we design a novel *expert selection module* which exploits expert selection parameters to explicitly select appropriate experts for each task. During the retraining process, the selected experts will be adaptively aggregated for each instance by the gating network. Second, we design an effective *end-to-end search space*, and the *expert module* in the search space can generate heterogeneous architectures for different experts. We also introduce a *feature interaction module* which can automatically search for effective interaction operations across input features. Third, we utilize an efficient differentiable NAS algorithm to explore the search space, and propose a novel search strategy with an auxiliary loss to stabilize the search process. In summary, the main contributions of this work are as follows:

- We propose the AutoMTL framework, which utilizes NAS to find optimal architectures for multi-task recommendation. AutoMTL is composed of feature interaction module, heterogeneous expert module, and expert selection module. It can not only alleviate the negative transfer problem by task-dependent expert selection but also enhance the representation diversity by heterogeneous expert architectures.
- Based on the AutoMTL framework, we design a general and expressive search space, which can cover both existing and unexplored MTL architectures.
- We further perform continuous relaxation on the proposed search space and utilize an effective differentiable architecture search algorithm to explore the huge search space. Moreover, a novel search strategy with an auxiliary loss is proposed to stabilize the search process.
- Extensive experiments on four public datasets demonstrate the effectiveness of AutoMTL, which can consistently outperform the state-of-the-art human-crafted MTL models.

2 RELATED WORK

2.1 Multi-Task Learning

Multi-task learning has been successfully adopted in many applications of machine learning, including natural language processing (NLP),computer vision, and recommender system [27]. MTL aims to make full use of the knowledge contained in multiple tasks to improve the overall performance. In fact, MTL can be viewed as a form of transfer learning [27], which transfers knowledge across different tasks and introduce inductive bias contained in these tasks to boost the performance. However, the multi-task learning model may not always outperform the specific single-task models [13, 19]. Since multiple tasks may have complex correlations and even conflict with each other, sharing information with a less related task may hurt performance, which is known as negative transfer.

DNNs-based MTL approaches mainly rely on parameter sharing [27]. There exist two typical schemes of parameter sharing, i.e., hard parameter sharing and soft parameter sharing. The hard parameter sharing is implemented by Shared-Bottom model [2], which shares several bottom hidden layers across all tasks. It may suffer significant performance degeneration when the tasks are less related. In contrast, soft parameter sharing can alleviate negative transfer through more flexible parameter sharing pattern. The method proposed in [6] adds L2 constraint between two single-task models' parameters, however it requires prior knowledge of task relationship. Cross-stitch network [22] and sluice network [28] both exploit linear combinations to fuse representations from different tasks selectively, but the representations are combined with the same static weights for all samples. MoE [29] first proposes to share multiple expert subnetworks at bottom and then aggregates the result of experts through a gating network. Then MMoE [19] extends MoE by utilizing different gating networks for different tasks. MMoE can also be regarded as an effective MTL paradigm, and many recent methods have followed the MMoE paradigm to design their architectures[31, 38]. However, negative transfer still exists in these soft parameter sharing methods.

2.2 MTL for Recommender System

Typically, the recommender systems are required to predict multiple objectives at the same time [39]. Inspired by the success of MTL in computer vision [22] and NLP [29], tremendous endeavors have been devoted to apply MLT to recommender systems. There are mainly two forms of multi-task recommendation methods, i.e., *Expert-Bottom-based* method and *Probability-Transfer-based* method [35].

The *Expert-Bottom-based* methods usually follow the MMoE paradigm. PLE [31] model separates task-common and task-specific experts explicitly to alleviate conflicts resulted from complex correlations among tasks. HMoE [14] further utilizes information from label space to learn the correlations of tasks on the basis of MMoE. MoSE [25] is proposed to model sequential user behaviors in multi-task recommendation. Similarly, SAME [23] utilizes self-attention to handle user behavior sequence in each expert and feeds scenario signals to the gating network to help it adapt to different scenarios. The *Probability-Transfer-based* methods utilize the dependency relationships of tasks to transfer probabilities in the output layers of different tasks. ESMM [20] and ESM^2 [34] utilize the sequential dependency of user behaviors to boost the prediction of conversion rate (CVR). AITM [35] models the sequential dependence among multi-step conversions and adaptively learns what and how much information to transfer.

Most of these methods are designed for specific scenarios and hardly to be transformed to other multi-task recommendation scenarios, which significantly limits their generalities. In this paper, we propose AutoMTL, which utilizes NAS to design superior MTL models for multi-task recommendation. And we tailor the search space for recommendation tasks. We don't consider the correlations among tasks for generality, and leave it for the future work.

2.3 Neural Architecture Search

Neural architecture search (NAS) aims to automatically design neural architectures in a data-driven manner. There are basically three existing frameworks for NAS [7]: Evolution-based NAS [10], Reinforcement Learning (RL) based NAS [24, 40], and gradient-based NAS [1, 16, 36]. As a pioneering work, [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., supernet) that connects smaller model components. Different candidates architectures can be generated by selecting a subset of components. ENAS [24] uses an RNN controller to sample candidate architectures from the supernet. DARTS [16], ProxylessNAS [1], and NASP [36] take a differentiable approach by relaxing the hard selection to continuous weight and applying bi-level optimization to jointly optimize the architecture parameters and the network weights. SPOS [10] first trains the supernet with randomly sampled architectures, then searches the optimal architecture by an evolutionary algorithm.

There are also some works utilize NAS to find good parameter sharing patterns for MTL. SNR [18] controls connections between subnetworks by binary random variables and exploits NAS to search for the optimal structure. Gumbel-matrix routing [21] learns the routing of MTL models by formulating it as a binary matrix with Gumbel-Softmax trick. MTNAS [3] efficiently finds suitable sparse sharing routing for given MTL problem. But these methods are not specifically designed for recommendation tasks and mainly focus on searching for parameter sharing routing, not expressive architectures. Different from above methods, our AutoMTL mainly focus on utilizing NAS to search for optimal architectures for recommendation tasks.

3 METHODOLOGY

We propose AutoMTL framework, which can automatically find effective architectures for multi-task recommendation. In this section, we first state the problem and define the goal of AutoMTL, then present an overview of our method, 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 find superior models for multi-task recommendation. The input features usually contain category and numerical features of user, item and contextual information. The category features will first be transformed to dense vectors by embedding lookup tables. There are multiple objectives $T_1, \cdots, T_K$ need to be predicted, such as "click", "favorite", "add-to-cart", and "purchase". Following a common used manner [31], we mainly focus on the performance of each task, and model the final objective as a weighted summation of all the tasks' objectives. Consequently, the objective function is defined as

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

where $\mathcal{L}_i$ denotes the loss function of task i, ω_i is the weight of $\mathcal{L}_i$ and $\sum_{i=1}^{K} \omega_i = 1$. $\mathcal{L}$ is the total loss function. The goal of AutoMTL can be easily 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 training data and validation data. $\mathcal{A}$ denotes the architecture search space, $\boldsymbol{w}$ denotes network weights, and $\boldsymbol{\alpha}$ denotes the architecture parameters.

3.2 Overall Framework

We design our framework on the basis of MMoE paradigm. Figure 2 depicts the overall framework of AutoMTL, which mainly contains three modules: feature interaction module, heterogeneous expert module, and expert selection module. The feature interaction module at the bottom of framework is used to search for effective feature interaction operations for recommendation tasks. All the experts in the heterogeneous expert module share the same inputs and common search space, but can learn different architectures to model different aspects of the data. Then the expert selection module will aggregate representations from experts for each task by explicitly learning the importance of experts. During retraining, experts with low importance will be dropped, and the gating networks will be used to adaptively aggregate results of selected experts. In the search stage, we first construct a supernet according to the search space, each layer of the supernet is regarded as a choice block which contains all the available operations. Then we

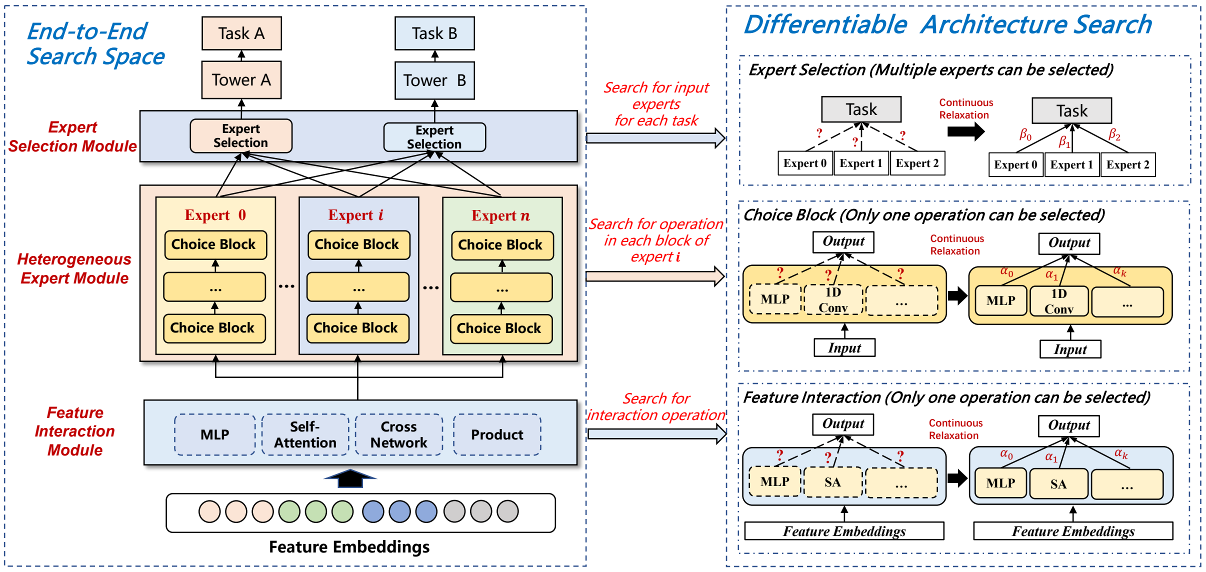

End-to-End Search Space
Task A
Task B
Tower A
Tower B
Expert Selection Module
Expert Selection
Expert Selection
Heterogeneous Expert Module
Expert 0
Expert i
Expert n
Choice Block
...
Feature Interaction Module
MLP
Self-Attention
Cross Network
Product
Feature Embeddings
Search for input experts for each task
Search for operation in each block of expert i
Search for interaction operation
Differentiable Architecture Search
Expert Selection (Multiple experts can be selected)
Task
Continuous Relaxation
Expert 0
Expert 1
Expert 2
Choice Block (Only one operation can be selected)
Output
Input
MLP
1D Conv
Feature Interaction (Only one operation can be selected)
SA
Feature Embeddings

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

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**
	Cross Network	4	*The number of cross layers:* **1, 2, 3, 4**
	Self-Attention (SA)	4	*The number of head:* **1, 2, 4, 8**
	Multi-layer Perceptron	5	*Hidden layer size:* **64, 128, 256, 512, 1024**
	Skip-Connection	1	/
Expert	Bit-level Gating Layer	1	/
	Multi-layer Perceptron	7	*Hidden layer size:* **16, 32, 64, 128, 256, 512, 1024**
	1D Convolution	3	*Kernel size:* **3, 5, 7**
	Skip-Connection	1	/

utilize a differentiable architecture search algorithm following the weight-sharing paradigm to search for optimal architectures. And we further propose a novel search strategy with an auxiliary loss to stabilize the search process.

3.3 Search Space Design

As Figure 2 illustrates, we follow the weight-sharing paradigm to design our search space and construct the supernet. The overall framework mainly contains there modules: feature interaction module, heterogeneous expert module and expert selection module. The feature interaction module is located at the bottom of the framework, and take the output of embedding layer as input. The representations learned by the feature interaction module are shared by all experts. The experts in the heterogeneous expert module are trained in parallel, and also share a common search space. Each layer in the feature interaction module and experts is regarded as a choice block which contains all the available operations. And only one operation will be finally selected in each choice block. Moreover, in order to adaptively learn the depth of architectures, we additionally add a "skip-connection" operation to each choice block. The expert selection module is responsible for aggregating the outputs of experts for each task. Finally, the aggregated representations will be fed to the task-specific towers to get predictions. We will describe the three modules in detail in the following.

3.3.1 Feature Interaction Module. Previous works [9, 15, 26, 30, 33] have proven that high-order feature interactions are crucial for the performances of recommendation models, so we also introduce a feature interaction module at the bottom to provide more effective feature representations for upper experts. We consider some common used feature interaction operations in this module such as Multi-layer Perceptron (MLP), Product Layer [26], Cross-Net [33], and Self-Attention [32]. More details can be found in Table 1. One operation can correspond to multiple block types with different hyperparameters, e.g., different numbers of hidden units in MLP represent different block types. We only use one feature interaction block here for simplicity. Note that this module can be easily extended by integrating more advanced feature interaction operations or combining multiple feature interaction blocks with sophisticated topologies. The representations learned by feature interaction module will be shared by all the experts.

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 AutoMTL, as described in Table 1, we mainly introduce three types of operations, i.e., Bit-level Gating Layer [12], Multi-layer Perceptron (MLP), and 1D Convolution, that are general enough for different types of tasks. The Gating Layer and 1D Convolution contain much fewer parameters and computations than normal fully-connected networks. As a result, the search space of expert module can not only boost the diversity of experts, but also obtain simpler expert architectures. Note that it is easy to put the feature interaction module into each expert to further promote the performance, but it will significantly increase the complexity of searched models. Also, other available operations can be easily integrated.

3.3.3 Expert Selection Module. As the gating network in MMoE, each task corresponds to an expert selection module in AutoMTL. The expert selection module explicitly learns the importance of experts for each task. When the search is complete, the expert selection module will select specific number of experts for each task according to the learned importance. By doing this, the negative transfer can be significantly alleviated. We treat the learnable importance weights as another type of architecture parameters named expert selection parameters, so it can be directly used to select experts after search. In the retrain stage, we use the gating network to adaptively aggregate representations from the task-dependent experts selected by the expert selection module for each instance. As a result, the expert selection modules are only introduced in the search stage. In addition, we propose an auxiliary loss for the expert selection parameters to encourage the expert selection modules to select diverse experts for different tasks. The formulation of the auxiliary loss will be presented in the next subsection.

3.3.4 Search Space Size Analysis. The search space size is determined by the number of block types in the feature interaction module $|\mathcal{I}|$, the number of block types in the expert $|\mathcal{B}|$, the number of blocks in each expert n, the number of experts N, the number of selected experts by the expert selection module c, and the number of tasks K. Thus, the search space size can be calculated as $|\mathcal{I}| \cdot |\mathcal{B}|^{n \cdot N} \cdot C_N^c \cdot K$. In our experiments, we use 15 types of feature interaction blocks, 11 types of blocks in the expert, 3-5 blocks in each expert, 8 experts and 6 selected experts for the 2 tasks scenario. The search space size is at least approximately 8.27×10^{27}. The huge search space requires efficient search algorithm.

3.4 Differentiable Neural Architecture Search

3.4.1 Continuous Relaxation and Search Strategy. We utilize efficient differentiable architecture search algorithm to explore the search space. As shown in Figure 2, for each choice block in an expert, the hard selection will be relaxed to continuous weights by standard softmax operator. The output of each choice block is represented as a weighted mixture of the candidate operations' outputs. Formally, let $O = \{o_1, \dots, o_M\}$ be a set of M candidate operations in a choice block B, the mixed output is formulated as:

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

where x denotes the input, each operation o_i is associated with a continuous coefficient α_{o_i} named as the architecture parameter. For the feature interaction module, the selection of the interaction operation also follows the same continuous relaxation mechanism.

For the expert selection module, we treat the learnable aggregation weights as another type of architecture parameters, i.e., expert selection parameters. The aggregation weights reflect the importance of experts for corresponding task. Let $\boldsymbol{\beta}^{(k)} = \{\beta_1^{(k)}, \dots, \beta_N^{(k)}\}$ be the expert selection parameters for task k, where N denotes the number of experts. The aggregated representation for task k can be formulated as:

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

where $E_i(x)$ denotes the output of the i-th expert. The aggregated representations will be fed to task-specific towers to obtain the final predictions.

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

3.4.2 Auxiliary Loss and Search Objective. In addition, in order to encourage diverse selections of experts for different tasks and help all the experts to be adequately trained, we propose an auxiliary loss for the expert selection parameters. The auxiliary loss is the extended negative Jensen-Shannon divergence (JSD) [17] for the distributions of expert selection parameters of different tasks, formally as,

$$\mathcal{L}_{JSD} = \sum_{i=1}^{N} m_i \log m_i - \sum_{k=1}^{K} \frac{1}{K} \sum_{i=1}^{N} p_i^{(k)} \log p_i^{(k)} \quad (2)$$

$$\text{where} \quad p_i^{(k)} = \frac{\exp(\beta_i^{(k)})}{\sum_{\beta \in \boldsymbol{\beta}^{(k)}} \exp(\beta)}, \quad m_i = \frac{1}{K} \sum_{k=1}^{K} p_i^{(k)}$$

In order to control its strength, we weight this loss by a coefficient w_{JSD}. During search, w_{JSD} will progressively decay to zero for reducing its impact on the learning of expert selection parameters. In fact, the auxiliary loss can be regarded as a strong priori regularization to the expert selection parameters, i.e., different tasks may prefer diverse representations from the experts.

Let $\boldsymbol{\alpha}$ denote the architecture parameters in the heterogeneous expert module and feature interaction module, and $\boldsymbol{\beta}$ denote the expert selection parameters. In conclusion, the search objective of

AutoMTL can be formalized as

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

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

Algorithm 1 Search algorithm of AutoMTL

```
Require: Training set and validation set, search space $\mathcal{A}$;
Ensure: Optimal searched architectures;
1: Initialize $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{w}, w_{JSD}$.
2: while not converged do
3:     Update architecture parameters $\boldsymbol{\alpha}$ via validation loss $\mathcal{L}_{val}$;
4:     Update supernet weights $\boldsymbol{w}$ via training loss $\mathcal{L}_{train}$;
5:     if not in warm-up stage then
6:         Update expert selection parameters $\boldsymbol{\beta}$ via training loss $\mathcal{L}_{train}$ and auxiliary loss $\mathcal{L}_{JSD}$;
7:         Decay $w_{JSD}$;
8:     end if
9: end while
10: Derive chosen operations base on $\boldsymbol{\alpha}$;
11: Select experts base on $\boldsymbol{\beta}$;
12: Introduce the gating networks for selected experts.
13: return Final optimal architectures;
```

3.4.3 Search Algorithm. To solve the bi-level optimization problem, we employ the common used approximation scheme [16] to optimize the architecture parameters.

$$\begin{aligned} &\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{val}(\boldsymbol{w}^*, \boldsymbol{\beta}^*, \boldsymbol{\alpha}) \\ \approx &\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{val}(\boldsymbol{w} - \xi_1 \nabla_{\boldsymbol{w}} \mathcal{L}_{train}(\boldsymbol{w}, \boldsymbol{\beta}, \boldsymbol{\alpha}), \\ &\qquad \boldsymbol{\beta} - \xi_2(\nabla_{\boldsymbol{\beta}} \mathcal{L}_{train}(\boldsymbol{w}, \boldsymbol{\beta}, \boldsymbol{\alpha}) + w_{JSD} \nabla_{\boldsymbol{\beta}} \mathcal{L}_{JSD}), \boldsymbol{\alpha}) \end{aligned} \tag{4}$$

where $\boldsymbol{w}$ and $\boldsymbol{\beta}$ denote the current network weights and expert selection parameters, and ξ_1 and ξ_2 are learning rates for a step of inner optimization. We also use the first-order approximation during architecture parameters training for efficiency. At the beginning of search, we introduce a worm-up stage, which fixes the expert selection parameters for several epochs to stabilize the search process. After search, the operation with the highest strength will be selected as operation in the final architecture. And the expert selection module will select specific number of experts for each task according to the expert selection parameters. Then the gating networks will be introduced for 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 report on our experimental results to evaluate the effectiveness of AutoMTL. Specifically, we will answer the following questions:

- **RQ1:** Can the performances of models searched by AutoMTL outperform state-of-the-art human-crafted ones on the benchmark datasets?
- **RQ2:** What are the impacts of different modules in AutoMTL?
- **RQ3:** How do the hyperparameters of AutoMTL influence it's performance?
- **RQ4:** What insights can we see from searched models?

4.1 Experiment Setup

Table 2: Statistics of the datasets

Dataset	#Samples	Category Fields	Numerical Fields	#Tasks
IJCAI-2015	2, 143, 260	7	0	2
UserBehavior-2017	520, 757	3	0	4
NL	12, 380, 042	17	63	2
US	17, 112, 045	17	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**[1], a public dataset from IJCAI-15 contest, contains millions of anonymized users' shopping logs in the past six months. Following the common preprocessing manner [11], we filter out users who purchase fewer than 20 unique items and items which are purchased by fewer than 10 unique users. We use two tasks: "favorite" and "purchase".
- **UserBehavior-2017**[2] is a public dataset of anonymized user behaviors from Alibaba. We filter out users who purchase fewer than 10 unique items and items which are purchased by fewer than 10 unique users. And we use four tasks: "click", "buy", "cart", and "favorite".
- **AliExpress**[3] is a public dataset gathered from traffic logs in AliExpress. This dataset is collected from five countries, and we use the data from the Netherlands (NL) and America (US) here. We filter out the users and items which appear less than 20 times, and use two tasks: "click" and "purchase".

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

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

- **AUC**. The Area Under the ROC Curve measures the goodness of order by ranking all the items with a predicted value in the test set. A higher AUC indicates better performance.
- **MTL-Loss**: The total loss value of all K tasks on the test set, i.e., the weighted summation of cross-entropy losses of all tasks. We use equivalent weights for all task.

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

It is worth noting that an improvement of AUC at 0.001 level is a significant improvement for the tasks like CTR prediction, which has also been pointed out in existing works [4, 9, 33].

4.1.3 Baselines. 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 the tasks and then each task has an individual "tower" of network on the top of the bottom representations.
- **MMoE** [19] shares some expert subnetworks across all tasks and then integrates experts 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] further 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. All hyperparameters are carefully tuned in the validation set, where early stopping strategy with the patience of 5 is applied. The category features are first transformed to dense vectors by embedding lookup tables, and the numerical features are linearly transformed to the same dimension as embedding. The embedding sizes are set to 16 for *IJCAI-2015* and *UserBehavior-2017*, and 64 for *AliExpress*. All the task-specific towers are implemented as two-layer MLP networks with layers $\{128, 64\}$. 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. For AutoMTL, we also set the number of experts to 8. SGD optimizer is used to optimize the network weights with initial learning rate 0.02 (cosine learning rate decay). The architecture parameters and expert selection parameters are optimized with Adam optimizer with learning rate 0.001. The initial weight of auxiliary loss is set to 1, and linearly decays to 0. All experiments are executed five times with different random seeds, and the average results are reported.

In order to alleviate the skip-connection aggregation problem which easily appears in the differentiable NAS approach, we borrow the idea of DARTS− [5] to additionally add an auxiliary skip-connection to each block to remove the unfair advantage and gradually decay its weight to zero.

4.2 Overall Comparison (RQ1)

To demonstrate the effectiveness of the proposed AutoMTL, we compare it with some state-of-the-art human-crafted models. The results are shown in Table 3, where we report the AUC scores and the MTL-Loss for each dataset. To clearly demonstrate the superiority of AutoMTL, we further report the relative improvements compared with the best baselines. From the results, we observe that: 1) For baseline models, one model is almost impossible to outperform others on all tasks, which illustrates that improving the performances of MTL for all tasks is non-trivial due to complex correlations among tasks. However, AutoMTL can achieve the best performances on all tasks even in the four tasks scenario as *UserBehavior-2017*. 2) More advanced PLE model can not always outperform other methods, and performs relative better on large datasets (e.g., it can achieve very competitive performances on several tasks in AliExpress-NL and AliExpress-US). PLE designs a more flexible progressive parameter sharing routing, however it will also increase the learning difficulty. 3) Sometimes, simple Shared-Bottom model can even obtain better performances, which illustrates that adequate parameter sharing may also be important for the performances of some tasks and datasets. 4) Our method can consistently outperform other human-crafted MTL models on all tasks and datasets, which demonstrates the effectiveness of our method. AutoMTL can alleviate negative transfer problem as well as achieve adequate parameter sharing through its three constructed modules and novel architecture search strategy, which can adapt to different tasks and datasets.

4.3 Ablation Study (RQ2)

To verify the impacts of different constructed modules in AutoMTL, we design some variants of AutoMTL: without feature interaction module (w/o INT), without expert selection module (w/o SEL), homogeneous expert architecture (hom Expert) and without auxiliary loss (w/o AUX). The results are also reported in Table 3, and we will discuss them in the following.

4.3.1 Feature Interaction Module. We remove the feature interaction module from the framework, i.e., w/o INT. The performances slightly drop on most tasks and datasets, which demonstrates the effectiveness of the feature interaction module. But actually the interaction module does not influence the performances so much. We suppose that these widely-used feature interaction operations may not general enough for multiple tasks. In addition to flexible parameter sharing pattern, obtaining more general feature representations which can adapt to multiple tasks is also an important way to boost the performance of MTL. Consequently, it is necessary to design more general feature interaction operations for multi-task recommendation. We will leave it for future work.

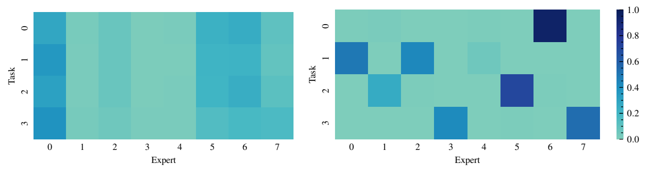
4.3.2 Heterogeneity of Experts. We mainly verify the impact of the heterogeneity of experts here, so we force all experts share homogeneous architecture, i.e., hom Expert. The performances also slightly drop, and the degrees of degradation are various across tasks. For example, it obtains much worse AUC on task_0 of *IJCAI-2015*, but relatively better on task_1. This phenomenon demonstrates the advantage of diverse experts for MMoE-based MTL models. Diverse experts can easily learn better representations for different tasks, and alleviate negative transfer to some extent.

4.3.3 Expert Selection Module. We remove the expert selection modules and replace it with raw gating networks both in search and retraining stages, i.e. w/o SEL. The performances drop relatively severe on most datasets compared with other variants. This is because the expert selection module not only can alleviate negative transfer by explicitly selecting experts, 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 to select diverse experts, and make all the experts to be trained as adequate as possible. We remove the auxiliary loss to verify its influence here, i.e., w/o AUX. The performances also obviously

Table 3: Overall performance comparison. "Impr." indicates relative improvements compared with the best baselines. The best results are marked in bold, and the best baselines are marked by underline. * denotes statistically significant improvement (measured by t-test with p-value < 0.005) over the best baselines.

Method	IJCAI-2015			UserBehavior-2017					AliExpress-NL			AliExpress-US		
	AUC0	AUC1	MTL-Loss	AUC0	AUC1	AUC2	AUC3	MTL-Loss	AUC0	AUC1	MTL-Loss	AUC0	AUC1	MTL-Loss
Shared-Bottom	0.7698	0.8504	0.3176	0.7198	0.7249	0.6930	0.8235	0.3090	0.7080	0.8117	0.0562	0.6960	0.7848	0.0520
MMoE	0.7695	0.8498	0.3177	0.7213	0.7271	0.6981	0.8331	0.3078	0.7064	0.8245	0.0568	0.6874	0.8298	0.0520
CGC	0.7689	0.8516	0.3172	0.7224	0.7256	0.7012	0.8340	0.3084	0.7050	0.8255	0.0574	0.6807	0.8192	0.0527
PLE	0.7695	0.8493	0.3190	0.7229	0.7240	0.6994	0.8274	0.3103	0.7064	0.8350	0.0574	0.6867	0.8480	0.0523
AutoMTL	**0.7711***	**0.8532***	**0.3161***	**0.7249***	**0.7272***	**0.7017***	**0.8378***	**0.3069***	**0.7082***	**0.8429***	**0.0560***	**0.6965***	**0.8504***	**0.0509***
Impr.	0.17%	0.19%	0.35%	0.50%	0.01%	0.52%	0.56%	0.29%	0.03%	2.23%	0.36%	0.07%	2.48%	2.12%
w/o INT	0.7700	0.8484	0.3178	0.7232	0.7257	0.7016	0.8373	0.3076	0.6934	0.8243	0.0577	0.6852	0.8481	0.0519
hom Expert	0.7691	0.8521	0.3171	0.7226	0.7272	0.6998	0.8388	0.3071	0.7008	0.8317	0.0570	0.6880	0.8319	0.0523
w/o SEL	0.7684	0.8471	0.3189	0.7221	0.7265	0.7003	0.8307	0.3076	0.6885	0.8424	0.0578	0.6840	0.8290	0.0525
w/o AUX	0.7703	0.8490	0.3179	0.7216	0.7265	0.7008	0.8348	0.3071	0.7028	0.8348	0.0567	0.6775	0.8403	0.0522



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

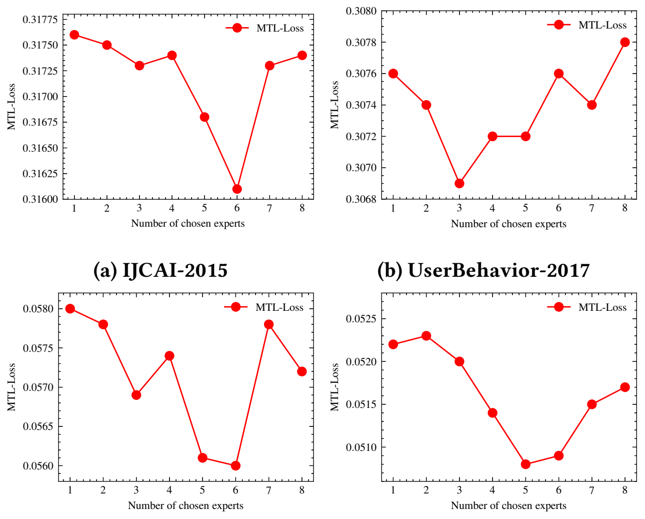
Figure 3: Distributions of expert selection parameters for different tasks on UserBehavior-2017.

drop on all tasks and datasets, which demonstrates the importance of auxiliary loss for AutoMTL.

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 present the result on *UserBehavior-2017* here for clarity, since it contains four tasks. The expert selection parameters determine how the expert selection module selects experts. We get two observations from the heat map: 1) Without adding auxiliary loss, the expert selection parameters for all tasks are concentrated on similar experts, and weights of experts are more uniform. This phenomenon is due to several experts may not be adequately trained due to not exactly fair initialization in the search process. It will cause these experts to be ignored during selection, and the negative transfer can't be alleviated by selecting different experts for different tasks. 2) With the addition of auxiliary loss, different tasks will concentrate on different experts, and the distributions are more sharp. It can easily select diverse experts for different tasks. The auxiliary loss can stabilize the search process and alleviate negative transfer in conjunction with the expert selection module.

4.4 Hyperparameter Study (RQ3)

4.4.1 Number of Chosen Experts. One important advantage of AutoMTL is that it can explicitly select specific 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 4, where we report the total MTL-Loss. We observe that: 1) The overall trends of MTL-Losses on all datasets first decrease and then increase as the number of chosen experts increases. When the number of chosen experts is too small, it is hard to obtain good representations for each task, and some parameters will be wasted due to idle experts. However, when too many experts are selected, negative transfer can easily occur. 2) For different datasets with different numbers of tasks, the optimal numbers of chosen experts are different. For example, when the number of task is relative more as in *UserBehavior-2017*, it is better to choose fewer experts. AutoMTL can alleviate negative transfer through choosing appropriate number of experts for each task. And the number of chosen experts is an important hyperparameter for AutoMTL, which needs to be carefully set.

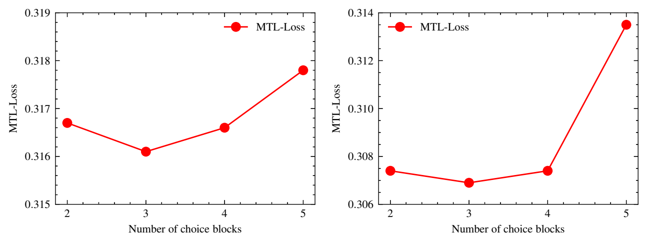


(a) IJCAI-2015 **(b) UserBehavior-2017**

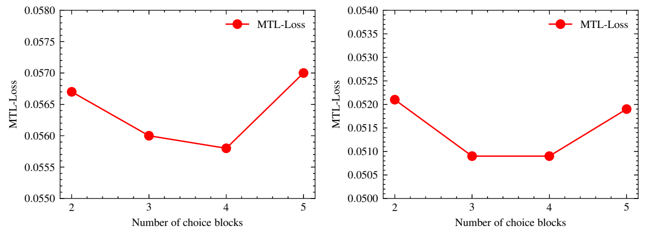
(c) AliExpress-NL **(d) AliExpress-US**

Figure 4: MTL-Loss with different numbers of chosen experts for each task.

4.4.2 Number of Choice Blocks in Each Expert. The number of choice blocks in each expert indicates the maximum number of layers of each expert in the final derived architectures. It is easier to obtain deeper and more complex models with greater number

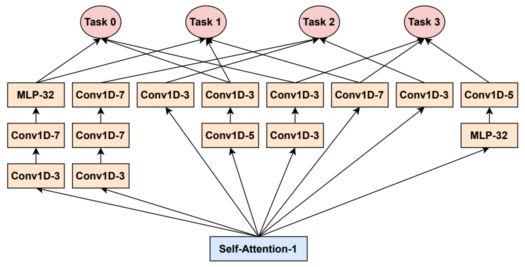


(a) IJCAI-2015 (b) UserBehavior-2017

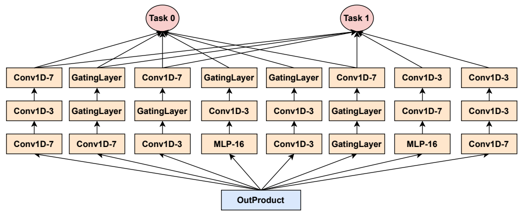


(c) AliExpress-NL (d) AliExpress-US

Figure 5: MTL-Loss with different numbers of choice blocks in each expert.



(a) The architecture searched on UserBehavior-2017



(b) The architecture searched on AliExperss-NL

Figure 6: The architectures searched by AutoMTL.

of choice blocks. Although AutoMTL can learn the depth of network through "skip-connection" operation in each block, greater number of choice blocks also means longer search time and bigger overfitting risk. And the search space size will exponentially expand with increasing number of choice blocks. Figure 5 shows the MTL-Losses with number of choice blocks from 2 to 5. The performances of AutoMTL also first increase and then decrease as the number of choice blocks increases. The performances on all datasets significantly drop when the number of choice blocks up to 5. The overfitting problem seems more severe in MLT. Moreover, an exponentially expanded search space will bring more difficulties for architecture search. Usually, we don't need complex architecture in each expert. Complex expert may bring negative influence, since one expert is only used to capture limited aspects of the data. It also has been mentioned in previous literature [19, 31].

4.5 Case Study (RQ4)

In Figure 6, we display the the best architectures searched by AutoMTL on *UserBehavior-2017* and *AliExpress-NL*. From the derived architectures, we can observe that: 1) AutoMTL can obtain diverse experts, which verifies the advantage of diverse experts in MMoE-based MTL models. 2) The architecture searched on *UserBehavior-2017* is much simpler than that on *AliExpress-NL*, where most experts contain fewer blocks. It is due to the scale of *AliExpress-NL* is much larger than *UserBehavior-2017*. Actually, AutoMTL is a data-specific method, which can find optimal architectures for given dataset. 3) Different tasks choose diverse experts in the searched architectures and several experts can also be selected by multiple tasks, which demonstrates that the expert selection module is able to alleviate negative transfer by selecting diverse experts for different tasks. 4) The 1D Convolution operations and Bit-level Gating Layer are common selections in the derived architectures. These operations contain much fewer parameters and computations than fully-connected structures, and general enough for different types of tasks. AutoMTL tends to generate simple architectures for expert subnetworks.

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

5 CONCLUSION AND FUTURE WORK

In this paper, we propose AutoMTL framework, which utilizes NAS to design superior architectures for multi-task recommendation in a data-specific manner. We tailor the search space for recommendation tasks, and the overall framework mainly contains three modules: feature interaction module for finding effective feature interaction operations, heterogeneous expert module for generating diverse expert subnetworks, and expert selection module for selecting appropriate experts for each task. We utilize efficient differentiable architecture search algorithm to explore the huge search space, and propose a novel search strategy which separates the training of architecture parameters and expert selection parameters into different stages. In addition, in order to encourage different tasks to select diverse experts and stabilize the search process, we propose an extended JS divergence auxiliary loss. Extensive experiments demonstrate that AutoMTL can consistently outperform other state-of-the-art human-crafted MTL models.

In the future, we plan to integrate more effective operations to the search space, and design more general feature interaction operations which can adapt to multiple tasks. And we will further explore and exploit the correlations among tasks in our framework. Moreover, we will design more efficient architecture search algorithm and more flexible strategy to select experts in the expert selection module.

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] 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.

[5] Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. 2020. Darts-: robustly stepping out of performance collapse without indicators. *arXiv preprint arXiv:2009.01027* (2020).

[6] 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.

[7] 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.

[8] 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.

[9] 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).

[10] 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.

[11] Yun He, Xue Feng, Cheng Cheng, Geng Ji, Yunsong Guo, and James Caverlee. 2022. MetaBalance: Improving Multi-Task Recommendations via Adapting Gradient Magnitudes of Auxiliary Tasks. In *Proceedings of the ACM Web Conference 2022*. 2205–2215.

[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] Pengcheng Li, Runze Li, Qing Da, An-Xiang Zeng, and Lijun Zhang. 2020. Improving multi-scenario learning to rank in e-commerce by exploiting task relationships in the label space. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2605–2612.

[15] 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.

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

[17] Jinghui Lu, Maeve Henchion, and Brian MacNamee. 2017. Extending jensen shannon divergence to compare multiple corpora. In *25th Irish Conference on Artificial Intelligence and Cognitive Science, Dublin, Ireland, 7-8 December 2017*. CEUR-WS. org.

[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] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[33] 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.

[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).