AutoLoss: Automated Loss Function Search in Recommendations

Xiangyu Zhao[1,2], Haochen Liu[2], Wenqi Fan[3*], Hui Liu[2], Jiliang Tang[2], Chong Wang[4]
[1]City University of Hong Kong, [2]Michigan State University, [3]The Hong Kong Polytechnic University, [4]Bytedance
{zhaoxi35,liuhaoc1,liuhui7,tangjili}@msu.edu,wenqifan@polyu.edu.hk,chong.wang@bytedance.com

ABSTRACT

Designing an effective loss function plays a crucial role in training deep recommender systems. Most existing works often leverage a predefined and fixed loss function that could lead to suboptimal recommendation quality and training efficiency. Some recent efforts rely on exhaustively or manually searched weights to fuse a group of candidate loss functions, which is exceptionally costly in computation and time. They also neglect the various convergence behaviors of different data examples. In this work, we propose an AutoLoss framework that can automatically and adaptively search for the appropriate loss function from a set of candidates. To be specific, we develop a novel controller network, which can dynamically adjust the loss probabilities in a differentiable manner. Unlike existing algorithms, the proposed controller can adaptively generate the loss probabilities for different data examples according to their varied convergence behaviors. Such design improves the model's generalizability and transferability between deep recommender systems and datasets. We evaluate the proposed framework on two benchmark datasets. The results show that AutoLoss outperforms representative baselines. Further experiments have been conducted to deepen our understandings of AutoLoss, including its transferability, components and training efficiency.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

AutoML; Recommender Systems; Loss Functions

ACM Reference Format:
Xiangyu Zhao[1,2], Haochen Liu[2], Wenqi Fan[3*], Hui Liu[2], Jiliang Tang[2], Chong Wang[4]. 2021. AutoLoss: Automated Loss Function Search in Recommendations. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3447548.3467208

1 INTRODUCTION

In the era of information explosion, recommender systems play a pivotal role in alleviating information overload, which vastly enhance user experiences in many commercial applications, such as generating playlists in video and music services [55, 63], recommending products in online stores [8, 59, 60, 62, 65], and suggesting locations for geo-social events [15, 34, 61]. With the recent growth of deep learning techniques, there have been increasing interests in developing deep recommender systems (DRS) [36, 49]. DRS has improved the recommendation quality since they can effectively learn feature representations and capture the nonlinear relationships between users and items via deep architectures [54]. Aside from developing sophisticated neural network architectures, well-designed loss functions have also been demonstrated to be effective in improving the performance in different recommendation tasks, such as item rating prediction (regression) [41], click-through rate prediction (binary classification) [11, 16], user behavior prediction (multi-class classification) [58], and item retrieval (clustering) [10].

To optimize DRS frameworks, most existing works are based on a predefined and fixed loss function, such as *mean-squared-error* (MSE) or *mean-absolute-error* (MAE) loss for regression tasks. Then DRS frameworks are optimized in a back-propagation manner, which computes gradients effectively and efficiently to minimize the given loss on the training dataset. During this process, the key step is to calculate gradients of network parameters for minimizing loss functions. However, it is often unclear whether the gradients generated from a given loss function are optimal. For example, in regression tasks, the MSE loss can ensure that the trained model has no outlier predictions with huge errors, while MAE performs better if we only want a well-rounded model that performs well on the majority [3, 6]. Therefore, solely utilizing a predefined and fixed loss function for all *data examples*, i.e., *user-item interactions*, cannot guarantee the optimal gradients throughout, especially when the interactions have varied convergence behaviors in the non-stationary environment of online recommendation platforms. In addition, there is often a gap between the model training and evaluation performance in real-world recommender systems. For instance, we usually train a predictive model by minimizing *cross-entropy loss* in online advertising, and evaluate the model performance by *click-through rate* (CTR). Consequently, it naturally raises a question - can we incorporate more loss functions in the training phase to enhance the model performance?

Efforts have been made to develop strategies to fuse multiple loss functions, which can take advantage of multiple loss functions in a weighted sum fashion. For example, Panoptic FPN [23] leverages a grid search to find better loss weights; and UPSNet [51] carefully investigates the weighting scheme of loss functions. However, these works rely on exhaustively or manually search for loss weights from a large candidate space, which would be an extremely costly execution in both computing power and time. Also, they aim to learn a set of unified and static weights over the loss functions, which entirely overlook the different convergence behaviors of data examples. Finally, retraining loss weights is always desired when

* Corresponding author.

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467208

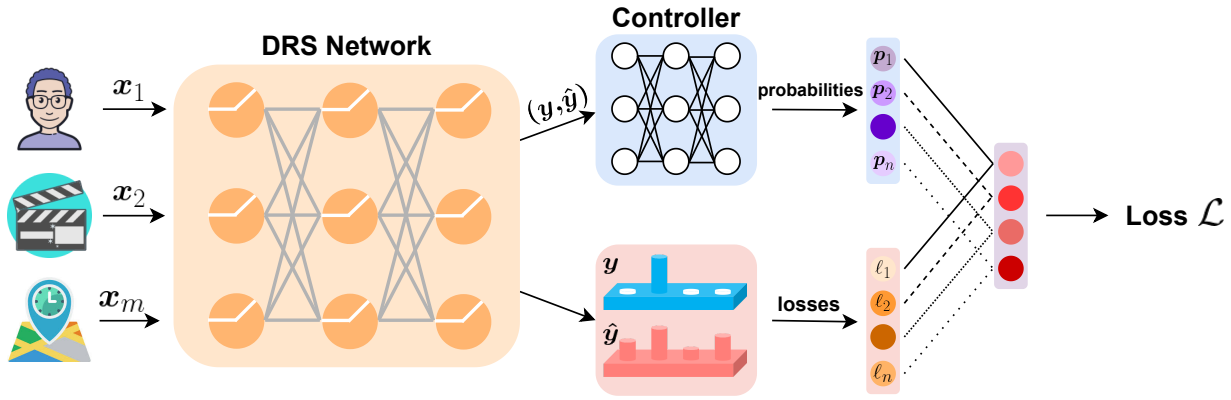

Figure 1: Overview of the AutoLoss framework.

switching among different DRS frameworks or recommendation datasets, which reduces their generalizability and transferability.

In order to obtain more accurate gradients to improve the recommendation performance and the training efficiency, we propose an automated loss function search framework, **AutoLoss**, which can dynamically and adaptively select appropriate loss functions for training DRS frameworks. Different from existing searching models with predefined and fixed loss functions, or the loss weights exhaustively or manually searched, the optimal loss function in AutoLoss is automatically selected for each data example in a differentiable manner. The experiments on two datasets demonstrate the effectiveness of the proposed framework. We summarize our major contributions as follows:

- We propose an end-to-end framework, AutoLoss, which can automatically select the proper loss functions for training DRS frameworks with better recommendation performance and training efficiency;
- A novel controller network is developed to adaptively adjust the probabilities over multiple loss functions according to different data examples' dynamic convergence behaviors during training, which enhances the model generalizability between different DRS frameworks and datasets;
- We empirically demonstrate the effectiveness of the proposed framework on real-world benchmark datasets. Extensive studies verify the importance of model components and the transferability of AutoLoss.

The rest of this paper is organized as follows. In Section 2, we detail the framework of automatically searching the probabilities over multiple loss functions, the architecture of the main DRS network and controller network, and propose an AutoML-based optimization algorithm. Section 3 carries out experiments based on real-world datasets and presents experimental results. Section 4 briefly reviews related work. Finally, Section 5 concludes this work and discusses future work.

2 THE PROPOSED FRAMEWORK

In this section, we will present an end-to-end framework, AutoLoss, which effectively tackles the aforementioned challenges in Section 1 via automatically and adaptively searching the optimal loss function from several candidates according to data examples' convergence behaviors. We will first provide an overview of the framework; next detail the architectures of the main DRS network; then introduce the loss function search method with a novel controller network; and finally provide an AutoML-based optimization algorithm.

2.1 An Overview

In this subsection, we will give an overview of the AutoLoss framework. AutoLoss aims to automatically select appropriate loss functions from a set of candidates for different data examples (i.e., user-item interactions). We demonstrate the framework in Figure 1. With a DRS network, a controller network and a set of pre-defined candidate loss functions, the learning process of AutoLoss mainly consists of two major steps.

The forward-propagation step. Given a mini-batch of data examples, the main DRS network first generates predictions $\hat{\boldsymbol{y}}$ based on the input features $\boldsymbol{x}$. Then, we can calculate the losses $\{\ell_i\}$ for each candidate loss function according to the ground truth labels $\boldsymbol{y}$ and predictions $\hat{\boldsymbol{y}}$. Meanwhile, the controller network takes $(\boldsymbol{y}, \hat{\boldsymbol{y}})$ and outputs the probabilities $\boldsymbol{p}$ over loss functions for each data example. Finally, the overall loss $\mathcal{L}$ can be calculated according to the losses from $\{\ell_i\}$ and the probabilities $\boldsymbol{p}$.

The backward-propagation step. We first fix the parameters of the controller network and update the main DRS network parameters upon the training data examples. Then, we fix the DRS parameters and optimize the controller network parameters based on a mini-batch of validation data examples. This alternative updating approach enhances the generalizability, and prevents AutoLoss from selecting probabilities that overfit the training data examples [29, 37]. Next, we will introduce the details of AutoLoss.

2.2 Deep Recommender System Network

AutoLoss is quite general for most existing deep recommender system frameworks [16, 27, 39, 42]. As visualized in Figure 2, they typically have four components: embedding layer, interaction layer, MLP layer and output layer. We now briefly introduce these components.

2.2.1 ***Embedding Layer.*** The raw input features of users and items are usually categorical or numeric, and in the form of multiple fields. Most DRS works first transform the input features into binary

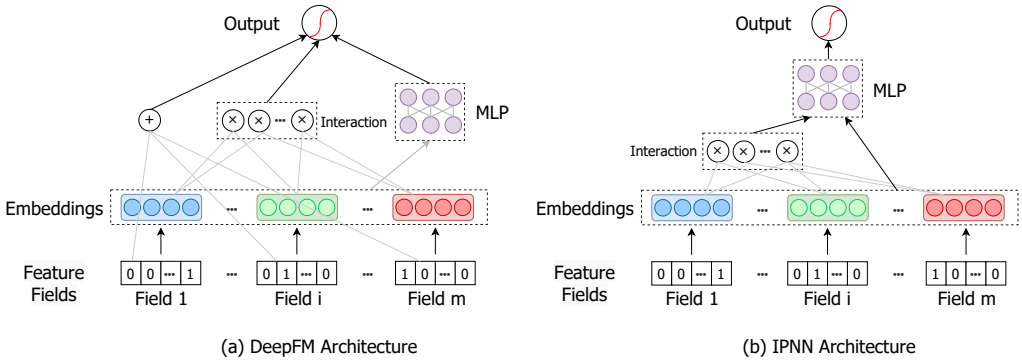

Figure 2: Architectures of DeepFM and IPNN.

vectors, and then embed them into continuous vectors using a field-wise embedding. In this way, a user-item interaction data example $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_m]$ can be represented as the concatenation of binary vectors from all feature fields:

$$\underbrace{[0,1,0,0,\ldots,0]}_{\boldsymbol{x}_1:\text{userid}}\underbrace{[\ 1,0\ ]}_{\boldsymbol{x}_2:\text{gender}}\underbrace{[0,1,0,0]}_{\boldsymbol{x}_3:\text{age}}\overbrace{\cdots\cdots}^{\text{other fields}}\underbrace{[0,1,0,1,\ldots,0]}_{\boldsymbol{x}_m:\text{itemid}}$$

where m is the number of feature fields and $\boldsymbol{x}_i$ is the binary vector of the i^{th} field. The categorical data are transformed into binary vectors via one-hot encoding, e.g., $[0, 1]$ for *gender* = *Female* and $[1, 0]$ for *gender* = *Male*. The numeric data are first partitioned into buckets, and then we have a binary vector for each bucket, e.g., we can use $[0, 0, 0, 1]$ for child whose *age*∈[0,14], $[0, 0, 1, 0]$ for youth whose *age*∈[15,24], $[0, 1, 0, 0]$ for adult whose *age*∈[25,64], and $[1, 0, 0, 0]$ for seniors whose *age*≥65.

Since vector $\boldsymbol{x}$ is high-dimensional and very sparse, and different feature fields have various lengths, DRS models usually introduce an embedding layer to transform each binary vector $\boldsymbol{x}_i$ into a low-dimensional continuous vector as:

$$\boldsymbol{e}_i = \boldsymbol{v}_i \boldsymbol{x}_i \tag{1}$$

where $\boldsymbol{v}_i \in R^{d \times u_i}$ is the weight matrix with u_i the number of unique feature values in the i^{th} feature field, and d is the pre-defined size of low-dimensional vectors[1]. Finally, the embedding layer will output the concatenation of embedding vectors from all feature fields:

$$\boldsymbol{E} = [\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_m] \tag{2}$$

2.2.2 ***Interaction Layer.*** After representing the input features as low-dimensional embeddings, DRS models usually develop an interaction layer to explicitly capture the interactions among feature fields. The most widely used method is factorization machine (FM) [42]. In addition to the linear interactions among features, FM can explicitly model the pairwise (second-order) feature interactions via the inner product of feature embeddings:

$$[\langle \boldsymbol{e}_1, \boldsymbol{e}_2 \rangle, \langle \boldsymbol{e}_1, \boldsymbol{e}_3 \rangle, \ldots, \langle \boldsymbol{e}_{m-1}, \boldsymbol{e}_m \rangle] \tag{3}$$

where $\langle \cdot, \cdot \rangle$ is the inner product of two embeddings, and the number of pairwise feature interactions is C_m^2. Then, the interaction layer will output:

$$l_{fm} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{i=1}^{m} \sum_{j>i}^{m} \langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle \tag{4}$$

Where $\boldsymbol{w}$ is the weight over the binary vector $\boldsymbol{x}$ of input features. The first term represents the impact of first-order feature interactions, and the second term reflects the impact of second-order feature interactions. FM can explicitly model even higher order interactions, such as $\sum_{i=1}^{m} \sum_{j>i}^{m} \sum_{t>j}^{m} \langle \boldsymbol{e}_i, \boldsymbol{e}_j, \boldsymbol{e}_t \rangle$ for third-order, but this will add a lot of computation.

2.2.3 ***MLP Layer.*** MLP Layer combines and transforms the features, e.g., $\boldsymbol{E}$ and l_{fm}, with several fully-connected layers and activations. The output of each layer is:

$$\boldsymbol{h}_{l+1} = \text{relu}\left(\boldsymbol{W}_l \boldsymbol{h}_l + \boldsymbol{b}_l\right) \tag{5}$$

where $\boldsymbol{W}_l$ is the weight matrix and $\boldsymbol{b}_l$ is the bias vector for the l^{th} hidden layer. $\boldsymbol{h}_0$ is the input of first fully-connected layer, and we denote the final output of MLP layer as MLP($\boldsymbol{h}_0$).

2.2.4 ***Output Layer.*** Finally, the output layer, which is subsequent to the previous layers, will generate the prediction $\hat{\boldsymbol{y}}$ of a user-item interaction data example. The input $\boldsymbol{h}_{out}$ of output layer can be different in different DRS models, e.g., $\boldsymbol{h}_{out} = [l_{fm}$+MLP($\boldsymbol{E}$)] in DeepFM [16] and $\boldsymbol{h}_{out}$ = MLP($l_{fm}, \boldsymbol{E}$) in IPNN [39], shown in Figure 2. The output layer will yield the prediction $\hat{\boldsymbol{y}}$ of the user-item interaction as:

$$\hat{\boldsymbol{y}} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}_{out} + \boldsymbol{b}_o\right) \tag{6}$$

where $\boldsymbol{W}_o$ and $\boldsymbol{b}_o$ are the weight matrix and bias vector for the output layer. Activation function $\sigma(\cdot)$ is selected based on different recommendation tasks, such as *sigmoid* for binary classification [16], and *softmax* for multi-class classification [46]. Finally, given a set of candidate loss functions, such as mean-squared-error, categorical hinge and cross-entropy, we can compute the candidate losses $\mathcal{L}_C$:

$$\mathcal{L}_C = [\ell_1(\boldsymbol{y}, \hat{\boldsymbol{y}}), \ell_2(\boldsymbol{y}, \hat{\boldsymbol{y}}), \cdots, \ell_n(\boldsymbol{y}, \hat{\boldsymbol{y}})] \tag{7}$$

where $\boldsymbol{y}$ is the ground truth label and n is the number of candidate loss functions.

2.3 Loss Function Search

AutoLoss aims to adaptively and automatically search the optimal loss function, which can enhance the prediction quality and training efficiency of the DRS network. This is naturally challenging

[1]For multi-valued features (e.g.,"Interest=Movie, Sports"), the feature embedding is the sum or average of multiple embeddings [5].

because of the complex relationship between the DRS parameters and the probabilities over candidate loss functions. To address this challenge, many existing works have focused on developing the fusing strategies for multiple loss functions, which can take advantage of multiple loss functions in a weighted sum manner:

$$\begin{aligned} \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\alpha}) &= \sum_{i=1}^{n} \alpha_i \cdot \ell_i(\boldsymbol{y}, \hat{\boldsymbol{y}}) \\ \text{s.t.} \sum_{i=1}^{n} \alpha_i &= 1, \quad \alpha_i > 0 \ \forall i \in [1, n] \end{aligned} \tag{8}$$

where $\boldsymbol{y}$ is the ground truth, $\hat{\boldsymbol{y}}$ is the prediction from DRS network, and ℓ_i is the i^{th} candidate loss function. The continuous loss weights $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \cdots, \alpha_n]$ measure the candidates' contributions in the final loss function $\mathcal{L}$. However, this method relies on exhaustively or manually search of loss weights from a large search space, which is extremely costly. Also, this soft fusing strategy cannot completely eliminate the impact of suboptimal candidate loss functions on the final loss function $\mathcal{L}$, thus, a hard selection method is desired. However, hard selection usually leads to the training framework not end-to-end differentiable.

Reinforcement learning (RL) is a potential solution to tackle the hard selection problem. However, since the RL is generally built upon the Markov decision process, it utilizes temporal-difference to make sequential actions. Consequently, the agent can only receive the reward until the optimal loss function is selected and the DRS is evaluated. In other words, the temporal-difference setting can suffer from delayed rewards. To address this issue, we introduce the Gumbel-softmax operation to simulate the hard selection over candidate loss functions, where the non-differentiable sampling is approximated from a categorical distribution based on a differentiable sampling from the Gumbel-softmax distribution [18].

Given the continuous loss weights $[\alpha_1, \cdots, \alpha_n]$ over candidate loss functions, we can draw a hard selection z through the Gumbel-max trick [13] as:

$$z = \text{one\_hot}\left(\arg\max_{i \in [1,n]} \left[\log \alpha_i + g_i\right]\right) \tag{9}$$

where $g_i = -\log(-\log(u_i))$ and $u_i \sim Uniform(0, 1)$. The independent and identically distributed (i.i.d) *gumbel noises* $\{g_i\}$ disturb the $\{\log \alpha_i\}$ terms. Also, they make the arg max operation equivalent to drawing a sample from loss weights $\alpha_1, \cdots, \alpha_n$. However, because of the arg max operation, this sampling method is non-differentiable. We tackle this problem by straight-through Gumbel-softmax [18], which leverages a softmax function as a differentiable approximation to the arg max operation:

$$p_i = \frac{\exp\left(\left(\log\left(\alpha_i\right) + g_i\right)/\tau\right)}{\sum_{j=1}^{n} \exp\left(\left(\log\left(\alpha_j\right) + g_j\right)/\tau\right)}, \quad \forall i \in [1, n] \tag{10}$$

where p_i is the probability of selecting the i^{th} candidate loss function. The temperature parameter τ is introduced to manage the smoothness of the Gumbel-softmax operation's output. Specifically, the output approaches a one-hot vector if τ is closer to zero. Then the final loss function $\mathcal{L}$ can be reformulated as:

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{p}) = \sum_{i=1}^{n} p_i \cdot \ell_i(\boldsymbol{y}, \hat{\boldsymbol{y}}) \tag{11}$$

In conclusion, the loss function search process becomes end-to-end differentiable by introducing the Gumbel-softmax operation with a similar hard selection performance. Next, we will discuss how to generate data example-level loss weights $[\alpha_1, \cdots, \alpha_n]$.

2.4 The Controller Network

As in Eq. (8), we suppose that $[\alpha_1, \cdots, \alpha_n]$ are the original (continuous) class probabilities over n candidate loss functions before the Gumbel-softmax operation. This assumption aims to learn a set of unified and static probabilities over the candidate loss functions. However, the environment of real-world commercial recommendation platforms is always non-stationary, and different user-item interaction examples have varying convergence behaviors. This cannot be handled by unified and static probabilities, resulting in suboptimal model performance, generalizability and transferability.

We propose a controller network to address this challenge, which learns to generate original class probabilities for each data example. Motivated by curriculum learning [1, 19], the original class probabilities should be generated according to the ground truth labels $\boldsymbol{y}$ and the output of DRS network $\hat{\boldsymbol{y}}$. Therefore, the input of the controller network is a mini-batch $\boldsymbol{(y, \hat{y})}$, followed by the MLP layer with several fully-connected layers like Eq. (5). Afterwards, the controller's output layer generates continuous class probabilities $[\alpha_1^b, \cdots, \alpha_n^b]$ $\forall b \in [1, B]$ for each data example in the mini-batch via a standard *softmax* activation, where B is the size of mini-batch. In other word, each data example has individual probabilities. The controller can enhance the recommendation quality, model generalizability and transferability, which is validated by the extensive experiments.

2.5 An Optimization Method

In above subsections, we formulate the loss function search as an architectural optimization problem and introduce the Gumbel-softmax that makes the framework end-to-end differentiable. Now, we discuss the optimization for the AutoLoss framework.

In AutoLoss, the parameters to be optimized are from two networks. We denote the main DRS network's parameters as $\boldsymbol{W}$, and the controller network's parameters as $\boldsymbol{V}$. Note that $\boldsymbol{p}$ are directly generated by the Gumbel-softmax operation based on the controller's output $\boldsymbol{\alpha}$ as in Eq. (10). Inspired by automated machine learning techniques [37], $\boldsymbol{W}$ and $\boldsymbol{V}$ should not be updated on the same training data batch like traditional supervised learning methods. This is because the optimization of them is highly dependent on each other. As a result, updating $\boldsymbol{W}$ and $\boldsymbol{V}$ on the same training batch can lead to the model over-fitting on the training examples.

According to the end-to-end differentiable property of AutoLoss, we update $\boldsymbol{W}$ and $\boldsymbol{V}$ through gradient descent utilizing the differentiable architecture search (DARTS) techniques [29]. To be specific, $\boldsymbol{W}$ and $\boldsymbol{V}$ are alternately updated on training and validation batches by minimizing the training loss $\mathcal{L}_{train}$ and validation loss $\mathcal{L}_{val}$, respectively. This forms a bi-level optimization problem [37], where controller parameters $\boldsymbol{V}$ and DRS parameters $\boldsymbol{W}$ are considered as the upper- and lower-level variables:

$$\begin{aligned} &\min_{\boldsymbol{V}} \mathcal{L}_{val}(\boldsymbol{W}^*(\boldsymbol{V}), \boldsymbol{V}) \\ &s.t.\ \boldsymbol{W}^*(\boldsymbol{V}) = \arg\min_{\boldsymbol{W}} \mathcal{L}_{train}(\boldsymbol{W}, \boldsymbol{V}^*) \end{aligned} \tag{12}$$

Algorithm 1 An Optimization Algorithm for AutoLoss via DARTS.

Input: features $\boldsymbol{x}$ and ground-truth labels $\boldsymbol{y}$
Output: well-learned parameters $\boldsymbol{W}^*$ and $\boldsymbol{V}^*$

1: **while** not converged **do**
2: Sample a mini-batch of validation data examples
3: Estimate the approximation of $\boldsymbol{W}^*(\boldsymbol{V})$ via Eq.(13)
4: Update $\boldsymbol{V}$ by descending $\nabla_{\boldsymbol{V}} \mathcal{L}_{val}(\boldsymbol{W}^*(\boldsymbol{V}), \boldsymbol{V})$
5: Sample a mini-batch of training data examples
6: Update $\boldsymbol{W}$ by descending $\nabla_{\boldsymbol{W}} \mathcal{L}_{train}(\boldsymbol{W}, \boldsymbol{V})$
7: **end while**

where directly optimizing $\boldsymbol{V}$ thoroughly via Eq.(12) is intractable since the inner optimization of $\boldsymbol{W}$ is extremely costly. To tackle this issue, we use an approximation scheme for the inner optimization:

$$\boldsymbol{W}^*(\boldsymbol{V}) \approx \boldsymbol{W} - \xi \nabla_{\boldsymbol{W}} \mathcal{L}_{train}(\boldsymbol{W}, \boldsymbol{V}) \tag{13}$$

where ξ is the predefined learning rate. This approximation scheme estimates $\boldsymbol{W}^*(\boldsymbol{V})$ by descending only one step toward the gradient $\nabla_{\boldsymbol{W}} \mathcal{L}_{train}(\boldsymbol{W}, \boldsymbol{V})$, rather than optimizing $\boldsymbol{W}(\boldsymbol{V})$ thoroughly. To further enhance the computation efficiency, we can set $\xi = 0$, i.e., the first-order approximation.

We detail the AutoLoss optimization via DARTS in Algorithm 1. More specifically, in each iteration, we first sample a mini-batch validation data examples of user-item interactions (line 2); next, we estimate (but do not update) $\boldsymbol{W}^*(\boldsymbol{V})$ via the approximation scheme in Eq.(13) (line 3); then, we update the controller parameters $\boldsymbol{V}$ by one step based on the estimation (line 4); afterward, we sample a mini-batch training data examples (line 5); and finally, we update the $\boldsymbol{W}$ via descending $\nabla_{\boldsymbol{W}} \mathcal{L}_{train}(\boldsymbol{W}, \boldsymbol{V})$ by one step (line 6).

3 EXPERIMENT

This section will conduct extensive experiments using various datasets to evaluate the effectiveness of AutoLoss. We first introduce the experimental settings, then compare AutoLoss with representative baselines, and finally conduct model component and transferability analysis.

3.1 Datasets

We evaluate our model on two datasets, including Criteo and ML-20m. Below we introduce these datasets and more statistics about them can be found in Table 1.

Criteo[2]: It is a real-world commercial dataset to assess click-through rate prediction models for online ads. It consists of 45 million data examples, i.e., users' click records on displayed ads. Each example contains $m = 39$ anonymous feature fields, where 13 fields are numerical and 26 fields are categorical. 13 numerical fields are converted into categorical features through bucketing.

ML-20m[3]: This is a benchmark dataset to evaluate recommendation algorithms, which contains 20 million users' 5-star ratings on movies. The dataset includes 27,278 movies and 138,493 users, i.e., $m = 2$ feature fields, where each user has at least 20 ratings.

[2]https://www.kaggle.com/c/criteo-display-ad-challenge/
[3]https://grouplens.org/datasets/movielens/20m/

Table 1: Statistics of the datasets.

Data	Criteo	ML-20m
# Interactions	45,840,617	20,000,263
# Feature Fields	39	2
# Feature Values	1,086,810	165,771
# Behavior	click or not	rating 1~5

3.2 Evaluation Metrics

AutoLoss is general for many recommendation tasks. To evaluate its effectiveness, we conduct *binary classification* (i.e., click-through rate prediction) on Criteo, and *multi-class classification* (i.e., 5-star ratings) on ML-20m. The two classification experiments are evaluated by AUC[4] and Logloss, where higher AUC or lower Logloss mean better performance. It is worth noting that slightly higher AUC and lower Logloss at 0.001-level are considered significant in recommendations [16].

3.3 Implementation

We implement AutoLoss based on a public library[5], which contains 16 representative recommendation models. We develop AutoLoss as an independent class, so we can easily apply our framework for all these models. In this paper, we only show the results on DeepFM [16] and IPNN [39] due to the page limitation. To be specific, AutoLoss framework mainly contains two networks, i.e., the DRS network and the controller network.

For the DRS network, (a) *Embedding layer*: we set the embedding size as 16 following the existing works [64]. (b) *Interaction layer*: we leverage factorization machine and inner product network to capture the interactions among feature fields for DeepFM and IPNN, respectively. (c) *MLP layer*: we have two fully-connected layers, and the layer size is 128. We also employ batch normalization, dropout ($rate = 0.2$) and ReLU activation for both layers. (d) *Output layer*: original DeepFM and IPNN are designed for click-through rate prediction, which use *sigmoid* activation for negative log-likelihood function. To fit the 5-class classification task on ML-20m, we modify the output layer correspondingly. i.e., the output layer is 5-dimensional with *softmax* activation.

For the controller network, (a) *Input layer*: the inputs are the ground truth labels $\boldsymbol{y}$ and the predictions $\hat{\boldsymbol{y}}$ from DRS network. (b) *MLP layer*: we also use two fully-connected layers with the layer size 128, batch normalization, dropout ($rate = 0.2$) and ReLU activation. (3) *Output layer*: the controller network will output continuous loss probabilities $\boldsymbol{\alpha}$ with *softmax* activation, whose dimension equals to the number of candidate loss functions.

For other hyper-parameters, (a) *Gumbel-softmax*: we use an annealing scheme for temperature $\tau = \max(0.01, 1 - 0.00005 \cdot t)$, where t is the training step. (b) *Optimization*: we set the learning rate as 0.001 for updating both DRS network and controller network with Adam optimizer and batch-size 2000. (c) We select the hyper-parameters of the AutoLoss framework via cross-validation, and we also do parameter-tuning for baselines correspondingly for a fair comparison.

[4]We evaluate the AUC for multiclass classification in a one-vs-rest manner.
[5]https://github.com/rixwew/pytorch-fm

Table 2: Performance comparison of different loss function search methods.

Dataset	Model	Metric	Methods								
			Focal	KL	Hinge	CE	MeLU	BOHB	DARTS	SLF	AutoLoss
Criteo	DeepFM	AUC ↑	0.8046	0.8042	0.8049	0.8056	0.8063	0.8065	0.8067	0.8081	**0.8092***
		Logloss ↓	0.4466	0.4469	0.4463	0.4457	0.4436	0.4435	0.4433	0.4426	**0.4416***
Criteo	IPNN	AUC ↑	0.8077	0.8072	0.8079	0.8085	0.8090	0.8092	0.8093	0.8098	**0.8108***
		Logloss ↓	0.4435	0.4437	0.4432	0.4428	0.4423	0.4422	0.4423	0.4418	**0.4409***
ML-20m	DeepFM	AUC ↑	0.7681	0.7682	0.7685	0.7692	0.7695	0.7695	0.7696	0.7705	**0.7717***
		Logloss ↓	1.2320	1.2317	1.2316	1.2310	1.2307	1.2305	1.2305	1.2299	**1.2288***
ML-20m	IPNN	AUC ↑	0.7721	0.7722	0.7725	0.7733	0.7735	0.7734	0.7736	0.7745	**0.7756***
		Logloss ↓	1.2270	1.2269	1.2266	1.2260	1.2256	1.2257	1.2255	1.2249	**1.2236***

"*" indicates the statistically significant improvements (i.e., two-sided t-test with $p < 0.05$) over the best baseline.
↑: the higher the better; ↓: the lower the better.

3.4 Overall Performance Comparison

AutoLoss is compared with the following loss function design and search methods:

- Fixed loss function: the first group of baselines leverages a predefined and fixed loss function. We utilize Focal loss, KL divergence, Hinge loss and cross-entropy (CE) loss for both classification tasks.
- Fixed weights over loss functions: this group of baselines aims to learn fixed weights over the loss functions in the first group, without considering the difference among data examples. In this group, we use the meta-learning method MeLU [24], as well as automated machine learning methods BOHB [7] and DARTS [29].
- Data example-wise loss weights: this group learns to assign different loss weights for different data examples according to their convergence behaviors. One existing work, stochastic loss function (SLF) [31], belongs to this group.

The overall performance is shown in Table 2. It can be observed that: (i) The first group of baselines achieves the worst recommendation performance in both recommendation tasks. Their optimizations are based on predefined and fixed loss functions during the training stage. This result demonstrates that leveraging a predefined and fixed loss function throughout can downgrade the recommendation quality. (ii) The methods in the second group outperform those in the first group. These methods try to learn weights over candidate loss functions according to their contributions to the optimization, and then combine them in a weighted sum manner. This validates that incorporating multiple loss functions in optimization can enhance the performance of deep recommender systems. (iii) The second group performs worse than the SLF, since the weights they learned are unified and static, which completely overlooks the various behaviors among different data examples. Therefore, SLF performs better by taking this factor into account. (iv) The decision network of SLF is optimized on the same training batch with the main DRS network via back-propagation, which can lead to over-fitting on the training batch. AutoLoss updates the DRS network on the training batch while updating the controller on the validation batch, which improves the model generalizability and results in better recommendation performance.

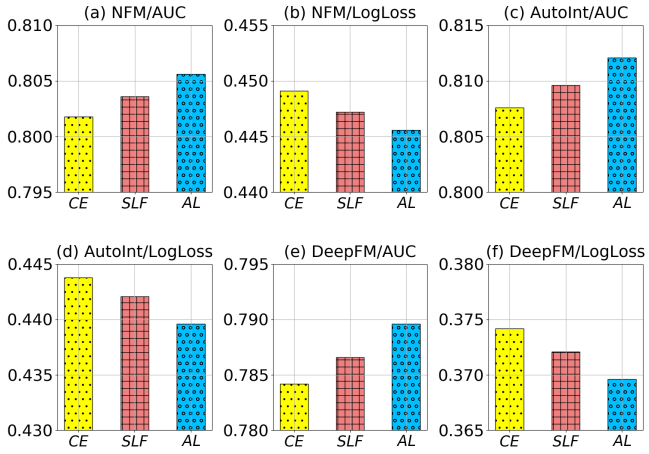

Figure 3: Transferability study results.

To summarize, AutoLoss achieves significantly better performance than state-of-the-art baselines on both datasets and tasks, which demonstrates its effectiveness.

3.5 Transferability Study

In this subsection, we study the transferability of the controller. Specifically, we want to investigate (i) whether the controller trained with one DRS model can be applied to other DRS models; and (ii) whether the controller learned on one dataset can be directly used on other datasets.

To study the transferability across different DRS models, we leverage the controller trained via DeepFM and AutoLoss on Criteo, fix its parameters and apply it to train NFM [17] and AutoInt [45] on Criteo. The results are demonstrated in Figure 3 (a)-(d), where (i) "*CE*" means that we directly train the new DRS model via minimizing the cross-entropy (CE) loss, which is the best single and fixed loss function in Table 2; (ii) "*SLF*" is that we use the controller upon DeepFM and SLF, which is the best baseline in Table 2; and (iii) "*AL*" denotes that we use the controller based on DeepFM and AutoLoss. From the figures, we can observe that *SLF* performs superior to

Table 3: Impact of model components.

Dataset	Model	Metric	Methods		
			AL-1	AL-2	AutoLoss
Criteo	DeepFM	AUC ↑	0.8052	0.8083	**0.8092***
		Logloss ↓	0.4460	0.4422	**0.4416***
Criteo	IPNN	AUC ↑	0.8081	0.8102	**0.8108***
		Logloss ↓	0.4431	0.4416	**0.4409***

"*" indicates the statistically significant improvements (i.e., two-sided t-test with $p < 0.05$) over the best baseline.
↑: the higher the better; ↓: the lower the better.

CE, which indicates that a pre-trained controller can improve other DRS models' training performance. More importantly, *AL* outperforms *SLF*, which validates AutoLoss's better transferability across different DRS models.

To study the transferability between different datasets, we train a controller upon Criteo dataset with DeepFM and AutoLoss, fix its parameters and apply it to train a new DeepFM on the Avazu dataset[6], i.e., "*AL*". Also, we denote that (i) "*CE*": DeepFM is directly optimized by minimizing cross-entropy (CE) loss on Avazu dataset; and (ii) "*SLF*": DeepFM is optimized on the new dataset with the assistance of a controller pre-trained with DeepFM+SLF on Criteo. In Figure 3 (e)-(f), *AL* shows superior performance over *CE* and *SLF*, which proves its better transferability between different datasets.

In summary, AutoLoss has better transferability across different DRS models and different recommendation datasets, which improves its usability in real-world recommender systems.

3.6 Impact of Model Components

In this subsection, in order to understand the contributions of important model components of AutoLoss, we systematically eliminate each component and define the following variants:

- **AL-1**: This variant aims to assess the contribution of the controller. Thus, we assign equivalent weights on four candidate loss functions, i.e., $[0.25, 0.25, 0.25, 0.25]$.
- **AL-2**: In this variant, we eliminate the Gumbel-softmax operation, and directly use the controller's output, i.e., the continuous loss probabilities $\boldsymbol{\alpha}$ from standard *softmax* activation, which aims to evaluate the impact of Gumbel-softmax.

The results on the Criteo dataset are shown in Table 3. First, AL-1 has worse performance than AutoLoss, which validates the necessity to introduce the controller network. It is noteworthy that, AL-1 performs worse than all loss function search methods, and even the fixed cross-entropy (CE) loss in Table 2, which indicates that equally incorporating all candidate loss functions cannot guarantee better performance. Second, AutoLoss outperforms AL-2. The main reason is that AL-2 always generates gradients based on all the loss functions, which introduces some noisy gradients from the suboptimal candidate loss functions. In contrast, AutoLoss can obtain appropriate gradients by filtering out suboptimal loss functions via Gumbel-softmax, which enhances the model robustness.

[6]Avazu is another benchmark dataset for CTR prediction, which contains 40 million user clicking behaviors in 11 days with $M = 22$ categorical feature fields. https://www.kaggle.com/c/avazu-ctr-prediction/

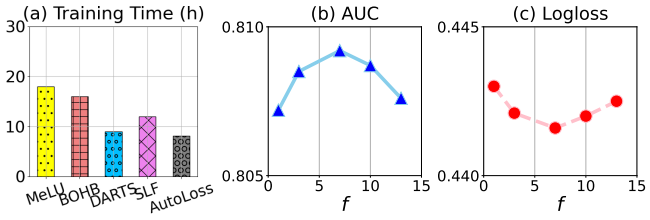

Figure 4: Efficiency study results.

3.7 Efficiency Study

This section compares AutoLoss's training efficiency with other loss function searching methods, which is an important metric to deploy a DRS model in real-world applications. Our experiments are based on one GeForce GTX 1060 GPU.

The results of DeepFM on Criteo dataset are illustrated in Figure 4 (a). We can observe that AutoLoss achieves the fastest training speed. The reasons are two-fold. First, AutoLoss can generate the most appropriate gradients to update DRS, which increases the optimization efficiency. Second, we update the controller once after every 7 times DRS is updated, i.e., the controller updating frequency $f = 7$. This trick not only reduces the training time ($\sim 60\%$ in this case) with fewer computations, but also enhances the performance. In Figure 4 (b)-(c) where x-axis is f, we find that DeepFM performs the best when $f = 7$, while updating too frequently/infrequently can lead to suboptimal AUC/Logloss.

To summarize, AutoLoss can efficiently achieve better performance, making it easier to be launched in real-world recommender systems.

4 RELATED WORK

In this section, we briefly introduce the works related to our study. We first go over the latest studies in loss function search and then review works about AutoML for recommendations.

4.1 Loss Function Search

The loss function plays an essential part in a deep learning framework. The choice of the loss function significantly affects the performance of the learned model. A lot of efforts have been made to design desirable loss functions for specific tasks. For example, in the field of image processing, Rahman and Wang [40] argued that the typical cross-entropy loss for semantic segmentation shows great limitations in aligning with evaluation metrics other than global accuracy. Ronneberger et al. [43], Wu et al. [50] designed loss functions by taking class frequency into consideration to cater to the mIoU metric. Caliva et al. [2], Qin et al. [38] designed losses with larger weights at boundary regions to improve the boundary F1 score. Liu et al. [33] proposed to replace the traditional Softmax loss with large margin Softmax (L-Softmax) loss to improve feature discrimination in classification tasks. Fan et al. [9] used sphere Softmax loss for the person re-identification task and obtained state-of-the-art results. The loss functions mentioned above are all designed manually, requiring ample expert knowledge, non-trivial time, and many human efforts.

Recently, automated loss function search draws increasing interests of researchers from various machine learning (ML) fields. Xu

et al. [52] investigated how to automatically schedule iterative and alternate optimization processes for ML models. A meta-learning framework was proposed to adaptively determine which loss function to use and which parameters to update at each optimization step. Liu and Lai [31] proposed to optimize the stochastic loss function (SLF), where the loss function of an ML model was dynamically selected. The loss function selection is determined by loss parameters, including a selective binary code and a weighting coefficient. During training, model parameters and the loss parameters are learned jointly. Li et al. [26] proposed automatically searching specific surrogate losses to improve different evaluation metrics in the image semantic segmentation task. Besides, Li et al. [25], Wang et al. [48] designed search spaces for a series of existing loss functions and developed algorithms to search for the best parameters of the probability distribution for sampling loss functions. However, their methods are designed exclusively for cross-entropy loss and its variants, making their methods not applicable in our tasks.

4.2 AutoML for Recommendation

AutoML techniques are now widely used to automatically design deep recommendation systems. Previous works mainly focused on the design of the embedding layer and the selection of feature interaction patterns.

In terms of the embedding layer, Joglekar et al. [20], Liu et al. [32], Zhao et al. [56] proposed novel methods to automatically select the best embedding size for different feature fields in a recommendation system. Liu et al. [30], Zhao et al. [57] proposed to dynamically search embedding sizes for users and items based on their popularity in the streaming setting. Similarly, Ginart et al. [12] proposed to use mixed dimension embeddings for users and items based on their query frequency. Kang et al. [21] proposed a multi-granular quantized embeddings (MGQE) technique to learn impact embeddings for infrequent items. Cheng et al. [4] proposed to perform embedding dimension selection with a soft selection layer, making the dimension selection more flexible. Guo et al. [14] focused on the embeddings of numerical features. They proposed AutoDis, which automatically discretizes features in numerical fields and maps the resulting categorical features into embeddings.

As for feature interaction, Luo et al. [35] proposed AutoCross that produces high-order cross features by performing beam search in a tree-structure feature space. Khawar et al. [22], Liu et al. [28], Song et al. [44], Xue et al. [53] proposed to automatically discover feature interaction architectures for click-through rate (CTR) prediction. Tsang et al. [47] proposed a method to interpret the feature interactions from a source recommendation model and apply them in a target recommendation model.

To the best of our knowledge, we are the first to investigate the automated loss function search for deep recommendation systems.

5 CONCLUSION

We propose a novel end-to-end framework, AutoLoss, to enhance recommendation performance and deep recommender systems' training efficiency by selecting appropriate loss functions in a data-driven manner. AutoLoss can automatically select the proper loss function for each data example according to their varied convergence behaviors. To be specific, we first develop a novel controller network, which generates continuous loss weights based on the ground truth labels and the DRS' predictions. Then, we introduce a Gumbel-softmax operation to simulate the hard selection over candidate loss functions, which filters out the noisy gradients from suboptimal candidates. Finally, we can select the optimal candidate according to the output from Gumbel-softmax. We conduct extensive experiments to validate the effectiveness of AutoLoss on two widely used benchmark datasets. The results show that our framework can improve recommendation performance and training efficiency with excellent transferability.

ACKNOWLEDGEMENTS

This work is supported by National Science Foundation (NSF) under grant numbers IIS1907704, IIS1928278, IIS1714741, IIS1715940, IIS1845081, CNS1815636, and an internal research fund from the Hong Kong Polytechnic University (project no. P0036200).

REFERENCES

[1] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*. 41–48.
[2] Francesco Caliva, Claudia Iriondo, Alejandro Morales Martinez, Sharmila Majumdar, and Valentina Pedoia. 2019. Distance map loss penalty term for semantic segmentation. *arXiv preprint arXiv:1908.03679* (2019).
[3] Samprit Chatterjee and Ali S Hadi. 2015. *Regression analysis by example*. John Wiley & Sons.
[4] Weiyu Cheng, Yanyan Shen, and Linpeng Huang. 2020. Differentiable Neural Input Search for Recommender Systems. *arXiv preprint arXiv:2006.04466* (2020).
[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.
[6] Ludwig Fahrmeir, Thomas Kneib, Stefan Lang, and Brian Marx. 2007. *Regression*. Springer.
[7] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*. PMLR, 1437–1446.
[8] Wenqi Fan, Tyler Derr, Xiangyu Zhao, Yao Ma, Hui Liu, Jianping Wang, Jiliang Tang, and Qing Li. 2020. Attacking Black-box Recommendations via Copying Cross-domain User Profiles. *arXiv preprint arXiv:2005.08147* (2020).
[9] Xing Fan, Wei Jiang, Hao Luo, and Mengjuan Fei. 2019. Spherereid: Deep hypersphere manifold embedding for person re-identification. *Journal of Visual Communication and Image Representation* 60 (2019), 51–58.
[10] Weihao Gao, Xiangjun Fan, Jiankai Sun, Kai Jia, Wenzhi Xiao, Chong Wang, and Xiaobing Liu. 2020. Deep Retrieval: An End-to-End Learnable Structure Model for Large-Scale Recommendations. *arXiv preprint arXiv:2007.07203* (2020).
[11] Yingqiang Ge, Shuchang Liu, Ruoyuan Gao, Yikun Xian, Yunqi Li, Xiangyu Zhao, Changhua Pei, Fei Sun, Junfeng Ge, Wenwu Ou, et al. 2021. Towards Long-term Fairness in Recommendation. *arXiv preprint arXiv:2101.03584* (2021).
[12] Antonio Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. 2019. Mixed Dimension Embeddings with Application to Memory-Efficient Recommendation Systems. *arXiv preprint arXiv:1909.11810* (2019).
[13] Emil Julius Gumbel. 1948. *Statistical theory of extreme values and some practical applications: a series of lectures*. Vol. 33. US Government Printing Office.
[14] Huifeng Guo, Bo Chen, Ruiming Tang, Zhenguo Li, and Xiuqiang He. 2020. AutoDis: Automatic Discretization for Embedding Numerical Features in CTR Prediction. *arXiv preprint arXiv:2012.08986* (2020).
[15] Hao Guo, Xin Li, Ming He, Xiangyu Zhao, Guiquan Liu, and Guandong Xu. 2016. CoSoLoRec: Joint Factor Model with Content, Social, Location for Heterogeneous Point-of-Interest Recommendation. In *International Conference on Knowledge Science, Engineering and Management*. Springer, 613–627.
[16] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. 1725–1731.
[17] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 355–364.
[18] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).
[19] Lu Jiang, Deyu Meng, Shoou-I Yu, Zhenzhong Lan, Shiguang Shan, and Alexander Hauptmann. 2014. Self-paced learning with diversity. In *Advances in Neural*

Information Processing Systems. 2078–2086.
[20] Manas R Joglekar, Cong Li, Mei Chen, Taibai Xu, Xiaoming Wang, Jay K Adams, Pranav Khaitan, Jiahui Liu, and Quoc V Le. 2020. Neural input search for large scale recommendation models. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2387–2397.
[21] Wang-Cheng Kang, Derek Zhiyuan Cheng, Ting Chen, Xinyang Yi, Dong Lin, Lichan Hong, and Ed H Chi. 2020. Learning Multi-granular Quantized Embeddings for Large-Vocab Categorical Features in Recommender Systems. *arXiv preprint arXiv:2002.08530* (2020).
[22] Farhan Khawar, Xu Hang, Ruiming Tang, Bin Liu, Zhenguo Li, and Xiuqiang He. 2020. AutoFeature: Searching for Feature Interactions and Their Architectures for Click-through Rate Prediction. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 625–634.
[23] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. 2019. Panoptic feature pyramid networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6399–6408.
[24] Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. 2019. MeLU: meta-learned user preference estimator for cold-start recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1073–1082.
[25] Chuming Li, Xin Yuan, Chen Lin, Minghao Guo, Wei Wu, Junjie Yan, and Wanli Ouyang. 2019. Am-lfs: Automl for loss function search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 8410–8419.
[26] Hao Li, Chenxin Tao, Xizhou Zhu, Xiaogang Wang, Gao Huang, and Jifeng Dai. 2020. Auto Seg-Loss: Searching Metric Surrogates for Semantic Segmentation. *arXiv preprint arXiv:2010.07930* (2020).
[27] 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*.
[28] Bin Liu, Chenxu Zhu, Guilin Li, Weinan Zhang, Jincai Lai, Ruiming Tang, Xiuqiang He, Zhenguo Li, and Yong Yu. 2020. AutoFIS: Automatic Feature Interaction Selection in Factorization Models for Click-Through Rate Prediction. *arXiv preprint arXiv:2003.11235* (2020).
[29] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).
[30] Haochen Liu, Xiangyu Zhao, Chong Wang, Xiaobing Liu, and Jiliang Tang. 2020. Automated Embedding Size Search in Deep Recommender Systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2307–2316.
[31] Qingliang Liu and Jinmei Lai. 2020. Stochastic Loss Function. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 4884–4891.
[32] Siyi Liu, Chen Gao, Yihong Chen, Depeng Jin, and Yong Li. 2021. Learnable Embedding Sizes for Recommender Systems. *arXiv preprint arXiv:2101.07577* (2021).
[33] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. 2016. Large-margin softmax loss for convolutional neural networks.. In *ICML*, Vol. 2. 7.
[34] Yiding Liu, Tuan-Anh Nguyen Pham, Gao Cong, and Quan Yuan. 2017. An experimental evaluation of point-of-interest recommendation in location-based social networks. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1010–1021.
[35] Yuanfei Luo, Mengshuo Wang, Hao Zhou, Quanming Yao, Wei-Wei Tu, Yuqiang Chen, Wenyuan Dai, and Qiang Yang. 2019. Autocross: Automatic feature crossing for tabular data in real-world applications. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1936–1945.
[36] Hanh TH Nguyen, Martin Wistuba, Josif Grabocka, Lucas Rego Drumond, and Lars Schmidt-Thieme. 2017. Personalized Deep Learning for Tag Recommendation. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer.
[37] 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*. 4095–4104.
[38] Xuebin Qin, Zichen Zhang, Chenyang Huang, Chao Gao, Masood Dehghan, and Martin Jagersand. 2019. Basnet: Boundary-aware salient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7479–7489.
[39] 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.
[40] Md Atiqur Rahman and Yang Wang. 2016. Optimizing intersection-over-union in deep neural networks for image segmentation. In *International symposium on visual computing*. Springer, 234–244.
[41] Logesh Ravi and Subramaniyaswamy Vairavasundaram. 2016. A collaborative location based travel recommendation system through enhanced rating prediction for the group of users. *Computational intelligence and neuroscience* 2016 (2016).
[42] Steffen Rendle. 2010. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 995–1000.
[43] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*. Springer, 234–241.
[44] Qingquan Song, Dehua Cheng, Hanning Zhou, Jiyan Yang, Yuandong Tian, and Xia Hu. 2020. Towards automated neural interaction discovery for click-through rate prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 945–955.
[45] 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.
[46] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. 17–22.
[47] Michael Tsang, Dehua Cheng, Hanpeng Liu, Xue Feng, Eric Zhou, and Yan Liu. 2020. Feature interaction interpretability: A case for explaining ad-recommendation systems via neural interaction detection. *arXiv preprint arXiv:2006.10966* (2020).
[48] Xiaobo Wang, Shuo Wang, Cheng Chi, Shifeng Zhang, and Tao Mei. 2020. Loss function search for face recognition. In *International Conference on Machine Learning*. PMLR, 10029–10038.
[49] Sai Wu, Weichao Ren, Chengchao Yu, Gang Chen, Dongxiang Zhang, and Jingbo Zhu. 2016. Personal recommendation using deep recurrent neural networks in NetEase. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*. IEEE, 1218–1229.
[50] Zifeng Wu, Chunhua Shen, and Anton van den Hengel. 2016. Bridging category-level and instance-level semantic image segmentation. *arXiv preprint arXiv:1605.06885* (2016).
[51] Yuwen Xiong, Renjie Liao, Hengshuang Zhao, Rui Hu, Min Bai, Ersin Yumer, and Raquel Urtasun. 2019. Upsnet: A unified panoptic segmentation network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8818–8826.
[52] Haowen Xu, Hao Zhang, Zhiting Hu, Xiaodan Liang, Ruslan Salakhutdinov, and Eric Xing. 2018. Autoloss: Learning discrete schedules for alternate optimization. *arXiv preprint arXiv:1810.02442* (2018).
[53] Niannan Xue, Bin Liu, Huifeng Guo, Ruiming Tang, Fengwei Zhou, Stefanos P Zafeiriou, Yuzhou Zhang, Jun Wang, and Zhenguo Li. 2020. AutoHash: Learning Higher-order Feature Interactions for Deep CTR Prediction. *IEEE Transactions on Knowledge and Data Engineering* (2020).
[54] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)* 52, 1 (2019), 1–38.
[55] Xiangyu Zhao, Changsheng Gu, Haoshenglun Zhang, Xiwang Yang, Xiaobing Liu, Hui Liu, and Jiliang Tang. 2021. DEAR: Deep Reinforcement Learning for Online Advertising Impression in Recommender Systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 750–758.
[56] Xiangyu Zhao, Haochen Liu, Hui Liu, Jiliang Tang, Weiwei Guo, Jun Shi, Sida Wang, Huiji Gao, and Bo Long. 2020. Memory-efficient Embedding for Recommendations. *arXiv preprint arXiv:2006.14827* (2020).
[57] Xiangyu Zhao, Chong Wang, Ming Chen, Xudong Zheng, Xiaobing Liu, and Jiliang Tang. 2020. AutoEmb: Automated Embedding Dimensionality Search in Streaming Recommendations. *arXiv preprint arXiv:2002.11252* (2020).
[58] Xiangyu Zhao, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2019. Toward Simulating Environments in Reinforcement Learning Based Recommendations. *arXiv preprint arXiv:1906.11462* (2019).
[59] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep Reinforcement Learning for Page-wise Recommendations. In *Proceedings of the 12th ACM Recommender Systems Conference*. ACM, 95–103.
[60] Xiangyu Zhao, Long Xia, Lixin Zou, Hui Liu, Dawei Yin, and Jiliang Tang. 2020. Whole-Chain Recommendations. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1883–1891.
[61] Xiangyu Zhao, Tong Xu, Qi Liu, and Hao Guo. 2016. Exploring the Choice Under Conflict for Social Event Participation. In *International Conference on Database Systems for Advanced Applications*. Springer, 396–411.
[62] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with Negative Feedback via Pairwise Deep Reinforcement Learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1040–1048.
[63] Xiangyu Zhao, Xudong Zheng, Xiwang Yang, Xiaobing Liu, and Jiliang Tang. 2020. Jointly learning to recommend and advertise. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3319–3327.
[64] Jieming Zhu, Jinyang Liu, Shuai Yang, Qi Zhang, and Xiuqiang He. 2020. FuxiCTR: An Open Benchmark for Click-Through Rate Prediction. *arXiv preprint arXiv:2009.05794* (2020).
[65] Lixin Zou, Long Xia, Yulong Gu, Xiangyu Zhao, Weidong Liu, Jimmy Xiangji Huang, and Dawei Yin. 2020. Neural Interactive Collaborative Filtering. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 749–758.