AutoLoss: Automated Loss Function Search in Recommendations

Xiangyu Zhao1,2, Haochen Liu2, Wenqi Fan3, Hui Liu2, Jiliang Tang2, Chong Wang4
1City University of Hong Kong, 2Michigan State University, 3The Hong Kong Polytechnic University, 4Bytedance
{zhaoxi35,liuhao1,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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

KDD ‘21, August 14–18, 2021, Virtual Event, Singapore
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8332-5/21/08...$15.00
https://doi.org/10.1145/3447548.3462708
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 provide 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 $\hat{y}$ based on the input features $x$. Then, we can calculate the losses $\{\ell_i\}$ for each candidate loss function according to the ground truth labels $y$ and predictions $\hat{y}$. Meanwhile, the controller network takes $(y, \hat{y})$ and outputs the probabilities $p$ over loss functions for each data example. Finally, the overall loss $L$ can be calculated according to the losses from $\{\ell_i\}$ and the probabilities $p$.

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 [30, 38]. 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, 28, 40, 43]. 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
vectors, and then embed them into continuous vectors using a field-wise embedding. In this way, a user-item interaction data example \( x = [x_1, x_2, \ldots, x_m] \) can be represented as the concatenation of binary vectors from all feature fields:

\[
[0, 1, 0, 0, \ldots, 0] [1, 0, 0, 1, 0, 0, \ldots, 0] [0, 1, 0, 1, 0, 1, 0, \ldots, 0]
\]

\( x_1: \) user id  \( x_2: \) gender  \( x_3: \) age  \( x_m: \) item id

where \( m \) is the number of feature fields and \( x_i \) 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 \( age \in [0,14)\), \([0, 1, 0, 1]\) for youth whose \( age \in [15, 24)\), \([0, 1, 0, 0]\) for adult whose \( age \in [25, 64]\), and \([1, 0, 0, 0]\) for seniors whose \( age \geq 65\).

Since vector \( x \) 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 \( x_i \) into a low-dimensional continuous vector as:

\[
e_i = u_i x_i
\]

where \( u_i \in \mathbb{R}^{d \times s_i} \) 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:

\[
E = [e_1, e_2, \ldots, e_m]
\]

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) \([43]\). 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:

\[
\{\langle e_1, e_2 \rangle, \langle e_1, e_3 \rangle, \ldots, \langle e_{m-1}, e_m \rangle\}
\]

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:

\[
I_{fm} = \langle w, x \rangle + \sum_{i=1}^{m} \sum_{j>i}^{m} \langle e_i, e_j \rangle
\]

Where \( w \) is the weight over the binary vector \( x \) 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 \( \sum_{i=1}^{m} \sum_{j>i}^{m} \sum_{j>j}^{m} \langle e_i, e_j, e_k \rangle \) 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., \( E \) and \( I_{fm} \), with several fully-connected layers and activations. The output of each layer is:

\[
h_{l+1} = \text{relu}(W_l h_l + b_l)
\]

where \( W_l \) is the weight matrix and \( b_l \) is the bias vector for the \( l^{th} \) hidden layer. \( h_0 \) is the input of first fully-connected layer, and we denote the final output of MLP layer as \( MLP(h_0) \).

2.2.4 Output Layer. Finally, the output layer, which is subsequent to the previous layers, will generate the prediction \( \hat{y} \) of a user-item interaction data example. The input \( h_{out} \) of output layer can be different in different DRS models, e.g., \( h_{out} = [I_{fm} + MLP(E)] \) in DeepFM \([16]\) and \( h_{out} = MLP(I_{fm}, E) \) in IPNN \([40]\), shown in Figure 2. The output layer will yield the prediction \( \hat{y} \) of the user-item interaction as:

\[
\hat{y} = \sigma(W_o h_{out} + b_o)
\]

where \( W_o \) and \( b_o \) 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 \([47]\). Finally, given a set of candidate loss functions, such as mean-squared-error, categorical hinge and cross-entropy, we can compute the candidate losses \( L_C \):

\[
L_C = [L_1(y, \hat{y}), L_2(y, \hat{y}), \ldots, L_n(y, \hat{y})]
\]

where \( y \) 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...
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:

\[
\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{\alpha}) = \sum_{i=1}^{n} \alpha_i \cdot \ell_i(\mathbf{y}, \hat{\mathbf{y}})
\]

\[
s.t. \sum_{i=1}^{n} \alpha_i = 1, \quad \alpha_i > 0 \quad \forall i \in [1, n]
\]

where \(\mathbf{y}\) is the ground truth, \(\hat{\mathbf{y}}\) is the prediction from DRS network, and \(\ell_i\) is the \(i^{th}\) candidate loss function. The continuous loss weights \(\mathbf{\alpha} = [\alpha_1, \alpha_2, \cdots, \alpha_n]\) 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 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:

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

where \(g_i = -\log (-\log (u_i))\) and \(u_i \sim \text{Uniform}(0,1)\). 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 introducing the Gumbel-softmax [18], which leverages a softmax function as a differentiable approximation to the arg max operation:

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

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

\[
\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{p}) = \sum_{i=1}^{n} p_i \cdot \ell_i(\mathbf{y}, \hat{\mathbf{y}})
\]

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 \(\mathbf{y}\) and the output of DRS network \(\hat{\mathbf{y}}\). Therefore, the input of the controller network is a mini-batch \((\mathbf{y}, \hat{\mathbf{y}})\), followed by the MLP layer with several fully-connected layers like Eq. (5). Afterwards, the controller’s output layer generates continuous class probabilities \([\alpha^b_1, \cdots, \alpha^b_n]\) 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 \(\mathbf{W}\), and the controller network’s parameters as \(\mathbf{V}\). Note that \(\mathbf{p}\) are directly generated by the Gumbel-softmax operation based on the controller’s output \(\mathbf{a}\) as in Eq. (10). Inspired by automated machine learning techniques [38], \(\mathbf{W}\) and \(\mathbf{V}\) 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 \(\mathbf{W}\) and \(\mathbf{V}\) 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 \(\mathbf{W}\) and \(\mathbf{V}\) through gradient descent utilizing the differentiable architecture search (DARTS) techniques [30]. To be specific, \(\mathbf{W}\) and \(\mathbf{V}\) are alternately updated on training and validation batches by minimizing the training loss \(\mathcal{L}_{\text{train}}\) and validation loss \(\mathcal{L}_{\text{val}}\), respectively. This forms a bi-level optimization problem [38], where controller parameters \(\mathbf{V}\) and DRS parameters \(\mathbf{W}\) are considered as the upper- and lower-level variables:

\[
\min_{\mathbf{V}} \mathcal{L}_{\text{val}}(\mathbf{W}^*(\mathbf{V}), \mathbf{V}) \quad s.t. \mathbf{W}^*(\mathbf{V}) = \arg \min_{\mathbf{W}} \mathcal{L}_{\text{train}}(\mathbf{W}, \mathbf{V}^*)
\]
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.

### 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\(^5\) 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 [40] 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 [65]. (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 \(y\) and the predictions \(\hat{y}\) 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 \(\alpha\) 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 \(T = \max(0.001, 1 - 0.00005 \cdot t)\), 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.

---

\(^2\)https://www.kaggle.com/c/criteo-display-ad-challenge/

\(^3\)https://grouplens.org/datasets/movielens/20m/

\(^5\)https://github.com/rixwew/pytorch-fm

\(^5\)We evaluate the AUC for multiclass classification in a one-vs-rest manner.
### 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 [25], as well as automated machine learning methods BOHB [7] and DARTS [30].
- **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) [32], 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 while updating the controller on the validation batch, which improves the model generalizability and results in better recommendation performance.

#### 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* |
|         | 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.

$\uparrow$: the higher the better; $\downarrow$: the lower the better.

![Figure 3: Transferability study results.](image)

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 [46] 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 |
|---------|-------|--------|---------|
| Criteo | DeepFM | AUC ↑ | AL-1 | 0.8052 | 0.8083 | 0.8092* |
|        |        | Logloss ↓ | AL-2 | 0.4460 | 0.4422 | 0.4416* |
| Criteo | IPNN   | AUC ↑ | AutoLoss | 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 dataset⁶, 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 systemically 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 $\alpha$ 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.

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 (~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 [41] 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. [44], Wu et al. [51] designed loss functions by taking class frequency into consideration to cater to the mIoU metric. Caliva et al. [2], Qin et al. [39] designed losses with larger weights at boundary regions to improve the boundary F1 score. Liu et al. [34] 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.

⁶Avazu 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/
Xu et al. [53] investigated how to automatically schedule iterative and alternate optimization processes for ML models. Liu and Lai [32] proposed to optimize the stochastic loss function (SLF), where the loss function of an ML model was dynamically selected. During training, model parameters and the loss parameters are learned jointly. Li et al. [27] proposed automatically searching specific surrogate losses to improve different evaluation metrics in the image semantic segmentation task. Jin et al. [20] composed multiple self-supervised learning tasks to jointly encode multiple sources of information and produce more generalizable representations, and developed two automated frameworks to search the task weights. Besides, Li et al. [26], Wang et al. [49] 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.

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. [21], Liu et al. [33], Zhao et al. [57] proposed novel methods to automatically select the best embedding size for different feature fields in a recommendation system. Liu et al. [31], Zhao et al. [58] 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. [22] 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. [36] proposed AutoCross that produces high-order cross features by performing beam search in a tree-structure feature space. Khawar et al. [23], Liu et al. [29], Song et al. [45], Xue et al. [54] proposed to automatically discover feature interaction architectures for click-through rate (CTR) prediction. Tsang et al. [48] 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 specify, 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] Youhua Bengio, Jérémie 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 Linqing 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 30th 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] Xiaobing Liu. 2020. Deep Retrieval: An End-to-End Learnable Structure Model for Point-of-Interest Recommendation. In Proceedings of the 26th International Joint Conference on Artificial Intelligence. 1725–1731.
[10] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. arXiv preprint arXiv:1701.07872 (2017).
[11] Xingnan 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.
[12] Antionio Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Hendler. 2019. Towards Long-term Fairness in Recommendation. arXiv preprint arXiv:1901.03854 (2021).
[13] 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).
[14] Emil Julius Gumbel. 1948. Statistical theory of extreme values and some practical applications: a series of lectures. Vol. 33. US Government Printing Office.
[15] Haofeng Guo, Ruiming Tang, Zhengu Li, and Xiuqiang He. 2020. AutoDis: Automatic Discretization for Embedding Numerical Features in CTR Prediction. arXiv preprint arXiv:2012.08986 (2020).
[16] Huijun Jin, Xiaorui Liu, Xiangyu Zhao, Yao Ma, Neil Shah, and Jiliang Tang. 2021. Automated Self-Supervised Learning for Graphs. arXiv:2106.05470 [cs.LG].
[21] Manas R Joglekar, Cong Li, Mei Chen, Taibai Xu, Xiaoming Wang, Jay K Adams, Pranav Khatian, 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.

[22] 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).

[23] Farhan Khawar, Xu Hang, Ruiming Tang, Bin Liu, Zhenqiu Li, and Xuqiang He. 2020. AutoFeature: Searching for Feature Interactions and Their Architectures for Click-Through Rate Prediction. In Proceedings of the 27th ACM International Conference on Information & Knowledge Management. 653–654.

[24] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. 2019. Panoptic feature pyramidal networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 6399–6408.

[25] Hoyee Lee, Jinbae Im, Seongwon Jang, Hyunsook Cho, and Seheon 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.

[26] Chuming Li, Xin Yuan, Chen Lin, Minghao Guo, Wei Wu, Junjie Yan, and Wanli Ouyang. 2019. A-Mils: Autolm for cold start recommendation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 8410–8419.

[27] Hao Li, Chenxun 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).

[28] Jianxun Lian, Xiaolun Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Nie, and Guangyong Sun. 2018. sde2cpp: Combining explicit and implicit feature interactions for recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.

[29] Bin Liu, Chenxu Zhu, Guolin Li, Weinan Zhang, Jiamei Lai, and Yong Yu. 2020. AutoFIS: Automatic Feature Interaction Selection in Factorization Models for Click-Through Rate Prediction. arXiv preprint arXiv:2003.11235 (2020).

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

[31] Haochen Liu, Xiangyu Zhao, Xudong Zheng, Xiaobing Liu, and Jiliang Tang. 2020. Higher-order Feature Interactions for Deep CTR Prediction. IEEE Transactions on Knowledge and Data Engineering.

[32] Haowen Xu, Hao Zhang, Zhiheng Hu, Xiaodan Liang, Ruslan Salakhutdinov, and Eric Xing. 2018. AutoFIS: Learning discrete schedules for alternate optimization. arXiv preprint arXiv:1810.02442 (2018).

[33] Haowen Xu, Bin Liu, Huifeng Guo, Ruiming Tang, Fengwei Zhou, Stefanos P Zaferiou, Yuzhou Zhang, Jun Wang, and Zhenqiu Li. 2020. AutoHash Learning: Higher-order Feature Interactions for Deep CTR Prediction. IEEE Transactions on Knowledge and Data Engineering.

[34] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. ACM Computing Surveys.

[35] Haowen Xu, Haochen Liu, Bin Liu, Jiliang Tang, Weiwei Guo, Jun Shi, Sida Wang, Huiji Gao, and Bo Long. 2020. Memory-efficient Embedding for Recommender Systems. arXiv preprint arXiv:2006.14827 (2020).

[36] Xiangyu Zhao, Changsheng Gu, Haoshenglun Zhang, Xiwang Yang, Xiaobing Liu, Hui Liu, and Jiliang Tang. 2020. Whole-Chain Recommendations. In Proceedings of the 24th ACM International Conference on Information & Knowledge Management. 793–801.

[37] Xiangyu Zhao, Xudong Zheng, Xiaobing Liu, and Jiliang Tang. 2020. AutoEmb: Automated Embedding Dimensionality Search in Streaming Recommendations. arXiv preprint arXiv:2002.11252 (2020).

[38] Xiangyu Zhao, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2019. Toward Seamless Environments in Reinforcement Learning Based Recommendations. IEEE Transactions on Knowledge and Data Engineering.

[39] Xiangyu Zhao, Long Xia, Chunhua Shen, and Anton van den Hengel. 2016. Bridging the analogy gap for cold-start recommendation. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 2307–2316.

[40] Xiangyu Zhao, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2019. Toward Seamless Environments in Reinforcement Learning Based Recommendations. IEEE Transactions on Knowledge and Data Engineering.