

AutoLossGen: Automatic Loss Function Generation for Recommender Systems

Zelong Li
Rutgers University
New Brunswick, NJ, US
zelong.li@rutgers.edu

Jianchao Ji
Rutgers University
New Brunswick, NJ, US
jianchao.ji@rutgers.edu

Yingqiang Ge
Rutgers University
New Brunswick, NJ, US
yingqiang.ge@rutgers.edu

Yongfeng Zhang
Rutgers University
New Brunswick, NJ, US
yongfeng.zhang@rutgers.edu

ABSTRACT

In recommendation systems, the choice of loss function is critical since a good loss may significantly improve the model performance. However, manually designing a good loss is a big challenge due to the complexity of the problem. A large fraction of previous work focuses on handcrafted loss functions, which needs significant expertise and human effort. In this paper, inspired by the recent development of automated machine learning, we propose an automatic loss function generation framework, AutoLossGen, which is able to generate loss functions directly constructed from basic mathematical operators without prior knowledge on loss structure. More specifically, we develop a controller model driven by reinforcement learning to generate loss functions, and develop iterative and alternating optimization schedule to update the parameters of both the controller model and the recommender model. One challenge for automatic loss generation in recommender systems is the extreme sparsity of recommendation datasets, which leads to the sparse reward problem for loss generation and search. To solve the problem, we further develop a reward filtering mechanism for efficient and effective loss generation. Experimental results show that our framework manages to create tailored loss functions for different recommendation models and datasets, and the generated loss gives better recommendation performance than commonly used baseline losses. Besides, most of the generated losses are transferable, i.e., the loss generated based on one model and dataset also works well for another model or dataset. Source code of the work is available at <https://github.com/rutgerswiselab/AutoLossGen>.

CCS CONCEPTS

- Computing methodologies → Machine learning; Reinforcement learning;
- Information systems → Recommender systems; Information retrieval.

KEYWORDS

Recommender Systems; Loss Learning; Loss Generation; Loss Function; Automatic Machine Learning

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 the author(s) 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.

SIGIR '22, July 11–15, 2022, Madrid, Spain

© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 978-1-4503-8732-3/22/07...\$15.00
<https://doi.org/10.1145/3477495.3531941>

ACM Reference Format:

Zelong Li, Jianchao Ji, Yingqiang Ge, Yongfeng Zhang. 2022. AutoLossGen: Automatic Loss Function Generation for Recommender Systems. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22), July 11–15, 2022, Madrid, Spain*. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3477495.3531941>

1 INTRODUCTION

In this era of information explosion, recommendation system (RS) has become an important platform to filter unrelated items and to provide users with items of personalized interest. Many researches are dedicated to optimizing RS models in order to promote the recommendation accuracy. However, a complete RS architecture consists of two vital parts: the RS model, and the loss function to optimize the RS model. Compared to the vast amount of research efforts on developing various kinds of RS models, the research on learning good loss function is still in its initial stage.

Actually, the choice of loss function may significantly influence the accuracy of the recommendation model. This is because the training of a RS model eventually depends on minimizing the loss function, and the gradient of the loss function supervises the optimization direction of the RS model. As a result, any inconsistency between the optimization goal and the optimization direction may hurt the model performance. An intuitive solution to this problem is directly using the optimization goal (e.g., the final evaluation metric) as the loss function. This can be effective when the evaluation metric is differentiable such as the root mean square error (RMSE) for some regression tasks. However, many metrics are non-differentiable and it is difficult to find derivatives such as the Area under the Curve (AUC) for classification tasks. For these cases, it is a challenge to design a good surrogate loss function to approximate the optimization goal, which needs comprehensive analysis and understanding of the task. Thanks to the meticulous design of researchers with their expertise and efforts, we have many useful handcrafted loss functions to solve the optimization problems under non-differentiable metrics [21, 31, 35, 40–43, 49].

Even though handcrafted loss functions have been used in various scenarios, it is still considerably beneficial to have methods that can automatically search and generate good loss functions. This is mainly for two reasons: 1) automatic loss generation helps to remove or reduce the manual efforts in loss design, and 2) the best loss could be different for different RS models and datasets, as a result, automatic loss generation can help to generate the best loss tailored to a specific model-dataset combination.

In recent years, we have witnessed the development of automated machine learning (AutoML) techniques and especially neural architecture search (NAS) [24, 46, 51], which can automatically design

model architectures that are on par with or surpass the manually designed model architectures. Inspired by the success of AutoML, we propose an automatic loss function generation (AutoLossGen) framework that can automatically generate loss functions for model optimization. AutoLossGen is different from existing loss learning research [32–34, 38, 39, 52, 55, 58] on two perspectives. First, AutoLossGen is particularly designed for recommender systems, which present unique challenges due to the extreme data sparsity of recommender systems. This leads to the sparse reward problem in automatic loss generation, and to solve the problem, we propose a reward filtering mechanism for efficient and effective loss generation. Second, previous work on loss learning for recommender systems mostly focuses on automatic loss combination [58], which adopts several handcrafted base losses and learns the weight/importance for each loss, and then combines the losses through weighted sum as the final loss function. Different from previous work, we do not assume any prior knowledge on the loss structure, instead, we directly assemble new loss functions based on very basic mathematical operators, which can help to generate completely new loss functions. Our work is complementary to rather than adversary with previous loss combination methods because our generated new losses can be used as base losses, which can be combined with other handcrafted losses for better loss combination.

This paper makes the following key contributions:

- We propose the AutoLossGen framework, which can generate loss functions directly constructed from basic mathematical operators without prior knowledge on loss structure.
- We develop proxy test and reward filtering mechanisms to speed up the generation process and mitigate the issues caused by the sparsity of RS datasets, so that the framework can produce reliable outcomes efficiently.
- We conduct experiments on two real-world datasets to show that the loss functions generated from the AutoLossGen framework outperform handcrafted base losses.
- We verify the transferability of our generated loss functions by showing that when applying them to other model-dataset settings, the losses can still achieve satisfactory performance.

In the following part of this paper, we first introduce the related work in Section 2. In Section 3, we show the high-level architecture of our AutoLossGen framework. In Section 4, we introduce the detailed design of the loss generation process along with methods for better generation efficiency. We provide and analyze the experimental results in Section 5, and finally conclude the work together with future directions in Section 6.

2 RELATED WORK

In this section, we first introduce related work on automated machine learning (AutoML), and then we introduce loss learning which is a sub-category of AutoML.

2.1 Automated Machine Learning

Automated Machine Learning (AutoML) has been an important direction in recent years, which aims for reducing or even removing the requirement of human intervention in machine learning tasks.

There are three typical applications of AutoML [57]: 1) automated model selection, such as Auto-sklearn [14] and Auto-WEKA

[29], which automatically selects a good machine learning model based on a library of models and hyper-parameter setting; 2) automated feature engineering, such as Data Science Machine [26], ExploreKit [27] and VEST [8], which generates or selects some useful features without manual intervention. Feature engineering is of great importance due to its great influence on model performance [44]; 3) neural architecture search (NAS), such as ENAS [45], DARTS [37], NASH [51], GNAS [24] and AmoebaNet-A [46], which enables to search an effective neural network for a given task without manual architecture design. Experiments have shown that networks generated from NAS are on par with or surpass human-crafted architectures in different tasks.

Our work is related to NAS among these three applications. A loss for model training is usually a function that can be described as terms and operators, while searched architectures from NAS can be described as computation cells and connections. Thus, we can leverage the idea of NAS to construct a loss function search model implemented by reinforcement learning (RL). Although some research employs evolutionary algorithm [46, 47] and hill-climbing procedure [51] for neural architecture search, RL has been shown effective in more research works, including [3, 5, 6, 36, 45, 60], and thus becomes the dominant method in this field. To the best of our knowledge, we are the first to develop automated loss function generation frameworks by RL for recommender systems (RS).

2.2 Loss Learning

Loss function plays an important role in machine learning, as it provides the direction for model training and significantly affects the performance [48]. Thus, besides model design, the choice of loss functions attracts more and more attention for specific tasks. Before the use of AutoML, loss functions are highly handcrafted and those handcrafted losses are shown to be effective and transferable under different scenarios. For regression tasks, mean absolute error (MAE) and root mean square error (RMSE) [1] are often employed in model evaluation [9, 54], and besides L1 and L2 losses, there are some loss variants including Smooth-L1 loss [20], Huber loss [25] and Charbonnier loss [10] for corresponding metrics. For classification tasks, since some metrics are non-differentiable, e.g., the Area under the ROC Curve (AUC) [7], more attempts on loss functions are made, including cross entropy (CE) [49], hinge loss [21] and its variants [31], softmax loss and its variants [40, 41], Focal loss [35], Savage loss [43] and tangent loss [42].

With recent development of AutoML, some researchers propose and study automated loss learning to avoid the significant requirement of human efforts and expertise in loss design. Xu et al. [55] design a framework to automatically select which loss to use and what parameters to update at each stage of the iterative and alternating optimization schedule. Li et al. [32] and Wang et al. [52] investigate the softmax loss to create an appropriate search space for loss learning and apply RL for the best parameter of the loss function. Liu et al. [39] provide a framework to automatically learn different weights of loss candidates for given data examples in a differentiable way. Li et al. [34] substitute non-differentiable operators in metrics with parameterized functions to automatically search surrogate losses. Although these methods aim to learn losses automatically, they still depend on human expertise in the loss search process to a large extent, because the search process starts from

Symbol	Description
U	The set of users in a recommender system
I	The set of items in a recommender system
u	A user ID in a recommender system
i	An item ID in a recommender system
e_u	Embedding vector of the user u
e_i	Embedding vector of the item i
y_{ui} or y	Ground-truth value of the pair (u, i)
\hat{y}_{ui} or \hat{y}	Predicted value of the pair (u, i)
S	Set of controller's currently maintained variables
L or L'	A list storing the generated candidate losses
B	A small batch of training data
θ	Parameters of the controller model
ω	Parameters of the recommender model
\mathcal{L}	Loss value
f	A sampled loss function from the controller model
$\pi(f, \theta)$	Policy of the controller model
ρ	Learning rate of the recommender model

Table 1: Summary of the notations in this work.

existing loss functions. Liu et al. [38] and Li et al. [33] search loss functions composed of primitive mathematical operators for several computer vision tasks by evolutionary algorithm, which is similar to our work, but we focus on different fields (recommender system) by the RL method and aim to address distinct challenges since the sparsity of recommender system datasets causes sparse reward issues if RL is directly applied. In the field of loss learning for RS, Zhao et al. [58] propose a framework to search for an appropriate loss for a given data example, which adopts a set of base loss functions and dynamically adjust the weight of these loss functions for loss combination. Our method is different from and complementary to their work since we focus on generating new losses instead of combining existing losses. More specifically, we construct a new loss function starting from basic variables and operators without prior knowledge of loss structure or predefined loss functions.

3 OVERALL AUTOLOSSGEN FRAMEWORK

In this section, we show the high-level architecture and the main components of the AutoLossGen framework. We will introduce the refined details of the key components in the next section.

3.1 Problem Formalization and Notations

Table 1 introduces the basic notations that will be used in this paper. In order to show our AutoLossGen framework is able to generate effective loss functions, we need to work on a concrete recommendation task so that evaluating the generated loss is possible. In the following part of the paper, we explore the binary like/dislike prediction task for each user-item pair (u, i) in recommender system. This can be formulated as either a 0-1 regression problem or a 0-1 classification problem, and we will explore both of them in the following. To formalize the task, given a pair of user $u \in U$ and item $i \in I$, the learned RS model is required to accurately predict the likeness of user u on item i as \hat{y}_{ui} , while the ground-truth likeness is either $y_{ui} = 1$ (like) or $y_{ui} = 0$ (dislike). Following standard treatment on this problem, we consider the 5-star rating scale in

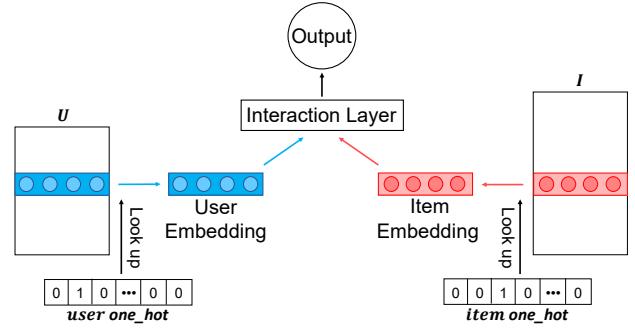


Figure 1: Structure of the recommender model

this paper, while ratings > 3 are considered as likes ($y_{ui} = 1$) and ratings ≤ 3 are considered as dislikes ($y_{ui} = 0$).

Our framework contains two components, an RS model and a loss generation model (also called as the controller). These two parts are introduced in the following subsections.

3.2 The Recommender System Model

AutoLossGen framework is quite flexible for different kinds of recommender models. In this paper, we choose two simple but representative models as examples to explore, which are Matrix Factorization (MF) [28] and the Multi-Layer Perceptron (MLP) network [13] for recommendation. The former stands for traditional shallow matching method for recommendation, while the latter represents the deep matching model for recommendation. As shown in Figure 1, the recommender model is composed of three layers: embedding layer, interaction layer and output layer, which are briefly introduced in the following.

3.2.1 Embedding Layer. In our recommendation task, we obtain the embedding vectors of users and items by their one-hot ID vectors. To formulate this, after transforming the user and item IDs to their corresponding one-hot vectors u and i , the user embedding e_u and item embedding e_i are calculated as:

$$e_u = M_u \cdot u, \text{ and } e_i = M_i \cdot i \quad (1)$$

where $M_u \in \mathbb{R}^{d \times |U|}$ and $M_i \in \mathbb{R}^{d \times |I|}$ are the matrices storing the embeddings of all users U and all items I , respectively, and they are learned during the training process. In this way, the sparse and high-dimensional one-hot vectors are compressed into low-dimensional embedding vectors, and we can retrieve their representations in this layer.

3.2.2 Interaction Layer. After the embedding layer, we feed the user and item representations into the interaction layer to make predictions. The structure of this layer is the main difference between MF and MLP. In MF model, we calculate the inner product of vectors with the bias terms, as shown in Eq.(2), where b_u , b_i and b_g are the user bias term, item bias term and global bias term, respectively. Together with e_u and e_i , they are the parameters of MF to be learned.

$$h_{out} = e_u \cdot e_i + b_u + b_i + b_g \quad (2)$$

For MLP model, the structure of the interaction layer is multiple hidden layers composed of fully-connected layer and activation

Operator	Expression	Arity
Add	$x + y$	2
Multi	$x \cdot y$	2
Max	$\max(x, y)$	2
Min	$\min(x, y)$	2
Neg	$-x$	1
Identical	x	1
Log	$\text{sign}(x) \cdot \log(x + \xi)$	1
Square	x^2	1
Reciprocal	$\text{sign}(x) / (x + \xi)$	1

Table 2: The set of basic mathematical operators. x and y are variables for operators. $\xi = 10^{-6}$ is a small value to avoid numerical error such as division by zero. Besides, the result of each operation is clamped in the interval $[\xi, 1/\xi]$ to avoid numerical explosion.

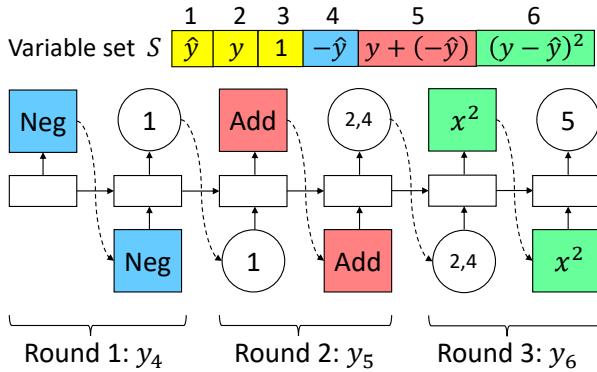


Figure 2: Taking the generation of MSE loss as an example to illustrate the loss generation process of the controller.

layer. The output of the l -th hidden layer is formulated as:

$$h_{l+1} = \text{ReLU}(W_l h_l + b_l) \quad (3)$$

where W_l is the weight matrix and b_l is the bias vector, and the model uses ReLU as the activation function. The input to the interaction layer of MLP is the concatenation of user and item embeddings, denoted as $h_0 = [e_u, e_i]$; the output is the result of the last activation layer, denoted as $h_{out} = h_n$ if there are n hidden layers in total. Besides, to unify the format of the interaction layer, the dimension of the last hidden output layer is 1, i.e., h_{out} is a real number. More implementation details are provided in the experiments.

3.2.3 Output Layer. The task of the output layer is to generate the final prediction \hat{y}_{ui} of the RS model. Therefore, it could be different based on the range of y , but we can define the unified formula as:

$$\hat{y}_{ui} = \sigma(h_{out}) \quad (4)$$

where $\sigma(\cdot)$ is the activation function. For our task, it is *sigmoid* since we limit the final output to a value between 0 and 1.

3.3 The Controller Model

The controller model is the most important part of the AutoLossGen framework, which is implemented as a recurrent neural network

(RNN). The controller is able to generate various loss functions starting from basic mathematical operators. Our loss search space contains all possible functions composed of the operators shown in Table 2. Most common handcrafted losses are also built from these operators, such as the cross-entropy (CE) loss and L2 loss. Thus, we can expect that the search space is large enough for our controller to generate some effective loss functions.

The role of variables and operators in loss function is similar to the role of computation cells and edge connections in neural networks. As a result, the loss generation process of RNN is similar to neural architecture search (NAS), shown as Figure 2. At first, there are three initial variables in the variable set S : \hat{y} , y and the number 1. During each round, RNN samples an operator first, and then based on the arity of the operator shown in Table 2, samples the corresponding number of distinct input variables from the set S . For example, in Figure 2, RNN samples *Negative* operator in the first round, so for the next step, one variable from the variable set S is sampled (suppose \hat{y} is sampled), and a new variable (i.e., $-\hat{y}$) is created and added into the variable set S ; in the second round, RNN samples *Add* operator, and thus two different variables are needed to execute the operation. The distribution of operator and variable sampling in each round is determined by the RNN's hidden output vector from the previous round, which goes through a soft-max layer to create a sampling probability vector with the same size as the current number of candidate operators and candidate variables in S . Besides, to encourage complex functions in fewer rounds, our framework allows variables to be used multiple times.

Finally, the controller takes the last variable in the variable set as the loss function. One thing to note here is that there is no significant relationship between the complexity and the performance of loss functions, i.e., the most effective loss function could have either complex or concise mathematical forms. As a result, we would not like to apply too many restrictions during the loss sampling process, and because of this, our framework is not only able to generate complex loss functions with various forms, but also includes short expressions in the search space. More precisely, the loss function search space includes every possible expression based on the pre-defined operators that utilizes at most m intermediate variables, where m is defined as the maximum number of rounds in RNN. However, the flexibility of sampled loss functions also brings new problems, including the zero-gradient problem and the duplicated function problem, and some bad functions may ruin the performance if used as loss. We will provide solutions to these problems in the loss generation process, specifically in Section 4.1.

4 LOSS GENERATION PROCESS

There are three phases when executing our AutoLossGen framework, as shown in Figure 3. In Phase I (loss search), we optimize all parameters of our framework in iterative and alternating schedule, and record the loss function we generated in every RL optimization loop. In Phase II (validation check), we remove those functions that cannot produce correct gradient direction for simulated (y, \hat{y}) pairs. In Phase III (effectiveness test), for each selected loss in Phase II, we randomly initialize a recommender model and train the model to convergence using the loss to obtain the final performance for the loss, and we keep the best performance loss as the finally selected

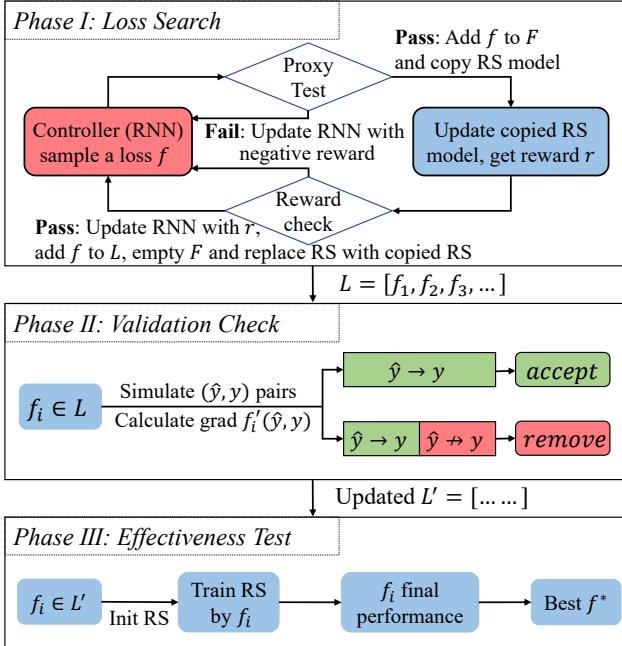


Figure 3: Overview of the loss generation process with three phases: loss search, validation check, and effectiveness test.

loss function. In the following part of this section, we introduce the loss generation phases with the proposed techniques in detail.

4.1 Loss Search Phase

4.1.1 Iterative and Alternating Optimization Schedule. As introduced in Section 3, both parameters in the recommender model ω and parameters in the RNN controller model θ need optimization in the search process. For the RS model, we perform stochastic gradient descent (SGD) on it to update ω , while for the controller, inspired by the success of neural architecture search (NAS) in [45, 59], we apply the REINFORCE [53] algorithm to update θ , and the performance increment on the validation dataset is used as the reward signal for policy gradient. We use the Area under the ROC Curve (AUC) for classification task and root mean square error (RMSE) for regression task to evaluate the performance increment and obtain the reward on validation set since they are prediction-sensitive, i.e., the metric values will be different with a very small fluctuation on predictions [1, 7], so that non-trivial reward can be calculated for better update on the controller.

The search process is a typical bi-level optimization problem [2]. Ideally, after the controller samples a new loss function, we cannot judge the performance of the loss and retrieve the reward signal until the RS model is updated to convergence. The optimization problem can be formulated as:

$$\begin{aligned} & \max_{\theta} \mathbb{E}_{f \sim \pi(f, \theta)} [\mathcal{R}_{val}(f, \omega^*(f))] \\ & \text{s.t. } \omega^*(f) = \arg \min_{\omega} \mathcal{L}_{train}(\omega, f) \end{aligned} \quad (5)$$

where $\mathbb{E}[\mathcal{R}_{val}]$ represents the expected reward on validation sets, and other symbols can be referred in Table 1. However, due to the large search space, the nested optimization process is too time-consuming to be put into practice. Referring to DARTS techniques

Algorithm 1: Iterative and alternating optimization schedule

```

1 Input: Controller, RS model, Reward threshold  $\eta$ 
2 Output:  $[f_1, f_2, f_3 \dots]$ 
3 while not converge do
4    $init \leftarrow$  performance of RS on validation data
5   do
6     | Controller generates a loss  $f(\hat{y}, y, 1)$ 
7     while  $f$  fails on the proxy test;
8     Copy current RS model
9     Update the copied RS by  $f$  with one epoch of train data
10     $updated \leftarrow$  performance of copied RS on validation data
11     $reward \leftarrow updated - init$ 
12    if  $reward \geq -\eta$  then
13      Replace RS model by copied RS
14      Record  $f$  into  $L$ 
15    end
16  end
17 return  $L$ 

```

[37], we utilize first-order approximation of the gradient for the RS model, and leverage the temporary reward to update the controller. To be specific, we update the RS model with one epoch of training data by the sampled loss function to approximate the optimization effect, shown as Eq.(6), and then use the performance increment of the current RS model on validation datasets as the reward of the sampled loss to update the controller. The process is called iterative and alternating optimization, since the updates of the RS model and the controller is alternating in each iteration. The pseudo code of our optimization method is described in Algorithm 1. In Section 5, we experimentally show that the first-order approximation is able to help our framework generate effective loss functions.

$$\omega^*(f) \approx \omega - \rho \nabla_{\omega} \mathcal{L}_{train}(\omega, f) \quad (6)$$

4.1.2 Proxy Test. In Algorithm 1, we mentioned a proxy test after the controller samples a loss. As analyzed in Section 3.3, we expect fewer restrictions on the form of losses when sampling functions, and as a side effect, zero-gradient functions and gradient-level duplicated functions may appear during the search process since the form of sampled functions can be very simple. Zero-gradient functions are the functions whose gradients over \hat{y} are always zero, such as $(y + 1)$, and gradient-level duplicated functions are the functions that have the same or very close gradient values over \hat{y} with already sampled functions. For instant, if MSE loss $(\hat{y} - y)^2$ is already sampled from our framework in the current RL optimization loop, then $(\hat{y} - y)^2 + y$ will be considered as a duplicated loss function since its gradient over \hat{y} is the same as MSE and thus their effects are equivalent in gradient-based optimization. To efficiently skip these losses, at the beginning of the whole loss search process, we sample a small batch of training data B , where $|B|$ is set between 5 to 20 according to the size of datasets and B will not be changed. After the controller samples a loss f , we first calculate the gradient of the RS model on B , denoted as $\nabla_{\omega}(B, f)$, and if:

$$\|\nabla_{\omega}(B, f)\| < \delta \quad (7)$$

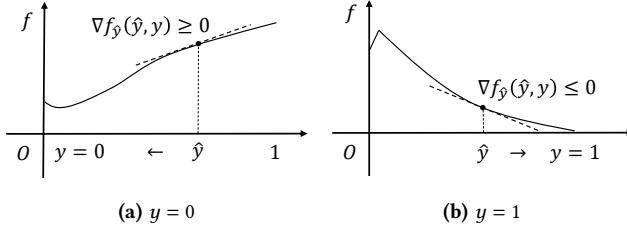


Figure 4: A toy example to show how a loss function can pass the validation check.

where δ is a small value set as 10^{-4} in implementation, then we treat f as a zero-gradient loss and provide a default negative reward to update the controller since we do not want zero-gradient losses in future rounds. Besides, we use a set F to temporarily store the already sampled losses in the current RL optimization loop, and if:

$$\exists f' \in F, \text{ s.t. } \|\nabla_{\omega}(B, f) - \nabla_{\omega}(B, f')\| < \delta \quad (8)$$

then we consider f as a duplicated loss and directly use the reward of f' to update the controller. If f is not considered as zero-gradient or duplicated losses (i.e., if f passes the proxy test in Eq.(7) and (8)), then we add f into F and calculate its reward over RS model. In this way, zero-gradient and gradient-level duplicated losses are quickly skipped after proxy test and thus we do not have to waste time training the RS model over such losses.

4.1.3 Reward Filtering Mechanism. During the training process of the RS model, in most cases, we would have to use more steps to correct the model if the model is trained along the wrong direction by a bad loss. As a result, to speed up the loss generation process and avoid degradation on RS model performance, we do not directly test a generated loss on the RS model, instead, we make a copy of the RS model and optimize the copied model over a generated loss to calculate the reward of the loss. Besides, we only replace the RS model with the copied RS model if $reward \geq -\eta$, otherwise, we discard the copied RS model and do not update the parameter of the RS model. Here, we allow some minor negative rewards to provide RL with some exploration ability on top of exploitation so as to avoid getting stuck in local optima. In Section 5.5, we will show through ablation study that without the reward filter mechanism, the performance of RS model may be unstable and no effective loss function can be generated during the search.

4.2 Validation Check and Effectiveness Test

We expect our generated losses can update a randomly initialized RS model from the beginning to convergence throughout the whole training process. However, the output loss functions from the loss search phase are only tested to be effective for a certain epoch of optimizing the RS model. Loss functions should perform well for various y and \hat{y} values in the domain of definition of the values. Out of such consideration, we design a validation check phase to filter the loss functions provided by the Loss Search Phase.

More specifically, we sample different values of \hat{y} and y from $\hat{y} \in [0, 1]$ and $y \in \{0, 1\}$ and create a set of synthesized (\hat{y}, y) pairs. For each candidate loss function f from the previous phase, we calculate the gradient of f over \hat{y} , denoted as $\nabla f_{\hat{y}}(\hat{y}, y)$. If $\nabla f_{\hat{y}}(\hat{y}, y) \geq 0$ when $y = 0$ and $\nabla f_{\hat{y}}(\hat{y}, y) \leq 0$ when $y = 1$ for all of the synthesized pairs,

Dataset	#Users	#Items	#Pos	#Neg	Density
ML-100K	943	1,682	55,375	44,625	6.30%
Electronics	192,403	63,001	1,356,067	333,121	0.014%

Table 3: Basic statistics of the datasets

then the loss function is considered valid, otherwise, it is removed from the loss candidate set. As shown in Figure 4, the intuition is that if the ground truth label is $y = 0$, then we hope the gradient direction on \hat{y} is positive, and because we use the inverse gradient direction for loss minimization, so \hat{y} will be optimized towards 0 during optimization. Similar for the case of $y = 1$. Based on this, we can filter out those losses whose gradients are infeasible for optimization.

Finally, in Phase III, we leverage each candidate loss to train a randomly initialized RS model to convergence and use the model performance on validation set as the effectiveness of the candidate loss, and we take the most effective loss as the final output loss of the whole loss learning process. In the experiment part, we will report the performance of the generated loss on test set.

5 EXPERIMENTS

In this section, we conduct experiments to evaluate the effectiveness of generated loss functions and to help better understand the loss generation process.¹

5.1 Experimental Setup

5.1.1 Dataset Description. Our experiments are conducted on two widely-used benchmark datasets of recommender systems, namely, ML-100K and Amazon Electronics [15–19, 22, 23, 56], which are both publicly available. The detailed statistics of the datasets are shown in Table 3, and we briefly introduce these two datasets in the following part.

- **ML-100K:** It is a widely used RS dataset maintained by GroupLens.² It contains 100,000 ratings from 943 users on 1,682 movies. The rating values are integers ranging from 1 to 5 (both included).
- **Amazon Electronics:** This is one of the Amazon 5-core e-commerce datasets³ that records the rating of items given by users on Amazon spanning from May 1996 to July 2014. We use the Electronics category with over one million interactions, which is larger and sparser than ML-100K. The rating values are integers ranging from 1 to 5 (both included).

As mentioned in Section 3.1, we would like to predict the explicit feedback from users, either as a classification task or as a regression task. Following standard treatment, ratings ≥ 4 are considered as positive (like) with label as 1, while ratings ≤ 3 are considered as negative (dislike) with label as 0. We use positive Leave-One-Out to create the train-validation-test datasets [11, 12, 50]. Specifically, for each user, based on timestamp, we put the user's last positive interaction, along with following negative interactions, into the test set, put the second-to-last positive interaction, together with

¹Source code available at <https://github.com/rutgerswiselab/AutoLossGen>.

²<https://grouplens.org/datasets/movielens/100k/>

³<http://jmcauley.ucsd.edu/data/amazon/index.html>

Model	Dataset	Task	Loss Name	Abbr.	Loss Formula
MF	ML-100K	Classification	Max of Ratio Loss	MaxR	$\max((\hat{y} + \epsilon)/(y + \epsilon), (y + \epsilon)/(\hat{y} + \epsilon))$
MF	Electronics		Sum of Reciprocal Loss	SumR	$(\hat{y} + y + \epsilon)/(\hat{y} \cdot y + \epsilon)$
MLP	ML-100K				
MLP	Electronics		Log Min Based Loss	LogMin	$\log((1 + \epsilon)/(\min(\hat{y}, y) + \epsilon)) \cdot (\hat{y} + y + \min(\hat{y}, y))$
*	*	Regression	Mean Square Error Loss	MSE	$(\hat{y} - y)^2$

Table 4: Loss generation result, ϵ is a tunable parameter representing the smoothing coefficient to prevent numerical errors such as division by zero. MSE loss is generated in all four model-dataset combinations for the regression task.

remaining following negative interactions, into the validation set, and put all of the remaining interactions into the training set. If a user has fewer than 5 interactions, we put all its interactions into the training set to avoid the cold start problem.

5.1.2 Baseline Losses. We compare with the following baseline losses in the experiment.

- **Mean Square Error (MSE)** [1]: MSE is a commonly used loss for regression tasks, but also shows good performance for classification. It minimizes the square of difference between label and prediction values.
- **Binary Cross Entropy (BCE)** [49]: BCE is a special form of cross entropy (CE) for binary classification task. It is one of the most widely used loss functions for the task. Besides, two well-known losses, logistic loss and Kullback–Leibler (KL) divergence [30], are different from BCE only in constants for binary classification task, and thus, we use BCE as a baseline loss to represent these types of losses.
- **Hinge Loss** [21]: Hinge loss is a margin-based loss function. It does not require the prediction to be exactly the same as the true value. Instead, if y and \hat{y} are close enough, the loss value would be 0, which is reasonable for classification tasks.
- **Focal Loss** [35]: Focal is recently proposed and revised from the CE loss. It is designed for models to concentrate on hard samples and reduce the weight of well-classified samples.

We are aware of some recent loss combination techniques such as SLF [39] and AutoLoss [58]. However, these models do not generate new loss functions. Instead, they learn a weighted sum of existing handcrafted losses as the final loss. However, our work aims to generate new individual losses rather than weighted combination of losses, as a result we only compare with individual loss functions. Actually, our work is complementary to rather than adversary with these loss combination methods, because our generated new losses can be used as base losses together with existing handcrafted losses for loss combination. As a result, our method will positively contribute to the loss combination methods if our generated loss functions are better than existing handcrafted loss functions.

5.1.3 Evaluation Metrics. To evaluate the final performance of the generated loss functions, we use the Area under the ROC Curve (AUC), F1-score and Accuracy for evaluating the classification task, and use mean absolute error (MAE) and root mean square error (RMSE) for evaluating the regression task.

5.1.4 Implementation Details. Our framework and all baselines are implemented by PyTorch, an open source library. As mentioned

in Section 3.2, we test on two types of RS models, a shallow matching model based on Matrix Factorization (MF) and a neural matching model based on Multi-Layer Perceptron (MLP). The implantation details of the RS models are as follow: (a) *Embedding layer*: we set the dimension of embedding vectors as 64 for both users and items. (b) *Interaction layer*: we do not have hyper-parameters for MF, and for the MLP model, we have three fully-connected layers with layer size 128×64 , 64×16 and 16×1 . For each layer of MLP, we leverage batch normalization, dropout (with rate as 0.2) and ReLU function for activation. (c) *Output layer*: we use *sigmoid* function since y is either 0 or 1, and \hat{y} is expected to be in-between 0 and 1.

Besides the RS model, another critical part of the AutoLossGen framework is the controller model. The controller RNN is implemented by a two-layer LSTM with 32 hidden units on each layer, and the weights of controller are uniformly initialized between -0.1 and 0.1. We use logit clipping with the tanh constant of 1.5 to limit the range of logits so as to control the sampling entropy. This can help to increase the sampling diversity and avoid premature convergence [4]. We also add the controller’s sampling entropy to the reward, weighted by 0.0001, to drive the sampling process towards a relatively stable status. The largest length of variable set is fixed to 10, i.e., our search space includes all functions that utilizes at most 10 intermediate variables including y , \hat{y} and 1.

During the loss generation process, we use iterative and alternating schedule to optimize the controller and RS models. When training the RS model, we fix the parameters of the controller θ , and use stochastic gradient descent (SGD) optimizer with learning rate as 0.01 to update the RS model; when training the controller, we fix the parameters of the RS model ω , and use Adam optimizer with learning rate as 0.001. To reduce random bias, we average the rewards of ten sampled loss functions to update the controller. ℓ_2 parameter regularization is adopted in both RS model and controller model optimization. For the proxy test in the loss search phase, the batch size $|B|$ is set as 5 for the smaller ML-100K dataset and 20 for the larger Electronics dataset. We want the exploration process to be as thorough as possible, as a result, the termination condition we set is that the RS model converges or the controller cannot sample an effective loss to pass the reward filtering mechanism over 24 hours. The running time of the exploration process varies from about two days to one week with different model-dataset combinations. However, the exploration and loss learning process is like gold mining—once the loss function is found, we do not have to re-run the process any more.

For the validation check in the second phase of Figure 3, the number of simulated pairs is 2,000. In the effectiveness test phase,

Model	Dataset	Metric	Handcrafted Loss				Generated Loss		
			MSE	BCE	Hinge	Focal	MaxR	SumR	LogMin
MF	ML-100K	AUC \uparrow	0.7808	0.7882	0.7848	0.7930	0.8087*	0.8086	0.7981
		F1-score \uparrow	0.6058	0.6073	0.6133	0.6121	0.6260*	0.6245	0.6160
		Accuracy \uparrow	0.6919	0.6972	0.7239	0.7011	0.7305	0.7645*	0.7398
MF	Electronics	AUC \uparrow	0.6510	0.6515	0.6689	0.6521	0.6697*	0.6695	0.6534
		F1-score \uparrow	0.8843	0.8843	0.8846	0.8843	0.8846	0.8846	0.8844
		Accuracy \uparrow	0.7927	0.7927	0.7937	0.7927	0.7937	0.7937	0.7927
MLP	ML-100K	AUC \uparrow	0.7655	0.7725	0.7472	0.7629	0.7747	0.7743	0.7752*
		F1-score \uparrow	0.5938	0.5985	0.5865	0.5890	0.6130*	0.6053	0.6064
		Accuracy \uparrow	0.6625	0.6797	0.6717	0.6437	0.7077*	0.6830	0.6909
MLP	Electronics	AUC \uparrow	0.6232	0.6228	0.6242	0.6205	0.6250	0.6404*	0.6318
		F1-score \uparrow	0.8843	0.8843	0.8843	0.8843	0.8844*	0.8843	0.8843
		Accuracy \uparrow	0.7926	0.7926	0.7926	0.7926	0.7928*	0.7927	0.7926

Table 5: Final performance on the classification task. \uparrow means the measure is the higher the better. Bold numbers indicate its performance is significantly better at $p < 0.01$ than all baseline losses, and * represents the best performance of each row.

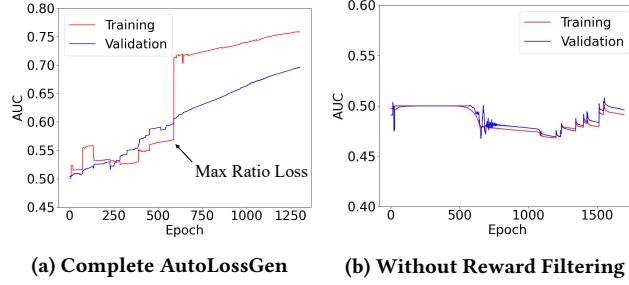


Figure 5: The performance of MF model on training and validation datasets of ML-100K during loss generation process under the classification task.

we randomly initialize a new RS model and use SGD for model optimization. There may exist tunable parameters to avoid numerical errors such as division by zero in the generated loss functions (e.g., the ϵ in Table 4), and we use grid search in $[1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}]$ on validation set to decide the value of the parameters. To prevent over-fitting, if the performance of the RS model on validation set is decreasing in 10 consecutive epochs, or the best performance on validation set is over 50 epochs before, then the training process will be early terminated.

5.2 The Generated Loss Functions

In this section, we would like to show the generated loss functions and the loss generation process of our framework described in Section 4 and Figure 3. We run the AutoLossGen framework under four model-dataset combinations for both the classification task and the regression task. The best loss function after Phase III under each setting is shown in Table 4, and more losses are provided in Appendix. The smoothing coefficient ϵ is included in each formulation when a division is calculated, which is to prevent numerical errors such as division by zero. The generated losses for the classification task are new losses and we name them MaxR, SumR and LogMin loss, respectively, while the generated loss for the regression task is the existing MSE loss for all of the four model-dataset combinations.

We take the generation process of the Max Ratio Loss (MaxR) as an example. Figure 5a shows the performance of the MF model on training and validation datasets during the loss search phase on the classification task. We can see that there is a significant increase on recommendation performance by using MaxR in that epoch, implying that MaxR is a good loss candidate to promote the performance. After the first phase, we confirm that MaxR is an effective loss based on the validation check in the second phase, which is further selected as the best loss in the third phase. And finally, we show that MaxR is indeed an effective loss on test set in Section 5.3.

Other generated loss functions also show similar improvements in Phase I and proven effective in Phase II, and finally selected by Phase III. Note that the generated loss under MF-Electronics setting is very similar with that under MLP-ML100K setting, with only a small difference in the constant term. Thus, we merge these two loss functions as SumR, whose name is due to the sum of the reciprocal of \hat{y} and y . The SumR loss may not be handcrafted by human experts since if separating the formula to the sum of the reciprocals (i.e., $\frac{1}{\hat{y}} + \frac{1}{y}$), we intuitively may not accept it as a good loss since it may cause numerical exception. However, after merging the reciprocals and adding smoothing coefficient ϵ , SumR proves to be a simple and effective loss, as we will show in the following experiment. For regression task, our AutoLossGen framework generates MSE as the loss in all of the four experimental settings, which is reasonable because MSE directly optimizes the target RMSE metric. This shows that when the ground-truth loss exists for a task, our framework is able to recover the ground-truth loss.

5.3 Performance Comparison

We compare the performance of the generated loss functions and four baseline losses on the test set for the classification task. Results under different model-dataset and evaluation metrics are shown in Table 5. We have the following observations from the results.

First and most importantly, the loss function generated in the corresponding model-dataset performs better on AUC than any

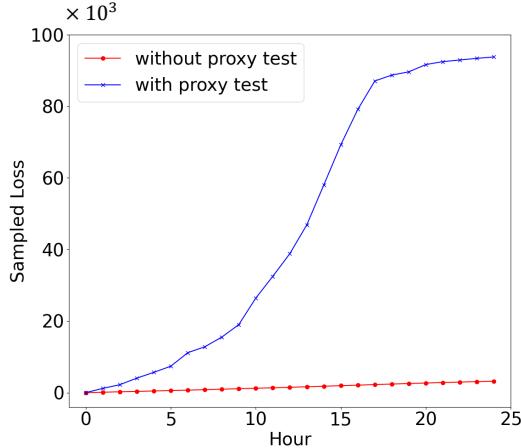


Figure 6: The accumulated number of sampled loss functions is shown in hours when using proxy test or not.

Model	Dataset	Metric	Handcrafted Loss			Generated Loss
			BCE	Hinge	Focal	
MF	ML-100K	RMSE ↓	0.4540	0.4602	0.4710	0.4480*
		MAE ↓	0.4386	0.4077	0.4673	0.3548*
MF	Electronics	RMSE ↓	0.4043	0.4075	0.4363	0.3945*
		MAE ↓	0.3224	0.3501	0.4237	0.3180*
MLP	ML-100K	RMSE ↓	0.4338	0.4643	0.4376	0.4268*
		MAE ↓	0.3802	0.3642	0.3811	0.3586*
MLP	Electronics	RMSE ↓	0.4005	0.4097	0.4153	0.4000*
		MAE ↓	0.3105	0.3633	0.3573	0.3100*

Table 6: Final performance on the regression task. ↓ means the measure is the lower the better. * represents the best performance for each row.

	#Sampled Loss	Speed-up
Without proxy test	3,200	1x
With proxy test	93,792	~ 30x

Table 7: Ablation study on proxy test. Speed-up shows how many times of losses are sampled than that w/o proxy test.

of the four baseline losses. The reason why the generated loss from our AutoLossGen framework outperforms the handcrafted loss functions is that during the loss generation process, the generated loss has been tested effective on both real data in Phase I (loss search) and on synthesized data in Phase II (validation check). Also, the performance on F1-score and accuracy of our generated losses is on par or better than that of all baselines. As a result, the generated losses from AutoLossGen can be more suitable for the corresponding model-dataset combination.

One interesting observation is that there is no globally best loss function, not only among the generated loss functions, but also for the handcrafted loss functions. For example, when we use MF as the RS model and Electronics as the dataset, hinge loss outperforms other baseline losses, however, if dataset switches

to ML-100K, Focal loss is the best among the handcrafted losses. Furthermore, for the combination of MLP and ML-100K, BCE loss defeats other baseline losses on all three metrics. This observation indicates the importance of using AutoLossGen framework when the environment changes so as to find the best loss function tailored to the environment.

For the regression task, the experimental results are shown in Table 6. MSE loss is the generated loss in this task and the performance of MSE loss is better than other losses as expected.

5.4 Loss Transferability

Even though the transferability of the generated loss functions is not the key focus of this paper, we still do not expect the loss from AutoGenLoss can only be applicable to one model or one dataset. Table 5 also shows the results when a loss generated from one model-dataset setting is applied on another model-dataset setting. We can see that even if applied to other experimental settings, our generated loss functions still outperform the baseline losses in most cases. The only exception is the LogMin loss under the MF-Electronics combination, where LogMin is slightly worse than the Hinge loss. However, LogMin is still better than all of the other three baseline losses under this setting. Besides, MaxR and SumR are both better than Hinge loss under this setting.

As a result, though it is best to apply the AutoLossGen framework to each specific experimental setting to obtain the most suitable loss for that setting, but to some extent, our generated loss functions are transferable to other experiment settings.

5.5 Ablation Study on Efficiency

In this section, we discuss the improvement on efficiency by the proposed proxy test mechanism in Section 4.1.2.

For proxy test, we compare the number of explored loss functions with and without the proxy test mechanism under the same amount of time. Here, without proxy test mechanism means that all of the sampled loss from the controller will directly pass the proxy test and update the copied RS model. For fair comparison, all experiments are run on a single NVIDIA GeForce 2080Ti GPU in 24 hours. The operating system is Ubuntu 20.04 LTS. The quantitative results are shown in Table 7 and the accumulated number of sampled loss functions over time (in hours) is shown in Figure 6. We can see that the loss search efficiency is significantly better when the proxy test is applied, which means that a lot more loss functions can be explored within the same amount of time. This is not surprising because we encourage fewer restrictions on the form of sampled functions, which can lead to a large number of zero-gradient and gradient-level duplicated functions during the search process. Without the proxy test mechanism, all functions will pass the test and thus a lot of time has to be spent on updating the copied RS model, which reduces the number of functions that can be explored in the same amount of time.

5.6 Ablation Study on Reward Filtering

We explore the role of the reward filtering mechanism in Section 4.1.3. We use the MF model under the ML-100K dataset as an example. Observations on other model-dataset combinations are similar. We plot the accuracy of the RS model when the reward filtering

mechanism is removed during the loss search phase as Figure 5b for better comparison with Figure 5a. We can see that when the reward filtering mechanism is removed, the AUC of the RS model will stay around 0.5, which means that the RS model is unable to learn useful information for prediction.

Intuitively, the reward filtering mechanism guarantees that the RS model will only be updated when the sampled loss is relatively good (i.e., reward $\geq -\eta$). Without the reward filtering mechanism, the RS model will always be updated by any sampled loss as long as the loss passes the proxy test, including those low-quality losses that lead to very negative rewards on the copied RS model. As a result, the reward filtering mechanism is important to guarantee the performance of the RS model and to filter out the low-quality losses from the candidate loss list.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we propose AutoGenLoss, an automatic loss function generation framework for recommender systems (RS), which is implemented by reinforcement learning (RL) and optimized in iterative and alternating schedules. Experiments show the better performance and transferability of the generated loss functions than commonly used handcrafted loss functions under various settings.

We will further extend our framework on several aspects in the future. For the controller model, although REINFORCE [53] has shown its effectiveness, more state-of-the-art RL algorithms may reduce the redundant sampling for better efficiency. Meanwhile, faster search makes it possible to include more operators with larger search space to locate better loss functions. Another line of potential research is to propose an end-to-end differentiable model and integrate coefficient search in loss generation, since coefficients in loss functions may influence the performance. Besides, we mainly focused on classification and regression loss generation in this work, while it is promising to generalize the framework for other tasks such as ranking in the future. Finally, through this work we can see that the generated loss function formulas have different forms—some are simple while some are complex; some are effective while some are non-effective. In the future, it will be very interesting to build systematic theories and/or methodologies to understand what are the key factors that make a loss formula effective and how can such understanding be encoded into the loss learning algorithm to search for the best loss function more efficiently and effectively.

ACKNOWLEDGEMENT

This work was supported in part by NSF IIS 1910154, 2007907, and 2046457. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors.

APPENDIX

In the AutoLossGen framework, multiple loss functions may pass Phase II (validation check) and enter Phase III (effectiveness test). In Table 4 we have listed the best loss f^* after Phase III (effectiveness test) on each dataset-model combination. Here, we use the MF on ML-100K combination as an example to show all of the generated loss functions that passed Phase II (i.e., passed the validation check on more than 90% of (y, \hat{y}) pairs) and entered Phase III, as shown in

Table 8. An observation from the results is that the generated loss is either effective (AUC close to known loss functions) or non-effective at all (AUC around 0.5, i.e., close to random guess). This implies that the effectiveness (in terms of AUC) of loss functions does not uniformly span across the effectiveness space, but instead tend to be binary, i.e., a loss function either works or does not work at all, and there may not be a loss function that partially works. As long as an effective loss exists, the AutoLossGen framework is able to generate the loss in Phase I and eventually find it through Phase II (validation check) and Phase III (effectiveness test). Our observation also indicates that there may exist some general knowledge about what key factors contribute to the effectiveness of loss formula. For example, the best loss functions (Table 4) tend to exhibit certain degrees of symmetry. In the future, it will be very interesting to build systematic theories and/or methodologies to understand what are the key factors that make a loss formula effective and how can such knowledge be embedded into the loss learning algorithm or process as prior knowledge to search for the best loss function more efficiently and effectively.

REFERENCES

- [1] David M Allen. 1971. Mean square error of prediction as a criterion for selecting variables. *Technometrics* 13, 3 (1971), 469–475.
- [2] G Anandalingam and Terry L Friesz. 1992. Hierarchical optimization: An introduction. *Annals of Operations Research* 34, 1 (1992), 1–11.
- [3] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2017. Designing Neural Network Architectures using Reinforcement Learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings*. OpenReview.net. <https://openreview.net/forum?id=S1c2cvqee>
- [4] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940* (2016).
- [5] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. 2017. Neural optimizer search with reinforcement learning. In *International Conference on Machine Learning*. PMLR, 459–468.
- [6] Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. 2018. Path-level network transformation for efficient architecture search. In *International Conference on Machine Learning*. PMLR, 678–687.
- [7] Toon Calders and Szymon Jaroszewicz. 2007. Efficient AUC optimization for classification. In *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 42–53.
- [8] Vitor Cerqueira, Nuno Moniz, and Carlos Soares. 2021. Vest: Automatic feature engineering for forecasting. *Machine Learning* (2021), 1–23.
- [9] Tianfeng Chai and Roland R Draxler. 2014. Root mean square error (RMSE) or mean absolute error (MAE)?—Arguments against avoiding RMSE in the literature. *Geoscientific model development* 7, 3 (2014), 1247–1250.
- [10] Pierre Charbonnier, Laure Blanc-Féraud, Gilles Aubert, and Michel Barlaud. 1994. Two deterministic half-quadratic regularization algorithms for computed imaging. In *Proceedings of 1st International Conference on Image Processing*, Vol. 2. IEEE, 168–172.
- [11] Hanxiong Chen, Yunqi Li, Shaoyun Shi, Shuchang Liu, He Zhu, and Yongfeng Zhang. 2022. Graph Collaborative Reasoning. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. 75–84.
- [12] Hanxiong Chen, Shaoyun Shi, Yunqi Li, and Yongfeng Zhang. 2021. Neural Collaborative Reasoning. In *Proceedings of the Web Conference 2021*. 1516–1527.
- [13] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.
- [14] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2015/file/11d0e6287202fcfd83f79975ec59a3a6-Paper.pdf>
- [15] Zuhui Fu, Yikun Xian, Yixin Zhu, Shuyuan Xu, Zelong Li, Gerard De Melo, and Yongfeng Zhang. 2021. HOOPS: Human-in-the-Loop Graph Reasoning for Conversational Recommendation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Positive Rate	Loss Formula	AUC (MF on ML-100K)
1.000	$\max(\hat{y}, y * \max(\hat{y}, \frac{1}{\hat{y}+\epsilon}))$	0.4882
1.000	$\hat{y} + \frac{1}{y * \hat{y} + \epsilon}$	0.4918
1.000	$\frac{\max(y, \hat{y})}{\hat{y} + \epsilon}$	0.4881
1.000	$1 + \hat{y} + \frac{1}{\min(\hat{y}, y) + \epsilon} + \max\left(\frac{1}{\min(\hat{y}, y) + \epsilon}, \max\left(\min(\hat{y}, y), \hat{y} + \frac{1}{\min(\hat{y}, y) + \epsilon}\right)\right)$	0.4863
1.000	$\hat{y} + y + \frac{1}{\min(y, \hat{y})^2 + \epsilon}$	0.4898
1.000	$\frac{1}{\min(y, \hat{y}) + \epsilon} + \max(\hat{y}, y) + \max(\hat{y}, y)^2$	0.4877
1.000	$1 + \min(\min(1, \hat{y} + y), \frac{1}{y + \hat{y} + \epsilon})$	0.7658
1.000	$\max(y, \hat{y}) + \frac{1}{\max(\hat{y} + 1, y) + \epsilon}$	0.7703
1.000	$\hat{y} + \frac{1}{\min(\hat{y}, y) + \epsilon}$	0.4872
1.000	$\left(\min\left(\frac{1}{y + \hat{y} + \epsilon}, \max(y, \hat{y})\right)\right)^2$	0.7782
0.996	$\min\left(\frac{1}{\max(\hat{y}, y + \hat{y}^2) + \epsilon}, \max(y, (\hat{y}^2)^2)\right)$	0.7787
0.902	$\hat{y}^2 + \frac{1}{\min(y, \hat{y}) + \epsilon}$	0.4884

Table 8: Loss generation result, changing random seed from 0 to 42. ϵ is a tunable parameter representing the smoothing coefficient to prevent numerical errors such as division by zero. Filter out the loss functions whose positive rate is less than 0.9 (i.e., randomly sample 2000 pairs of (y, \hat{y}) and fewer than 1800 pairs lead \hat{y} towards y) in Validation Check.

2415–2421.

[16] Yingqiang Ge, Juntao Tan, Yan Zhu, Yinglong Xia, Jiebo Luo, Shuchang Liu, Zuohui Fu, Shijie Geng, Zelong Li, and Yongfeng Zhang. 2022. Explainable Fairness in Recommendation. *SIGIR* (2022).

[17] Yingqiang Ge, Shuyuan Xu, Shuchang Liu, Zuohui Fu, Fei Sun, and Yongfeng Zhang. 2020. Learning Personalized Risk Preferences for Recommendation. In *Proceedings of the 43rd SIGIR*. 409–418.

[18] Yingqiang Ge, Shuyuan Xu, Shuchang Liu, Shijie Geng, Zuohui Fu, and Yongfeng Zhang. 2019. Maximizing marginal utility per dollar for economic recommendation. In *The World Wide Web Conference*. 2757–2763.

[19] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. 2022. Recommendation as Language Processing (RLP): A Unified Pretrain, Personalized Prompt & Predict Paradigm (P5). *arXiv preprint arXiv:2203.13366* (2022).

[20] Ross Girshick. 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*. 1440–1448.

[21] Bharath Hariharan, Lihui Zelnik-Manor, SVN Vishwanathan, and Manik Varma. 2010. Large scale max-margin multi-label classification with priors. In *ICML*.

[22] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *AcM transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.

[23] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*. 507–517.

[24] Siyu Huang, Xi Li, Zhi-Qi Cheng, Zhongfei Zhang, and Alexander Hauptmann. 2018. Gnas: A greedy neural architecture search method for multi-attribute learning. In *Proceedings of the 26th ACM international conference on Multimedia*. 2049–2057.

[25] Peter J Huber. 1992. Robust estimation of a location parameter. In *Breakthroughs in statistics*. Springer, 492–518.

[26] James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*. IEEE, 1–10.

[27] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. 2016. Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 979–984.

[28] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[29] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. 2019. Auto-WEKA: Automatic model selection and hyperparameter optimization in WEKA. In *Automated Machine Learning*. Springer, Cham, 81–95.

[30] Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The annals of mathematical statistics* 22, 1 (1951), 79–86.

[31] Ching-Pei Lee and Chih-Jen Lin. 2013. A study on L2-loss (squared hinge-loss) multiclass SVM. *Neural computation* 25, 5 (2013), 1302–1323.

[32] 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.

[33] Hao Li, Tianwen Fu, Jifeng Dai, Hongsheng Li, Gao Huang, and Xizhou Zhu. 2021. AutoLoss-Zero: Searching Loss Functions from Scratch for Generic Tasks. *arXiv preprint arXiv:2103.14026* (2021).

[34] Hao Li, Chenxin Tao, Xizhou Zhu, Xiaogang Wang, Gao Huang, and Jifeng Dai. 2021. Auto Seg-Loss: Searching Metric Surrogates for Semantic Segmentation. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=MJAqnaC2vO1>

[35] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*. 2980–2988.

[36] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*. 19–34.

[37] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=S1eYHoC5FX>

[38] Peidong Liu, Gengwei Zhang, Bochao Wang, Hang Xu, Xiaodan Liang, Yong Jiang, and Zhenguo Li. 2021. Loss Function Discovery for Object Detection via Convergence-Simulation Driven Search. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=5jzlpHvvRk>

[39] Qingliang Liu and Jinmei Lai. 2020. Stochastic Loss Function. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 04 (Apr. 2020), 4884–4891. <https://doi.org/10.1609/aaai.v34i04.5925>

[40] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. 2017. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 212–220.

[41] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. 2016. Large-Margin Softmax Loss for Convolutional Neural Networks. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*. Maria Florina Balcan and Kilian Q. Weinberger (Eds.). PMLR, New York, New York, USA, 507–516. <https://proceedings.mlr.press/v48/liu16.html>

[42] Hamed Masnadi-Shirazi, Vijay Mahadevan, and Nuno Vasconcelos. 2010. On the design of robust classifiers for computer vision. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 779–786.

[43] Hamed Masnadi-Shirazi and Nuno Vasconcelos. 2008. On the design of loss functions for classification: theory, robustness to outliers, and SavageBoost. In *Proceedings of the 21st International Conference on Neural Information Processing Systems*. 1049–1056.

[44] Tom Mitchell. 1997. *Machine learning*. McGraw hill Burr Ridge.

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

[46] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized Evolution for Image Classifier Architecture Search. *Proceedings of the AAAI Conference on Artificial Intelligence* 33, 01 (Jul. 2019), 4780–4789. <https://doi.org/10.1609/aaai.v33i01.33014780>

[47] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Sue-matsu, Jie Tan, Quoc V Le, and Alexey Kurakin. 2017. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*. PMLR, 2902–2911.

[48] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. 2004. Are loss functions all the same? *Neural computation* 16, 5 (2004), 1063–1076.

[49] Reuven Rubinstein. 1999. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability* 1, 2 (1999), 127–190.

[50] Shaoyun Shi, Hanxiang Chen, Weizhi Ma, Jiaxin Mao, Min Zhang, and Yongfeng Zhang. 2020. Neural Logic Reasoning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1365–1374.

[51] Frank Hutter Thomas Elsken, Jan Hendrik Metzen. 2018. Simple and efficient architecture search for Convolutional Neural Networks. <https://openreview.net/forum?id=SySaJ0xCZ>

[52] 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.

[53] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3 (1992), 229–256.

[54] Cort J Willmott and Kenji Matsuura. 2005. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate research* 30, 1 (2005), 79–82.

[55] Haowen Xu, Hao Zhang, Zhiting Hu, Xiaodan Liang, Ruslan Salakhutdinov, and Eric Xing. 2019. AutoLoss: Learning Discrete Schedule for Alternate Optimization. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=BjgK6iA5KX>

[56] Shuyuan Xu, Yingqiang Ge, Yunqi Li, Zuohui Fu, Xu Chen, and Yongfeng Zhang. 2021. Causal collaborative filtering. *arXiv preprint arXiv:2102.01868* (2021).

[57] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. 2018. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306* (2018).

[58] Xiangyu Zhao, Haochen Liu, Wenqi Fan, Hui Liu, Jiliang Tang, and Chong Wang. 2021. AutoLoss: Automated Loss Function Search in Recommendations. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14–18, 2021*, Feida Zhu, Beng Chin Ooi, and Chunyan Miao (Eds.). ACM, 3959–3967. <https://doi.org/10.1145/3447548.3467208>

[59] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings*. OpenReview.net. <https://openreview.net/forum?id=r1Ue8Hcxg>

[60] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8697–8710.