

# A Survey on Dropout Methods and Experimental Verification in Recommendation

Yangkun Li, *Member, IEEE*, Weizhi Ma, *Member, IEEE*, Chong Chen, *Member, IEEE*,  
Min Zhang, *Member, IEEE*, Yiquan Liu, *Senior Member, IEEE*, Shaoping Ma, and Yuekui Yang

**Abstract**—Overfitting is a common problem in machine learning, which means the model too closely fits the training data while performing poorly in the test data. Among various methods of coping with overfitting, dropout is one of the representative ways. From randomly dropping neurons to dropping neural structures, dropout has achieved great success in improving model performances. Although various dropout methods have been designed and widely applied in past years, their effectiveness, application scenarios, and contributions have not been comprehensively summarized and empirically compared by far. It is the right time to make a comprehensive survey.

In this paper, we systematically review previous dropout methods and classify them into three major categories according to the stage where dropout operation is performed. Specifically, more than seventy dropout methods published in top AI conferences or journals (e.g., TKDE, KDD, TheWebConf, SIGIR) are involved. The designed taxonomy is easy to understand and capable of including new dropout methods. Then, we further discuss their application scenarios, connections, and contributions. To verify the effectiveness of distinct dropout methods, extensive experiments are conducted on recommendation scenarios with abundant heterogeneous information. Finally, we propose some open problems and potential research directions about dropout that worth to be further explored.

**Index Terms**—Dropout, Neural Network Model, Recommendation.

## 1 INTRODUCTION

### 1.1 Backgrounds

OVERFITTING is a common problem in the training process of neural network models [1]. Due to the large number of parameters and strong fitting ability, most neural models perform well on the training set, while they may perform poorly on the test set. Some methods have been proposed in previous studies to address the overfitting problem, such as adding a regularization term to penalize the total size of model parameters [2] and applying Batch Normalization [3] or Weight Normalization [4] to regularize deep neural networks.

In 2012, Hinton et al. proposed Dropout [5] to cope with overfitting. The idea is to randomly drop neurons of the neural network during training. This is, in each parameter update, only part of the model parameters will be updated. Through this process, it can prevent complex co-adaptations of neurons on training data. It is important to note that in the testing phase, dropout must be disabled, and the whole network is used for prediction. From this beginning, numerous dropout-based training methods have been proposed and achieved better performances.

In the beginning, this dropout-based training method was applied only to fully connected layers [5], [6], [7]. Later, it is extended to more network structures such as convolu-

tional layers in convolutional neural networks (CNNs) [8], [9], [10], residual networks (ResNet) [11], [12], [13], recurrent layers in recurrent neural networks (RNNs) [14], [15], [16], etc. In terms of the stage where the dropout operation is performed, there are not only dropout of model structure, but also dropout of input information [17], [18], [19] and dropout of embeddings [20], [21], [22]. In terms of contributions, dropout methods were first used only to prevent overfitting. Besides, many studies have been made to explore other aspects of its usefulness, such as model compression [23], [24], [25], model uncertainty measurement [26], data augmentation [27], enhancing data representations in the pre-training phase [28], and prevention of over-smoothing problem in graph neural networks [29].

Despite their wide applications, a dropout method that works for one model structure may have no significant effect on another. For example, the use of standard dropout in CNNs does not improve the effect significantly [8]. The same is true for the direct use of standard dropout to the recurrent connections of RNNs [15]. Dropout at different stages of a machine learning task achieves different purposes. Therefore, in this paper, we classify these wide varieties of dropout methods and summarize their effectiveness, application scenarios, connections, and contributions.

Most of the methods with commonality also fail to compare their results under the same scenario. Since dropout methods are applied to different forms of input information and model structures, a proper comparison scenario should have rich heterogeneous input information and a variety of different model structures. With the rapid development of the internet, various recommendation models have been proposed and widely used to extract user and item features to improve user experience in many online scenarios [30],

- Y. Li, C. Chen, M. Zhang, Y. Liu, S. Ma, and Y. Yang are with Department of Computer Science and Technology, Institute for Artificial Intelligence, Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing 100084, China.  
E-mail: lyk21@mails.tsinghua.edu.cn, z-m@tsinghua.edu.cn
- W. Ma is with Institute for AI Industry Research (AIR), Tsinghua University, Beijing 100084, China.
- Y. Yang is also with Tencent AI Platform Department, China.
- Min Zhang is the corresponding author.

[31]. They utilize various heterogeneous information, such as: user and item interaction histories, content features, or social connections [32]. Such a variety of models and heterogeneous input information provides a suitable environment for our comparisons and verification of different dropout methods. Therefore, we evaluate and compare different dropout methods under the recommendation scenario to fairly compare the effect of these dropout methods, providing references for future works related to dropout.

## 1.2 Contributions

Our contributions in this paper are three fold:

First, we provide a comprehensive review of more than seventy dropout methods. We propose a new taxonomy based on the stage where dropout operations are performed in machine learning tasks. Each category is then supplemented with operation granularity and application scenarios for more detailed classification and discussion.

Second, we discuss the connections between dropout methods of different categories and summarize their various contributions other than preventing overfitting.

Third, we experimentally investigate the effect of dropout methods in recommendation scenarios. The rich heterogeneous input information makes recommendation scenarios suitable for the comparison between different types of dropout methods. We verify and compare dropout methods' effectiveness under a unified experimental environment. And finally, we provide potential research directions about dropout methods.

## 1.3 Outline

The organization of this paper is as follows: Section 2 introduces background concepts of dropout methods and recommendation systems. In Section 3, we review dropout methods according to the stage where the dropout operation is performed in a machine learning task. We summarize their applications on different neural models and discuss their connections. In Section 4, we analyze the contributions of dropout methods other than preventing overfitting. In Section 5, we present an experimental verification of dropout methods on recommendation models. In Section 6, we provide further discussions on dropout methods and analyze potential research directions in this field. Finally, we conclude the entire paper in Section 7.

# 2 BACKGROUND CONCEPTS

This section introduces the fundamental knowledge of dropout and recommender systems by summarizing related works of these two fields.

## 2.1 Dropout methods

Dropout is a class of training methods effectively coping with overfitting. Hinton et al. [5] proposed the original dropout, whose idea is to randomly drop neurons of the neural network during training. Through this process, it prevents complex co-adaptations of neurons on training data. From this beginning, numerous drop-based methods have been proposed, achieving better results and higher performances. For example, DropConnect [33] randomly

drops neuron connections instead of neurons, while Annealed Dropout [34] and Curriculum Dropout [35] adjust dropout ratio throughout the training process.

Besides dropping individual neurons, a series of dropout methods that drop neuron groups are proposed for certain neural model structures. For CNNs, SpatialDropout [8] randomly drops feature map, and DropBlock [36] drops continuous region of neurons to prevent overfitting. For RNNs, early applications of dropout [14] only drop feed-forward connections, in order to preserve the memory ability of RNN. Later approaches including RNNDrop [16] and Recurrent Dropout [37] allow dropping recurrent connections as well. For ResNets, there are also specified dropout methods such as Stochastic Depth [11] and ShakeDrop [38].

Except for dropping model structure during training, dropping input information or embeddings of input data is also applied in several scenarios. DropoutNet [20] and ACCM [21] drop embeddings of users and items in recommendation scenarios to handle the cold-start problem. BERT [28] randomly masks tokens at pre-training stage, enhancing data representations in NLP tasks. CutOut [18] and GridMask [39] randomly drop part of the input images during training, serving as a regularization and data augmentation technique. GraphSAGE [19] and DropEdge [29] randomly drop nodes or edges during GCN training, preventing overfitting and over-smoothing problem in GCN.

The former survey about dropout methods [40] was made by Labach et al. in 2019, which is the only comprehensive survey about this topic. It reviews dropout methods from the aspect of neural models that dropout methods are performed on, including fully connected layers, convolutional layers and recurrent layers.

Our work has three major differences with this former one. First, we cover a wider range of dropout methods, including those proposed in recent three years. Especially new methods published in the top AI conferences. Second, we present a more precise and general classification. In the former survey, dropout methods were classified according to the neural models they perform on, which means classification requires that when a new model structure appears, the corresponding dropout methods are classified into a new category. Nevertheless, dropout methods themselves may be similar to the ones that already existed before, so it seems cumbersome to put them into a new category just because their applications change. Our work classifies dropout methods according to the stage where dropout operations are performed in machine learning tasks. With this setting, new methods must fit into one existing category, making our classification more reasonable. We also review these methods from the perspective of application scenarios, their interconnections, and contributions other than preventing overfitting. Third, there is no experimental comparison of the effectiveness of dropout methods in [40], while we experimentally verify and compare their effectiveness under recommendation scenarios.

## 2.2 Recommender Systems

In terms of the input information types, recommender systems are mainly categorized into collaborative filtering (CF) based, content-based (CB), and hybrid [41]. CF based

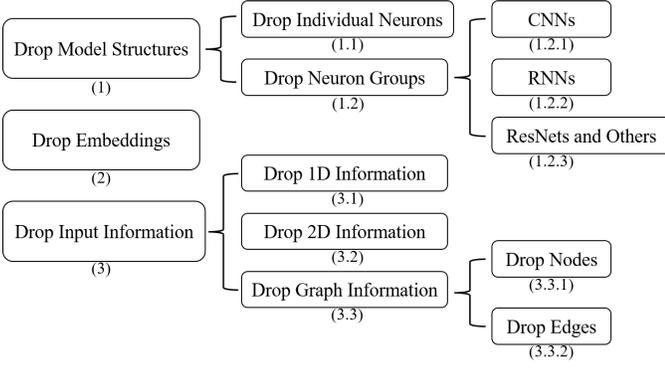


Fig. 1. Classification of dropout methods. The number below each box is referring to the corresponding category of dropout methods.

recommender systems make predictions based on the interaction histories of users and items [42], [43], while CB recommender systems use content feature of users and items [44]. Hybrid recommender systems use multiple types of input information to extract interaction similarity as well as content similarity, such as users’ social networks [45], [46] and item reviews [47], [48]. In recent years, some recommendation algorithms specified for certain tasks utilize specific form of input data, such as in sequential recommendation the input data is structured as temporal sequences [49], [50], and in graph recommendation the input data is structured as graphs [51], [52]. However, the input information in real world scenarios may not be sufficient for the recommender systems to make good predictions, and this problem is called *Cold Start*. Works addressing cold-start problem have been emerging in recent years [53], [54], [55], [56], [57], [58].

Such rich variety of input information in recommendation scenario provides a good environment for the verification and comparison of different dropout methods. Besides conducting a survey on dropout methods, we also do experimental verification and put them under the same scenario for comparison.

### 3 SURVEY OF DROPOUT METHODS

In this section, we review papers of dropout methods. Based on the stage where the dropout operation is performed in a machine learning algorithm, we classify them into three main categories: drop model structures (Section 3.1), drop embeddings (Section 3.2), and drop input information (Section 3.3) as shown in Figure 1. We introduce how these methods perform dropout, their effectiveness, and their applications in different neural models. Finally, we discuss the interconnections between dropout methods in different categories (Section 3.4).

#### 3.1 Drop Model Structures

Methods in this category drop model structures, which is, randomly setting part of neuron outputs to zero or other value during training. We first introduce methods dropping individual neurons, then methods dropping neuron groups.

##### 3.1.1 Drop Individual Neurons

Hinton et al. [5] first proposed the standard dropout in 2012, and further detailed in [6]. Specifically, for a neural network

with  $L$  hidden layers,  $l \in 1, \dots, L$ ,  $z^{(l)}$  and  $y^{(l)}$  are the input and output of the  $l$ th layer respectively, and  $y^{(0)}$  is the input of the network.  $W^{(l)}$  and  $b^{(l)}$  are the weight matrix and bias vectors (biases) of the  $l$ th layer, respectively. The standard feed-forward operation when dropout is not performed is

$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)} \mathbf{y}^{(l)} + b_i^{(l+1)}, \quad y_i^{(l+1)} = f(z_i^{(l+1)}) \quad (1)$$

Dropout sets a proportion of  $p$  of the neuron outputs to zero during training, where  $p \in (0, 1)$  is the dropout ratio. When performing dropout, the feed-forward operation becomes

$$\begin{aligned} r_j^{(l)} &\sim \text{Bernoulli}(p), \quad \tilde{\mathbf{y}}^{(l)} = \mathbf{r}^{(l)} * \mathbf{y}^{(l)} \\ z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \tilde{\mathbf{y}}^{(l)} + b_i^{(l+1)}, \quad y_i^{(l+1)} = f(z_i^{(l+1)}) \end{aligned} \quad (2)$$

During testing no dropout is performed so all neurons have output. To keep training and testing conditions consistent, all the weights need to be multiplied by  $p$  during testing, i.e., take  $\mathbf{W}_{test}^{(l)} = p\mathbf{W}^{(l)}$ ; or it multiplies all outputs by  $1/p$  during training so that the expected output is consistent with testing time. Srivastava et al. [6] also mentioned that besides generating the dropout mask from Bernoulli distribution, it is also possible to generate it from a continuous distribution with the same expectation and variance, such as Gaussian distribution  $\mathcal{N} \sim (1, \alpha)$ ,  $\alpha = p/(1-p)$ . The standard dropout has achieved good performance since it was proposed, and the authors showed the effectiveness of standard dropout on image classification tasks [59] in 2013.

Following the standard dropout, a series of methods that randomly drop individual neurons have been proposed.

Ba et al. [7] proposed Standout in 2013. The method treats dropout as a Bayesian learning process. A Bayesian network (Belief Network) is added above the original network to control the dropout ratio:

$$y = f(\mathbf{W}\mathbf{x}) \circ m, \quad m \sim \text{Bernoulli}(g(\mathbf{W}_s\mathbf{x})) \quad (3)$$

where  $\mathbf{W}_s$  and  $g(\cdot)$  is the weight and the activation function of each layer of the Bayesian network. In application the authors find that  $\mathbf{W}_s$  can be chosen as an affine transformation of  $\mathbf{W}$ , and the test output  $y$  is computed as

$$\mathbf{W}_s = \alpha\mathbf{W} + \beta, \quad y = f(\mathbf{W}\mathbf{x}) \circ g(\mathbf{W}_s\mathbf{x}) \quad (4)$$

Wang and Manning [60] proposed Fast Dropout in 2013. In standard dropout, only one of the possible network structures is sampled at a time; a proportion of  $p$  neurons are not trained in each epoch, making the network’s training slower. Fast Dropout, on the other hand, explains dropout from a Bayesian perspective, showing that the output of a layer that has undergone dropout can be considered as sampling from a potential approximate Gaussian distribution. We can then sample directly from this distribution to obtain results or use its parameters to propagate information about the entire dropout set. This allows for faster training than the standard dropout and is also known as Gaussian Dropout.

Wan et al. [33] proposed DropConnect in 2013. Compared to standard dropout which randomly zeroes the output of neurons, DropConnect randomly zeroes elements of the weight matrix of each layer:

$$\mathbf{y} = f((\mathbf{W} \circ \mathbf{M})\mathbf{x}), \quad m_{ij} \sim \text{Bernoulli}(1-p) \quad (5)$$

This approach removes “connections” in fully connected layers, hence the name “DropConnect”.

TABLE 1  
Table of methods that drop model structures.

Method	Year	Category	Brief Description	Original Scenario	Source
Dropout [5], [6]	2012	1.1†	Randomly drop neurons	FCL*	JMLR
Standout [7]	2013	1.1	Add a Bayesian NN to control the dropout ratio	FCL	NeurIPS
Fast Dropout [60]	2013	1.1	Sample outputs directly from a distribution	FCL	ICML
DropConnect [33]	2013	1.1	Drop weights instead of neurons	FCL	ICML
Maxout [61]	2013	1.1	Computes several outputs for each input	FCL	ICML
Annealed Dropout [34]	2014	1.1	Dropout ratio decreases with training epochs	FCL	SLT
Variational Dropout [62]	2015	1.1	Dropout ratio can be learned in training	FCL	NeurIPS
Monte Carlo Dropout [26]	2016	1.1	Interpret dropout as a Bayesian approximation of deep Gaussian process	FCL	ICML
DropIn [63]	2016	1.1	Pass dropped values directly to the next layer	FCL	CVPR
Evolutional Dropout [64]	2016	1.1	Calculate dropout ratio from input	FCL	NeurIPS
Concrete Dropout [65]	2017	1.1	Automatically adjust dropout ratio compared to Monte Carlo Dropout	FCL	NeurIPS
Curriculum Dropout [35]	2017	1.1	Dropout ratio increases with training epochs	FCL	ICCV
Targeted Dropout [25], [66]	2018	1.1	Dropout for neural pruning	FCL	NeurIPS
Ising-Dropout [67]	2019	1.1	Incorporate Ising model	FCL	ICASSP
EDropout [68]	2021	1.1	Use EBM to decide pruning state	FCL	TNNLS
LocalDrop [69]	2021	1.1	Based on local Rademacher complexity	FCL	TPAMI
SimCSE [70]	2021	1.1	Data augmentation by dropout twice	FCL	EMNLP
Child-Tuning [71]	2021	1.1	Mask gradient when back-propagation	FCL	EMNLP
R-Drop [72]	2021	1.1	Dropout twice to regularize	FCL	arxiv
AS-Dropout [73]	2021	1.1	Adaptive sparse dropout	FCL	Neurocomput.
SpatialDropout [8]	2015	1.2.1†	Drop feature maps in CNN	CNN	CVPR
Max-pooling Dropout [9]	2015	1.2.1	Drop neurons before pooling layer	CNN	NN
Convolutional Dropout [9]	2015	1.2.1	Drop neurons before convolutional layer	CNN	NN
Max-drop [10]	2016	1.2.1	Drop feature maps with high activations	CNN	ACCV
Stochastic Dropout [10]	2016	1.2.1	Dropout ratio sampled from normal distribution	CNN	ACCV
DropBlock [36]	2018	1.2.1	Drop contiguous regions on each feature map	CNN	NeurIPS
Spectral Dropout [74]	2018	1.2.1	Dropout in the frequency domain	CNN	NN
Drop-Conv2d [75]	2019	1.2.1	Dropout before convolution instead of BN	CNN	arxiv
Weighted Channel Dropout [76]	2019	1.2.1	Drop weighted feature channels	CNN	AAAI
CorrDrop [77]	2021	1.2.1	Drop neurons according to feature correlation	CNN	PR
LocalDrop [69]	2021	1.2.1	Based on local Rademacher complexity	CNN	TPAMI
AutoDropout [78]	2021	1.2.1	Optimize dropout patterns by RL	CNN	AAAI
Vanilla drop for RNN [14], [15]	2014	1.2.2†	Drop feed-forward connections only	RNN	ICFHR
RNNDrop [16]	2015	1.2.2	One dropping mask for each layer	RNN	ASRU
Variational RNN Dropout [79]	2015	1.2.2	Variational inference based dropout	RNN	NeurIPS
Recurrent Dropout [37]	2016	1.2.2	Drop only the vectors generating hidden states	RNN	COLING
Zoneout [80]	2016	1.2.2	Residual connections between timestamps	RNN	ICLR
Weighted-dropped LSTM [81]	2017	1.2.2	Drop weights like DropConnect	RNN	ICLR
Fraternal Dropout [82]	2018	1.2.2	Train two identical RNNs with different dropout masks	RNN	ICLR
Stochastic Depth [11]	2016	1.2.3†	Drop blocks and retain only residual connections	ResNet	ECCV
Shakeout [12]	2016	1.2.3	Assign new weights to neurons	ResNet	AAAI
Whiteout [13]	2016	1.2.3	Introduce Gaussian noise compared to Shakeout	ResNet	arxiv
Swapout [83]	2016	1.2.3	A synthesis of standard Dropout and Stochastic Depth	ResNet	NeurIPS
DropPath [84]	2016	1.2.3	Drop subpaths in Fractalnet	DNN	ICLR
Shake-Shake [85]	2017	1.2.3	Assign weights in 3-way ResNet	ResNet	arxiv
ShakeDrop [38]	2018	1.2.3	Improve Shake-Shake to other form of ResNet	ResNet	IEEE Access
Scheduled DropPath [86]	2018	1.2.3	Dropout ratio increases linearly	DNN	CVPR
DropHead [87]	2020	1.2.3	Drop attention heads of Transformer	Transformer	EMNLP

† 1.1 refers to dropping individual neurons, 1.2.1 dropping 2D neuron groups, 1.2.2 dropping recurrent connections, and 1.2.3 dropping residual connections or others.

\* FCL refers to Fully Connected Layers.

Goodfellow et al. [61] proposed Maxout in 2013. Maxout is an improvement of standard dropout [5]. Specifically, the output of each hidden layer is computed as

$$h_i(\mathbf{x}) = \max_{j \in [1, k]} z_{ij}, \text{ where } z_{ij} = \mathbf{x}^T \mathbf{W}_{:ij} + b_{ij} \quad (6)$$

where the weight matrix  $\mathbf{W} \in \mathbb{R}^{d \times m \times k}$  and the bias vector  $\mathbf{b} \in \mathbb{R}^{m \times k}$  are training parameters.  $\mathbf{x}$ ,  $d$ ,  $m$ , and  $k$  are the input, the input dimension, the output dimension, and Maxout parameter, respectively. As can be seen, unlike in standard dropout [5] where only one output is computed for each input at each layer, Maxout computes  $k$  outputs

for each input at each layer. Then it takes the maximum of the  $k$  outputs as the output of this layer. This operation makes Maxout essentially a nonlinear activation function, which gives the method its name. Maxout has been applied in computer vision tasks including object detection [88].

Kingma et al. [62] proposed Variational Dropout in 2015. This work studies Stochastic Gradient Variational Bayesian Inference (SGVB) problem and found its connection with dropout: Gaussian Dropout proposed in [6] is a local reparameterization of SGVB. This paper thus proposes Variational Dropout so that the dropout ratio  $p$  is not a pre-

set hyperparameter that requires human adjusting but a parameter that can be learned through training. In [23] the authors show that Variational Dropout is a efficient way to perform model compression, which can significantly reduce the number of parameters of neural networks with a negligible decrease of accuracy.

Gal and Ghahramani proposed Monte Carlo Dropout [26] in 2016. The authors interpret dropout as a Bayesian approximation of deep Gaussian processes. The output of a deep Gaussian process is a probability distribution, and using standard dropout in testing phase can estimate some properties of this potential distribution. For example, the estimated variance can be used to characterize the uncertainty of the model output, and this estimating method is called Monte Carlo Dropout. Monte Carlo Dropout has been applied in a series of works [89], [90], [91], [92]. It can mitigate the problem of representing uncertainty in deep learning more efficiently without sacrificing test accuracy.

Smith et al. [63] proposed DropIn in 2016. The feed-forward operation for each layer performing DropIn is:

$$\begin{aligned} \tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} \circ \mathbf{y}^{(l)}, \quad \mathbf{z}^{(l+1)} = \mathbf{W}^{(l+1)} \tilde{\mathbf{y}}^{(l)} + \mathbf{b}^{(l+1)} \\ \mathbf{y}^{(l+1)} &= f(\mathbf{z}^{(l+1)}) + (1 - \mathbf{r}^{(l)}) \circ \mathbf{y}^{(l)} \end{aligned} \quad (7)$$

As can be seen, in addition to passing the kept outputs ( $\mathbf{r}^{(l)} \circ \mathbf{y}^{(l)}$ ) to the next layer, DropIn also passes the values of dropped positions ( $(1 - \mathbf{r}^{(l)}) \circ \mathbf{y}^{(l)}$ ) directly to the next layer without going through the nonlinear activation function. This operation increases the depth of the network while avoiding vanishing gradient problem while serving the same regularization effect as standard dropout.

Li et al. [64] proposed Evolutional Dropout in 2016. Intuitively, the importance of neurons corresponding to different features in a neural network is different, so their corresponding dropout probabilities should be different. This paper applies this idea to both shallow and deep neural networks: for shallow networks, the dropout ratio is calculated from the second-order statistics of the input data features; for deep networks, the dropout ratio of each layer is calculated in real-time from the output of that layer of each batch. Compared with the standard dropout, Evolutional Dropout improves the accuracy of the results while greatly increasing the convergence speed.

Gal et al. [65] proposed Concrete Dropout in 2017. Concrete Dropout is an improvement on Monte Carlo Dropout [26]. Monte Carlo Dropout can estimate the model uncertainty, achieved by performing a grid search on the dropout ratio parameter. This is unfeasible for deeper models (e.g., those in computer vision tasks) and reinforcement learning models because of the excessive computational time and resources. Based on the development of Bayesian learning, this paper uses a continuous relaxation of the dropout discrete mask. Concrete Dropout proposes a new objective function which allows automatic adjustment of the dropout parameters on large models, reducing the time required for experiments. It also allows the agent in reinforcement learning to dynamically adjust its uncertainty as the training process goes on and more training data is exposed. Experiments show that Concrete Dropout can decrease the time of model training by weeks by automatically learning the dropout probabilities in reinforcement learning compared to conventional dropout methods.

Rennie et al. [34] proposed Annealed Dropout in 2014. As the name implies, “annealed” dropout is the decrease of dropout ratio as the number of training epochs increases:

$$p[t] = p[t - 1] + \alpha_t(\theta) \quad (8)$$

$\alpha_t(\theta)$  is the parameter that controls the dropout ratio. A simple approach is to decrease linearly: the initial dropout ratio is  $p[0]$  and decreases to 0 after  $N$  rounds:

$$p[t] = \max(0, 1 - \frac{t}{N})p[0] \quad (9)$$

The explanation for this approach is that at the beginning when we are exposed to little training data, we only need to “explain” the data with a simple model, i.e., we only make fewer neurons work, and more neurons are dropped out. Later on, when more training data is exposed, we can allow a more “complex” model to “explain” the data, reducing the dropout ratio and making more neurons work.

Morerio et al. [35] proposed Curriculum Dropout in 2017. Inspired by curriculum learning [93], in contrast to Annealed Dropout [34], Curriculum Dropout increases dropout ratio as the number of training epochs increases. The explanation for this approach is to simulate the learning process from easy to difficult in human learning: the dropout ratio is small at the beginning, introducing less noise and analogous to the “easy” task; then the dropout ratio increases, introducing more noise and making the task “harder”.

Gomez et al. [25], [66] proposed Targeted Dropout in 2018. Neural pruning is a neural network compression method [94] that can be used to reduce the number of network parameters and improve training efficiency. The Targeted Dropout selects and drops those neurons whose absence can make the model most suitable for neural network pruning, facilitating model compression. Targeted Dropout is able to achieve better performance with only half of total number of parameters compared to the original networks without dropout.

Salehinejad and Valaee [67] proposed Ising-Dropout in 2019. Borrowing the concept of Ising model in physics, Ising-Dropout adds an image Ising model to a neural network to detect and drop out those neurons that are least useful. It could compress the number of parameters up to 41.18% and 55.86% for the classification task on the MNIST and Fashion-MNIST datasets respectively. The authors also proposed EDropout [68] working for neural pruning in 2021. It utilizes an Energy-Based Model (EBM) to decide the pruning state.

In 2020, İrsoy and Alpaydın [95] proposed a dropout method for hierarchically gated models [96], [97] to prevent overfitting in decision-tree-like models. Ragusa et al. [98] employ dropout on Internet of Things (IoT) models.

Gao et al. [70] proposed SimCSE in 2021. SimCSE is a contrastive learning method for NLP. Specifically, it performs dropout twice for the same instance to get two positive samples and treats all other in-batch instances as negative. Performing data augmentation by dropout twice achieves good results on this contrastive learning task. Child-Tuning [71] is another application of dropout in NLP. It randomly masks gradient when back-propagation.

Liang et al. [72] proposed R-Drop (“R” for “Regularized”) in 2021. R-Drop generalized the idea of “dropout twice” [70] from contrastive learning to general tasks. A

training instance goes through the network twice with random dropout. On the one hand, we want the two predictions as close as possible to the label, by which we compute cross-entropy loss  $\mathcal{L}^{(CE)}$ , on the other hand, we want the two predictions as close as possible to each other, by which we compute Kullback-Leibler divergence loss  $\mathcal{L}^{(KL)}$ . Thus we get final loss function consists of two parts:

$$\mathcal{L} = \mathcal{L}^{(CE)} + \alpha \mathcal{L}^{(KL)} \quad (10)$$

and the second part is “regularizing” the first part.

Chen and Yi [73] proposed AS-Dropout (Adaptive Sparse Dropout) in 2021. AS-Dropout calculates dropout probability adaptively according to the neuron’s activation function, such that only a small proportion of neurons are active in each training epoch.

Dropout methods that drop individual neurons are summarized in Table 1. The application scenario for the dropout of individual neurons is usually fully connected layers. The basic operation of this type of methods is easy to implement and can be applied to a wide range of neural models. It is likely to lead to a stable improvement on the model performance in most cases, as will be shown in Section 5. The generality and effectiveness has made it the most popular type of dropout methods. However, it is not suitable for models with specific structure (e.g. CNN, RNN, Transformer), for which the dropout methods usually drop neuron groups.

### 3.1.2 Drop Neuron Groups

Dropout methods in 3.1.1 are mainly performed on fully connected layers. For neural networks with special structures, such as convolutional neural networks, residual networks, and recurrent neural networks, neurons are aggregated together forming specific structures to perform certain functions. Directly dropping individual neurons randomly may not have expected effect on these networks, so a series of dropout methods have been proposed specified for them.

**3.1.2.1 CNNs:** Tompson et al. [8] first proposed SpatialDropout specifically for convolutional neural networks in 2014. In CNNs, for the same feature map, all pixel features within the coverage of the same convolutional kernel are used to compute the output of the next layer, resulting a strong gradient correlation between adjacent pixels. The standard dropout removes individual pixel features randomly, which has little effect on reducing the interdependence between neurons, and is therefore ineffective. In contrast, for a feature tensor with size  $n_{\text{feats}} \times \text{height} \times \text{width}$ , SpatialDropout selects only  $n_{\text{feats}}$  dropout values, i.e., the whole feature map is either dropped for all or kept for all. This reduces the interdependence between neurons in CNN and has a good regularization effect.

Wu and Gu [9] proposed Max-pooling Dropout in 2015. For a neural network containing convolutional layers and pooling layers, if the  $l$ th layer is immediately followed by a pooling layer, the feed-forward operation is expressed as

$$a_j^{(l+1)} = \text{pool}(a_1^{(l)}, \dots, a_i^{(l)}, \dots, a_n^{(l)}), \quad i \in R_j^{(l)} \quad (11)$$

where  $R_j^{(l)}$  is the  $j$ th pooling region of the  $l$ th layer,  $a_i^{(l)}$  is the activation value of each neuron, and  $\text{pool}()$  is the pooling function. Two common choices are average pooling, which averages the outputs of all neurons, and max-pooling, which

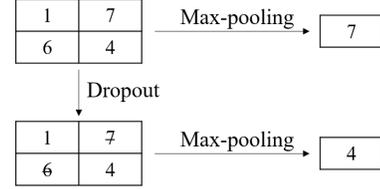


Fig. 2. Max-pooling Dropout [9] drops neuron outputs before they are passed to pooling layer.

takes the maximum of all neuron outputs. The authors take the latter one. Max-pooling Dropout randomly drops out the output  $a_i^{(l)}$  of individual neurons during training phase, which is then passed into the pooling layer:

$$\begin{aligned} \hat{a}_i^{(l)} &= m_i^{(l)} * a_i^{(l)}, \quad m_i^{(l)} \sim \text{Bernoulli}(p) \\ a_j^{(l+1)} &= \text{pool}(\hat{a}_1^{(l)}, \dots, \hat{a}_i^{(l)}, \dots, \hat{a}_n^{(l)}), \quad i \in R_j^{(l)} \end{aligned} \quad (12)$$

as shown in Figure 2. This is therefore equivalent to selecting neuron outputs from a multinomial distribution:

$$\Pr(a_j^{(l+1)} = a_i^{(l)}) = p_i = pq^{n-i}, \quad i = 1, 2, \dots, n \quad (13)$$

If the  $l$ th layer is immediately followed by a convolutional layer, this paper also proposes Convolutional Dropout. The feed-forward operation in the training phase is

$$\begin{aligned} m_k^{(l)}(i) &\sim \text{Bernoulli}(p), \quad \hat{a}_k^{(l)} = a_k^{(l)} * m_k^{(l)}, \\ z_j^{(l+1)} &= \sum_{k=1}^{n^{(l)}} \text{conv}(W_j^{(l+1)}, \hat{a}_k^{(l)}), \\ a_j^{(l+1)} &= f(z_j^{(l+1)}). \end{aligned} \quad (14)$$

where  $a_k^{(l)}$  is the  $k$ th feature map of the  $l$ th layer. No dropout is performed during testing and all neuron outputs need to be multiplied by the retain probability of the training phase.

In 2016, Park and Kwak [10] makes two improvements to SpatialDropout [8]. The first is to select and drop out high activation values on the feature maps or channels; the second is that the dropout ratio is not a fixed value but is obtained by sampling from a normal distribution. The authors refer to these two improved dropout methods as Max-drop and Stochastic Dropout, respectively.

Ghiasi et al. [36] proposed DropBlock in 2018. For each layer of the feature map, DropBlock randomly drops multiple contiguous regions of size  $\text{block\_size} \times \text{block\_size}$ . When  $\text{block\_size} = 1$ , DropBlock is reduced to standard dropout; when  $\text{block\_size} = \text{feature\_map\_size}$ , i.e., one block can cover the whole layer of feature map, DropBlock is equivalent to SpatialDropout [8].

Khan et al. [74] proposed Spectral Dropout in 2018. A Spectral Dropout operation is added between two layers of a CNN. The operation has three steps: transforming activation values of the previous layer to frequency domain; dropping the components below a certain threshold in frequency domain; and changing back to the original value domain. The feed-forward operation between two layers without Spectral Dropout is expressed in the following equation:

$$\mathbf{A}'_l = f(\mathbf{F}_l \otimes \mathbf{A}_{l-1} + \mathbf{b}_l) \quad (15)$$

When performing Spectral Dropout, the operation becomes

$$\mathbf{A}_l = \mathcal{T}^{-1}(\mathbf{M} \circ \mathcal{T}(f(\mathbf{F}_l \otimes \mathbf{A}_{l-1} + \mathbf{b}_l))) \quad (16)$$

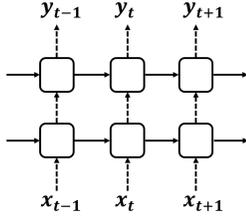


Fig. 3. Vanilla dropout for RNNs, which only drops feed-forward connections but not recurrent connections. [15]

where  $\mathcal{T}$  denotes the frequency transformation and  $\mathbf{M}$  is the masking matrix of dropout in frequency domain. This approach serves to filter input noises and effectively speeds up the convergence of network training.

In 2019, Cai et al. [75] analyze why standard dropout does not work in convolutional neural networks: it conflicts with the effect of Batch Normalization [3]. It is experimentally verified that putting dropout operation before convolution operation instead of batch normalization operation can effectively improve the dropout effect. Then the authors proposed Drop-Conv2d method to improve the training effect by combining dropout at feature-channel level and dropout at forward-path level in CNN.

Hou and Wang [76] proposed Weighted Channel Dropout for feature channel dropout in 2019. The operation steps are in two stages: scoring feature channels and selecting feature channels. In the first stage, feature channels are scored using Global Average Pooling (GAP) method:

$$score_i = \frac{1}{W \times H} \sum_{j=1}^W \sum_{k=1}^H x_i(j, k) \quad (17)$$

In the second stage, the weighted random selection (WRS) and random number generation (RNG) steps are used to select feature channels for dropout and retention.

Zeng et al. [77] proposed CorrDrop in 2021. CorrDrop drops out CNN neurons based on their feature correlation and can do it in both spatial manner and channel manner.

Lu et al. [69] proposed LocalDrop in 2021. This regularization method is based on theoretical analysis of local Rademacher complexity and can be applied to both fully connected layers and convolutional layers.

Pham and Le [78] proposed AutoDropout in 2021. AutoDropout uses reinforcement learning to train a controller selecting the optimal dropout pattern to train the model. The controller eliminates the need of manually adjusting dropout patterns as in previous methods such as DropBlock.

**3.1.2.2 RNNs:** Pachitariu et al. [99] applied standard dropout directly to RNNs in 2013 to randomly drop the outputs of neurons in RNNs. Bayer et al. [100], in the same year, applied Fast Dropout [60] directly to RNN.

Pham et al. [14] first proposed a dropout method specific to RNN structure in 2014, rather than just applying random dropout directly to RNNs. Instead of dropping connections between hidden states at different timestamps (recurrent connections), only the connections from input to output direction (feed-forward connections) are dropped. The dropout is performed in the same way as the standard dropout, i.e., training with a mask  $\mathbf{m}$  of Bernoulli distribution and an elementary multiplication of the hidden state

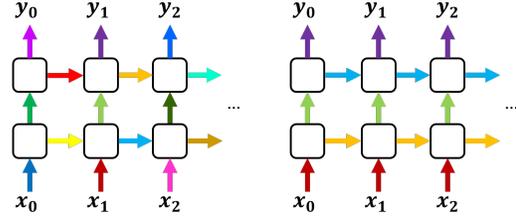


Fig. 4. RNNDrop (right) outperforms standard dropout (left) by generating one mask for each layer in RNN and keeping it throughout the sequence. [16], [40]

vector at each layer, and testing with all neurons working thus multiplying the output by the retention probability  $p$ :

$$\mathbf{h}_{\text{train}} = \mathbf{m} \odot \mathbf{h}, \quad \mathbf{h}_{\text{test}} = p\mathbf{h} \quad (18)$$

Zaremba et al. [15] also elaborated this method more systematically in 2014. For multilayer LSTMs [101], only the connections between layers are dropped, not for connections at different timestamps within the same layer. Let  $h_t^l$  be the hidden state at moment  $t$  of the  $l$ th layer,  $c_t^l$  be the memory unit at moment  $t$  of the  $l$ th layer, and  $T_{n,m} : \mathbb{R}^n \rightarrow \mathbb{R}^m$  be an affine transformation, then performing dropout only to the connections between layers is:

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} T_{2n,4n} \left( \mathbf{D} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} \right), \quad (19)$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g, \quad h_t^l = o \odot \tanh(c_t^l)$$

where  $\mathbf{D}$  is dropout operation matrix, which acts only on the output  $h_t^{l-1}$  of previous layer at the same timestamp, and not on the output  $h_{t-1}^l$  at the previous timestamp of this layer, as shown in Figure 3 [15]. The dashed lines in Figure 3 indicate the connections to which dropout is applied and may be dropped, while the solid lines indicate the connections retained.

The motivation of both the above articles [14], [15] is that the strength of RNN is its memory capacity, but if recurrent connections are dropped, the memory capacity of RNN will be impaired. To test this idea, the authors of [14] did a series of experiments in [102] 2015 to discuss dropout for RNNs and where in the network dropout operation should be performed. The authors examined the effect of adding dropout layers before LSTM input, on LSTM recurrent connection direction, and after LSTM output, respectively. They found that in most cases, adding dropout layers on LSTM input and output directions is better than adding them on recurrent connection direction, which experimentally verifies the ideas in the previous two papers.

Moon et al. [16] proposed RNNDrop in 2015, which gives a method for dropping recurrent connections between different timestamps of the same layer. This is done by generating only one dropping mask for each layer and then using this one mask at all timestamps of that layer. In this way, elements that are dropped at the first timestamp will not be used at subsequent timestamps, and elements that are kept at the first timestamp will be passed through to the last timestamp. The schematic is shown in Figure 4 [40]. The left side of the figure shows the dropout of RNNs using random dropout method, and the right shows how

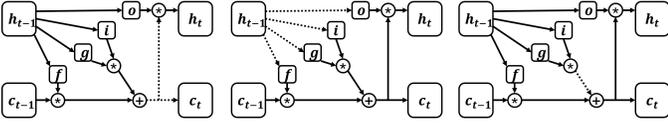


Fig. 5. Comparison of RNNDrop, Variational RNN Dropout and Recurrent Dropout [37].

RNNDrop working, with the same colors indicating the same dropout masks. In this way, the model is regularized with dropout while retaining RNN memory capability.

Gal and Ghahramani [79] proposed Variational RNN Dropout in 2015. The authors view dropout as an approximate inference process in Bayesian neural networks, which can also perform dropout of both feed-forward connections and recurrent connections. In this regard, Variational RNN Dropout can be seen as a variant of RNNDrop [16].

Recurrent Dropout proposed by Semeniuta et al. [37] in 2016 is another dropout method that preserves memory capacity of RNNs and generates random dropout masks at each step, just like standard dropout does [5], [6]. This is done by dropping only the vectors used to generate hidden state vectors, but not dropping hidden state vectors themselves. Recurrent Dropout is specialized for gated RNNs, such as LSTM [101] and GRU [103]. For LSTM in Eq. 20,

$$\begin{pmatrix} \mathbf{i}_t \\ \mathbf{f}_t \\ \mathbf{o}_t \\ \mathbf{g}_t \end{pmatrix} = \begin{pmatrix} \sigma(\mathbf{W}_i[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_i) \\ \sigma(\mathbf{W}_f[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_f) \\ \sigma(\mathbf{W}_o[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_o) \\ f(\mathbf{W}_g[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_g) \end{pmatrix}, \quad (20)$$

$$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \mathbf{g}_t, \quad \mathbf{h}_t = \mathbf{o}_t * f(\mathbf{c}_t)$$

Variational RNN Dropout [79] performs dropout as Equation 21,

$$\begin{pmatrix} \mathbf{i}_t \\ \mathbf{f}_t \\ \mathbf{o}_t \\ \mathbf{g}_t \end{pmatrix} = \begin{pmatrix} \sigma(\mathbf{W}_i[\mathbf{x}_t, d(\mathbf{h}_{t-1})] + \mathbf{b}_i) \\ \sigma(\mathbf{W}_f[\mathbf{x}_t, d(\mathbf{h}_{t-1})] + \mathbf{b}_f) \\ \sigma(\mathbf{W}_o[\mathbf{x}_t, d(\mathbf{h}_{t-1})] + \mathbf{b}_o) \\ f(\mathbf{W}_g[\mathbf{x}_t, d(\mathbf{h}_{t-1})] + \mathbf{b}_g) \end{pmatrix} \quad (21)$$

RNNDrop [16] performs dropout as Equation 22.

$$\mathbf{c}_t = d(\mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \mathbf{g}_t) \quad (22)$$

This method, on the other hand, performs dropout as Equation 23.

$$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * d(\mathbf{g}_t) \quad (23)$$

Dropout operations performed by RNNDrop [16], Variational RNN Dropout [79] and Recurrent Dropout are shown in Figure 5 [37].

Krueger et al. [80] proposed Zoneout in 2016. RNNs sometimes face the problem of vanishing gradient [104], [105]. Inspired by dropout of residual networks with the method Stochastic Depth [11] and Swapout [83], Zoneout randomly replaces the output at a certain timestamp in RNN with the output of a previous timestamp. Denote the hidden state transfer operation as  $\mathcal{T}$ , where  $\mathcal{T}$  is often an affine transformation. To perform dropout operation is to replace the original transfer operation  $\mathcal{T}$  with a new operation  $\tilde{\mathcal{T}}$ . Standard dropout and Zoneout can be expressed as follows, respectively:

$$\begin{aligned} \text{Dropout : } & \tilde{\mathcal{T}} = d_t \odot \mathcal{T} + (1 - d_t) \odot 0 \\ \text{Zoneout : } & \tilde{\mathcal{T}} = d_t \odot \mathcal{T} + (1 - d_t) \odot 1 \end{aligned} \quad (24)$$

where  $d_t$  is a mask vector obeying Bernoulli distribution. In this way, Zoneout passes the output of previous timestamp to the next timestamp with probability  $p$  instead of dropping it with probability  $p$ . In this way Zoneout solves vanishing gradient problem while acting as a regularizer.

Merity et al. [81] proposed Weighted-dropped LSTM in 2017. Borrowing the idea of DropConnect [33], instead of dropping neuron activations, Weighted-dropped LSTM drops elements in the weight matrix. That is, for LSTM in Eq. 20, Weighted-dropped LSTM drops weight matrix  $[\mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o]$  instead of hidden state  $\mathbf{h}_{t-1}$ .

In 2017, Melis et al. [106] did a comprehensive review about the effectiveness of RNN applications in language models. The covered dropout methods perform on the input of RNN, feed-forward connections, recurrent connections, and the output of the last layer.

Zolna et al. [82] proposed Fraternal Dropout in 2018. According to Ma et al.'s analysis [107], for networks trained with dropout, the expected outputs of training and prediction will differ. Using different dropout masks results in different outputs, i.e., the outputs are related to dropout masks, which is not what we want. We want model outputs to be irrelevant to dropout masks, they should be as same as possible under different masks, and their variance be as small as possible. Following this idea, Fraternal Dropout, trains two neural networks with the same structure and shared parameters. The only difference between them is their dropout masks. Fraternal Dropout optimize both objective functions of the two networks and the difference between their outputs. The authors also prove that the upper bound of the regularization term here is the objective function of expected linear dropout in [107].

**3.1.2.3 ResNets and others:** Residual network (ResNet) [108] is a structure designed to solve problems such as vanishing gradient [104], [105] and long training time caused by excessive network depth. Let the output of the  $l$ th layer of network be  $h_l$  and the transfer function from the  $l - 1$ th layer to the  $l$ th layer be  $f_l(\cdot)$  (which may contain one or more convolution functions, batch normalization functions, and activation functions), then the feed-forward operation containing residual block is

$$h_l = \text{ReLU}(f_l(h_{l-1}) + \text{id}(h_{l-1})) \quad (25)$$

where  $\text{id}$  denotes identity function, i.e.,  $h_{l-1}$  is passed to the  $l$ th layer directly. A series of dropout methods are also proposed for models with ResNet structure.

Huang et al. [11] proposed Stochastic Depth in 2016. Stochastic Depth randomly drops some operation blocks and retains only residual connections of that layer:

$$h_l = \text{ReLU}(b_l f_l(h_{l-1}) + \text{id}(h_{l-1})) \quad (26)$$

where  $b_l \sim \text{Bernoulli}(p_l)$  is retention probability. When  $b_l = 0$ , the layer has only one identity function, which is equivalent to directly copying results of the previous layer, i.e., the network becomes shallower. In this way, it is possible to train a network with a desired shallow depth and use the whole deep network during testing, alleviating the vanishing gradient problem [104], [105] and the problem of long training time.

Kang et al. [12] proposed Shakeout in 2016. Different from standard dropout, Shakeout acts on the neurons not

by making them choose between 0 and the original value, but between two new weights. The authors also show that Shakeout regularization combines three regularization terms,  $L_0$ ,  $L_1$  and  $L_2$ .

Li and Liu [13] proposed Whiteout in 2016. Both standard dropout [5], [6] and Shakeout [12] only introduce Bernoulli noise, while Whiteout introduces Gaussian noise into training process. This is done by adding additive or multiplicative Gaussian noise to the output of each neuron. Whiteout is the first noise injection regularization technique (NIRT) that imposes an extensive  $L_\gamma$ ,  $\gamma \in (0, 2)$  sparse regularization without involving  $L_2$  regularization.

Gastaldi [85] proposed Shake-Shake in 2017. Shake-Shake is used for three-way ResNet. The original feed-forward operation is

$$x_{i+1} = \sigma(x_i + \mathcal{F}(x_i, \mathcal{W}_i^{(1)}) + \mathcal{F}(x_i, \mathcal{W}_i^{(2)})) \quad (27)$$

Shake-Shake introduces random variables  $\alpha_i \sim U(0, 1)$  to assign weights to the two-way transfer function:

$$x_{i+1} = \sigma(x_i + \alpha_i \mathcal{F}(x_i, \mathcal{W}_i^{(1)}) + (1 - \alpha_i) \mathcal{F}(x_i, \mathcal{W}_i^{(2)})) \quad (28)$$

It takes a weight of 0.5 for each path when testing:

$$x_{i+1} = \sigma(x_i + 0.5 \mathcal{F}(x_i, \mathcal{W}_i^{(1)}) + 0.5 \mathcal{F}(x_i, \mathcal{W}_i^{(2)})) \quad (29)$$

Yamada et al. [38] proposed ShakeDrop in 2018. Stochastic Depth [11] simply drops or retains a layer, Shake-Shake [85] can assign weights to different pathways but is applied only to a three-way ResNet. ShakeDrop combines both functions. It has two parameters  $\alpha$  and  $\beta_l$  to control the assigned weights.  $b_l \sim \text{Bernoulli}(p_l)$  is the dropout ratio. ShakeDrop is expressed as

$$G(x) = \begin{cases} x + (b_l + \alpha - b_l \alpha) F(x), & \text{in train-fwd} \\ x + (b_l + \beta - b_l \beta) F(x), & \text{in train-bwd} \\ x + \mathbb{E}[b_l + \alpha - b_l \alpha] F(x), & \text{in test} \end{cases} \quad (30)$$

Larsson et al. [84] proposed DropPath in 2016. The authors first proposed a network structure Fractalnet, which achieves extremely deep neural networks based on self-similarity of structure. It is shown that Fractalnet can also alleviate vanishing gradient problem in deep neural networks, just as ResNet does. The authors then proposed a regularization approach for Fractalnet, which is randomly dropping sub-paths between input and output within each fractal block. Just as standard dropout can reduce dependencies between neurons, this operation reduces dependencies between sub-paths to act as a regularizer [84].

Zoph et al. [86] proposed Scheduled DropPath in 2018 as an improvement to DropPath [84]. Dropout ratio increases linearly with the number of training epochs rather than a fixed value, and Scheduled DropPath achieves better performance than the original DropPath.

Singh et al. [83] proposed Swapout in 2016, a synthesis of standard dropout and Stochastic Depth. Let  $X$  be the input of a block in neural network, and the block computes the output  $Y = F(X)$ . The output of the  $u$ th neuron in the block is noted as  $F^{(u)}(X)$ .  $\Theta$  is a tensor with the same shape as  $F(X)$  whose elements obey Bernoulli distribution. Standard dropout makes the output of each neuron choosed from  $\{0, F^{(u)}(X)\}$ . Stochastic Depth is such that the output of each neuron is choosed from  $\{X^{(u)}, X^{(u)} + F^{(u)}(X)\}$ .

Swapout, on the other hand, extends the range of possible output values. For a layer with  $N$  blocks  $F_1, \dots, F_N$ , define  $N$  independent Bernoulli tensor  $\Theta_1, \dots, \Theta_N$  such that the output is computed as

$$Y = \sum_{i=1}^N \Theta_i \odot F_i(X) \quad (31)$$

In this way, the output of each neuron has  $2^N$  possible values. Consider the simplest case of  $N = 2$  in ResNet,

$$Y = \Theta_1 \odot X + \Theta_2 \odot F(X) \quad (32)$$

The output of each neuron can take 4 values:  $\{0, X^{(u)}, F^{(u)}(X), X^{(u)} + F^{(u)}(X)\}$ . Each value corresponds to neuron's state as

- 1) 0: dropped
- 2)  $X^{(u)}$ : skipped by the residual connection
- 3)  $F^{(u)}(X)$ : normal
- 4)  $X^{(u)} + F^{(u)}(X)$ : a complete residual unit

Zhou et al. proposed DropHead [87] in 2020, dropping attention heads in multi-head attention mechanism, which is a core component of Transformer. It also adaptively adjusts dropout ratio during training to achieve better performance.

Dropout methods that drop neuron groups are summarized in Table 1. The application scenarios of dropping neuron groups are usually the models with certain structures such as CNN, RNN, ResNet, or Transformer. This type of dropout methods can boost the performance of these models, while its implementation is also subject to the model structures.

### 3.2 Drop Embeddings

In some machine learning tasks, the input data is first converted to embeddings then goes through the model. Dropout methods introduced in this section drop embeddings during training. For example, in recommendation, some dropout methods drop embeddings of user and item interaction histories during training to cope with the cold-start problem, while others randomly drop user and item feature embeddings to handle the possible missing information problem in real scenarios.

Volkovs et al. [20] proposed DropoutNet in 2017. The embedding matrices of users and items are noted as  $\mathbf{U}$  and  $\mathbf{I}$ . The embedding vectors of the  $u$ th user and  $i$ th item are  $\mathbf{U}_u$  and  $\mathbf{I}_i$ , respectively. The context information matrices of users and items are  $\Phi^{\mathcal{U}}$  and  $\Phi^{\mathcal{I}}$ , respectively. DropoutNet proposed a new form of objective dealing with the problem of missing interaction history. Previous models would add extra terms of context information into objective, hoping these newly added terms can be useful when the terms of interaction history are not available. However, it is difficult to determine the weights of these two components. DropoutNet's objective handles this problem automatically:

$$L = \sum_{u,i} (\mathbf{U}_u \mathbf{I}_i^T - f_{\mathcal{U}}(\mathbf{U}_u, \Phi^{\mathcal{U}}) f_{\mathcal{I}}(\mathbf{I}_i, \Phi^{\mathcal{I}})^T)^2 \quad (33)$$

During training, DropoutNet randomly drops a portion of input  $\mathbf{U}_u$  or  $\mathbf{I}_i$  by setting them to zero. For training instances that are kept, the objective will make the model ignore context information part ( $\Phi^{\mathcal{U}}$  and  $\Phi^{\mathcal{I}}$ ) as much as possible

like Equation 33 shows. For training instances with user or item inputs dropped, the objective will make the model rely as much as possible on context information part, as shown in Equation 34.

$$\begin{aligned} u \text{ cold start: } L_{ui} &= (\mathbf{U}_u \mathbf{I}_i^T - f_U(\mathbf{0}, \Phi_u^U) f_I(\mathbf{I}_i, \Phi_i^I)^T)^2 \\ i \text{ cold start: } L_{ui} &= (\mathbf{U}_u \mathbf{I}_i^T - f_U(\mathbf{U}_u, \Phi_u^U) f_I(\mathbf{0}, \Phi_i^I)^T)^2 \end{aligned} \quad (34)$$

Shi et al. [21] proposed ACCM model in 2018. DropoutNet [20], while automatically processing both interaction history information and context information, is implemented in such a way that its objective function is completely backward to one side. When interaction history is available, the model tends to completely rely on it and ignore attribute information; when interaction history is missing, the model tends to completely rely on attribute information. ACCM model, on the other hand, achieves flexible control of the weights of the two components by using attention mechanism [131].

The model contains a user part and an item part. Each part computes embeddings by both interaction history and context information. For the user part, the attention network computes attention weights for two kinds of information separately, and then obtains the final embedding  $\mathbf{u}$ :

$$\begin{aligned} h_{CF}^u &= \mathbf{h}^T \tanh(\mathbf{W} \mathbf{u}_{CF} + \mathbf{b}), \quad h_{CB}^u = \mathbf{h}^T \tanh(\mathbf{W} \mathbf{u}_{CB} + \mathbf{b}) \\ a_{CF}^u &= \frac{\exp(h_{CF}^u)}{\exp(h_{CF}^u) + \exp(h_{CB}^u)} = 1 - a_{CB}^u \\ \mathbf{u} &= a_{CF}^u \mathbf{u}_{CF} + a_{CB}^u \mathbf{u}_{CB} \end{aligned} \quad (35)$$

Item embedding  $\mathbf{v}$  is generated in the same way as  $\mathbf{u}$ . The model prediction is

$$y = b_g + b_u + b_v + \mathbf{u} \mathbf{v} \quad (36)$$

Like DropoutNet, the interaction history embeddings are randomly dropped during training, replaced with random vectors:

$$\begin{aligned} \mathbf{u} &= a_{CF}^u [(1 - c^u) \mathbf{u}_{CF} + c^u \mathbf{u}_r] + a_{CB}^u \mathbf{u}_{CB} \\ \mathbf{v} &= a_{CF}^v [(1 - c^v) \mathbf{v}_{CF} + c^v \mathbf{v}_r] + a_{CB}^v \mathbf{v}_{CB} \\ y &= b_g + c^u b_u + c^v b_v + \mathbf{u} \mathbf{v} \end{aligned} \quad (37)$$

where  $c^u, c^v \sim \text{Bernoulli}(p)$ ,  $p$  is the dropout probability.  $\mathbf{u}_r, \mathbf{v}_r$  are random vectors with the same initial distribution of user and item vectors. By using attention mechanism and randomly dropping embeddings, ACCM better solves cold start problem in recommendation.

Similar to missing interaction history, missing content information is sometimes encountered in recommendation. Since cold-start problem can be better solved by embedding dropout and attention mechanism together, missing attribute problem can also be solved in a similar way, which is the idea of AFS (Adaptive Feature Sampling) [22]. AFS drops a portion of user and item context information randomly during training to simulate missing attribute values, making the model more robust when testing.

Dropout methods that drop embeddings are summarized in Table 2. Recommendation models are usually the application scenarios of dropping embeddings, whose input information usually needs to be converted into vector representations for model operations. Dropping embeddings can be effective in such scenarios, while it can only be used when there are embeddings of input data.

### 3.3 Drop Input Information

Some dropout methods drop part of input information directly during training, which serves for various purposes under different scenarios, such as regularization, data augmentation, or data representation enhancement of pre-training stage.

#### 3.3.1 One-dimensional Information

Sennrich et al. [17] in 2016 use WordDropout in machine translation to drop out words from input data.

Ghazvinine et al. [109] proposed Mask-Predict in 2019. While most machine translation systems generate text from left to right, Mask-Predict uses a masking approach to train the model. It first predicts all target words and then iteratively masks and regenerates a subset of words in which the model has least confidence. Unlike BERT [28], this paper does not use masked language model for pre-training but uses it directly with Mask-Predict to generate text.

Zhang et al. [115] in 2020 proposed Token Drop mechanism for neural network machine translation. WordDropout [17] randomly drops words from sentences, while Token Drop method drops tokens.

Devlin et al. [28] proposed BERT in 2019. In the pre-training phase, 15% of the tokens are randomly masked. These masked tokens are then predicted in both directions using a self-attentive transformer to enhance the data representation in pre-training phase. After BERT came out, many BERT-like or BERT-based methods have been proposed for enhancing data representation in NLP pre-training.

Cui et al. [111] in 2019 proposed Whole Word Masking for Chinese language models. BERT randomly masks words for English language models, but randomly masking Chinese characters is less appropriate because Chinese characters may not be a complete semantic unit. Whole Word Masking masks words instead of Chinese characters when training Chinese language models.

Sun et al. [110] proposed ERNIE in 2019. ERNIE introduces human knowledge into word vector training. It achieves this by considering masking operations of three levels. The first level, like BERT, randomly masks English or Chinese words. The second level randomly masks phrases identified by existing toolkits, incorporating phrase information into training. The third level randomly masks entities pre-defined by human, incorporating prior human knowledge into the training of word vectors.

Wu et al. [112] proposed Mask and Infill in 2019. The pre-training phase is divided into a masking phase and a filling phase used to accomplish the task of sentiment transfer.

Ye et al. [113] proposed AMS method in 2019. A general knowledge Q&A dataset is generated through ConceptNet [132], and the general knowledge concepts in each utterance of this dataset are masked and predicted so that the model learns general knowledge through this process.

Zhang et al. [114] in 2019 proposed PEGASUS for summary generation tasks. During pre-training, not only word tokens are randomly masked, but also important sentences. These sentences are part of the summary to be generated and need to be predicted by the model.

Wang et al. [133] in 2019 imitates BERT to introduce dropout in speech recognition, which randomly masks acoustic signals and features in the input audio.

TABLE 2  
Table of methods that drop embeddings or input information.

Method	Year	Category	Brief Description	Original Scenario	Source
DropoutNet [20]	2017	2†	Randomly drop interactions	Recom.	NeurIPS
ACCM [21]	2018	2	Drop interactions & use attention mechanism	Recom.	CIKM
AFS [22]	2019	2	Drop interactions and attribute values	Recom.	CIKM
WordDropout [17]	2016	3.1†	Drop words in machine translation	NLP	WMT16
BERT [28]	2018	3.1	Mask and predict tokens in pre-training phase	NLP	NAACL
Mask-Predict [109]	2019	3.1	Mask and regenerate words in machine translation	NLP	EMNLP
ERNIE [110]	2019	3.1	Incorporate human knowledge into pre-training	NLP	arxiv
Whole Word Masking [111]	2019	3.1	Randomly mask chinese words	NLP	arxiv
Mask and Infill [112]	2019	3.1	Mask and infill tokens in pre-training	NLP	IJCAI
AMS [113]	2019	3.1	Incorporate general knowledge using ConceptNet	NLP	arxiv
PEGASUS [114]	2019	3.1	Mask sentences for summary generation	NLP	ICML
Token Drop [115]	2020	3.1	Drop tokens instead of words	NLP	COLING
Selective Masking [116]	2020	3.1	Introduce a task-guided pre-training stage	NLP	EMNLP
S3-Rec [117]	2020	3.1	Enhance interaction sequence like BERT	Recom.	CIKM
CutOut [18]	2017	3.2†	Drop a square region on the input image	CV	arxiv
Random Erasing [118]	2017	3.2	Drop a rectangular region on the input image	CV	AAAI
Hide-and-Seek [119]	2017	3.2	Drop several square regions	CV	ICCV
Mixup [120]	2017	3.2	Take linear interpolations of training instances as input	CV	ICLR
Manifold Mixup [121]	2019	3.2	Generalize Mixup to feature level	CV	ICML
CutMix [122]	2019	3.2	Replace regions of one image with another’s	CV	ICCV
GridMask [39]	2020	3.2	Drop regularly tiled square regions	CV	arxiv
Attentive CutMix [123]	2020	3.2	Improve CutMix with attention mechanism	CV	ICASSP
MAE [124]	2021	3.2	Mask and reconstruct patches of input images	CV	arxiv
GraphSAGE [19]	2017	3.3†	Randomly sample nodes	Graph	NeurIPS
FastGCN [125]	2018	3.3	Sample nodes from the whole graph	Graph	ICLR
AS-GCN [126]	2018	3.3	Node sampling layer by layer	Graph	NeurIPS
GAT [127]	2018	3.3	Attention on edges	Graph	ICLR
LADIES [128]	2019	3.3	Adaptively sample nodes by layer	Graph	NeurIPS
GRAND [129]	2020	3.3	Random propagation on graph	Graph	NeurIPS
SGAT [130]	2020	3.3	Learn sparse attention coefficients on graph	Graph	TKDE
DropEdge [29]	2020	3.3	Randomly drop edges	Graph	ICLR

† 2 refers to dropping embeddings, 3.1 dropping one-dimensional information, 3.2 dropping two-dimensional information, and 3.3 dropping graph information.

Selective Masking [116] proposed by Gu et al. in 2020 introduced a task-guided pre-training stage between general pre-training and fine-tuning stage.

There are similarities between the input of sequential recommendation and the input of NLP tasks, for both of them are temporal one-dimensional information. So there are also recommendation models that borrows the idea of BERT: Zhou et al. [117] proposed S3-Rec in 2020. It divides the recommendation task into a pre-training stage and a fine-tuning stage just like BERT. S3-Rec randomly masks a portion of item ids and attributes in pre-training phase to enhance the representation between item ids, item attributes, and item sequences.

Dropout methods that drop one-dimensional input information are summarized in Table 2, which are mainly applied in NLP or sequential recommendation tasks, where input data is organized as temporal one-dimensional sequences.

### 3.3.2 Two-dimensional Information

Existing data enhancement methods can be broadly classified into three categories: spatial transformation, color distortion, and information dropping. Dropping two-dimensional input information is generally regarded as a data enhancement method of information dropping.

DeVries and Taylor [18] proposed CutOut in 2017. For every training image, CutOut randomly selects a square region and sets the pixel values within this region to zero. The difference of CutOut with methods in Section 3.1.2 such as SpatialDropout [8] or DropBlock [36] is that CutOut

is performed at the level of input information. Compared to dropping model structure, it is easier to implement by directly dropping a part of the input image. In addition, dropping input information is equivalent to generating a new training sample, so there is no need to multiply the neuron output by a factor to eliminate the bias during testing as the methods in Section 3.1 does.

Zhong et al. [118] proposed Random Erasing in 2017. Similar to CutOut [18], the training image is covered with a rectangular box with random position and random size. The pixel values within the rectangular box are also random.

Singh et al. [119] proposed Hide-and-Seek in 2017. CutOut [18] and Random Erasing [118] drop only one rectangular region for each input image, while Hide-and-Seek divides the image into  $S \times S$  small squares, each square is dropped with  $p_{hide}$  probability. The purpose of Hide-and-Seek is to let the model be capable of extracting features from other parts of the image after the most discriminative part has been dropped, preventing the model from relying too much on certain parts.

Chen et al. [39] proposed GridMask in 2020. Dropout pattern of GridMask is a number of equally spaced square regions tiled on a plane, determined by four parameters  $(r, d, \delta_x, \delta_y)$ . The dropout pattern of GridMask is more regular compared to CutOut, Random Erasing and Hide-and-Seek. Compared to AutoAugment [134] which employs reinforcement learning to search for dropout patterns, GridMask consumes much less training cost. The above four methods are schematically shown in Figure 6.

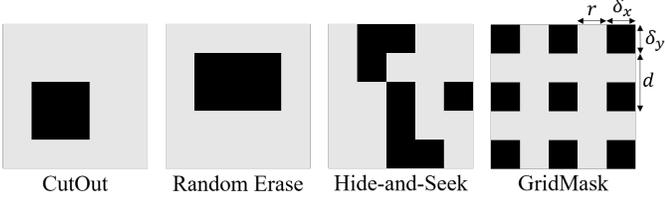


Fig. 6. Comparison of dropout patterns of CutOut [18], Random Erasing [118], Hide-and-Seek [119] and GridMask [39].

Dropping input data can also be seen as *introducing noise* to input data, while this noise is Bernoulli noise. Some of the following data enhancement methods do not necessarily *drop* input data, but *introduce noise* to input data.

Zhang et al. [120] proposed Mixup in 2017. Mixup augments data in a simple way: the linear interpolations of training instances are also taken as training instances. Specifically, for training instances  $(\mathbf{x}_i, y_i)$  and  $(\mathbf{x}_j, y_j)$ ,

$$\begin{aligned}\tilde{\mathbf{x}} &= \lambda \mathbf{x}_i + (1 - \lambda) \mathbf{x}_j \\ \tilde{y} &= \lambda y_i + (1 - \lambda) y_j\end{aligned}\quad (38)$$

Mixup takes  $(\tilde{\mathbf{x}}, \tilde{y})$  as a training instance as well.

Verma et al. [121] proposed Manifold Mixup in 2019. Manifold Mixup generalizes Mixup [120] operation to feature level. The motivation is that features have higher-order semantic information, and interpolation in feature level could yield more meaningful samples.

Yun et al. [122] proposed CutMix in 2019, which is an improvement on Mixup [120] and Cutout [18]. Cutout fills part of the image with meaningless regions, which is not conducive to the model making full use of training data. Mixup uses linear interpolation to augment the data, however these newly produced images are not natural images. CutMix, on the other hand, randomly selects some rectangular regions of the image  $\mathbf{x}_A$  and replaces them with regions at the same locations of image  $\mathbf{x}_B$ . The corresponding labels are replaced by a combination of the two images:

$$\begin{aligned}\tilde{\mathbf{x}} &= \mathbf{M} \odot \mathbf{x}_A + (1 - \mathbf{M}) \odot \mathbf{x}_B \\ \tilde{y} &= \lambda y_A + (1 - \lambda) y_B\end{aligned}\quad (39)$$

where  $\mathbf{M}$  is the masking matrix. Walawalkar et al. [123] proposed Attentive CutMix in 2020, which further improved CutMix by using an attention mechanism to select the most discriminative regions for replacement. Operations of Mixup, CutOut, CutMix and Attentive CutMix are shown in Figure 7 from the original paper [123].

He et al. [124] proposed Masked Autoencoders (MAE) in 2021. It randomly masks patches of the input image and reconstructs the missing pixels during pre-training.

Dropout methods that drop two-dimensional information are summarized in Table 2. They are mainly applied in computer vision tasks, where input data is organized as pixel matrices.

### 3.3.3 Graph Information

Graph neural networks (GNNs) have a wide range of applications in various tasks such as node classification, cluster detection, and recommender systems [135], [136], [137]. In the training of GNNs, some methods randomly drop nodes or edges and use only part of graph information for training, serving as a regularization technique.

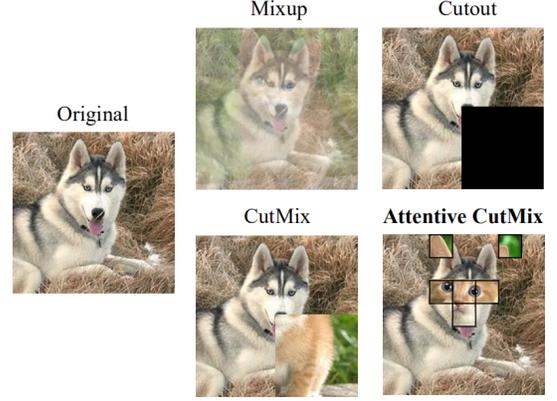


Fig. 7. Comparison of dropout patterns of Mixup [121], CutOut [18], CutMix [122] and Attentive CutMix. [123]

**3.3.3.1 Drop Nodes:** Hamilton et al. [19] proposed GraphSAGE (SAmple and AGGreGatE) in 2017. Before this, GCNs were generally trained in the way of *transductive learning*, which requires all nodes to be visible at training time and needs to calculate a node’s embedding by all its neighbors. GraphSAGE, on the other hand, adopts *inductive learning*, which requires only some of the neighbors to predict a node’s embedding. To achieve this, GraphSAGE does not directly train node embeddings but trains aggregation functions, which compute node embeddings from its neighbors. When a new node is added to the graph during testing, the trained aggregation functions can predict its embedding from its neighbors. Parameters of the aggregation functions are trained with an unsupervised learning objective that makes the representations of closer nodes more similar and farther nodes less similar,

$$\begin{aligned}J_G(\mathbf{z}_u) &= -\log(\sigma(\mathbf{z}_u^T \mathbf{z}_v)) \\ &\quad - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log(\sigma(-\mathbf{z}_u^T \mathbf{z}_{v_n}))\end{aligned}\quad (40)$$

$v$  is some node reachable within a fixed number of steps near  $u$ .  $P_n$  is the negative sampling distribution, and  $Q$  is the number of negative samples. This objective can be replaced with any task-specific objective for other downstream tasks. This training method of sampling only some nodes makes GCN more generalizable, reduces the computational complexity of training, and improves the model performance.

A series of node-dropout training methods have been proposed after GraphSAGE. Chen et al. [125] proposed FastGCN in 2018, whose idea is similar to GraphSAGE [19]. The difference is that FastGCN randomly samples all nodes in the whole graph, not only the neighbors of a certain node. Considering node sampling efficiency, FastGCN is significantly faster than the original GCN as well as GraphSAGE, while maintaining comparable prediction performance.

Huang et al. [126] proposed AS-GCN in 2018. Again, the nodes are sampled during training; its differences with GraphSAGE [19] and FastGCN [125] are threefold: AS-GCN samples nodes layer by layer instead of independently; the sampler is adaptive; and AS-GCN skips some edges when transferring information between two nodes with long distances, enhancing the efficiency of information propagation. The experiments on running time show that AS-GCN is faster than the original GCN and the former node-wise sampling methods.

Zou et al. [128] proposed LADIES in 2019. The previous single-node-based sampling methods suffer from the problem of too many neighbors, while the layer-based sampling methods suffer from too sparse connections. LADIES also samples layer by layer, but it calculates the importance of each node in the next layer and samples the most critical nodes among them. This alleviates the problem of sparse connections while limiting the number of neighbors.

Feng et al. [129] proposed GRAND in 2020. It performs data augmentation using Random Propagation method. For the graph feature matrix, the authors compare two dropout methods: random dropout of matrix elements and random dropout matrix rows. The former is equivalent to a standard dropout in feature level, while the latter is dropping nodes. The latter performs better in experimental comparison.

**3.3.3.2 Drop Edges:** Veličković et al. [127] proposed GAT (Graph Attention Networks) in 2017. GAT uses attention mechanism to compute the importance of different edges and train GNN using more important information. In 2021 Ye and Ji [130] improved on GAT and proposed SGAT. SGAT learns sparse attention coefficients on the graph and produces an edge-sparsified graph.

Rong et al. [29] proposed DropEdge in 2020. GCN training process is prone to overfitting and over-smoothing problems. Over-smoothing is a phenomenon that representations of all nodes on the graph tend to be the same, occurring when GCN is too deep. The authors address these two problems by randomly dropping edges during training: dropping edges can be seen as a data augmentation method introducing noise into input data to prevent overfitting; dropping edges also reduces the information propagation through edges, thus preventing the over-smoothing problem.

Dropout methods that drop graph information are widely used in GCN, and we summarize them in Table 2.

Dropping input information can be an effective way of data augmentation or regularization. As will be shown in 5, it is a good way of augmenting sequences in recommendation. It can be applied to a wide range of scenarios as all machine learning tasks have input information. Meanwhile, it is not performed on model parameters but input data, so its regularization effectiveness is not as stable as dropping model structures.

### 3.4 Summary and Interconnections between Dropout Methods

Based on where in a machine learning task the dropout operation performs, we classify commonly used dropout methods into three major categories: dropping model structures, dropping embeddings and dropping input information. Dropping model structures is divided into two subcategories of dropping individual neurons and dropping neuron groups, according to the granularity of dropout operation. Dropping input information is divided into three subcategories of dropping one-dimensional information, two-dimensional information, and graph information according to the form of input information.

The three types of dropout methods, dropout of model structure, dropout of input information, and dropout of embedding, can be represented as ①, ②, and ③ in Figure 8, respectively. They are performed at three different stages of the training process, which is our classification criteria.

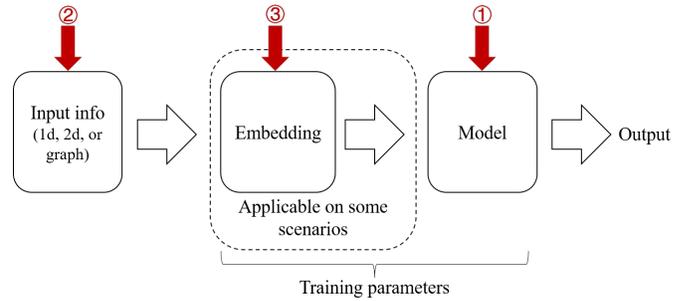


Fig. 8. Training procedure and dropout position in neural models. We classify dropout methods into three major categories based on where their operations perform in a training process: dropping input information (①), dropping embeddings (②), and dropping model structures (③).

Within each block, there may also be different layers. For example, for the model part, if we use a deep network, we can perform dropout in every layer or just do it for some layers. If we use a recurrent neural network, we can perform dropout in its feed-forward direction, its recurrent direction, or part of the layers. Since a common implementation of dropout is to zero the neuron outputs, if the output of layer  $l$  is considered as the input data of layer  $l + 1$ , then dropping neurons of layer  $l$  can be considered as dropping the input data of layer  $l + 1$ . Thus dropping model structure and dropping input data are not clear-cut, they are highly correlated. This is more clear in computer vision tasks where convolutional neural networks are used: methods in Section 3.1.2.1 and Section 3.3.2 operate at different levels, but their actual implementations are highly similar. By performing dropout operation in Section 3.1.2.1 on the input image level, it becomes the operation in Section 3.3.2.

## 4 CONTRIBUTIONS OF DROPOUT METHODS

In Section 3, we classify commonly used dropout methods into three major categories according to the stage at which the dropout operation is performed, discuss their applications in neural models and analyze their interconnections. In this section, we discuss the contributions of dropout methods from the perspective of effectiveness and efficiency.

### 4.1 Improving Effectiveness

Dropout makes models to better utilize training data and promotes model effectiveness in many ways.

- **Preventing overfitting.** Most of the dropout methods are used as regularization methods [138], [139] to prevent overfitting. For the methods of dropping model structure, during the training process, a part of neurons is dropped randomly to reduce the interdependence between neurons and prevent overfitting. The method of dropping input information enhances the robustness of the model by introducing noise into the input data. Meanwhile, many dropout methods have contributions other than preventing overfitting.

- **Simulating testing phase.** Some methods are used to simulate the possible situations in the testing phase [20], [21], [22]. The testing phase may face an information deficit, and the model needs to give predictions under the absence of information. Therefore, these methods drop part of the information at training time so that the model does not

rely excessively on this possibly missing information and improves the performance of the testing phase.

- **Data augmentation.** Some methods are used for data augmentation [18], [27], [39], [118], [122]. Noise is introduced into the input data to create more training samples and improve the training effect of the model. Dropping training data can be seen as introducing Bernoulli noise into data.
- **Enhancing data representation.** Some methods are used to enhance data representation in pre-training phase [110], [112], [114], [117]. These BERT [28] based methods randomly mask part of the input data and use the unmasked part to predict the masked part to enhance data representation.
- **Preventing over-smoothing.** Dropout methods in graph neural networks can also solve the over-smoothing problem [29]. Over-smoothing occurs when GCN is too deep that the representation of all nodes on the graph tends to be the same. Randomly dropping edges during training can reduce information propagation through edges and prevent the over-smoothing problem.

## 4.2 Improving Efficiency

Besides improving model effectiveness, some dropout methods can also improve model efficiency for certain tasks.

- **Accelerating GCN training.** In GCN scenarios, node sampling technique proposed by GraphSAGE [19] efficiently accelerates GCN training. It only needs some of the nodes to perform the training process instead of needing all node neighbors. Later works like FastGCN [125] and AS-GCN [126] improve this sampling technique, making it faster or sample in an adaptive way.
- **Model compression.** Some methods are used for model compression [23], [24], [25], [66], [67], [68]. These methods make the model structure easier to compress after random dropout of neurons, e.g., easier to perform neural pruning. Model compression reduces model parameters, which can improve training efficiency and prevent overfitting.
- **Model uncertainty measurement.** Some methods are used to measure the model uncertainty [26], [65], [89], [92]. These methods view dropout as a Bayesian learning process. For example, in Monte Carlo Dropout [26], the authors interpret dropout as a Bayesian approximation of a deep Gaussian process. Monte Carlo Dropout estimates the uncertainty of the model output by performing a grid search on the dropout rate, which is almost unusable for deeper models (those in computer vision tasks) and reinforcement learning models because of the excessive computational time and computational resources consumed. Thanks to the development of Bayesian learning, Concrete Dropout [65] uses a continuous relaxation of the dropout discrete mask. A new objective function is proposed to automatically adjust the dropout rate on large models, reducing the time required for experiments. It also allows the agent in reinforcement learning to dynamically adjust its uncertainty as the training process proceeds and more training data is observed.

## 5 DROPOUT EXPERIMENTS IN RECOMMENDATION MODELS

We have reviewed multiple types of dropout methods and discussed their interconnections and contributions. However, each work has its own experiments to verify the effect

of its dropout method, so the methods’ effectiveness actually has not been investigated under a unified framework and evaluation system. In this section, we experimentally investigate four classes of dropout methods on recommendation models. Choosing recommendation models as our experiment scenario is because they utilize various heterogeneous information, which are transformed into different forms, from input data, to embeddings, and to model structures, covering the range of dropout operations we reviewed in Section 3. Such a variety of information sources and forms provides a suitable environment for our comparisons and verification of different dropout methods.

We first introduce the selected recommendation models, the implementations of the four dropout methods on each of them (Section 5.1), the datasets and experimental settings (Section 5.2). Then, we analyze the experimental results and present comparison to evaluate the effectiveness of each dropout method (Section 5.3). Finally, we explore the effect of dropout ratio on model performances (Section 5.4).

### 5.1 Recommendation Models and Implementations of Dropout Methods

We choose five recommendation models belonging to four classes:

- Traditional recommendation model: BPRMF [140]
- Neural recommendation model utilizing context information: NFM [141]
- Sequential recommendation model: GRU4Rec [49] and SASRec [50]
- Graph recommendation model: LightGCN [52]

The four dropout methods are dropout of model structure, dropout of input information, dropout of embeddings, and dropout of graph information. Since there are significant differences between graph information and other input information in recommender systems, we treat them as different methods. We elaborate on the implementations of the four dropout methods on each recommendation model in Appendix B.

### 5.2 Datasets and Experiment Settings

TABLE 3  
Dataset Statistics

	#user	#item	#interaction	density(%)
MovieLens-1M	6040	3883	1000209	4.26
ml1m-cold	6040	3883	797675	3.40
Amazon Baby 5-core	19445	7050	160792	0.117

We use two datasets for experiments: MovieLens-1M-cold and Amazon Baby 5-core [142], [143].

MovieLens-1M-cold is obtained by artificially creating a cold-start condition based on MovieLens-1M [144]. Each user in MovieLens-1M has at least 20 interactions, so there is no cold-start scenario for users. To make our experimental environment closer to real-world recommendation, we randomly select 10% of users and 10% of items and remove all of their interactions in training set, constructing a group of cold-start users and items. We also randomly select user and item attributes and set the value to zero (unknown) so that unknown attribute values in the dataset account for 10% of

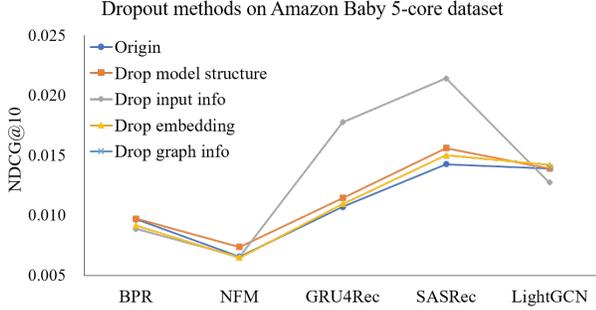


Fig. 9. NDCG@10 on Amazon Baby 5-core

all attribute values. After this process, the resulting dataset is named MovieLens-1M-cold, or ml1m-cold for short.

The statistics of the datasets are shown in Table 3. The density of Amazon Baby 5-core is only 1/30 of ml1m-cold. We choose a dense and a sparse dataset to make our experiment results more general.

We consider Top-N recommendation task. The parameter values taken for all models in common are shown in Table 5 in Appendix C. This ensures a consistent evaluation environment and comparability of evaluation results. According to the analysis of Krichene et al. [145], negative sampling during testing can bias the results. Therefore, we conduct non-sampling evaluation for all our experiments. Parameters specific to each model are shown in Table 6 in Appendix C. The training batch size is set to 1024 on ml1m-cold for faster training.

We searched  $L_2$ -coefficient and choose  $1e^{-6}$  for all models on ml1m-cold dataset. On Amazon Baby 5-core, we choose  $1e^{-5}$  for BPR, NFM, and GRU4Rec;  $1e^{-6}$  for SASRec; and  $1e^{-4}$  for LightGCN.

## 5.3 Results and Analysis

### 5.3.1 Overall Results

TABLE 4  
Overall NDCG@10 results

	Origin	Drop Model Structure	Drop Input Info	Drop Embedding
Amazon Baby 5-core				
BPR	0.00969	0.00973	0.00888**	0.00916*
NFM	0.00657	0.00737**	0.00654	0.00647
GRU4Rec	0.01071	0.01146	0.01777**	0.01096
SASRec	0.01428	0.01562*	0.02143**	0.01502
LightGCN†	0.01392	0.01392	0.01275	0.01420
ml1m-cold				
BPR	0.0339	0.0364**	0.0331	0.0332
NFM	0.0335	0.0353*	0.0334	0.0354**
GRU4Rec	0.0964	0.1010**	0.1084**	0.1013**
SASRec	0.1064	0.1092**	0.1093	0.1063
LightGCN†	0.0377	0.0388	0.0361**	0.0376

\*:  $p < 0.05$ , \*\*:  $p < 0.01$ , compared to the origin value (not using any dropout methods)

†: Drop graph info for LightGCN on Amazon Baby 5-core is 0.01403, on ml1m-cold is 0.0383

We present the overall results of NDCG@10 in Table 4, and each value in the table is the best result for one dropout method on one model with the dropout ratio among  $\{0.1, 0.2, 0.3\}$ . We plot the results in lines showing in Figure 9 and 10. All detailed experimental results of other

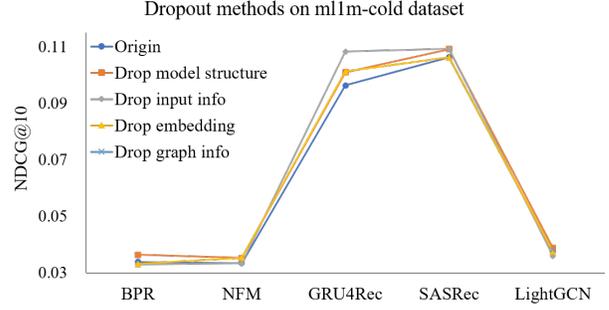


Fig. 10. NDCG@10 on ml1m-cold

evaluation metrics (NDCG@5, 20, 50; HR@10, 20) and each dropout ratio are in Appendix D.

According to the experimental results, we summarize the effect of different dropout methods in the five recommendation models.

For traditional recommendation models that do not use neural networks, dropping model structure can have a regularizing effect and improve model performance. Dropping input information and dropping embeddings can be detrimental to the performance of the model.

For neural network models using contextual information, both dropping model structure and dropping embeddings can improve the model performance. This can be because the former acts as a regularizer, and the latter allows the model to take advantage of multiple aspects of information to better cope with cold start situations. Dropping input information does not affect model effectiveness.

For sequential models, all three dropout methods lead to improved model effects. Among them, dropping input information has the most significant improvement if dropout ratio is properly chosen. This is because dropping items in input sequences can be viewed as a type of sequence augmentation [146]. Dropping model structure has the most stable improvement. Dropping embeddings also allows the model to get improved or, at least, remain unchanged.

For the graph recommendation model, this paper only explores the lightweight model LightGCN, which contains a small quantity of parameters thus does not require too much regularization according to the original paper. So all the four dropout methods do not affect the model performance, or even have a detrimental effect.

### 5.3.2 Discussion on the Applications of the Four Dropout Methods

As described in Section 3, the four dropout methods belong to different levels and may have different contributions, and Section 5.3.1 shows their performances vary in different scenarios. Therefore, it is not feasible to simply determine which of them is better or worse generally, but we can analyze their features. This section provides some analysis and discussions on the properties of the four dropout methods.

Dropping model structure is the most stable dropout method in our experiment. It makes the model performance gain or at least unchanged for all models, where most of them have significant improvement. This reveals that this classical dropout method is still the most effective way of regularization, and dropout according to the structural properties of the model can achieve good results, except for

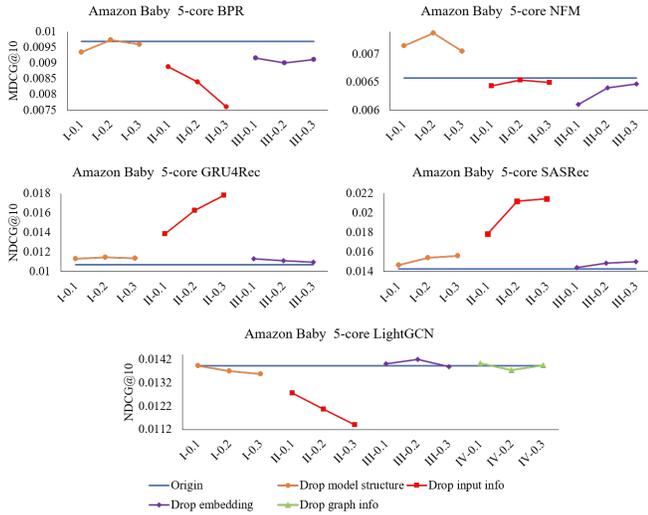


Fig. 11. Effect of dropout ratio on Amazon Baby 5-core

the model with very few parameters (like LightGCN) which does not need much regularization.

Dropping input information has significant side effects on the performance of traditional models and graph recommendation models, and has no effect on the neural model utilizing contextual information. This indicates that recommendation models utilizing less information and having fewer parameters, such as BPRMF and LightGCN in this paper, do not need too much regularization, and dropping input information will harm their performances. However, it significantly improves sequential recommendation models and is the most effective one among the four methods. This is because dropping input information of sequential models can be viewed as a way of sequence augmentation.

Dropping embedding has no significant effect on model performance in most cases, while slightly improves NFM. The method of embedding dropout is from [21], [22], which use the attention mechanism to make the model select the information to be exploited automatically. Then the model can automatically rely on the information that is kept after dropping part of the embeddings. In contrast, the model used in this experiment does not have this attention mechanism, so the improving effect is limited.

For dropping edges in graph, there is neither an improvement nor a decrease on model performance. This is determined by the nature of LightGCN [52], which only has a small quantity of parameters. Dropping edges or nodes may be effective to other models with a larger number of parameters such as NGCF [147].

### 5.4 Effect of Dropout Ratio

In this section, we analyze the parameter sensitivity: how does dropout ratio affect model performances?

For each dataset, each model, and each evaluation metric, we test dropout ratios of {0.1, 0.2, 0.3}. We plotted the values of NDCG@10 in Figure 11 and 12.

As can be seen in Figure 11 and 12, the orange line keeps staying above the blue line, indicating that dropping model structure almost always leads to a stable improvement on model performances. For the two sequential recommendation models, GRU4Rec and SASRec, the red line is higher

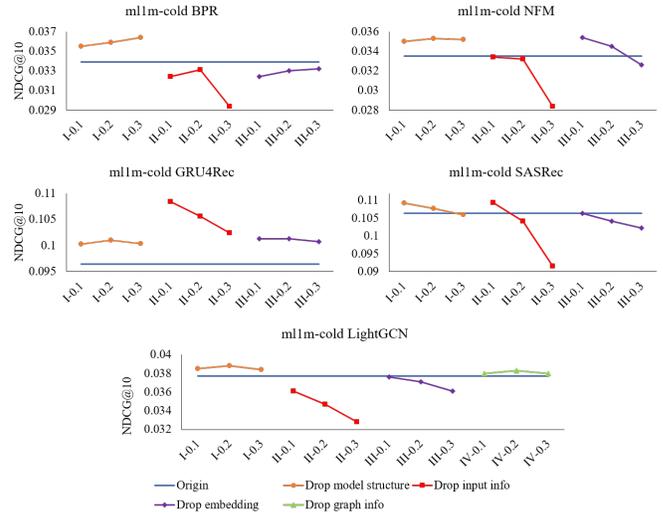


Fig. 12. Effect of dropout ratio on ml1m-cold

than the other three colored lines, indicating that dropping input information has the most significant improvement effect on sequential recommendation models, when choosing dropout ratio properly. The appropriate range of dropout rate for SASRec on ml1m-cold is low, and 0.2 and 0.3 are too large, causing its performance to decline. The purple line improves NFM on ml1m-cold because ml1m-cold contains rich contextual information, and dropping embeddings allows NFM to utilize multifaceted information more robustly.

## 6 FUTURE DIRECTIONS

Based on our review and experiments, in this section, we further discuss several topics about dropout and analyzes some potential research directions in this field.

### 6.1 Transfer of Dropout Strategies in Different Domains

Most dropout methods that drop input information are domain specific, designed for a certain domain like NLP or CV, as we have reviewed in Section 3.3. However recently, MAE [124] brought the randomly masking strategy from NLP pre-training to CV and achieved good results. Some dropout methods dropping model structure are also linked to the ideas of other domain. SimCSE [70] adopted dropout in NLP self-supervised learning tasks, in a way like the data augmentation techniques in self-supervised learning in CV. This indicates that the applications of dropout methods in different domain could be embarking on similar trajectories.

### 6.2 Selection of Dropout Methods

Though specific dropout methods achieve impressive improvement on certain neural models, deciding which type of dropout method to use is not an easy task. Based on our experimental results in Section 5, distinct dropout methods work best for various types of models, like dropping input information for sequential recommendation models, dropping embedding for recommendation models that utilize contextual information, and dropping model structure for all models. But can we determine which dropout method should be used according to the characteristic of the model before we conduct experiments? If so, lots of time on enumerating dropout methods during training could be saved.

### 6.3 Hyperparameter Optimization of Dropout

In the beginning, dropout methods require manually setting dropout ratio and patterns. Standard Dropout [5] and DropConnect [33] need manually setting dropout ratio, and the dropout patterns of the methods like DropBlock [36] and GridMask [39] need to be deliberately designed. Some later methods make progress to automate this process to some extent. As more data is exposed throughout the training process, Variational Dropout [62], Concrete Dropout [65], and Curriculum Dropout [35] can automatically adjust dropout ratio towards more suitable values. Instead of randomly dropping neurons, Targeted Dropout [25] and Ising-Dropout [67] calculate the most suitable neurons to drop, making the dropout pattern design more automatic. AutoDropout [78] uses reinforcement learning to train a controller, which decides the dropout patterns in CNN and Transformers. In the future, more efficient ways of optimizing dropout hyperparameters could be explored.

### 6.4 Efficient Dropout

Besides effectiveness, efficiency is also needed to be considered when using dropout, since the dropout operation itself takes time. Some aforementioned methods in Section 3 add additional attention parts to calculate dropout patterns, which may prolong the training time; edge dropping operation in Section 5 slows down the training; and the methods that use reinforcement learning [78] requires excessive computational time and resources.

Fast Dropout [60] takes the first step of improving the efficiency of dropout operation itself, and many methods have tried to improve dropout efficiency: Concrete Dropout optimizes the model uncertainty estimation process of Monte Carlo Dropout; the series of GCN-based dropout methods [19], [126], [129] have been making improvement on node-dropping training. However, as more complicated and time-consuming techniques like reinforcement learning have been adopted for dropout, improving their efficiency to speed up the dropout operation is still worth to be further explored.

### 6.5 Understanding Dropout Theoretically

The effectiveness of dropout has been irrefutably verified by experiments of hundreds of works. However, mathematical proofs of the validity of dropout has been rare. Baldi and Sadowski in 2013 gave a mathematical formality of dropout and use it to analyze the averaging and regularization properties of dropout [148]. Gal and Ghahramani in 2016 cast dropout network training as approximate inference of Bayesian neural networks, achieving a significant improvement in experiment results without increasing time complexity [149]. Gal and Ghahramani's other works [26], [79] also perform mathematical derivation on the validity of proposed dropout methods. In the future, proving the effectiveness of dropout not only from intuitive explanation and experimental verification but also from mathematical proof could be a challenging research direction.

## 7 CONCLUSIONS

In this paper, we investigate more than seventy dropout methods in neural network models and classify them into

three major categories and six subcategories according to the stage where the dropout operation is performed. We discuss their applications in neural models, their contributions and interconnections.

We conduct experiments on five recommendation models to verify the effectiveness of each type of dropout method under our classification framework, and find that dropping model structure has the most general and stable improvement effect on the models, while dropping input information and dropping embeddings are model-specific.

Finally, we present some open problems and potential research directions, hoping to promote the research in this field. Dropout methods are basic and universally used training techniques in today's neural model, helping with our steps towards better machine learning and artificial intelligence. We hope this survey paper can help readers better understand the works in this research area.

## ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China (2018YFC0831900), Natural Science Foundation of China (Grant No. 62002191, 61672311, 61532011) and Tsinghua University Guoqiang Research Institute.

## REFERENCES

- [1] T. Dietterich, "Overfitting and undercomputing in machine learning," *ACM computing surveys (CSUR)*, vol. 27, pp. 326–327, 1995.
- [2] T. Van Laarhoven, "L2 regularization versus batch and weight normalization," *arXiv:1706.05350*, 2017.
- [3] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*. PMLR, 2015, pp. 448–456.
- [4] T. Salimans and D. P. Kingma, "Weight normalization: a simple reparameterization to accelerate training of deep neural networks," in *30th NeurIPS*, 2016, pp. 901–909.
- [5] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv:1207.0580*, 2012.
- [6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *JMLR*, vol. 15, pp. 1929–1958, 2014.
- [7] L. J. Ba and B. Frey, "Adaptive dropout for training deep neural networks," in *26th NeurIPS-Volume 2*, 2013, pp. 3084–3092.
- [8] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks," in *CVPR*, 2015, pp. 648–656.
- [9] H. Wu and X. Gu, "Towards dropout training for convolutional neural networks," *Neural Networks*, vol. 71, no. C, pp. 1–10, 2015.
- [10] S. Park and N. Kwak, "Analysis on the dropout effect in convolutional neural networks," in *ACCV*. Springer, 2016, pp. 189–204.
- [11] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *ECCV*, 2016, pp. 646–661.
- [12] G. Kang, J. Li, and D. Tao, "Shakeout: a new regularized deep neural network training scheme," in *30th AAAI*, 2016.
- [13] Y. Li and F. Liu, "Whiteout: Gaussian adaptive noise regularization in deep neural networks," *arXiv:1612.01490*, 2016.
- [14] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout improves recurrent neural networks for handwriting recognition," in *14th ICFHR*. IEEE, 2014, pp. 285–290.
- [15] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv:1409.2329*, 2014.
- [16] T. Moon, H. Choi, H. Lee, and I. Song, "Rnndrop: A novel dropout for rnns in asr," in *IEEE Workshop on ASRU*, 2015.
- [17] R. Sennrich, B. Haddow, and A. Birch, "Edinburgh neural machine translation systems for wmt 16," in *WMT16*, 2016.
- [18] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv:1708.04552*, 2017.

- [19] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *31st NeurIPS*, 2017.
- [20] M. Volkovs, G. Yu, and T. Poutanen, "Dropoutnet: addressing cold start in recommender systems," in *31st NeurIPS*, 2017.
- [21] S. Shi, M. Zhang, Y. Liu, and S. Ma, "Attention-based adaptive model to unify warm and cold starts recommendation," in *27th ACM CIKM*, 2018, pp. 127–136.
- [22] S. Shi, M. Zhang, X. Yu, Y. Zhang, B. Hao, Y. Liu, and S. Ma, "Adaptive feature sampling for recommendation with missing content feature values," in *28th ACM CIKM*, 2019, pp. 1451–1460.
- [23] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *34th ICML*, 2017.
- [24] K. Neklyudov, D. Molchanov, A. Ashukha, and D. Vetrov, "Structured bayesian pruning via log-normal multiplicative noise," in *31st NeurIPS*, 2017, pp. 6778–6787.
- [25] A. N. Gomez, I. Zhang, K. Swersky, Y. Gal, and G. E. Hinton, "Targeted dropout," *2018 CDNNRIA Workshop at the 32nd Conference on NeurIPS*, 2018.
- [26] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *ICML*. PMLR, 2016, pp. 1050–1059.
- [27] X. Bouthillier, K. Konda, P. Vincent, and R. Memisevic, "Dropout as data augmentation," *arXiv:1506.08700*, 2015.
- [28] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019, pp. 4171–4186.
- [29] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *ICLR*, 2019.
- [30] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W.-Y. Ma, "Collaborative knowledge base embedding for recommender systems," in *22nd ACM SIGKDD*, 2016, pp. 353–362.
- [31] H. Zhao, Q. Yao, J. Li, Y. Song, and D. L. Lee, "Meta-graph based recommendation fusion over heterogeneous information networks," in *23rd ACM SIGKDD*, 2017, pp. 635–644.
- [32] B. Hu, C. Shi, W. X. Zhao, and P. S. Yu, "Leveraging meta-path based context for top-n recommendation with a neural co-attention model," in *24th ACM SIGKDD*, 2018, pp. 1531–1540.
- [33] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *ICML*, 2013.
- [34] S. J. Rennie, V. Goel, and S. Thomas, "Annealed dropout training of deep networks," in *SLT Workshop*. IEEE, 2014, pp. 159–164.
- [35] P. Morerio, J. Cavazza, R. Volpi, R. Vidal, and V. Murino, "Curriculum dropout," in *IEEE ICCV*, 2017, pp. 3544–3552.
- [36] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "Dropblock: a regularization method for convolutional networks," in *32nd NeurIPS*, 2018.
- [37] S. Semeniuta, A. Severyn, and E. Barth, "Recurrent dropout without memory loss," in *COLING*, 2016, pp. 1757–1766.
- [38] Y. Yamada, M. Iwamura, T. Akiba, and K. Kise, "Shakedrop regularization for deep residual learning," *IEEE Access*, vol. 7, pp. 186 126–186 136, 2019.
- [39] P. Chen, S. Liu, H. Zhao, and J. Jia, "Gridmask data augmentation," *arXiv:2001.04086*, 2020.
- [40] A. Labach, H. Salehinejad, and S. Valaee, "Survey of dropout methods for deep neural networks," *arXiv:1904.13310*, 2019.
- [41] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *TKDE*, vol. 17, no. 6, pp. 734–749, 2005.
- [42] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in artificial intelligence*, vol. 2009, 2009.
- [43] Z. Li, J. Tang, and T. Mei, "Deep collaborative embedding for social image understanding," *TPAMI*, vol. 41, no. 9, 2018.
- [44] Z. Sun, Q. Guo, J. Yang, H. Fang, G. Guo, J. Zhang, and R. Burke, "Research commentary on recommendations with side information: A survey and research directions," *Electronic Commerce Research and Applications*, vol. 37, p. 100879, 2019.
- [45] P. Massa and P. Avesani, "Trust-aware recommender systems," in *ACM RecSys*, 2007, pp. 17–24.
- [46] M. Jamali and M. Ester, "Trustwalker: a random walk model for combining trust-based and item-based recommendation," in *15th ACM SIGKDD*, 2009, pp. 397–406.
- [47] L. Zheng, V. Noroozi, and P. S. Yu, "Joint deep modeling of users and items using reviews for recommendation," in *WSDM*, 2017.
- [48] Y. Xu, Y. Yang, J. Han, E. Wang, F. Zhuang, and H. Xiong, "Exploiting the sentimental bias between ratings and reviews for enhancing recommendation," in *ICDM*, 2018, pp. 1356–1361.
- [49] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," *arXiv:1511.06939*, 2015.
- [50] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *ICDM*. IEEE, 2018, pp. 197–206.
- [51] H. Wang, M. Zhao, X. Xie, W. Li, and M. Guo, "Knowledge graph convolutional networks for recommender systems," in *WWW'19*.
- [52] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "Lightgcn: Simplifying and powering graph convolution network for recommendation," in *43rd ACM SIGIR*, 2020, pp. 639–648.
- [53] Y. Xu, L. Zhu, Z. Cheng, J. Li, Z. Zhang, and H. Zhang, "Multi-modal discrete collaborative filtering for efficient cold-start recommendation," *TKDE*, 2021.
- [54] T. Qian, Y. Liang, Q. Li, and H. Xiong, "Attribute graph neural networks for strict cold start recommendation," *TKDE*, 2020.
- [55] Y. Zhu, J. Lin, S. He, B. Wang, Z. Guan, H. Liu, and D. Cai, "Addressing the item cold-start problem by attribute-driven active learning," *TKDE*, vol. 32, no. 4, pp. 631–644, 2019.
- [56] Y. Zhang, I. Tsang, H. Yin, G. Yang, D. Lian, and J. Li, "Deep pairwise hashing for cold-start recommendation," *TKDE*, 2020.
- [57] J. Li, K. Lu, Z. Huang, and H. T. Shen, "On both cold-start and long-tail recommendation with social data," *TKDE*, vol. 33, no. 1, pp. 194–208, 2019.
- [58] Y. Lu, Y. Fang, and C. Shi, "Meta-learning on heterogeneous information networks for cold-start recommendation," in *26th ACM SIGKDD*, 2020, pp. 1563–1573.
- [59] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in NeurIPS*, vol. 25, pp. 1097–1105, 2012.
- [60] S. Wang and C. Manning, "Fast dropout training," in *ICML*, 2013.
- [61] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *ICML*, 2013, pp. 1319–1327.
- [62] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," in *NeurIPS*, 2015.
- [63] L. N. Smith, E. M. Hand, and T. Doster, "Gradual dropout of layers to train very deep neural networks," in *CVPR*, 2016.
- [64] Z. Li, B. Gong, and T. Yang, "Improved dropout for shallow and deep learning," in *30th NeurIPS*, 2016, pp. 2531–2539.
- [65] Y. Gal, J. Hron, and A. Kendall, "Concrete dropout," in *31st NeurIPS*, 2017, pp. 3584–3593.
- [66] A. N. Gomez, I. Zhang, S. R. Kamalakar, D. Madaan, K. Swersky, Y. Gal, and G. E. Hinton, "Learning sparse networks using targeted dropout," *arXiv:1905.13678*, 2019.
- [67] H. Salehinejad and S. Valaee, "Ising-dropout: A regularization method for training and compression of deep neural networks," in *ICASSP*. IEEE, 2019, pp. 3602–3606.
- [68] H. Salehinejad and S. Valaee, "Edropout: Energy-based dropout and pruning of deep neural networks," *IEEE TNLS*, 2021.
- [69] Z. Lu, C. Xu, B. Du, T. Ishida, L. Zhang, and M. Sugiyama, "Localdrop: A hybrid regularization for deep neural networks," *IEEE TPAMI*, 2021.
- [70] T. Gao, X. Yao, and D. Chen, "Simcse: Simple contrastive learning of sentence embeddings," *arXiv:2104.08821*, 2021.
- [71] R. Xu, F. Luo, Z. Zhang, C. Tan, B. Chang, S. Huang, and F. Huang, "Raise a child in large language model: Towards effective and generalizable fine-tuning," in *EMNLP*, 2021.
- [72] X. Liang, L. Wu, J. Li, Y. Wang, Q. Meng, T. Qin, W. Chen, M. Zhang, and T.-Y. Liu, "R-drop: Regularized dropout for neural networks," *arXiv:2106.14448*, 2021.
- [73] Y. Chen and Z. Yi, "Adaptive sparse dropout: Learning the certainty and uncertainty in deep neural networks," *Neurocomputing*, vol. 450, pp. 354–361, 2021.
- [74] S. Khan, M. Hayat, and F. Porikli, "Regularization of deep neural networks with spectral dropout." *Neural Networks: the Official Journal of the INNS*, vol. 110, pp. 82–90, 2018.
- [75] S. Cai, Y. Shu, G. Chen, B. C. Ooi, W. Wang, and M. Zhang, "Effective and efficient dropout for deep convolutional neural networks," *arXiv:1904.03392*, 2019.
- [76] S. Hou and Z. Wang, "Weighted channel dropout for regularization of deep convolutional neural network," in *AAAI*, vol. 33, no. 01, 2019, pp. 8425–8432.
- [77] Y. Zeng, T. Dai, B. Chen, S.-T. Xia, and J. Lu, "Correlation-based structural dropout for convolutional neural networks," *Pattern Recognition*, p. 108117, 2021.
- [78] H. Pham and Q. Le, "Autodropout: Learning dropout patterns to regularize deep networks," in *AAAI*, vol. 35, no. 11, 2021.

- [79] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *30th NeurIPS*, 2016.
- [80] D. Krueger, T. Maharaj, J. Kramár, M. Pezeshki, N. Ballas, N. R. Ke, A. Goyal, Y. Bengio, A. Courville, and C. Pal, "Zoneout: Regularizing rnn's by randomly preserving hidden activations," *arXiv:1606.01305*, 2016.
- [81] S. Merity, N. S. Keskar, and R. Socher, "Regularizing lstm language models," in *ICLR*, 2018.
- [82] K. Zolna, D. Arpit, D. Suhrddy, and Y. Bengio, "Fraternal dropout," in *ICLR*, 2018.
- [83] S. Singh, D. Hoiem, and D. Forsyth, "Swapout: learning an ensemble of deep architectures," in *30th NeurIPS*, 2016, pp. 28–36.
- [84] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," *arXiv:1605.07648*, 2016.
- [85] X. Gastaldi, "Shake-shake regularization," *arXiv:1705.07485*, 2017.
- [86] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *IEEE CVPR*, 2018, pp. 8697–8710.
- [87] W. Zhou, T. Ge, F. Wei, M. Zhou, and K. Xu, "Scheduled dropout: A regularization method for transformer models," in *2020 EMNLP: Findings*, 2020, pp. 1971–1980.
- [88] H. Zhou, Z. Li, C. Ning, and J. Tang, "Cad: Scale invariant framework for real-time object detection," in *ICCV Workshops*. IEEE, 2017, pp. 760–768.
- [89] Y. Gal, "Uncertainty in deep learning," Ph.D. dissertation, Department of Engineering, University of Cambridge, Sept. 2016.
- [90] L. Zhu and N. Laptev, "Deep and confident prediction for time series at uber," in *ICDM Workshop*. IEEE, 2017, pp. 103–110.
- [91] A. Jungo, R. McKinley, R. Meier, U. Knecht, L. Vera, J. Pérez-Beteta, D. Molina-García, V. M. Pérez-García, R. Wiest, and M. Reyes, "Towards uncertainty-assisted brain tumor segmentation and survival prediction," in *International MICCAI Brainlesion Workshop*. Springer, 2017, pp. 474–485.
- [92] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *31st NeurIPS*, 2017, pp. 6405–6416.
- [93] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *26th ICML*, 2009, pp. 41–48.
- [94] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *arXiv:1510.00149*, 2015.
- [95] O. İrsoy and E. Alpaydın, "Dropout regularization in hierarchical mixture of experts," *Neurocomputing*, vol. 419, pp. 148–156, 2021.
- [96] M. K. Titsias and A. Likas, "Mixture of experts classification using a hierarchical mixture model," *Neural Computation*, vol. 14, no. 9, pp. 2221–2244, 2002.
- [97] G.-J. Qi, "Hierarchically gated deep networks for semantic segmentation," in *IEEE CVPR*, 2016, pp. 2267–2275.
- [98] E. Ragusa, C. Gianoglio, R. Zunino, and P. Gastaldo, "Random-based networks with dropout for embedded systems," *Neural Computing and Applications*, vol. 33, no. 12, pp. 6511–6526, 2021.
- [99] M. Pachitariu and M. Sahani, "Regularization and nonlinearities for neural language models: when are they needed?" *arXiv:1301.5650*, 2013.
- [100] J. Bayer, C. Osendorfer, D. Korhammer, N. Chen, S. Urban, and P. van der Smagt, "On fast dropout and its applicability to recurrent networks," *arXiv:1311.0701*, 2013.
- [101] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [102] T. Bluche, C. Kermorvant, and J. Louradour, "Where to apply dropout in recurrent neural networks for handwriting recognition?" in *2015 13th ICDAR*. IEEE, 2015, pp. 681–685.
- [103] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *NeurIPS Workshop on Deep Learning*, 2014.
- [104] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," Master's thesis, Institut für Informatik, Technische Universität München, 1991.
- [105] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.
- [106] G. Melis, C. Dyer, and P. Blunsom, "On the state of the art of evaluation in neural language models," in *ICLR*, 2018.
- [107] X. Ma, Y. Gao, Z. Hu, Y. Yu, Y. Deng, and E. Hovy, "Dropout with expectation-linear regularization," *arXiv:1609.08017*, 2016.
- [108] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016, pp. 770–778.
- [109] M. Ghazvininejad, O. Levy, Y. Liu, and L. Zettlemoyer, "Mask-predict: Parallel decoding of conditional masked language models," in *EMNLP-IJCNLP*, 2019, pp. 6114–6123.
- [110] Y. Sun, S. Wang, Y. Li, S. Feng, X. Chen, H. Zhang, X. Tian, D. Zhu, H. Tian, and H. Wu, "Ernie: Enhanced representation through knowledge integration," *arXiv:1904.09223*, 2019.
- [111] Y. Cui, W. Che, T. Liu, B. Qin, Z. Yang, S. Wang, and G. Hu, "Pre-training with whole word masking for chinese bert," *arXiv:1906.08101*, 2019.
- [112] X. Wu, T. Zhang, L. Zang, J. Han, and S. Hu, "" mask and infill": Applying masked language model to sentiment transfer," *arXiv:1908.08039*, 2019.
- [113] Z.-X. Ye, Q. Chen, W. Wang, and Z.-H. Ling, "Align, mask and select: A simple method for incorporating commonsense knowledge into language representation models," *arXiv:1908.06725*, 2019.
- [114] J. Zhang, Y. Zhao, M. Saleh, and P. Liu, "Pegasus: Pre-training with extracted gap-sentences for abstractive summarization," in *ICML*. PMLR, 2020, pp. 11 328–11 339.
- [115] H. Zhang, S. Qiu, X. Duan, and M. Zhang, "Token drop mechanism for neural machine translation," in *28th COLING*, 2020, pp. 4298–4303.
- [116] Y. Gu, Z. Zhang, X. Wang, Z. Liu, and M. Sun, "Train no evil: Selective masking for task-guided pre-training," in *2020 EMNLP*, 2020, pp. 6966–6974.
- [117] K. Zhou, H. Wang, W. X. Zhao, Y. Zhu, S. Wang, F. Zhang, Z. Wang, and J.-R. Wen, "S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization," in *29th ACM CIKM*, 2020, pp. 1893–1902.
- [118] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," in *AAAI*, vol. 34, no. 07, 2020.
- [119] K. K. Singh and Y. J. Lee, "Hide-and-peek: Forcing a network to be meticulous for weakly-supervised object and action localization," in *2017 IEEE ICCV*. IEEE Computer Society, 2017, pp. 3544–3553.
- [120] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *arXiv:1710.09412*, 2017.
- [121] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, D. Lopez-Paz, and Y. Bengio, "Manifold mixup: Better representations by interpolating hidden states," in *ICML*, 2019.
- [122] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, "Cutmix: Regularization strategy to train strong classifiers with localizable features," in *IEEE/CVF ICCV*, 2019, pp. 6023–6032.
- [123] D. Walawalkar, Z. Shen, Z. Liu, and M. Savvides, "Attentive cutmix: An enhanced data augmentation approach for deep learning based image classification," in *ICASSP*, 2020.
- [124] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," *arXiv preprint arXiv:2111.06377*, 2021.
- [125] J. Chen, T. Ma, and C. Xiao, "Fastgcn: Fast learning with graph convolutional networks via importance sampling," in *ICLR*, 2018.
- [126] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *32nd NeurIPS*, 2018, pp. 4563–4572.
- [127] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv:1710.10903*, 2017.
- [128] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," *Advances in NeurIPS*, 2019.
- [129] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang, "Graph random neural networks for semi-supervised learning on graphs," *NeurIPS*, 2020.
- [130] Y. Ye and S. Ji, "Sparse graph attention networks," *TKDE*, 2021.
- [131] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *31st NeurIPS*, 2017, pp. 6000–6010.
- [132] R. Speer, J. Chin, and C. Havasi, "Conceptnet 5.5: an open multilingual graph of general knowledge," in *AAAI*, 2017.
- [133] C. Wang, Y. Wu, Y. Du, J. Li, S. Liu, L. Lu, S. Ren, G. Ye, S. Zhao, and M. Zhou, "Semantic mask for transformer based end-to-end speech recognition," *Proc. Interspeech 2020*, pp. 971–975, 2020.
- [134] E. D. Cubuk, B. Zoph, D. Mané, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation strategies from data," in *IEEE/CVF CVPR*. IEEE Computer Society, 2019, pp. 113–123.

- [135] Y. Weng, X. Chen, L. Chen, and L. Wei, "Gain: Graph attention & interaction network for inductive semi-supervised learning over large-scale graphs," *TKDE*, 2020.
- [136] Z. Huan, Y. Quanming, and T. Weiwei, "Search to aggregate neighborhood for graph neural network," in *37th ICDE*, 2021.
- [137] Z. Wang, T. Xia, R. Jiang, X. Liu, K.-S. Kim, X. Song, and R. Shibasaki, "Forecasting ambulance demand with profiled human mobility via heterogeneous multi-graph neural networks," in *37th ICDE*. IEEE, 2021, pp. 1751–1762.
- [138] S. Wager, S. Wang, and P. Liang, "Dropout training as adaptive regularization," in *26th NeurIPS-Volume 1*, 2013, pp. 351–359.
- [139] D. P. Helmbold and P. M. Long, "On the inductive bias of dropout," *JMLR*, vol. 16, no. 1, pp. 3403–3454, 2015.
- [140] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," in *25th UAI*, 2009, pp. 452–461.
- [141] X. He and T.-S. Chua, "Neural factorization machines for sparse predictive analytics," in *40th ACM SIGIR*, 2017, pp. 355–364.
- [142] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *38th ACM SIGIR*, 2015, pp. 43–52.
- [143] R. He and J. McAuley, "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering," in *25th WWW*, 2016, pp. 507–517.
- [144] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM TITS*, vol. 5, no. 4, pp. 1–19, 2015.
- [145] W. Krichene and S. Rendle, "On sampled metrics for item recommendation," in *26th ACM SIGKDD*, 2020, pp. 1748–1757.
- [146] Z. Liu, Z. Fan, Y. Wang, and P. S. Yu, "Augmenting sequential recommendation with pseudo-prior items via reversely pre-training transformer," *arXiv:2105.00522*, 2021.
- [147] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," in *42nd ACM SIGIR*, 2019, pp. 165–174.
- [148] P. Baldi and P. Sadowski, "Understanding dropout," in *26th NeurIPS-Volume 2*, 2013, pp. 2814–2822.
- [149] Y. Gal and Z. Ghahramani, "Bayesian convolutional neural networks with bernoulli approximate variational inference," *arXiv:1506.02158*, 2015.



**Yangkun Li** is currently pursuing a Ph.D. degree in Department of Computer Science and Technology at Tsinghua University. He received his B.E. from Department of Computer Science and Technology, Tsinghua University in 2021. His research interests focus on personalized recommender systems, user modeling, and data mining.



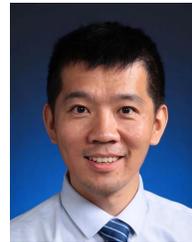
**Weizhi Ma** is currently a research assistant professor at the Institute for AI Industry Research (AIR), Tsinghua University. He received his Ph.D. from Tsinghua University in 2019. His major research interests are in recommender systems, user modeling, and information retrieval. He has served as PC member for top conferences including KDD, WSDM, AAI, theWebConf, SIGIR, EMNLP, COLING, and CIKM, and as reviewer for journals including TKDE, TOIS, and TNNLS.



**Chong Chen** is now a final year Ph.D. student in Dept. of Computer Science and Technology in Tsinghua University. He received his B.E. from Tsinghua University in 2017. He has over 10 publications appeared in top conferences and journals including AAI, WSDM, theWebConf, SIGIR, and TOIS. His research interests include deep learning, user modeling, and recommendation.



**Min Zhang** is currently an associate professor of Dept. of Computer Science and Technology, Tsinghua University. She received her Ph.D. in Tsinghua University in 2003. Her research interests include web information retrieval and recommendation, user behavior analysis and profiling, machine learning, and data mining. She is currently serving as Editor-in-Chief of ACM TOIS, and has served as co-chair, senior PC member, or area chair for top conferences including WSDM, theWebConf, SIGIR, and CIKM.



**Yiqun Liu** is working as a professor and department co-chair at Dept. of Computer Science and Technology in Tsinghua University. He received his Ph.D. in Tsinghua University in 2007. His major research interests are in web search, user behavior analysis, and information retrieval. He serves as co-Editor-in-Chief of FnTIR and has served as co-chair, senior PC member, or area chair for top conferences including IJCAI, theWebConf, SIGIR, CIKM, and NTCIR.



**Shaoping Ma** is a professor of Dept. of Computer Science and Technology in Tsinghua University. He received his Ph.D. in Tsinghua University in 1997. He devotes his life to exploring and researching intelligent information processing, including information retrieval, recommendation and machine learning. He has authored over 300 book chapters, journals, and top conference papers in his research area.



**Yuekui Yang** is currently pursuing an Eng.D. degree in Department of Computer Science and Technology, Tsinghua University. He received his B.E. from Taiyuan University of Technology in 2006. His research interests focus on recommendation systems and computational advertising. He is now working as a director in Tencent AI Platform Department.

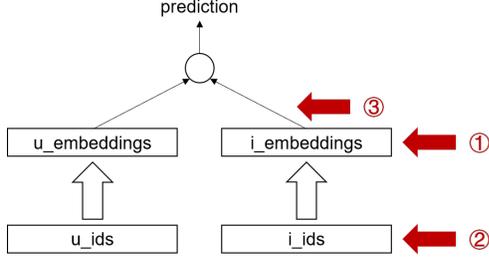


Fig. 13. Dropout on BPR.

## APPENDIX A PROTOCOL USED IN THE SELECTION PROCESS OF THE ARTICLES IN THIS SURVEY

We searched with the query “dropout” on Google Scholar, selected the papers that were about dropout methods in neural models by checking the title and abstract of the articles, and we got around 200 articles. We further considered each article carefully, selecting the articles that met the following requirements:

- The article was published on top AI conferences or journals.
- The article had proposed new dropout methods, not just applied existing dropout methods.
- Some articles available on arxiv but not published on conferences or journals, which have proposed good dropout methods with some citations, were also included.

Then we checked the references of these articles, added the articles they cited and meeting the above requirements but missed by the search engine into our list. In this way, we finally got the current about 80 articles.

We sort the articles first according to our classification taxonomy. Within the same category, the articles are sorted mainly based on their publication date. Some highly related papers which are introduced together are placed in adjacent positions.

## APPENDIX B IMPLEMENTATION DETAILS OF DROPOUT METHODS ON RECOMMENDATION MODELS

We first introduce the criterion we adopt to select the implementations of dropout methods, then we introduce the implementation details of dropout methods on each recommendation model.

### B.1 Criterion of the Implementations

For dropping model structures, we implement dropout according to the original papers of the recommendation models: NFM [141], GRU4Rec [49], SASRec [50], and LightGCN [52]. Because dropping model structure is a universal type of dropout, all the neural recommendation models have the corresponding implementations. The original paper of BPR [140] is published before dropout was proposed, so we randomly drops the parameters in BPR model, which is consistent with the definition of dropping model structure. Our implementation of dropping graph information is also

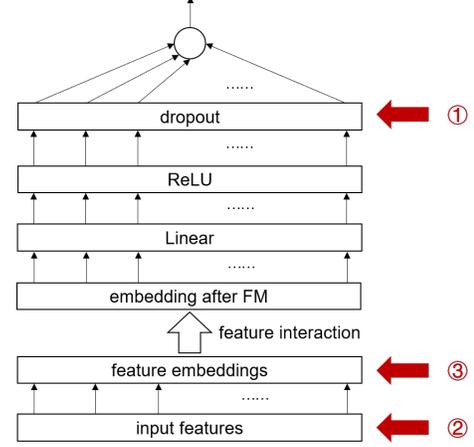


Fig. 14. Dropout on NFM.

according to the original paper of LightGCN, which referred its implementation to NGCF [147].

For dropping embeddings, we implement dropout according to the original papers (ACCM [21] and AFS [22]) where embedding dropout was first proposed as a formal method.

For dropping input information, we implement dropout according to the definition of “input information” in recommendation models, i.e., user ids, item ids, and features.

Following subsections are the implementation details of dropout methods on each recommendation model.

### B.2 Dropout on BPR

- **Drop model structure:** standard dropout, achieved by adding a dropout layer after the user and item embedding matrix.
- **Drop input information:** randomly set some of the user and item numbers in each batch to random numbers.
- **Drop embedding:** randomly set the user and item embeddings in each batch to random embeddings.

The schematic diagram is shown in figure 13. The ① in the figure indicates the dropout of model structure, ② indicates the dropout of input information, and ③ indicates the dropout of embeddings.

### B.3 Dropout on NFM

- **Drop model structure:** standard dropout, achieved by adding a dropout layer after the ReLU layer in the deep part.
- **Drop input information:** randomly drops the attribute information of users and items in each batch by setting them to random values.
- **Drop embedding:** randomly drops the embeddings corresponding to the attribute information of users and items in each batch by setting them to random embeddings.

The schematic diagram is shown in figure 14. The ① in the figure indicates the dropout of model structure, ② indicates the dropout of input information, and ③ indicates the dropout of embeddings.

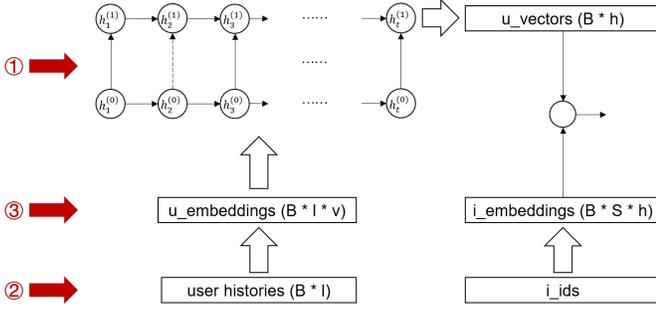


Fig. 15. Dropout on GRU4Rec.

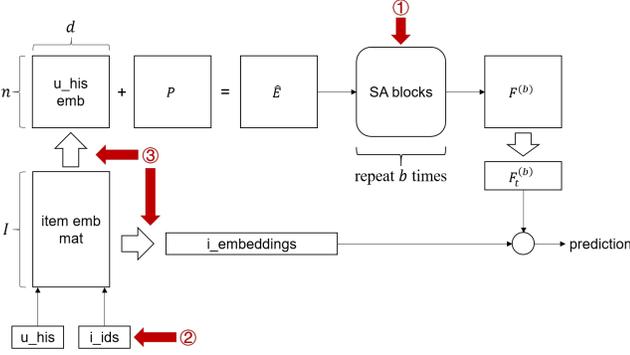


Fig. 16. Dropout on SASRec.

#### B.4 Dropout on GRU4Rec

- **Drop model structure:** randomly dropout of feed-forward connections, following Zaremba et al. [15].
- **Drop input information:** randomly set some of the user and item numbers in each batch to random numbers.
- **Drop embedding:** randomly set the user and item embeddings in each batch to random embeddings.

The schematic diagram is shown in figure 15. The ① in the figure indicates the dropout of model structure, ② indicates the dropout of input information, and ③ indicates the dropout of embeddings.

#### B.5 Dropout on SASRec

- **Drop model structure:** adding dropout layers within the self-attentive block to dropout neuron outputs, following the original article [50].
- **Drop input information:** randomly set some of the user and item numbers in each batch to random numbers.
- **Drop embedding:** randomly set the user and item embeddings in each batch to random embeddings.

The schematic diagram is shown in figure 16. The ① in the figure indicates the dropout of model structure, ② indicates the dropout of input information, and ③ indicates the dropout of embeddings.

#### B.6 Dropout on LightGCN

- **Drop model structure:** add a dropout layer after the user-item embedding matrix.
- **Drop input information:** randomly set some of the user and item numbers in each batch to random numbers.

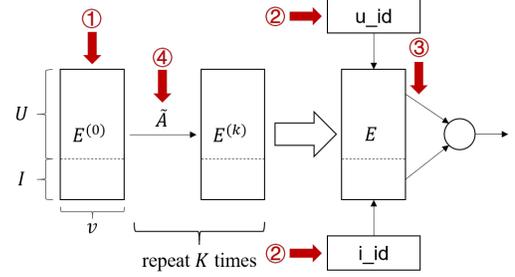


Fig. 17. Dropout on LightGCN.

- **Drop embedding:** randomly set the user and item embeddings in each batch to random embeddings.
- **Drop graph information:** for each batch, randomly drop some edges in the graph. This can be achieved by randomly dropping elements of the symmetrically normalized adjacency matrix  $\tilde{A}$  [52].

The schematic is shown in figure 17. The ① in the figure indicates the dropout of model structure, ② indicates the dropout of input information, ③ indicates the dropout of embeddings, and ④ indicates the dropout of the edges.

## APPENDIX C

### EXPERIMENTAL SETTINGS AND MODEL PARAMETERS

Parameter values taken for all models in common are shown in Table 5. Parameters specific to each model are shown in Table 6.

TABLE 5  
Global Parameters

Parameter	Value
Learning rate	0.001
Optimizer	Adam
Batch size	128
Early stop	50
Validation metrics	NDCG@10
Evaluation metrics	NDCG@5,10,20,50; HR@10,20
Neg. sample during training	1
Neg. sample during testing	all [145]
Embedding size	64
Loss function	BPR loss [140]

We grid search all main parameters and choose the parameter set that achieves the best performance on validation set. We use leave-one-out strategy to get the validation set and the test set. We adopt early stop, stopping training when model performance has not increased on validation set for 50 epochs. We run each experiment for ten times with ten random seeds, and do significance test using t-test.

## APPENDIX D

### EXPERIMENT DATA

TABLE 6  
Parameters specific to each model

Model	Parameter	Value
NFM	Number of layers	1
	Hidden state size	64
GRU4Rec	User history length	20
	Number of layers	2
	Hidden vector size	64
SASRec	User history length	20
	Number of self-attention heads	1
LightGCN	Number of layers	3

TABLE 7  
Detailed results for dropout methods on BPRMF, ml1m-cold dataset.

Dropout Methods	Dropout Ratio	NDCG @5	NDCG @10	NDCG @20	NDCG @50	HR @10	HR @20
Origin	-	0.0251	0.0339	0.0458	0.0667	0.0678	0.1155
Drop Model Structure	0.1	0.0258	0.0355*	0.0479*	0.0697**	0.0718*	0.1213*
	0.2	<b>0.0267*</b>	0.0359**	0.0487**	<b>0.0709**</b>	0.0715**	<b>0.1225**</b>
	0.3	<b>0.0267**</b>	<b>0.0364**</b>	<b>0.0488**</b>	0.0701**	<b>0.0721**</b>	0.1220**
Drop Input Info	0.1	0.0233**	0.0324*	0.0444*	0.0658	0.0668	0.1148
	0.2	0.0241*	0.0331	0.0448	0.0660	0.0671	0.1138
	0.3	0.0217**	0.0294**	0.0394**	0.0573**	0.0586**	0.0988**
Drop Embedding	0.1	0.0236*	0.0324*	0.0450	0.0660	0.0658	0.1162
	0.2	0.0236*	0.0330	0.0451	0.0660	0.0672	0.1155
	0.3	0.0241	0.0332	0.0453	0.0658	0.0672	0.1154

\* for  $p < 0.05$ , \*\* for  $p < 0.01$ , compared to the origin (not using any dropout methods). Bold numbers are the best results of each column.

TABLE 8  
Detailed results for dropout methods on BPRMF, Amazon Baby 5-core dataset.

Dropout Methods	Dropout Ratio	NDCG @5	NDCG @10	NDCG @20	NDCG @50	HR @10	HR @20
Origin	-	0.00709	0.00969	0.01300	0.01936	0.01942	0.03263
Drop Model Structure	0.1	0.00693	0.00935	0.01253	0.01946	0.01872	0.03145
	0.2	<b>0.00723</b>	<b>0.00973</b>	<b>0.01311</b>	<b>0.01991</b>	<b>0.01943</b>	0.03292
	0.3	0.00706	0.00959	<b>0.01311</b>	0.01984*	0.01907	<b>0.03315</b>
Drop Input Info	0.1	0.00631*	0.00888**	0.01229*	0.01892	0.01839*	0.03202
	0.2	0.00610**	0.00839**	0.01180**	0.01836**	0.01720**	0.03078*
	0.3	0.00544**	0.00761**	0.01077**	0.01671**	0.01581**	0.02844**
Drop Embedding	0.1	0.00664*	0.00916*	0.01271	0.01933	0.01864	0.03283
	0.2	0.00641**	0.00900**	0.01241**	0.01943	0.01868*	0.03231
	0.3	0.00661	0.00911*	0.01241*	0.01914	0.01862	0.03177

\* for  $p < 0.05$ , \*\* for  $p < 0.01$ , compared to the origin (not using any dropout methods). Bold numbers are the best results of each column.

TABLE 9  
Detailed results for dropout methods on NFM, ml1m-cold dataset.

Dropout Methods	Dropout Ratio	NDCG @5	NDCG @10	NDCG @20	NDCG @50	HR @10	HR @20
Origin	-	0.0246	0.0335	0.0454	0.0664	0.0680	0.1153
Drop Model Structure	0.1	0.0255	0.0350*	0.0472*	0.0682**	0.0708*	<b>0.1193*</b>
	0.2	<b>0.0260*</b>	0.0353*	0.0473*	0.0687**	0.0704*	0.1184
	0.3	0.0257*	0.0352**	<b>0.0474**</b>	0.0684**	0.0705*	0.1191*
Drop Input Info	0.1	0.0242	0.0334	0.0457	0.0678**	0.0677	0.1167
	0.2	0.0240	0.0332	0.0458	0.0675*	0.0670	0.1172
	0.3	0.0135**	0.0184**	0.0244**	0.0371**	0.0369**	0.0608**
Drop Embedding	0.1	0.0259**	<b>0.0354**</b>	0.0473**	<b>0.0691**</b>	<b>0.0716**</b>	0.1188*
	0.2	0.0251	0.0345	0.0470	0.0686**	0.0695	<b>0.1193*</b>
	0.3	0.0231**	0.0326	0.0449	0.0666	0.0667	0.1159

\* for  $p < 0.05$ , \*\* for  $p < 0.01$ , compared to the origin (not using any dropout methods). Bold numbers are the best results of each column.

TABLE 10  
Detailed results for dropout methods on NFM, Amazon Baby 5-core dataset.

Dropout Methods	Dropout Ratio	NDCG @5	NDCG @10	NDCG @20	NDCG @50	HR @10	HR @20
Origin	-	0.00458	0.00657	0.00926	0.01444	0.01376	0.02451
Drop Model Structure	0.1	0.00515*	0.00715*	0.01010**	0.01592**	0.01496**	0.02677**
	0.2	<b>0.00518*</b>	<b>0.00737**</b>	<b>0.01046**</b>	<b>0.01618**</b>	<b>0.01545**</b>	<b>0.02778**</b>
	0.3	0.00499	0.00705	0.00985	0.01534	0.01475	0.02588
Drop Input Info	0.1	0.00460	0.00643	0.00898	0.01441	0.01357	0.02386
	0.2	0.00465	0.00654	0.00923	0.01489	0.01388	0.02469
	0.3	0.00468	0.00649	0.00899	0.01467	0.01379	0.02381
Drop Embedding	0.1	0.00424	0.00610	0.00858	0.01348	0.01264	0.02255
	0.2	0.00460	0.00640	0.00892	0.01440	0.01352	0.02359
	0.3	0.00473	0.00647	0.00913	0.01479	0.01369	0.02435

\* for  $p < 0.05$ , \*\* for  $p < 0.01$ , compared to the origin (not using any dropout methods). Bold numbers are the best results of each column.

TABLE 11  
Detailed results for dropout methods on GRU4Rec, ml1m-cold dataset.

Dropout Methods	Dropout Ratio	NDCG @5	NDCG @10	NDCG @20	NDCG @50	HR @10	HR @20
Origin	-	0.0752	0.0964	0.1196	0.1496	0.1818	0.2739
Drop Model Structure	0.1	0.0782*	0.1003**	0.1233**	0.1536**	0.1892**	0.2805**
	0.2	0.0792**	0.1010**	0.1245**	0.1544**	0.1895**	0.2830**
	0.3	0.0780*	0.1004**	0.1239**	0.1538**	0.1902**	0.2834**
Drop Input Info	0.1	<b>0.0859**</b>	<b>0.1084**</b>	<b>0.1323**</b>	<b>0.1625**</b>	<b>0.2012**</b>	<b>0.2957**</b>
	0.2	0.0829**	0.1056**	0.1297**	0.1598**	0.1973**	0.2931**
	0.3	0.0811**	0.1024**	0.1255**	0.1555**	0.1904**	0.2818**
Drop Embedding	0.1	0.0791**	0.1013**	0.1246**	0.1552**	0.1911**	0.2840**
	0.2	0.0784**	0.1012**	0.1245**	0.1554**	0.1925**	0.2852**
	0.3	0.0779*	0.1007**	0.1248**	0.1557**	0.1920**	0.2875**

\* for  $p < 0.05$ , \*\* for  $p < 0.01$ , compared to the origin (not using any dropout methods). Bold numbers are the best results of each column.

TABLE 12  
Detailed results for dropout methods on GRU4Rec, Amazon Baby 5-core dataset.

Dropout Methods	Dropout Ratio	NDCG @5	NDCG @10	NDCG @20	NDCG @50	HR @10	HR @20
Origin	-	0.00762	0.01071	0.01487	0.02261	0.02199	0.03856
Drop Model Structure	0.1	0.00786	0.01132*	0.01555*	0.02337*	0.02347**	0.04036
	0.2	0.00810	0.01146	0.01579*	0.02342	0.02374*	0.04101*
	0.3	0.00800	0.01138*	0.01565**	0.02348*	0.02340**	0.04043**
Drop Input Info	0.1	0.01013**	0.01383**	0.01864**	0.02698**	0.02781**	0.04700**
	0.2	0.01185**	0.01622**	0.02170**	0.03073**	0.03257**	0.05440**
	0.3	<b>0.01318**</b>	<b>0.01777**</b>	<b>0.02343**</b>	<b>0.03301**</b>	<b>0.03543**</b>	<b>0.05797**</b>
Drop Embedding	0.1	0.00817	0.01129	0.01556	0.02327	0.02311	0.04010
	0.2	0.00792	0.01109	0.01536	0.02311	0.02294	0.03991
	0.3	0.00785	0.01096	0.01514	0.02275	0.02255	0.03926

\* for  $p < 0.05$ , \*\* for  $p < 0.01$ , compared to the origin (not using any dropout methods). Bold numbers are the best results of each column.

TABLE 13  
Detailed results for dropout methods on SASRec, ml1m-cold dataset.

Dropout Methods	Dropout Ratio	NDCG @5	NDCG @10	NDCG @20	NDCG @50	HR @10	HR @20
Origin	-	0.0840	0.1064	0.1296	0.1593	0.1981	0.2903
Drop Model Structure	0.1	0.0864*	0.1092**	0.1326*	0.1627**	0.2013	0.2941
	0.2	0.0852	0.1077	0.1308	0.1606	0.1996	0.2912
	0.3	0.0836	0.1059	0.1290	0.1585	0.1961	0.2878
Drop Input Info	0.1	<b>0.0868*</b>	<b>0.1093</b>	<b>0.1330*</b>	<b>0.1632*</b>	<b>0.2019</b>	<b>0.2957*</b>
	0.2	0.0816*	0.1041*	0.1282	0.1589	0.1942*	0.2901
	0.3	0.0705**	0.0915**	0.1142**	0.1445**	0.1742**	0.2645**
Drop Embedding	0.1	0.0835	0.1063	0.1301	0.1606	0.1980	0.2923
	0.2	0.0814*	0.1042	0.1278	0.1588	0.1962	0.2901
	0.3	0.0797*	0.1022*	0.1259*	0.1572	0.1935	0.2876

\* for  $p < 0.05$ , \*\* for  $p < 0.01$ , compared to the origin (not using any dropout methods). Bold numbers are the best results of each column.

TABLE 14  
Detailed results for dropout methods on SASRec, Amazon Baby 5-core dataset.

Dropout Methods	Dropout Ratio	NDCG @5	NDCG @10	NDCG @20	NDCG @50	HR @10	HR @20
Origin	-	0.01039	0.01428	0.01913	0.02787	0.02889	0.04824
Drop Model Structure	0.1	0.01048	0.01468	0.01990	0.02873	0.03016	0.05100*
	0.2	0.01140**	0.01542*	0.02067**	0.02961**	0.03076*	0.05170**
	0.3	0.01145*	0.01562*	0.02095**	0.03004**	0.03154*	0.05277**
Drop Input Info	0.1	0.01308**	0.01783**	0.02375**	0.03326**	0.03600**	0.05961**
	0.2	0.01595**	0.02119**	0.02714**	0.03750**	0.04182**	0.06563**
	0.3	<b>0.01612**</b>	<b>0.02143**</b>	<b>0.02768**</b>	<b>0.03828**</b>	<b>0.04207**</b>	<b>0.06701**</b>
Drop Embedding	0.1	0.01045	0.01440	0.01964	0.02849	0.02934	0.05020*
	0.2	0.01071	0.01484	0.01985	0.02904	0.03010	0.05008
	0.3	0.01094	0.01502	0.02009	0.02950*	0.03030	0.05056*

\* for  $p < 0.05$ , \*\* for  $p < 0.01$ , compared to the origin (not using any dropout methods). Bold numbers are the best results of each column.

TABLE 15  
Detailed results for dropout methods on LightGCN, ml1m-cold dataset.

Dropout Methods	Dropout Ratio	NDCG @5	NDCG @10	NDCG @20	NDCG @50	HR @10	HR @20
Origin	-	0.0281	0.0377	0.0504	0.0722	0.0748	0.1255
Drop Model Structure	0.1	0.0287	0.0385	0.0510	0.0732	0.0761	0.1261
	0.2	<b>0.0291*</b>	<b>0.0388</b>	<b>0.0516</b>	<b>0.0737*</b>	<b>0.0765</b>	<b>0.1277</b>
	0.3	0.0286	0.0384	0.0513	0.0730	0.0757	0.1273
Drop Input Info	0.1	0.0264**	0.0361**	0.0484**	0.0705**	0.0721**	0.1213**
	0.2	0.0254**	0.0347**	0.0466**	0.0674**	0.0690**	0.1163**
	0.3	0.0239**	0.0328**	0.0443**	0.0642**	0.0653**	0.1113**
Drop Embedding	0.1	0.0280	0.0376	0.0505	0.0723	0.0748	0.1259
	0.2	0.0272	0.0371	0.0497	0.0709	0.0741	0.1242
	0.3	0.0264*	0.0361**	0.0482**	0.0696**	0.0724*	0.1205*
Drop Graph Info	0.1	0.0282	0.0380	0.0508	0.0723	0.0754	0.1265
	0.2	0.0284	0.0383	0.0511	0.0729	0.0764	0.1273
	0.3	0.0283	0.0380	0.0509	0.0727	0.0756	0.1269

\* for  $p < 0.05$ , \*\* for  $p < 0.01$ , compared to the origin (not using any dropout methods). Bold numbers are the best results of each column.

TABLE 16  
Detailed results for dropout methods on LightGCN, Amazon Baby 5-core dataset.

Dropout Methods	Dropout Ratio	NDCG @5	NDCG @10	NDCG @20	NDCG @50	HR @10	HR @20
Origin	-	0.01032	0.01392	0.01843	0.02738	0.02779	0.04577
Drop Model Structure	0.1	0.01017	0.01392	0.01835	0.02743	0.02809	0.04577
	0.2	0.01001	0.01370	0.01807	0.02730	0.02763	0.04502
	0.3	0.01001	0.01357	0.01811	0.02703	0.02693	0.04509
Drop Input Info	0.1	0.00936**	0.01275**	0.01710**	0.02598**	0.02559**	0.04301**
	0.2	0.00872**	0.01207**	0.01651**	0.02532**	0.02435**	0.04209**
	0.3	0.00824**	0.01140**	0.01583**	0.02456**	0.02321**	0.04095**
Drop Embedding	0.1	<b>0.01043</b>	0.01402	0.01866	0.02754	0.02781	0.04633
	0.2	0.01034	<b>0.01420</b>	<b>0.01869</b>	<b>0.02773*</b>	<b>0.02853</b>	<b>0.04651</b>
	0.3	0.01029	0.01389	0.01841	0.02757	0.02798	0.04595
Drop Graph Info	0.1	0.01020	0.01403	0.01838	0.02733	0.02821	0.04552
	0.2	0.01005*	0.01375	0.01811*	0.02722	0.02766	0.04503
	0.3	0.01021	0.01395	0.01843	0.02736	0.02799	0.04584

\* for  $p < 0.05$ , \*\* for  $p < 0.01$ , compared to the origin (not using any dropout methods). Bold numbers are the best results of each column.