DeepGBM: A Deep Learning Framework Distilled by GBDT for Online Prediction Tasks

Guolin Ke
Microsoft Research
guolin.ke@microsoft.com

Zhenhui Xu*
Peking University
zhenhui.xu@pku.edu.cn

Jia Zhang
Microsoft Research
jia.zhang@microsoft.com

Jiang Bian
Microsoft Research
jiang.bian@microsoft.com

Tie-Yan Liu
Microsoft Research
tie-yan.liu@microsoft.com

ABSTRACT

Online prediction has become one of the most essential tasks in many real-world applications. Two main characteristics of typical online prediction tasks include *tabular input space* and *online data generation*. Specifically, *tabular input space* indicates the existence of both sparse categorical features and dense numerical ones, while *online data generation* implies continuous task-generated data with potentially dynamic distribution. Consequently, effective learning with tabular input space as well as fast adaption to online data generation become two vital challenges for obtaining the online prediction model. Although Gradient Boosting Decision Tree (GBDT) and Neural Network (NN) have been widely used in practice, either of them yields their own weaknesses. Particularly, GBDT can hardly be adapted to dynamic online data generation, and it tends to be ineffective when facing sparse categorical features; NN, on the other hand, is quite difficult to achieve satisfactory performance when facing dense numerical features. In this paper, we propose a new learning framework, *DeepGBM*, which integrates the advantages of the both NN and GBDT by using two corresponding NN components: (1) *CatNN*, focusing on handling sparse categorical features. (2) *GBDT2NN*, focusing on dense numerical features with distilled knowledge from GBDT. Powered by these two components, DeepGBM can leverage both categorical and numerical features while retaining the ability of efficient online update. Comprehensive experiments on a variety of publicly available datasets have demonstrated that DeepGBM can outperform other well-recognized baselines in various online prediction tasks.

KEYWORDS

Neural Network; Gradient Boosting Decision Tree

ACM Reference Format:
Guolin Ke, Zhenhui Xu, Jia Zhang, Jiang Bian, and Tie-Yan Liu. 2019. DeepGBM: A Deep Learning Framework Distilled by GBDT for Online Prediction Tasks. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330858

*Work primarily done while visiting Microsoft Research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
KDD '19, August 4–8, 2019, Anchorage, AK, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330858

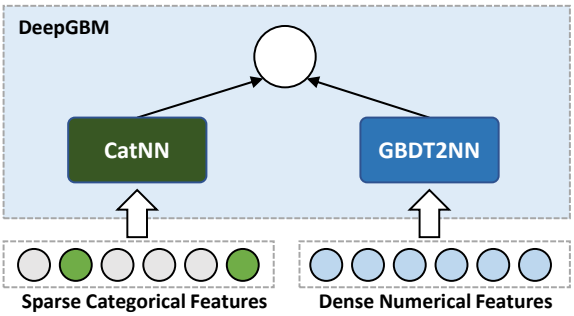

Figure 1: The framework of DeepGBM, which consists of two components, CatNN and GBDT2NN, to handle the sparse categorical and dense numerical features, respectively.

1 INTRODUCTION

Online prediction represents a certain type of tasks playing the essential role in many real-world industrial applications, such as click prediction [21, 22, 36, 51] in sponsored search, content ranking [1, 6, 7] in Web search, content optimization [9, 10, 47] in recommender systems, travel time estimation [31, 49] in transportation planning, etc.

A typical online prediction task usually yields two specific characteristics in terms of the *tabular input space* and the *online data generation*. In particular, the *tabular input space* means that the input features of an online prediction task can include both categorical and numerical tabular features. For example, the feature space of the click prediction task in sponsored search usually contains categorical ones like the ad category as well as numerical ones like the textual similarity between the query and the ad. In the meantime, the *online data generation* implies that the real data of those tasks are generated in the online mode and the data distribution could be dynamic in real time. For instance, the news recommender system can generate a massive amount of data in real time, and the ceaseless emerging news could give rise to dynamic feature distribution at a different time.

Therefore, to pursue an effective learning-based model for the online prediction tasks, it becomes a necessity to address two main challenges: (1) how to learn an effective model with tabular input space; and (2) how to adapt the model to the online data generation. Currently, two types of machine learning models are widely used to

solve online prediction tasks, i.e., Gradient Boosting Decision Tree (GBDT) and Neural Network (NN)[1]. Unfortunately, neither of them can simultaneously address both of those two main challenges well. In other words, either GBDT or NN yields its own pros and cons when being used to solve the online prediction tasks.

On one side, GBDT's main advantage lies in its capability in handling dense numerical features effectively. Since it can iteratively pick the features with the largest statistical information gain to build the trees [20, 45], GBDT can automatically choose and combine the useful numerical features to fit the training targets well [2]. That is why GBDT has demonstrated its effectiveness in click prediction [33], web search ranking [6], and other well-recognized prediction tasks [8]. Meanwhile, GBDT has two main weaknesses in online prediction tasks. First, as the learned trees in GBDT are not differentiable, it is hard to update the GBDT model in the online mode. Frequent retraining from scratch makes GBDT quite inefficient in learning over online prediction tasks. This weakness, moreover, prevents GBDT from learning over very large scale data, since it is usually impractical to load a huge amount of data into the memory for learning [3].

The second weakness of GBDT is its ineffectiveness in learning over sparse categorical features[4]. Particularly, after converting categorical features into sparse and high-dimensional one-hot encodings, the statistical information gain will become very small on sparse features, since the gain of imbalance partitions by sparse features is almost the same as non-partition. As a result, GBDT fails to use sparse features to grow trees effectively. Although there are some other categorical encoding methods [41] that can directly convert a categorical value into a dense numerical value, the raw information will be hurt in these methods as the encode values of different categories could be similar and thus we cannot distinguish them. Categorical features also could be directly used in tree learning, by enumerating possible binary partitions [16]. However, this method often over-fits to the training data when with sparse categorical features, since there is too little data in each category and thus the statistical information is biased [29]. In short, while GBDT can learn well over dense numerical features, the two weaknesses, i.e., the difficulty in adapting to online data generation and the ineffectiveness in learning over sparse categorical features, cause GBDT to fail in many online prediction tasks, especially those requiring the model being online adapted and those containing many sparse categorical features.

On the other side, NN's advantages consist of its efficient learning over large scale data in online tasks since the batch-mode back-propagation algorithm as well as its capability in learning over sparse categorical features by the well-recognized embedding structure [35, 38]. Some recent studies have revealed the success of employing NN in those online prediction tasks, including click prediction [22, 36, 51] and recommender systems [9, 10, 32, 38, 47]. Nevertheless, the main challenge of NN lies in its weakness in learning over dense numerical tabular features. Although a Fully Connected Neural Network (FCNN) could be used for dense numerical features directly, it usually leads to unsatisfactory performance, because its fully connected model structure leads to very complex optimization hyper-planes with a high risk of falling into local optimums [15]. Thus, in many tasks with dense numerical tabular features, NN often cannot outperform GBDT [8]. To sum up, despite NN can effectively handle sparse categorical features and be adapted to online data generation efficiently, it is still difficult to result in an effective model by learning over dense numerical tabular features.

Table 1: Comparison over different models.

	NN	GBDT	GBDT+NN	**DeepGBM**
Sparse Categorical Feature	✓	✗	✓	✓
Dense Numerical Feature	✗	✓	✓	✓
Online update & Large-scale data	✓	✗	✗	✓

As summarized in Table 1, either NN or GBDT yields its own pros and cons for obtaining the model for online prediction tasks. Intuitively, it will be quite beneficial to explore how to combine the advantages of both NN and GBDT together, to address the two major challenges in online prediction tasks, i.e., *tabular input space* and *online data generation*, simultaneously.

In this paper, we propose a new learning framework, *DeepGBM*, which integrates NN and GBDT together, to obtain a more effective model for generic online prediction tasks. In particular, the whole DeepGBM framework, as shown in Fig. 1, consists of two major components: *CatNN* being an NN structure with the input of categorical features and *GBDT2NN* being another NN structure with the input of numerical features. To take advantage of GBDT's strength in learning over numerical features, GBDT2NN attempts to distill the knowledge learned by GBDT into an NN modeling process. Specifically, to boost the effectiveness of knowledge distillation [24], GBDT2NN does not only transfer the output knowledge of the pre-trained GBDT but also incorporates the knowledge of both feature importance and data partition implied by tree structures from obtained trees. In this way, in the meantime achieving the comparable performance with GBDT, GBDT2NN, with the NN structure, can be easily updated by continuous emerging data when facing the online data generation.

Powered by two NN based components, *CatNN* and *GBDT2NN*, DeepGBM can indeed yield strong learning capacity over both categorical and numerical features while retaining the vital ability of efficient online learning. To illustrate the effectiveness of the proposed DeepGBM, we conduct extensive experiments on various publicly available datasets with tabular data. Comprehensive experimental results demonstrate that DeepGBM can outperform other solutions in various prediction tasks.

In summary, the contributions of this paper are multi-fold:

- We propose DeepGBM to leverage both categorical and numerical features while retaining the ability of efficient online update, for all kinds of prediction tasks with tabular data, by combining the advantages of GBDT and NN.
- We propose an effective solution to distill the learned knowledge of a GBDT model into an NN model, by considering the selected inputs, structures and outputs knowledge in the learned tree of GBDT model.

[1] We use NN to refer all kinds of differentiable models, including Logistic Regression and Factorization Machine [38] in this paper.
[2] Other decision tree based models, such as Random Forest, have the same advantages. We focus on GBDT in this paper due to its popularity.
[3] Although we can use distributed learning [34] or in-disk learning [8] to learn from more data, these solutions have overheads and thus are not efficient.
[4] Sparse categorical features refer to the categorical features with high cardinality.

- Extensive experiments show that DeepGBM is an off-the-shelf model, which can be ready to use in all kinds of prediction tasks and achieves state-of-the-art performance.

2 RELATED WORK

As aforementioned, both GBDT and NN have been widely used to learn the models for online prediction tasks. Nonetheless, either of them yields respective weaknesses when facing the tabular input space and online data generation. In the following of this section, we will briefly review the related work in addressing the respective weaknesses of either GBDT or NN, followed by previous efforts that explored to combine the advantages of GBDT and NN to build a more effective model for online prediction tasks.

2.1 Applying GBDT for Online Prediction Tasks

Applying GBDT for online prediction tasks yields two main weaknesses. First, the non-differentiable nature of trees makes it hard to update the model in the online mode. Additionally, GBDT fails to effectively leverage sparse categorical features to grow trees. There are some related works that tried to address these problems.

Online Update in Trees. Some studies have tried to train tree-based models from streaming data [4, 11, 18, 28], however, they are specifically designed for the single tree model or multiple parallel trees without dependency, like Random Forest [3], and are not easy to apply to GBDT directly. Moreover, they can hardly perform better than learning from all data at once. Two well-recognized open-sourced tools for GBDT, i.e., XGBoost [8] and LightGBM [29], also provide a simple solution for updating trees by online generated data. In particular, they keep the tree structures fixed and update the leaf outputs by the new data. However, this solution can cause performance far below satisfaction. Further efforts Son et al. [44] attempted to re-find the split points on tree nodes only by the newly generated data. But, as such a solution abandons the statistical information over historical data, the split points found by the new data is biased and thus the performance is unstable.

Categorical Features in Trees. Since the extremely sparse and high-dimensional features, representing high cardinality categories, may cause very small statistical information gain from imbalance partitions, GBDT cannot effectively use sparse features to grow trees. Some other encoding methods [41] tried to convert a categorical value into a dense numerical value such that they can be well handled by decision trees. CatBoost [12] also used the similar numerical encoding solution for categorical features. However, it will cause information loss. Categorical features also could be directly used in tree learning, by enumerating possible binary partitions [16]. However, this method often over-fits to the training data when with sparse categorical features, since there is too little data in each category and thus the statistical information is biased [29].

There are some other works, such as DeepForest [52] and mGBDT [14], that use trees as building blocks to build multi-layered trees. However, they cannot be employed to address either the challenge of online update or that of learning over the categorical feature. In a word, while there were continuous efforts in applying GBDT to online prediction tasks, most of them cannot effectively address the critical challenges in terms of how to handle online data generation and how to learn over categorical features.

2.2 Applying NN for Online Prediction Tasks

Applying NN for online prediction tasks yields one crucial challenge, i.e. NN cannot learn effectively over the dense numerical features. Although there are many recent works that have employed NN into prediction tasks, such as click prediction [22, 36, 51] and recommender systems [9, 10, 32, 47], they all mainly focused on the sparse categorical features, and far less attention has been put on adopting NN over dense numerical features, which yet remains quite challenging. Traditionally, Fully Connected Neural Network (FCNN) is often used for dense numerical features. Nevertheless, FCNN usually fails to reach satisfactory performance [15], because its fully connected model structure leads to very complex optimization hyper-planes with a high risk of falling into local optimums. Even after employing the certain normalization [27] and regularization [43] techniques, FCNN still cannot outperform GBDT in many tasks with dense numerical features [8]. Another widely used solution facing dense numerical features is discretization [13], which can bucketize numerical features into categorical formats and thus can be better handled by previous works on categorical features. However, since the bucketized outputs will still connect to fully connected layers, discretization actually cannot improve the effectiveness in handling numerical features. And discretization will increase the model complexity and may cause over-fitting due to the increase of model parameters. To summarize, applying NN to online prediction tasks still suffers from the incapability in learning an effective model over dense numerical features.

2.3 Combining NN and GBDT

Due to the respective pros and cons of NN and GBDT, there have been emerging efforts that proposed to combine their advantages. In general, these efforts can be categorized into three classes:

Tree-like NN. As pointed by Ioannou et al. [26], tree-like NNs, e.g. GoogLeNet [46], have decision ability like trees to some extent. There are some other works [30, 40] that introduce decision ability into NN. However, these works mainly focused on computer vision tasks without attention to online prediction tasks with tabular input space. Yang et al. [50] proposed the soft binning function to simulate decision trees in NN, which is, however, very inefficient as it enumerates all possible decisions. Wang et al. [48] proposed NNRF, which used tree-like NN and random feature selection to improve the learning from tabular data. Nevertheless, NNRF simply uses random feature combinations, without leveraging the statistical information over training data like GBDT.

Convert Trees to NN. Another track of works tried to convert the trained decision trees to NN [2, 5, 25, 39, 42]. However, these works are inefficient as they use a redundant and usually very sparse NN to represent a simple decision tree. When there are many trees, such conversion solution has to construct a very wide NN to represent them, which is unfortunately hard to be applied to realistic scenarios. Furthermore, these methods use the complex rules to convert a single tree and thus are not easily used in practice.

Combining NN and GBDT. There are some practical works that directly used GBDT and NN together [23, 33, 53]. Facebook [23] used the leaf index predictions as the input categorical features of a Logistic Regression. Microsoft [33] used GBDT to fit the residual errors of NN. However, as the online update problem in GBDT is not resolved, these works cannot be efficiently used online. In fact,

Facebook also pointed up this problem in their paper [23], for the GBDT model in their framework needs to be retrained every day to achieve the good online performance.

As a summary, while there are increasing efforts that explored to combine the advantages of GBDT and NN to build a more effective model for online prediction tasks, most of them cannot totally address the challenges related to *tabular input space* and *online data generation*. In this paper, we propose a new learning framework, *DeepGBM*, to better integrates NN and GBDT together.

3 DEEPGBM

In this section, we will elaborate on how the new proposed learning framework, *DeepGBM*, integrates NN and GBDT together to obtain a more effective model for generic online prediction tasks. Specifically, the whole DeepGBM framework, as shown in Fig. 1, consists of two major components: *CatNN* being an NN structure with the input of categorical features and *GBDT2NN* being another NN structure distilled from GBDT with focusing on learning over dense numerical features. We will describe the details of each component in the following subsections.

3.1 CatNN for Sparse Categorical Features

To solve online prediction tasks, NN has been widely employed to learn the prediction model over categorical features, such as Wide & Deep [9], PNN [36], DeepFM [22] and xDeepFM [32]. Since the target of CatNN is the same as these works, we can directly leverage any of existing successful NN structures to play as the CatNN, without reinventing the wheel. In particular, the same as previous works, CatNN mainly relies on the embedding technology, which can effectively convert the high dimensional sparse vectors into dense ones. Besides, in this paper, we also leverage FM component and Deep component from previous works [9, 22], to learn the interactions over features. Please note CatNN is not limited by these two components, since it can use any other NN components with similar functions.

Embedding is the low-dimensional dense representation of a high-dimensional sparse vector, and can denote as

$$E_{\boldsymbol{V}_i}(x_i) = embedding\_lookup(\boldsymbol{V}_i, x_i), \tag{1}$$

where x_i is the value of i-th feature, $\boldsymbol{V}_i$ stores all embeddings of the i-th feature and can be learned by back-propagation, and $E_{\boldsymbol{V}_i}(x_i)$ will return the corresponding embedding vector for x_i. Based on that, we can use FM component to learn linear (order-1) and pair-wise (order-2) feature interactions, and denote as

$$y_{FM}(\boldsymbol{x}) = w_0 + \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{i=1}^{d} \sum_{j=i+1}^{d} \langle E_{\boldsymbol{V}_i}(x_i), E_{\boldsymbol{V}_j}(x_j) \rangle x_i x_j \ , \tag{2}$$

where d is the number of features, w_0 and $\boldsymbol{w}$ are the parameters of linear part, and $\langle \cdot, \cdot \rangle$ is the inner product operation. Then, Deep component is used to learn the high-order feature interactions:

$$y_{Deep}(\boldsymbol{x}) = \mathcal{N}\left(\left[E_{\boldsymbol{V}_1}(x_1)^T, E_{\boldsymbol{V}_2}(x_2)^T, \ldots, E_{\boldsymbol{V}_d}(x_d)^T\right]^T ; \boldsymbol{\theta}\right) , \tag{3}$$

where $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\theta})$ is a multi-layered NN model with input $\boldsymbol{x}$ and parameter $\boldsymbol{\theta}$. Combined with two components, the final output of CatNN is

$$y_{Cat}(\boldsymbol{x}) = y_{FM}(\boldsymbol{x}) + y_{Deep}(\boldsymbol{x}) . \tag{4}$$

3.2 GBDT2NN for Dense Numerical Features

In this subsection, we will describe the details about how we distill the learned trees in GBDT into an NN model. Firstly, we will introduce how to distill a single tree into an NN. Then, we will generalize the idea to the distillation from multiple trees in GBDT.

3.2.1 ***Single Tree Distillation.*** Most of the previous distillation works only transfer model knowledge in terms of the learned function, in order to ensure the new model generates a similar output compared to the transferred one.

However, since tree and NN are naturally different, beyond traditional model distillation, there is more knowledge in the tree model could be distilled and transferred into NN. In particular, the feature selection and importance in learned trees, as well as data partition implied by learned tree structures, are indeed other types of important knowledge in trees.

Tree-Selected Features. Compared to NN, a special characteristic of the tree-based model is that it may not use all input features, as its learning will greedily choose the useful features to fit the training targets, based on the statistical information. Therefore, we can transfer such knowledge in terms of tree-selected features to improve the learning efficiency of the NN model, rather than using all input features. In particular, we can merely use the tree-selected features as the inputs of NN. Formally, we define $\mathbb{I}^t$ as the indices of the used features in a tree t. Then we can only use $\boldsymbol{x}[\mathbb{I}^t]$ as the input of NN.

Tree Structure. Essentially, the knowledge of tree structure of a decision tree indicates how to partition data into many non-overlapping regions (leaves), i.e., it clusters data into different classes and the data in the same leaf belongs to the same class. It is not easy to directly transfer such tree structure into NN, as their structures are naturally different. Fortunately, as NN has been proven powerful enough to approximate any functions [19], we can use an NN model to approximate the function of the tree structure and achieve the structure knowledge distillation. Therefore, as illustrated in Fig.2, we can use NN to fit the cluster results produced by the tree, to let NN approximate the structure function of decision tree. Formally, we denote the structure function of a tree t as $C^t(\boldsymbol{x})$, which returns the output leaf index, i.e. the cluster result produced by the tree, of sample $\boldsymbol{x}$. Then, we can use an NN model to approximate the structure function $C^t(\cdot)$ and the learning process can denote as

$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}^{'}\left(\mathcal{N}\left(\boldsymbol{x}^i[\mathbb{I}^t]; \boldsymbol{\theta}\right), \boldsymbol{L}^{t,i}\right) , \tag{5}$$

where n is the number of training samples, $\boldsymbol{x}^i$ is the i-th training sample, $\boldsymbol{L}^{t,i}$ is the one-hot representation of leaf index $C^t(\boldsymbol{x}^i)$ for $\boldsymbol{x}^i$, $\mathbb{I}^t$ is the indices of used features in tree t, $\boldsymbol{\theta}$ is the parameter of NN model $\mathcal{N}$ and can be updated by back-propagation, $\mathcal{L}^{'}$ is the loss function for the multiclass problem like cross entropy. Thus, after learning, we can get an NN model $\mathcal{N}(\cdot\,; \boldsymbol{\theta})$. Due to the strong expressiveness ability of NN, the learned NN model should perfectly approximate the structure function of decision tree.

Tree Outputs. Since the mapping from tree inputs to tree structures is learned in the previous step, to distill tree outputs, we only need to know the mapping from tree structures to tree outputs. As there is a corresponding leaf value for a leaf index, this mapping

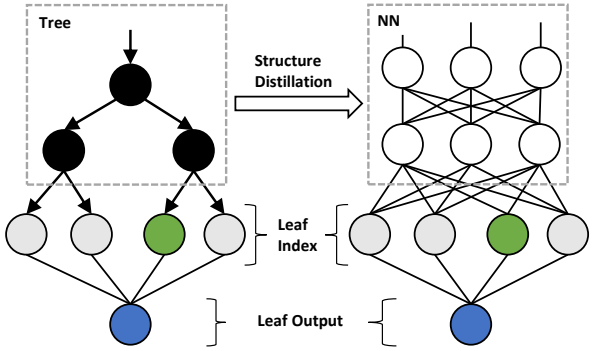

Figure 2: Tree structure distillation by leaf index. NN will approximate the tree structure by fitting its leaf index.

is actually not needed to learn. In particular, we denote the leaf values of tree t as $\boldsymbol{q}^t$ and q_i^t represents the leaf value of i-th leaf. Then we can map $\boldsymbol{L}^t$ to the tree output by $p^t = \boldsymbol{L}^t \times \boldsymbol{q}^t$.

Combined with the above methods for single tree distillation, the output of NN distilled from tree t can denote as

$$y^t(\boldsymbol{x}) = \mathcal{N}\left(\boldsymbol{x}[\mathbb{I}^t]; \boldsymbol{\theta}\right) \times \boldsymbol{q}^t. \tag{6}$$

3.2.2 ***Multiple Tree Distillation***. Since there are multiple trees in GBDT, we should generalize the distillation solution for the multiple trees. A straight-forward solution is using $\#NN = \#tree$ NN models, each of them distilled from one tree. However, this solution is very inefficient due to the high dimension of structure distillation targets, which is $O(|\boldsymbol{L}| \times \#NN)$. To improve the efficiency, we propose *Leaf Embedding Distillation* and *Tree Grouping* to reduce $|\boldsymbol{L}|$ and $\#NN$ respectively.

Leaf Embedding Distillation. As illustrated in Fig.3, we adopt embedding technology to reduce the dimension of structure distillation targets $\boldsymbol{L}$ while retraining the information in this step. More specifically, since there are bijection relations between leaf indices and leaf values, we use the leaf values to learn the embedding. Formally, the learning process of embedding can denote as

$$\min_{\boldsymbol{w}, w_0, \boldsymbol{\omega}^t} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}'' \left(\boldsymbol{w}^T \mathcal{H}(\boldsymbol{L}^{t,i}; \boldsymbol{\omega}^t) + w_0, p^{t,i} \right), \tag{7}$$

where $\boldsymbol{H}^{t,i} = \mathcal{H}(\boldsymbol{L}^{t,i}; \boldsymbol{\omega}^t)$ is an one-layered fully connected network with parameter $\boldsymbol{\omega}^t$ that converts the one-hot leaf index $\boldsymbol{L}^{t,i}$ to the dense embedding $\boldsymbol{H}^{t,i}$, $p^{t,i}$ is the predict leaf value of sample $\boldsymbol{x}^i$, $\mathcal{L}''$ is the same loss function as used in tree learning, $\boldsymbol{w}$ and w_0 are the parameters for mapping embedding to leaf values. After that, instead of sparse high dimensional one-hot representation $\boldsymbol{L}$, we can use the dense embedding as the targets to approximate the function of tree structure. This new learning process can denote as

$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(\mathcal{N}\left(\boldsymbol{x}^i[\mathbb{I}^t]; \boldsymbol{\theta}\right), \boldsymbol{H}^{t,i} \right), \tag{8}$$

where $\mathcal{L}$ is the regression loss like L2 loss for fitting dense embedding. Since the dimension of $\boldsymbol{H}^{t,i}$ should be much smaller than $\boldsymbol{L}$, Leaf Embedding Distillation will be more efficient in the multiple

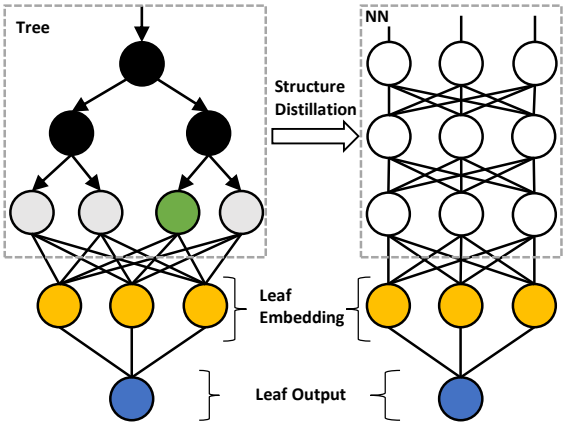

Figure 3: Tree structure distillation by leaf embedding. The leaf index is first transformed to leaf embedding. Then NN will approximate tree structure by fitting the leaf embedding. Since the dimension of leaf embedding can be significantly smaller than the leaf index, this distillation method will be much more efficient.

tree distillation. Furthermore, it will use much fewer NN parameters and thus is more efficient.

Tree Grouping. To reduce the $\#NN$, we can group the trees and use an NN model to distill from a group of trees. Subsequently, there are two problems for grouping: (1) how to group the trees and (2) how to distill from a group of trees. Firstly, for the grouping strategies, there are many solutions. For example, the equally randomly grouping, equally sequentially grouping, grouping based on importance or similarity, etc. In this paper, we use the equally randomly grouping. Formally, assuming there are m trees and we want to divide them into k groups, there are $s = \lceil m/k \rceil$ trees in each group and the trees in j-th group are $\mathbb{T}_j$, which contains random s trees from GBDT. Secondly, to distill from multiple trees, we can extend the Leaf Embedding Distillation for multiple trees. Formally, given a group of trees $\mathbb{T}$, we can extend the Eqn.(7) to learn leaf embedding from multiple trees

$$\min_{\boldsymbol{w}, w_0, \boldsymbol{\omega}^{\mathbb{T}}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}'' \left(\boldsymbol{w}^T \mathcal{H}\left(\|_{t \in \mathbb{T}} (\boldsymbol{L}^{t,i}); \boldsymbol{\omega}^{\mathbb{T}} \right) + w_0, \sum_{t \in \mathbb{T}} p^{t,i} \right), \tag{9}$$

where $\|(\cdot)$ is the concatenate operation, $\boldsymbol{G}^{\mathbb{T},i} = \mathcal{H}\left(\|_{t \in \mathbb{T}} (\boldsymbol{L}^{t,i}); \boldsymbol{\omega}^{\mathbb{T}} \right)$ is an one-layered fully connected network that convert the multi-hot vectors, which is the concatenate of multiple one-hot leaf index vectors, to a dense embedding $\boldsymbol{G}^{\mathbb{T},i}$ for the trees in $\mathbb{T}$. After that, we can use the new embedding as the distillation target of NN model, and the learning process of it can denote as

$$\mathcal{L}^{\mathbb{T}} = \min_{\boldsymbol{\theta}^{\mathbb{T}}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(\mathcal{N}\left(\boldsymbol{x}^i[\mathbb{I}^{\mathbb{T}}]; \boldsymbol{\theta}^{\mathbb{T}}\right), G^{\mathbb{T},i} \right), \tag{10}$$

where $\mathbb{I}^{\mathbb{T}}$ is the used features in tree group $\mathbb{T}$. When the number of trees in $\mathbb{T}$ is large, $\mathbb{I}^{\mathbb{T}}$ may contains many features and thus hurt the feature selection ability. Therefore, as an alternate, we can only use top features in $\mathbb{I}^{\mathbb{T}}$ according to feature importance. To sum up, combined with above methods, the final output of the NN distilled

from a tree group $\mathbb{T}$ is

$$y_{\mathbb{T}}(\boldsymbol{x}) = \boldsymbol{w}^T \times \mathcal{N}\left(\boldsymbol{x}[\mathbb{I}^{\mathbb{T}}]\,; \theta^{\mathbb{T}}\right) + w_0\ . \tag{11}$$

And the output of a GBDT model, which contains k tree groups, is

$$y_{GBDT2NN}(\boldsymbol{x}) = \sum_{j=1}^{k} y_{\mathbb{T}_j}(\boldsymbol{x})\ . \tag{12}$$

In summary, owing to Leaf Embedding Distillation and Tree Grouping, GBDT2NN can efficiently distill many trees of GBDT into a compact NN model. Furthermore, besides tree outputs, the feature selection and structure knowledge in trees are effectively distilled into the NN model as well.

3.3 Training for DeepGBM

We will describe how to train the DeepGBM in this subsection, including how to train it end-to-end offline and how to efficiently update it online.

3.3.1 ***End-to-End Offline Training.*** To train DeepGBM, we first need to use offline data to train a GBDT model and then use Eqn.(9) to get the leaf embedding for the trees in GBDT. After that, we can train DeepGBM end-to-end. Formally, we denote the output of DeepGBM as

$$\hat{y}(\boldsymbol{x}) = \sigma^{'}\left(w_1 \times y_{GBDT2NN}(\boldsymbol{x}) + w_2 \times y_{Cat}(\boldsymbol{x})\right)\ , \tag{13}$$

where w_1 and w_2 are the trainable parameters used for combining GBDT2NN and CatNN, $\sigma^{'}$ is the output transformation, such as *sigmoid* for binary classification. Then, we can use the following loss function for the end-to-end training

$$\mathcal{L}_{offline} = \alpha \mathcal{L}^{''}(\hat{y}(\boldsymbol{x}),\ y) + \beta \sum_{j=1}^{k} \mathcal{L}^{\mathbb{T}_j}\ , \tag{14}$$

where y is the training target of sample $\boldsymbol{x}$, $\mathcal{L}^{''}$ is the loss function for corresponding tasks such as cross-entropy for classification tasks, $\mathcal{L}^{\mathbb{T}}$ is the embedding loss for tree group $\mathbb{T}$ and defined in Eqn.(10), k is the number of tree groups, α and β are hyper-parameters given in advance and used for controlling the strength of end-to-end loss and embedding loss, respectively.

3.3.2 ***Online Update.*** As the GBDT model is trained offline, using it for embedding learning in the online update will hurt the online real-time performance. Thus, we do not include the $\mathcal{L}^{\mathbb{T}}$ in the online update, and the loss for the online update can denote as

$$\mathcal{L}_{online} = \mathcal{L}^{''}(\hat{y}(\boldsymbol{x}),\ y)\ , \tag{15}$$

which only uses the end-to-end loss. Thus, when using DeepGBM online, we only need the new data to update the model by $\mathcal{L}_{online}$, without involving GBDT and retraining from scratch. In short, DeepGBM will be very efficient for online tasks. Furthermore, it is also very effective since it can well handle both the dense numerical features and sparse categorical features.

Table 2: Details of the datasets used in experiments. All these datasets are publicly available. #Sample is the number of data samples, #Num is the number of numerical features, and #Cat is the number of categorical features.

Name	#Sample	#Num	#Cat	Task
Flight	7.79M	5	7	Classification
Criteo	45.8M	13	26	Classification
Malware	8.92M	12	69	Classification
AutoML-1	4.69M	51	23	Classification
AutoML-2	0.82M	17	7	Classification
AutoML-3	0.78M	17	54	Classification
Zillow	90.3K	31	27	Regression

4 EXPERIMENT

In this section, we will conduct thorough evaluations on DeepGBM[5] over a couple of public tabular datasets and compares its performance with several widely used baseline models. Particularly, we will start with details about experimental setup, including data description, compared models and some specific experiments settings. After that, we will analyze the performance of DeepGBM in both offline and online settings to demonstrate its effectiveness and advantage over baseline models.

4.1 Experimental Setup

Datasets: To illustrate the effective of DeepGBM, we conduct experiments on a couple of public datasets, as listed in Table 2. In particular, Flight[6] is an airline dataset and used to predict the flights are delayed or not. Criteo[7], Malware[8] and Zillow[9] are the datasets from Kaggle competitions. AutoML-1, AutoML-2 and AutoML-3 are datasets from "AutoML for Lifelong Machine Learning" Challenge in NeurIPS 2018[10]. More details about these datasets can be found in Appendix A.1. As these datasets are from real-world tasks, they contain both categorical and numerical features. Furthermore, as time-stamp is available in most of these datasets, we can use them to simulate the online scenarios.

Compared Models: In our experiments, we will compare DeepGBM with the following baseline models:

- *GBDT* [17], which is a widely used tree-based learning algorithm for modeling tabular data. We use LightGBM [29] for its high efficiency.
- *LR*, which is Logistic Regression, a generalized linear model.
- *FM* [38], which contains a linear model and a FM component.
- *Wide&Deep* [9], which combines a shallow linear model with deep neural network.
- *DeepFM* [22], which improves Wide&Deep by adding an additional FM component.
- *PNN* [36], which uses pair-wise product layer to capture the pair-wise interactions over categorical features.

[5]We released the source code at: https://github.com/motefly/DeepGBM
[6]http://stat-computing.org/dataexpo/2009/
[7]https://www.kaggle.com/c/criteo-display-ad-challenge/data
[8]https://www.kaggle.com/c/malware-classification
[9]https://www.kaggle.com/c/zillow-prize-1
[10]https://www.4paradigm.com/competition/nips2018

Table 3: Offline performance comparison. AUC (higher is better) is used for binary classification tasks, and MSE (lower is better) is used for regression tasks. All experiments are run 5 times with different random seeds, and the *mean* ± *std* results are shown in this table. The top-2 results are marked bold.

Model	Binary Classification						Regression
	Flight	Criteo	Malware	AutoML-1	AutoML-2	AutoML-3	Zillow
LR	0.7234 ±5e-4	0.7839 ±7e-5	0.7048 ±1e-4	0.7278 ±2e-3	0.6524 ±2e-3	0.7366 ±2e-3	0.02268 ±1e-4
FM	0.7381 ±3e-4	0.7875 ±1e-4	0.7147 ±3e-4	0.7310 ±1e-3	0.6546 ±2e-3	0.7425 ±1e-3	0.02315±2e-4
Wide&Deep	0.7353 ±3e-3	0.7962 ±3e-4	0.7339 ±7e-4	0.7409 ±1e-3	0.6615 ±1e-3	0.7503 ±2e-3	0.02304 ±3e-4
DeepFM	0.7469 ±2e-3	0.7932 ±1e-4	0.7307 ±4e-4	0.7400 ±1e-3	0.6577 ±2e-3	0.7482 ±2e-3	0.02346 ±2e-4
PNN	0.7356 ±2e-3	0.7946 ±8e-4	0.7232 ±6e-4	0.7350 ±1e-3	0.6604 ±2e-3	0.7418 ±1e-3	0.02207 ±2e-5
GBDT	0.7605 ±1e-3	0.7982 ±5e-5	0.7374 ±2e-4	0.7525 ±2e-4	0.6844 ±1e-3	0.7644 ±9e-4	0.02193 ±2e-5
DeepGBM (D1)	0.7668 ±5e-4	**0.8038 ±3e-4**	0.7390 ±9e-5	0.7538 ±2e-4	0.6865 ±4e-4	**0.7663 ±3e-4**	0.02204 ±5e-5
DeepGBM (D2)	**0.7816 ±5e-4**	0.8006 ±3e-4	**0.7426 ±5e-5**	**0.7557 ±2e-4**	**0.6873 ±3e-4**	0.7655 ±2e-4	**0.02190 ±2e-5**
DeepGBM	**0.7943 ±2e-3**	**0.8039 ±3e-4**	**0.7434 ±2e-4**	**0.7564 ±1e-4**	**0.6877 ±8e-4**	**0.7664 ±5e-4**	**0.02183 ±3e-5**

Besides, to further analyze the performance of DeepGBM, we use additional two degenerated versions of DeepGBM in experiments:

- *DeepGBM (D1)*, which uses GBDT directly in DeepGBM, rather than GBDT2NN. As GBDT cannot be online updated, we can use this model to check the improvement brought by DeepGBM in online scenarios.
- *DeepGBM (D2)*, which only uses GBDT2NN in DeepGBM, without CatNN. This model is to examine the standalone performance of GBDT2NN.

Experiments Settings: To improve the baseline performance, we introduce some basic feature engineering in the experiments. Specifically, for the models which cannot handle numerical features well, such as LR, FM, Wide&Deep, DeepFM and PNN, we discrete the numerical features into categorical ones. Meanwhile, for the models which cannot handle categorical feature well, such as GBDT and the models based on it, we convert the categorical features into numerical ones, by label-encoding [12] and binary-encoding [41]. Based on this basic feature engineering, all models can use the information from both categorical and numerical features, such that the comparisons are more reliable. Moreover, all experiments are run five times with different random seeds to ensure a fair comparison. For the purpose of reproducibility, all the details of experiments settings including hyper-parameter settings will be described in Appendix A and the released codes.

4.2 Offline Performance

We first evaluate the offline performance for the proposed DeepGBM in this subsection. To simulate the real-world scenarios, we partition each benchmark dataset into the training set and test set according to the time-stamp, i.e., the older data samples (about 90%) are used for the training and the newer samples (about 10%) are used for the test. More details are available in Appendix A.

The overall comparison results could be found in Table 3. From the table, we have following observations:

- GBDT can outperform other NN baselines, which explicitly shows the advantage of GBDT on the tabular data. Therefore, distilling GBDT knowledge will definitely benefit DeepGBM.
- GBDT2NN (DeepGBM (D2)) can further improve GBDT, which indicates that GBDT2NN can effectively distill the trained GBDT model into NN. Furthermore, it implies that the distilled NN model can be further improved and even outperform GBDT.
- Combining GBDT and NN can further improve the performance. The hybrid models, including DeepGBM (D1) and DeepGBM, can all reach better performance than single model baselines, which indicates that using two components to handle categorical features and numerical features respectively can benefit performance for online prediction tasks.
- DeepGBM outperforms all baselines on all datasets. In particular, DeepGBM can boost the accuracy over the best baseline GBDT by 0.3% to 4.4%. as well as the best of NN baselines by 1% to 6.3%.

To investigate the convergence of DeepGBM, Fig. 4 demonstrates the performance in terms of AUC on the test data by the model trained with increasing epochs. From these figures, we can find that DeepGBM also converges much faster than other models.

4.3 Online Performance

To evaluate the online performance of DeepGBM, we use Flight, Criteo and AutoML-1 datasets as the online benchmark. To simulate the online scenarios, we refer to the setting of the "AutoML for Lifelong Machine Learning" Challenge in NeurIPS 2018 [37]. Specifically, we partition each dataset into multiple consecutive batches along with the time. We will train the model for each batch from the oldest to latest in sequence. And, at i-th batch, it only allows to use the samples in that batch to train or update the model; after that, the $(i+1)$-th batch is used for the evaluation. More details are available in Appendix A.

Note that, as the data distribution may change along with different batches during the online simulation, we would like to examine if the online learned models can perform better than their offline versions, i.e., the models without the online update. Thus, we also check the performance of offline DeepGBM as another baseline to compare with the online learned DeepGBM.

All the comparison results are summarized in Fig 5, and we have following observations:

- GBDT cannot perform well in the online scenarios as expected. Although GBDT yields good result in the first batch (offline stage), it declines obviously in the later (online) batches.
- The online performance of GBDT2NN is good. In particular, GBDT2NN (DeepGBM (D2)) can significantly outperform GBDT. Furthermore, DeepGBM outperforms DeepGBM (D1), which uses GBDT instead of GBDT2NN, by a non-trivial gain. It indicates that the distilled NN model by GBDT could be further improved and effectively used in the online scenarios.

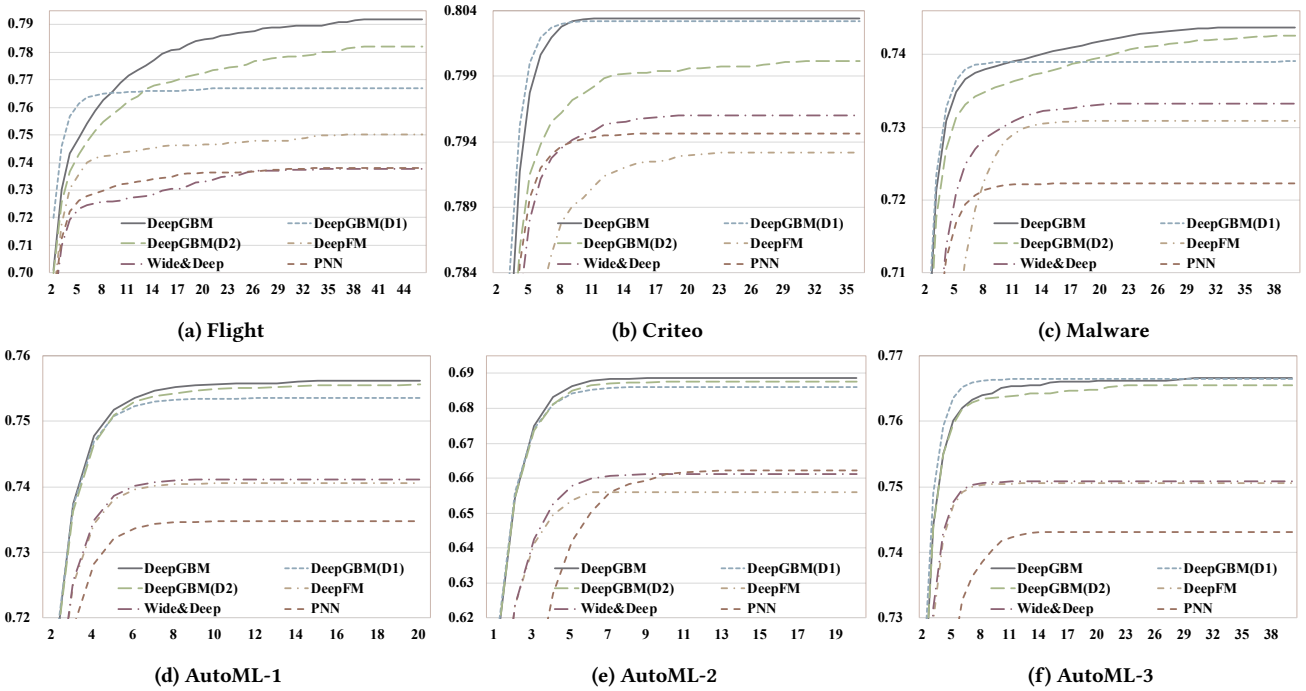

Figure 4: Epoch-AUC curves over test data, in the offline classification experiments. We can find that DeepGBM converges much faster than other baselines. Moreover, the convergence points of DeepGBM are also much better.

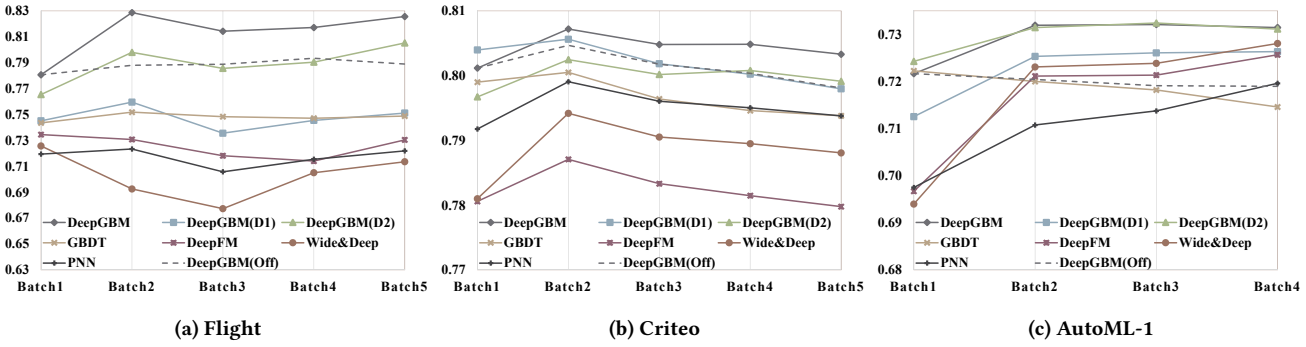

Figure 5: Online performance comparison. For the models that cannot be online updated, we did not update them during the online simulation. All experiments are run 5 times with different random seeds, and the mean results (AUC) are used.

- DeepGBM outperforms all other baselines, including its offline version (the dotted lines). It explicitly proves the proposed DeepGBM indeed yields strong learning capacity over both categorical and numerical tabular features while retaining the vital ability of efficient online learning.

In short, all above experimental results demonstrate that DeepGBM can significantly outperform all kinds of baselines in both offline and online scenarios.

5 CONCLUSION

To address the challenges of *tabular input space*, which indicates the existence of both sparse categorical features and dense numerical ones, and *online data generation*, which implies continuous task-generated data with potentially dynamic distribution, in online prediction tasks, we propose a new learning framework, *DeepGBM*, which integrates NN and GBDT together. Specifically, DeepGBM consists of two major components: *CatNN* being an NN structure with the input of sparse categorical features and *GBDT2NN* being another NN structure with the input of dense numerical features. To further take advantage of GBDT's strength in learning over dense numerical features, GBDT2NN attempts to distill the knowledge learned by GBDT into an NN modeling process. Powered by these two NN based components, DeepGBM can indeed yield the strong learning capacity over both categorical and numerical tabular features while retaining the vital ability of efficient online learning. Comprehensive experimental results demonstrate that DeepGBM can outperform other solutions in various prediction tasks, in both offline and online scenarios.

ACKNOWLEDGEMENT

We thank Hui Xue (Microsoft) for the discussion about the idea and the comments on an earlier version of the manuscript.

REFERENCES

[1] Eugene Agichtein, Eric Brill, and Susan Dumais. 2006. Improving web search ranking by incorporating user behavior information. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 19–26.

[2] Arunava Banerjee. 1997. Initializing neural networks using decision trees. *Computational learning theory and natural learning systems* 4 (1997), 3–15.

[3] Iñigo Barandiaran. 1998. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence* 20, 8 (1998).

[4] Yael Ben-Haim and Elad Tom-Tov. 2010. A streaming parallel decision tree algorithm. *Journal of Machine Learning Research* 11, Feb (2010), 849–872.

[5] Gérard Biau, Erwan Scornet, and Johannes Welbl. 2016. Neural random forests. *Sankhya A* (2016), 1–40.

[6] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[7] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*. ACM, 129–136.

[8] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 785–794.

[9] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 7–10.

[10] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 191–198.

[11] Pedro Domingos and Geoff Hulten. 2000. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 71–80.

[12] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. 2018. CatBoost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363* (2018).

[13] James Dougherty, Ron Kohavi, and Mehran Sahami. 1995. Supervised and unsupervised discretization of continuous features. In *Machine Learning Proceedings 1995*. Elsevier, 194–202.

[14] Ji Feng, Yang Yu, and Zhi-Hua Zhou. 2018. Multi-Layered Gradient Boosting Decision Trees. *arXiv preprint arXiv:1806.00007* (2018).

[15] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. 2014. Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research* 15, 1 (2014), 3133–3181.

[16] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The elements of statistical learning*. Vol. 1. Springer series in statistics New York, NY, USA:.

[17] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[18] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. 2005. Mining data streams: a review. *ACM Sigmod Record* 34, 2 (2005), 18–26.

[19] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[20] Krzysztof Grabczewski and Norbert Jankowski. 2005. Feature selection with decision tree criterion. In *null*. IEEE, 212–217.

[21] Thore Graepel, Joaquin Quinonero Candela, Thomas Borchert, and Ralf Herbrich. 2010. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. Omnipress.

[22] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247* (2017).

[23] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*. ACM, 1–9.

[24] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[25] K. D. Humbird, J. L. Peterson, and R. G. McClarren. 2017. Deep neural network initialization with decision trees. *ArXiv e-prints* (July 2017). arXiv:1707.00784

[26] Yani Ioannou, Duncan Robertson, Darko Zikic, Peter Kontschieder, Jamie Shotton, Matthew Brown, and Antonio Criminisi. 2016. Decision forests, convolutional networks and the models in-between. *arXiv preprint arXiv:1603.01250* (2016).

[27] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[28] Ruoming Jin and Gagan Agrawal. 2003. Efficient decision tree construction on streaming data. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 571–576.

[29] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*. 3146–3154.

[30] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. 2015. Deep neural decision forests. In *Proceedings of the IEEE international conference on computer vision*. 1467–1475.

[31] Yaguang Li, Kun Fu, Zheng Wang, Cyrus Shahabi, Jieping Ye, and Yan Liu. 2018. Multi-task representation learning for travel time estimation. In *International Conference on Knowledge Discovery and Data Mining,(KDD)*.

[32] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. *arXiv preprint arXiv:1803.05170* (2018).

[33] Xiaoliang Ling, Weiwei Deng, Chen Gu, Hucheng Zhou, Cui Li, and Feng Sun. 2017. Model ensemble for click prediction in bing search ads. In *Proceedings of the 26th International Conference on World Wide Web Companion*. International World Wide Web Conferences Steering Committee, 689–698.

[34] Qi Meng, Guolin Ke, Taifeng Wang, Wei Chen, Qiwei Ye, Zhi-Ming Ma, and Tie-Yan Liu. 2016. A communication-efficient parallel algorithm for decision tree. In *Advances in Neural Information Processing Systems*. 1279–1287.

[35] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[36] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 1149–1154.

[37] Yao Quanming, Wang Mengshuo, Jair Escalante Hugo, Guyon Isabelle, Hu Yi-Qi, Li Yu-Feng, Tu Wei-Wei, Yang Qiang, and Yu Yang. 2018. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306* (2018).

[38] Steffen Rendle. 2010. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 995–1000.

[39] David L Richmond, Dagmar Kainmueller, Michael Y Yang, Eugene W Myers, and Carsten Rother. 2015. Relating cascaded random forests to deep convolutional neural networks for semantic segmentation. *arXiv preprint arXiv:1507.07583* (2015).

[40] Samuel Rota Bulo and Peter Kontschieder. 2014. Neural decision forests for semantic image labelling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 81–88.

[41] Scikit-learn. 2018. categorical_encoding. https://github.com/scikit-learn-contrib/categorical-encoding.

[42] Ishwar Krishnan Sethi. 1990. Entropy nets: from decision trees to neural networks. *Proc. IEEE* 78, 10 (1990), 1605–1613.

[43] Ira Shavitt and Eran Segal. 2018. Regularization Learning Networks: Deep Learning for Tabular Datasets. In *Advances in Neural Information Processing Systems*. 1386–1396.

[44] Jeany Son, Ilchae Jung, Kayoung Park, and Bohyung Han. 2015. Tracking-by-segmentation with online gradient boosting decision tree. In *Proceedings of the IEEE International Conference on Computer Vision*. 3056–3064.

[45] V Sugumaran, V Muralidharan, and KI Ramachandran. 2007. Feature selection using decision tree and classification through proximal support vector machine for fault diagnostics of roller bearing. *Mechanical systems and signal processing* 21, 2 (2007), 930–942.

[46] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[47] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1235–1244.

[48] Suhang Wang, Charu Aggarwal, and Huan Liu. 2017. Using a random forest to inspire a neural network and improving on it. In *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 1–9.

[49] Zheng Wang, Kun Fu, and Jieping Ye. 2018. Learning to estimate the travel time. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 858–866.

[50] Yongxin Yang, Irene Garcia Morillo, and Timothy M Hospedales. 2018. Deep Neural Decision Trees. *arXiv preprint arXiv:1806.06988* (2018).

[51] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep learning over multi-field categorical data. In *European conference on information retrieval*. Springer, 45–57.

[52] Zhi-Hua Zhou and Ji Feng. 2017. Deep forest: Towards an alternative to deep neural networks. *arXiv preprint arXiv:1702.08835* (2017).

[53] Jie Zhu, Ying Shan, JC Mao, Dong Yu, Holakou Rahmanian, and Yi Zhang. 2017. Deep embedding forest: Forest-based serving with deep embedding features. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1703–1711.

Appendix A REPRODUCIBILITY DETAILS

For the reproducibility, we released the source code at: https://github.com/motefly/DeepGBM. Furthermore, we use this supplementary material to provide some important details about datasets and model settings.

A.1 Dataset Details

Following are the details of the used datasets in the experiments:

- Flight, which is used as a binary classification dataset. In particular, the classification target is whether a flight is delayed (more than 15 minutes) or not.
- Criteo, which is a click prediction dataset and widely used in the experiments of many previous works.
- Malware, which is a binary classification dataset from Kaggle competitions.
- AutoML, which are the binary classification datasets from "AutoML for Lifelong Machine Learning" Challenge in NeurIPS 2018. There are total 5 datasets, and we use "A", "B", "D" datasets from them in this paper. Although there are 10 batches in each dataset, the last 5 batches are not publicly available. Thus, we only can use the first 5 batches in the experiments.
- Zillow, which is a regression dataset from Kaggle competitions.

There are some ordinal features, such as "day of week", in these datasets. We treat these ordinal features as both categorical features and numerical features in the experiments.

The details of data partitions in the offline experiments are listed in Table 4.

Table 4: Data partition for offline experiments.

Name	Training	Test
Flight	all samples in 2007	random 0.5M samples in 2008
Criteo	first 90%	last 10%
Malware	first 90%	last 10%
AutoML-1	first 90%	last 10%
AutoML-2	first 90%	last 10%
AutoML-3	first 90%	last 10%
Zillow	first 90%	last 10%

The details of batch partitions in the online experiments are listed in Table 5.

Table 5: Data partition for Online experiments.

Dataset	Flight	Criteo	AutoML-1
#Batch	6	6	5
Batch 1	Year 2007	first 50%	Original 5 batches from data itself, for the data is provided by batch fashion.
Batch 2	Jan 2008	50% - 60%	
Batch 3	Feb 2008	60% - 70%	
Batch 4	Mar 2008	70% - 80%	
Batch 5	Apr 2008	80% - 90%	
Batch 6	May 2008	last 10%	

In the online simulation, the first batch is used for the offline pre-train. Then at the later batches, we can only use the data in that batch to update the model. And the data at $(i+1)$-th batch is used to evaluate the model from i-th batch. For the models that cannot be online updated, such as GBDT, we will not update them during the simulation, and the model from the first batch will be used in the evaluation for all batches.

A.2 Model Details

For GBDT based model, our implementation is based on LightGBM. For the NN based model, our implementation is based on pytorch. All the implementation codes are available at https://github.com/motefly/DeepGBM.

As the distributions of the used datasets in experiments are different with each other, we use the different hyper-parameters for different datasets. We first list the common hyper-parameters for GBDT and NN based models in Table 6. And these hyper-parameters are shared in all models.

The model-specific hyper-parameters are shown in Table 7.

- *Deep Part Structure.* The deep part structures of DeepFM and Wide&Deep are the same, which are shown in the table. We refer to their open-sourced versions[11, 12, 13] for these structure settings. We also tried the wider or deeper hidden layers for the deep part, but it caused over-fitting and poor test results.
- *PNN.* Consulting the settings and results in PNN paper [36], we use three hidden layers, one layer more than DeepFM and Wide&Deep, in PNN. And this indeed is better than two two hidden layers.
- *GBDT2NN.* The number of tree groups for different datasets are listed in the table. The dimension of leaf embedding for a tree group is set to 20 on all datasets. The structure of the distilled NN model is a fully connected networks with "100-100-100-50" hidden layers. Besides, we adopt the feature selection in each tree group. More specifically, we first sort the features according to the information gain, and the top k of them are selected as the inputs of distilled NN model. The number k is shown in the table.
- *DeepGBM.* The trainable weights w_1 and w_2 are initialized to 1.0 and 0.0, respectively. The hyper-parameters of CatNN are the same as DeepFM. For the offline training, we adopt a exponential decay strategy for β in Eqn.(14), to let the loss focuses more on embedding fitting at the beginning. More specifically, β (initialed to 1.0) is decayed exponentially by a factor at a certain frequency (along with epochs), and α is set to $(1-\beta)$ in our experiments. The decay factors and frequencies are listed in the table.

[11] https://github.com/ChenglongChen/tensorflow-DeepFM
[12] https://github.com/nzc/dnn_ctr
[13] https://github.com/shenweichen/DeepCTR

Table 6: Shared hyper-parameters for GBDT and NN based models.

Models	Parameters	Flight	Criteo	Malware	AutoML-1	AutoML-2	AutoML-3	Zillow
GBDT based models	Number of trees	200	200	200	100	100	100	100
	Learning rate	0.15	0.15	0.15	0.1	0.1	0.1	0.15
	Max number of leaves	128	128	128	64	64	64	64
NN based models	Training batch size	512	4096	1024	512	128	128	128
	Learning rate	1e-3						
	Optimizer	AdamW						
	Offline epoch	45	35	40	20	20	40	40
	Online update epoch	1						

Table 7: More hyper-parameters. For the NN structures listed in this table, we only report the hidden layers of them.

Models	Parameters	Flight	Criteo	Malware	AutoML-1	AutoML-2	AutoML-3	Zillow
GBDT2NN	#Tree Groups	20	20	20	5	10	10	10
	#Top Features	128	128	128	128	64	64	64
DeepFM, Wide&Deep	Deep part structure	32-32	32-32	64-64	16-16	16-16	16-16	32-32
PNN	Structure	32-32-32	32-32-32	64-64-64	16-16-16	16-16-16	16-16-16	32-32-32
DeepGBM	β decay frequency	2	3	2	2	2	2	10
	β decay factor	0.7	0.9	0.9	0.7	0.7	0.7	0.7