Model Ensemble for Click Prediction in Bing Search Ads

Xiaoliang Ling
Microsoft Bing
No. 5 Dan Ling Street
Beijing, China
xiaoling@microsoft.com

Weiwei Deng
Microsoft Bing
No. 5 Dan Ling Street
Beijing, China
dedeng@microsoft.com

Chen Gu
Microsoft Bing
No. 5 Dan Ling Street
Beijing, China
chengu@microsoft.com

Hucheng Zhou
Microsoft Research
No. 5 Dan Ling Street
Beijing, China
huzho@microsoft.com

Cui Li*
Microsoft Research
No. 5 Dan Ling Street
Beijing, China
v-cuili@microsoft.com

Feng Sun
Microsoft Bing
No. 5 Dan Ling Street
Beijing, China
fengsun@microsoft.com

ABSTRACT

Accurate estimation of the click-through rate (CTR) in sponsored ads significantly impacts the user search experience and businesses' revenue, even 0.1% of accuracy improvement would yield greater earnings in the hundreds of millions of dollars. CTR prediction is generally formulated as a supervised classification problem. In this paper, we share our experience and learning on model ensemble design and our innovation. Specifically, we present 8 ensemble methods and evaluate them on our production data. Boosting neural networks with gradient boosting decision trees turns out to be the best. With larger training data, there is a nearly 0.9% AUC improvement in offline testing and significant click yield gains in online traffic. In addition, we share our experience and learning on improving the quality of training.

Keywords

click prediction; DNN; GBDT; model ensemble

1. INTRODUCTION

Search engine advertising has become a significant element of the web browsing experience. Choosing the right ads for a query and the order in which they are displayed greatly affects the probability that a user will see and click on each ad. Accurately estimating the click-through rate (CTR) of ads [10, 16, 12] has a vital impact on the revenue of search businesses; even a 0.1% accuracy improvement in our production would yield hundreds of millions of dollars in additional earnings. An ad's CTR is usually modeled as a classification problem, and thus can be estimated by machine learning models. The training data is collected from historical ads impressions and the corresponding clicks. Because of the simplicity, scalability and online learning capability, logistic regression (LR) is the most widely used model that has been studied by Google [21], Facebook [14] and Yahoo! [3]. Recently, factorization machines (FMs) [24, 5, 18, 17], gradient boosting decision trees (GBDTs) [25] and deep neural networks (DNNs) [29] have also been evaluated and gradually adopted in industry.

A single model would lead to suboptimal accuracy, and the above-mentioned models all have various different advantages and disadvantages. They are usually ensembled together in an industry setting (or even machine learning competition like Kaggle [15]) to achieve better prediction accuracy. For instance, apps recommendation in Google adopts Wide&Deep [7] that co-trains LR (wide) and DNN (deep) together; ad CTR in Facebook [14] uses GBDT for non-linear feature transformation and feeds them to LR for the final prediction; Yandex [25] boosts LR with GBDT for CTR prediction; and there also exists work [29] on ads CTR that feeds the FM embedding learned from sparse features to DNN. Simply replicating them does not yield the best possible level of accuracy. In this paper, we share our experience and learning on designing and optimizing model ensembles to improve the CTR prediction in Microsoft Bing Ads.

The challenge lies in the large design space: which models are ensembled together; which ensemble techniques are used; and which ensemble design would achieve the best accuracy? In this paper, we present 8 ensemble variants and evaluate them in our system. The ensemble that boosts the NN with GBDT, i.e., initializes the sample target for GBDT with the prediction score of NN, is considered to be the best in our setting. With larger training data, it shows near 0.9% AUC improvement in offline testing and significant click yield gains in online traffic. To push this new ensemble design into the system also brings system challenges on a fast and accurate trainer, considering that multiple models are trained and each trainer must have good scalability and accuracy. We share our experience with identifying accuracy-critical factors in training.

The rest of the paper is organized as follows. We first provide a brief primer on the ad system in Microsoft Bing Ads in Section 2. We then present several model ensemble design in detail in Section 3, followed by the corresponding evaluation against production data. The means of improving model accuracy and system performance is described in Section 5. Related work is listed in Section 6 and we conclude in Section 7.

2. ADS CTR OVERVIEW

In this section, we will describe the overview of the ad system in Microsoft Bing Ads and the basic models and features we use.

*This work was done during her internship in Microsoft Research.

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4914-7/17/04.
http://dx.doi.org/10.1145/3041021.3054192

2.1 Ads System Overview

Sponsored search typically uses keyword based auction. Advertisers bid on a list of keywords for their ad campaigns. When a user searches with a query, the search engine matches the user query with bidding keywords, and then selects and shows proper ads to the user. When a user clicks any of the ads, the advertiser will be charged with a fee based on the generalized second price [2, 1]. A typical system involves several steps including selection, relevance filtration, CTR prediction, ranking and allocation. The input query from the user is first used to retrieve a list of candidate ads (`selection`). Specifically, the selection system parses the query, expands it to relevant ad keywords and then retrieves the ads from advertisers' campaigns according to their bidding keywords. For each selected ad candidate, a relevance model estimates the relevance score between query and ad, and further filters out the least relevant ones (`relevance filtration`). The remaining ads are estimated by the click model to predict the click probability (*pClick*) given the query and context information (`click prediction`). In addition, a ranking score is calculated for each ad candidate by $bid * pClick$ where *bid* is the corresponding bidding price. These candidates are then sorted by their ranking score (`ranking`). Finally, the top ads with a ranking score larger than the given threshold are allocated for impression (`allocation`), such that the number of impressions is limited by total available slots. The click probability is thus a key factor used to rank the ads in appropriate order, place the ads in different locations on the page, and even to determine the price that will be charged to the advertiser if a click occurs. Therefore, ad click prediction is a core component of the sponsored search system.

2.2 Models

Consider a training data set $D = \{(x_i, y_i)\}$ with n examples (i.e., $|D| = n$), where each sample has m features $x_i \in R^m$ with observed label $y_i \in \{0,1\}$. We formulate click prediction as a supervised learning problem, and binary classification models are often used for click probability estimation $p(click = 1|user, query, ad)$. Given the observed label $y \in \{0,1\}$, the prediction p gets the resulting LogLoss (logistic loss), given as:

$$\ell(p) = -y \cdot \log p - (1-y) \cdot \log(1-p), \quad (1)$$

which means the negative log-likelihood of y given p. In the following, we will give a brief description on two basic models used in our production.

Logistic Regression. LR predicts the click probability as $p = \sigma(w \cdot x + b)$, where w is the feature weight, b is the bias, and $\sigma(a) = \frac{1}{1+exp(-a)}$ is the sigmoid function. It is straightforward to get the gradient as $\nabla\ell(w) = (\sigma(w \cdot x) - y) \cdot x = (p - y) \cdot x$ that is used in an optimization process like SGD. The left part in Figure 1 depicts the LR model structure. LR is a generalized linear model that memorizes the frequent co-occurrence between feature and label, with the advantages of simplicity, interpretability and scalability. LR essentially works by `memorization` that can be achieved effectively using cross-product transformations over sparse features. For instance, the term co-occurrence between the query and ad can be cross combined to capture their correlation, e.g., the binary feature "AND(car, vehicle)" has value 1 if "car" occurs in the query and "vehicle" occurs in the ad title. This explains how the co-occurrence of a crossed feature correlates with the target label. However, since the LR model itself can only model the linear relation among features, the non-linear relation has to be combined manually. Even worse, memorization does not generalize to query-ad pairs that have never occurred in the past.

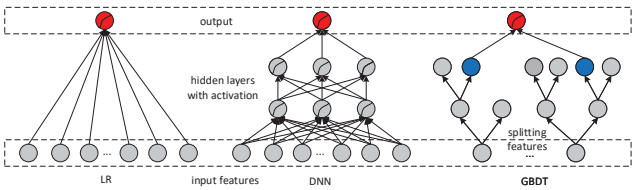


Figure 1: Graphical illustration of basic models: LR, DNN and GBDT.

Deep Neural Network. DNN generalizes to previously unseen query-ad feature pairs by learning a low-dimensional dense embedding vector for both query and ad features, with less burden of feature engineering. The middle model in Figure 1 depicts four layers of the DNN structure, including two hidden layers each with u neuron units, one input layer with m features and one output layer with one single output. With a top-down description, the output unit is a real number $p \in (0,1)$ as the predicted CTR with $p = \sigma(w_2 \cdot x_2 + b_2)$, where $\sigma(a) = \frac{1}{1+exp(-a)}$ is the logistic activation function. $w_2 \in R^{1\times u}$ is the parameter matrix between output layer and the connected hidden layer, $b_2 \in R$ is the bias. $x_2 \in R^u$ is the activation output of the last hidden layer computed as $x_2 = \sigma(w_1 \cdot x_1 + b_1)$, where $w_1 \in R^{u\times u}, b_1 \in R^u, x_1 \in R^u$. Similarly, $x_1 = \sigma(w_0 \cdot x_0 + b_0)$ where $w_0 \in R^{u\times m}, b_0 \in R^u$ and $x_0 \in R^m$ is the input sample. Different hidden layers can be regarded as different internal functions capturing different forms of representations of a data instance. Compared with the linear model, DNN thus has better for catching intrinsic data patterns and leads to better generalization. The sigmoid activation can be replaced as a tanh or ReLU [19] function.

2.3 Training data

The training data is collected from an ad impressions log, that each sample (x_i, y_i) represents whether or not the impressed ad allocated by the ad system has been clicked by a user. The output variable y_i is 1 if the ad has been clicked, and y_i is 0 otherwise. The input features x_i consist of different sources that describe different `domains` of an impression. 1). `query` features that include query term, query classification, query length, etc. 2). `ad` features that include ad ID, advertiser ID, campaign ID, and the corresponding terms in ad keyword, title, body, URL domain, etc. 3). `user` features that include user ID, demographics, and user click propensity [6], etc. 4). `context` features that describe date, and location. and 5). `crossing` features among them, e.g., *QueryId_X_AdId* (*X* means crossing) that cross the user ID with the ad ID in an example.

One-Hot Encoding Features. These features can be simply represented as `one-hot encoding`, e.g., *QueryId_X_AdId* is 1 if the user-ad pair occurs in the example. Consider that there would be hundreds of million of users and ads, as well as millions of terms, and even more crossing features. The feature space has extremely high dimensionality, and they are meanwhile extremely sparse in a sample. This high dimensionality and sparsity introduces constraints on the model design and also introduces challenges on the corresponding model training and serving.

Statistic Features. They can be classified into three types: 1). `Counting features` that include statistics like the number of clicks, the number of impressions, and the historical CTR over different domains (basic and crossing). e.g., *QueryId_X_adId_Click_6M*, and *QueryId_X_AdId_Impression_6M* that counts the number of clicks for specific `(QueryId, AdId)` pair in last six months. To account for this display position bias [9], we use position-normalized statistics such as expected clicks `(ECs)` and clicks over expected

clicks (COEC) [6]:

$$COEC = \frac{\sum_{r=1}^{R} c_r}{\sum_{r=1}^{R} i_r \cdot EC_r} \quad (2)$$

where the numerator is the total number of clicks received by a query-ad pair; the denominator can be interpreted as the expected clicks (ECs) that an average ad would receive after being impressed i_r times at rank r, and EC_r is the average CTR for each position in the result page (up to R), computed over all pairs of query and ad. We thus can obtain COEC statistics for specific query-ad pairs. The counting feature is essential to convert huge amounts of discrete one-hot encoding features (billions) to only hundreds of dense real-valued features. A hash table is used to store the statistics and they are looked up online through the key likes "iPhone case_Ad3735". The statistics are refreshed regularly with a moving time window. 2). For some lookup keys (e.g., the long tail ones), there are too few impressions and clicks thus the statistics are pretty noisy. However, they still occupy a large amount of hash table storage. The solution is to assign these low impression/click data to a "garbage group", and the statistic that corresponds to this group is the default value if the key is missing in the hash table. A `garbage feature` with a binary value thus indicates whether or not current sample is in garbage group. 3). `Semantic feature` such as BM25. We also have a query/ad term based logistic regression model to capture the semantic relationship between the query term and ad term. The prediction output is treated as a feature.

Position Feature. We also record the specific position in which an ad is impressed. A search result page view (SRPV) may contain multiple ads at different positions, either in the mainline right after the search bar or in the right sidebar. Position feature w.r.t a specific position is the expected CTR based on a portion of traffic with randomized ad order. The specialty of position feature is that it never interacts with other statistic features (during feature engineering and model learning), but separates them out independently. The underlying consideration lies in the displayed position and the ad quality being two independent factors that affect the final click probability. Actually, we treat the position feature as a position prior as $p(click = 1|ad, position) \propto p(click = 1|ad) \cdot p(position)$. This separation of position features and other features is also validated by our experiments where it outperforms the model that interacts with them together. Since we do not know the position where the ad will be displayed, a default position (ML-1) is used to predict click probability online. In this way, we mainly compare the ad quality in the click prediction stage, i.e., all ads are set with the same value for position feature, and the specific position is finally determined in the ads allocation stage. Note that we still collect the click log into training data even when the corresponding clicked position is not ML1, as this is helpful for enriching the training data.

2.4 Baseline Model

Figure 2 depicts the baseline model we use, where several LRs and a NN model are ensembled together. Several LR models are first trained [1] so that each is fitted based on the one-hot features (with up to billions), and their prediction scores are treated as statistic features. Combined with the statistic and position feature listed above (Section 2.3), they are then fed into an NN model. NN is a "special" DNN with one single hidden layer. NN rather than DNN is selected since adding more layers and more units would have substantial offline gain, but the online gain is poor and not stable.

[1]We adopt FTRL ("Follow The (Proximally) Regularized Leader") [21, 20] or L1 regularization [11] to produce sparse model.

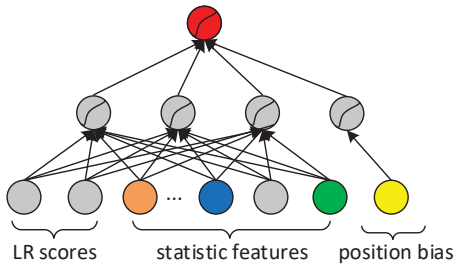


Figure 2: NN model used in production. There are three parts in input features: 1). the predicted scores of LRs; 2). statistic features; 3). position bias.

Besides, DNN introduces much more system costs in both training and serving. The postilion bias is only connected to a special hidden unit of NN to avoid the interaction. This cascading ensemble (stacking) shows good offline and online accuracy, and is considered the `baseline` that is compared with several novel ensembles described in Section 3.

On the one hand, we do not use a single model like LR and combine one-hot features and statistics features together, since it is hard to fit a good linear model with comparable cost, consider that there are a large number (1B) of sparse features and a small number of (100-500) dense features. Moreover, since the historical correlation in one-hot features is represented as the corresponding weights (parameters), thus the corresponding model needs to be updated frequently even with online learning to fit the latest trends. As a comparison, the statistic feature is updated in real-time that the corresponding model does not need to be re-trained frequently, e.g., the historical CTR of an advertiser can be updated as soon as a click or an impression of that advertiser occurs [2]. Lastly, the dimensionality of statistic features is much less than one-hot features, posing less challenge to offline training and online serving. Based on these factors, we choose to use NN as the baseline model that is fit from these statistic features. Note that all features including position features are first normalized by means of $\frac{x-min}{max-min}$. On the other hand, however, if we only keep the statistic feature, the tail cases could have poor prediction accuracy, since there are few impressions in training data and they will fall into the garbage group. Therefore, the NN model trained from the statistic features has no discrimination among these rare cases, thus leads to over-generalize and make less accurate prediction [7]. As a comparison, with more fine-grained term-level one-hot features with cross-product feature transformations, linear models (LRs) can memorize these "exception rules" and can learn different term-crossing weight [3]. Two LR models are ensembled, one is trained from older dataset and another is trained from the latest dataset. To mitigate the potential loss, our solution thus resorts to ensemble LR and NN together.

3. MODEL ENSEMBLE DESIGN

Different models can complement each other, and a model ensemble that combines multiple models into one model is a common practice in an industry setting to achieve better accuracy. In this section, we describe the different model ensemble designs and the cor-

[2]We actually have long term and real-time counting feature, the long term ones are updated per day and the real time ones are updated in seconds.

[3]We can record these term-level counts as statistic features but with more overheads. One feasible approach is to feed the dense embedding of these sparse term-crossing features to DNN, and we treat it as future work.

responding design consideration, which aim to provide better prediction accuracy than the baseline model.

3.1 Ensemble

Ensemble approaches. There are different ensemble [26] techniques that aim to decrease variance and bias, improve predictive accuracy (stacking), etc. The following is a short description of these methods: 1). **Bagging** stands for bootstrap aggregation. The idea behind bagging is that an overfitted model would have high variance but low bias in bias/variance tradeoff. Bagging decreases the variance of prediction by generating additional data from the original dataset using sampling with repetitions. 2). **Boosting** works with the under-fitted model that has high bias and low variance, i.e., the model cannot completely describe the inherent relationship in the data. With the insight that the model residuals still contain useful information, boosting at its heart repeatedly fits a new model on the remaining residuals. The final result is predicted by summing all models together. GBDT is the most widely used boosting model. 3). **Stacking** also first applies several models to the original data and the final prediction is the linear combination of these models. It introduces a meta-level and uses another model or approach to estimate the weight of each model, i.e., to determine which model performs well given these input data. 4). **Cascading** model A to model B means the results of model A are treated as new features to model B. Compared with stacking, cascading is more like joint training [7], with the difference that the cascaded models are not trained together but are trained separately without knowing each other. In contrast, joint training optimizes all parameters simultaneously by taking parameters of all models as well as the weights of their sum into account at training time. To simplify the description, we represent cascading A to B as $A2B$ and boosting A with B as $A+B$.

There are still questions regarding specific ensemble design that remain unanswered. Which models are ensembled together? Which ensemble techniques are used? Which ensemble design would achieve the best accuracy? Sometimes bagging or boosting works great, sometimes one or the other approach is mediocre or even negative. To answer them in our setting, we will present 8 ensemble variants in the next part.

3.2 Ensemble design

Design principles. There are several principled rules taken into consideration when we design the ensemble: 1). We do not consider the bagging approach since the variance of DNN is not significant especially if we regularize the model complexity, e.g., NN instead of DNN is used in production. The gain from bagging would be marginal. 2). Diversity is key to ensemble design. Non-parametric models such as decision trees are introduced to increase diversity since it differs largely with parametric models such as LR and DNN. Parametric models are usually optimized with gradient descent, while non-parametric models are fitted by greedily distinguishing the examples via clustering (K-Means) or splitting (decision tree). We believe the ensemble among non-parametric and parametric models would get more complementary benefits for accuracy. Boosting is commonly associated with gradient boosting decision trees (GBDTs). 3). Co-training between non-parametric models such as GBDT and parametric models such as LR/DNN is difficult even infeasible thus we do not consider joint training in this paper. Note that it is not easy to co-train multiple parametric models when they are optimized with different optimizers (FTRL of LR VS. AdaGrad of DNN) with different mini-batches and different asynchronization requirements. 4). We skip the ensemble of DNN and LR on statistic features. Our baseline model actually has the ensemble between DNN and LR already on one-hot features. However, in the last prediction stage, there are only statistic features. In this situation, the ensemble between DNN and LR is unnecessary since DNN is considered more powerful than LR such that for any given LR model there is always a DNN that has the same or larger representation capability. 5). Cascading is also emphasized. On the one hand, it is considered to have the benefits of co-training. [4] On the other hand, unlike co-training, it can ensemble the parametric and non-parametric models together. In the next, we will describe 9 ensemble variants that are all based on the same training data as our baseline model.

GBDT. The Gradient-Boosted Decision Tree (GBDT) is the ensemble of decision trees, and is widely used as it can model non-linear correlation, obtain interpretable results and does not need extra feature preprocessing such as normalization. GBDT iteratively trains T decision trees in order to minimize a loss function. During each iteration, the algorithm uses the current ensemble to predict the label of each sample and then compare the prediction with the true label. The dataset is re-labeled with the corresponding "residual" to put more emphasis on training instances with poor predictions. Thus, in the next iteration, a new decision tree will be fitted to correct for previous mistakes. The specific mechanism for re-labeling instances is defined by a loss function. Specifically, the t-th tree (f_t) is added to minimize the following objective:

$$\ell^t = \sum_{i=1}^{n} \ell(y_i, y_i^{t-1} + f_t(x_i)), \text{ where } f_t \in \Gamma \tag{3}$$

where y_i^t is the prediction of the i-th instance at the t-th iteration. $\Gamma = \{f(x) = w_{q(x)}\}, (q : R^m \to L, w \in R^L)$ is the structure space of decision trees. Here q represents the tree structure that maps a sample to the corresponding index of exit leaf ($q(x)$). Each leaf has a score (w). L is the number of leaves in the tree. Given sample x, GBDT uses T additive functions to predict the output, each subtree corresponds to a scoring function f_t and a shrinkage rate γ_t:

$$p = \sigma(\bar{y}); \ \bar{y} = \sum_{t=1}^{T} \gamma_t \cdot f_t(x) \tag{4}$$

The right part of Figure 1 depicts a GBDT model where nodes in blue color are the exit leaves. The specialty of our design is that the sample score in the first tree is initialized as the corresponding position bias whose value is roughly the expected CTR of all samples collected from an bucket traffic with randomized ad order. Note that we use the `inverse position bias`, i.e., given $pb = \sigma(x)$, we use x instead of pb. As pointed out in Section 2 that position bias cannot be crossed with other features, thus we never split position feature during training.

GBDT2LR: Cascading GBDT to LR. As pointed out by He, et al. [14], GBDT is a powerful way to implement non-linear and crossing transformations on input features. Specifically, we treat each individual tree as a categorical feature that takes as feature value the index of the leaf where a sample falls in. They are represented as one-hot encoding. These newly transformed features are then fed into LR as input feature. Essentially, GBDT based transformation is considered a supervised feature encoding that converts a real-valued feature vector into a compact binary-valued vector. A traversal from the root node to a leaf node represents a rule on the splitting features along the path. Fitting a linear classifier on the resulting binary vector is to learn the weights for these rules.

[4] It has part of the benefit since only one model's parameters are changed.

LR2GBDT: Cascading LR to GBDT. Conversely, we can also cascade LR to GBDT (with T subtrees). It first trains an LR model and uses as an input feature the prediction score of LR to a GBDT model. Given a sample x, the specific scoring formula is as follows:

$$p = \sigma(\bar{y}); \;\; \bar{y} = \sum_{t=1}^{T} \gamma_t \cdot f_t(x, \sigma(\overline{y_{lr}})) \tag{5}$$

Position bias here is only used in LR and never used in GBDT. LR has better accuracy if we use the inverse position bias rather than the normalized value. This is because the position bias is the expected CTR, i.e., the expected value of LR prediction, and the linear combination of non-position features (i.e., logit) can be regarded as the adjustment to the expected CTR. The inverse position bias is essentially to convert the position feature to the expected logit.

GBDT2DNN: Cascading GBDT to DNN. This is a cascading ensemble that first trains a GBDT model, and the predicting score of GBDT is fed as input feature (x_{gbdt}) into a DNN model. Given a sample x, the specific scoring formula is as follows:

$$\begin{aligned} &p = \sigma(\bar{y}); \;\; \bar{y} = \sigma(w_1 \cdot x_1 + b_1); \\ &x_1 = \sigma(w_0 \cdot (x_0, x_{gbdt}) + b_0) \end{aligned} \tag{6}$$

The position feature is only used to initialize the GBDT and not used in DNN to avoid the cross interaction. DNN here has only one hidden layer to simplify the description. Unlike `GBDT2LR`, we do not feed the transformed categorical features from GBDT to DNN, since DNN resorts to embedding to deal with the categorical features. Considering that we have a large number of trees and each tree has a large number of leaves, this introduces scalability issue on the DNN trainer. [5]

DNN2GBDT: Cascading DNN to GBDT. The opposite direction that cascades DNN to GBDT also should be tried. Specifically, it first trains a DNN model, and the DNN's predicting score is then fed as an input feature to a GBDT model. Given a sample x, the specific scoring formula is as follows:

$$p = \sigma(\bar{y}); \;\; \bar{y} = \sum_{t=1}^{T} \gamma_t \cdot f_t(x, \overline{y_{dnn}}) \tag{7}$$

Here position feature is used normally in DNN and GBDT training, however, the predicting score of DNN (input feature to GBDT) does not count the position bias to avoid cross interaction, i.e., the weight of position bias is set to 0 during prediction.

GBDT+DNN: stacking GBDT and DNN. DNN and GBDT are first trained separately used the same training data. Given a sample x, the final result is the average of prediction scores, with the following formula:

$$p = \sigma(\bar{y}); \;\; \bar{y} = \frac{1}{2} \cdot \overline{y_{dnn}} + \overline{y_{gbdt}} \tag{8}$$

We do not average the final predicted probability directly, instead the scores are averaged first and then input to sigmoid that returns the final probability.

LR+GBDT: Boosting LR with GBDT. It initializes the GBDT with a linear combination of input features learned by the LR. In other word, the pseudo target of a sample is initialized as the "residual" between the prediction score of LR and the real target before fitting the first tree $f_1(x)$. Although LR is a quite simple model, its prediction result has already good accuracy, i.e., the residual is quite small. Therefore, it is much easier to train compared with the original GBDT. The prediction scores of these T weak learners are added (boosted) sequentially to the prediction result of LR. Given a sample x, the specific ensemble is represented as:

$$\begin{aligned} &p = \sigma(\overline{y_{lr}} + \overline{y_{gbdt}}); \;\; \overline{y_{lr}} = w \cdot x + b; \\ &\overline{y_{gbdt}} = \gamma_1 \cdot f_1(x) + \sum_{t=2}^{T} \gamma_t \cdot f_t(x); \\ &f_1(x) = \arg\min_f \sum_{i=1}^{N} \ell(y_i, \overline{y_{lr}}_i + f(x_i)); \end{aligned} \tag{9}$$

Yandex [25] has adopted this boosting design for their ads CTR prediction. Instead of adding the predicted probability of LR directly, we actually add the logit computed by LR ($w \cdot x + b$) first and then apply the sigmoid to get the final prediction. Position feature (or inverse position bias) is only used in LR, we do not use it in GBDT to avoid the interaction between position feature and other features.

DNN+GBDT: boosting DNN with GBDT. Lastly, like `LR+GBDT`, DNN can also be boosted by GBDT. It first trains a DNN model, and the prediction score is used to initialize the GBDT (with T subtrees), i.e., the GBDT try to fit the residual between the optimal solution and DNN's result . Similarly, the prediction score of these T weak learners are added (boosted) sequentially to the prediction score of DNN, and feed the sum to sigmoid that returns the final probability. Given a sample x, the specific ensemble formula is as follows:

$$\begin{aligned} &p = \sigma(\overline{y_{dnn}} + \overline{y_{gbdt}}); \\ &\overline{y_{gbdt}} = \gamma_1 \cdot f_1(x) + \sum_{t=2}^{T} \gamma_t \cdot f_t(x); \\ &f_1(x) = \arg\min_f \sum_{i=1}^{n} \ell(y_i, \overline{y_{dnn}}_i + f(x_i)); \end{aligned} \tag{10}$$

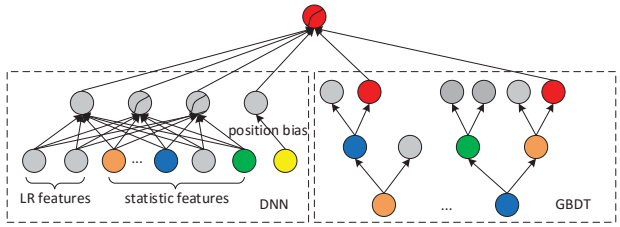


Figure 3: DNN+GBDT.

Figure 3 depicts the model structure, where the position feature is only used in DNN with the normalized (rather than reversed) value.

4. EVALUATION

We have compared these model ensembles against our baseline setting, and `DNN+GBDT` turns out to have the best accuracy in terms of offline testing AUC and click yields in online traffic.

4.1 Evaluation setup

Datasets. Training data used in this study consists of 56M examples which are randomly sampled from the logs generated in one month. For each sample, there are several hundreds of statistic features. To reduce the training cost, non-click cases are further down-sampled with a 50% sampling ratio. Each non-click sample is thus weighted by 2 such that the distribution is unchanged during the training. The model predicting accuracy is tested against dataset with 40M samples that are randomly drawn from the log generated in next week right after the training log. Without explicit description, all experiments have been applied to this dataset.

[5] Wide&Deep [7] can train heterogenous feature combination with both sparse features and dense embedding.

Accuracy Metric. The Area under Receiver Operating Characteristic Curve (AUC) [8] and Relative Information Gain (RIG) [14] are computed against the testing data to evaluate offline prediction accuracy. We calculate AUC normally, but with a small difference in RIG calculation that is defined as:

$$
\begin{aligned}
LL_{predict} &= -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log p_i + (1-y_i)\cdot \log(1-p_i), \\
LL_{empirical} &= -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log p_e + (1-y_i)\cdot \log(1-p_e) \\
RIG &= \frac{LL_{predict}}{LL_{empirical}} - 1
\end{aligned}
\tag{11}
$$

where y_i is the observed label of testing sample i, p_i is the predicted probability, and p_e is the empirical CTR that is calculated by $\frac{\#clicks}{\#impressions}$ in testing set. $LL_{predict}$ represents the mean cross entropy (i.e., the average log-loss per impression), and $LL_{empirical}$ is the average log-loss per impression if the CTR is predicted by a naive model that always predicts with the average empirical CTR. Dividing by $LL_{empirical}$ makes RIG insensitive to the average empirical CTR. AUC essentially evaluates the rank order and RIG measures the goodness of predicted value. For example, if we apply a global multiplier 0.5 to all predicted values, RIG will change even though AUC remains the same. Modelling with higher AUC and RIG value is considered to have better accuracy. Note that we compute AUC or RIG both at position=ALL and position=ML1, position=ALL is computed against the entire testing set, while position=ML1 is computed against a testing subset that consists of all samples impressed at ML1. In production, we care about AUC at position=ML1 more since the ads ranking, allocation and bidding are all based ML1 position, and in our experience, an AUC gain of 0.03% is statistically significant that exceeds the normal AUC variance and should not be neglected as noise.

The ensemble with significant offline accuracy will be picked for online A/B testing where we schedule two randomly sampled traffic buckets from full traffic as control and treatment. These two traffics have the same configuration settings through the whole serving stack except the click prediction model. We draw conclusion only when the online KPIs are statistic significant. We use the normalized click yield (CY) which removes the impact from the difference of impression yield.

Configuration of Model Ensembles. The evaluated model ensembles are described in Table 1, respectively. All model ensembles are trained and tested using the same dataset. Note that position features are handled differently in different ensembles (Section 3). In this section, DNN is configured to share the same configuration as our baseline model (Figure 2), i.e., a simple NN that has only one single hidden layer with 30 hidden units, and the activation is a sigmoid function. All features fed to LR and NN are first normalized by means of $\frac{x-min}{max-min}$ to ensure the value in [0,1]. The learning rate in DNN training starts from 0.005 and multiply by 0.2 every 4 iterations. The number of iterations (epoches) is 20 to avoid over-fitting based on the AUC gain trending on a validation set. Mini-batch size is 1 by default. LR is trained in full batch with LBFGS since the data size is small. There are 300 trees and each tree has 200 leaf nodes in GBDT, and the shrinkage ratio is 0.05 by default. The evaluation on different hyperparameter settings will be described in the next section.

4.2 Experiment Results

The experiment results of various ensembles are listed in Table 1. All the results are compared with the baseline NN model. We care much more about the accuracy at the ML1 position, but the results at ML=ALL are still listed for reference. We can draw the observations and the corresponding explanations as follows: **1).** The GBDT model has the best predictive accuracy among single models that has AUC lift 0.14% and RIG lift 0.36% than baseline NN, while LR is the worst with about 1.81% AUC loss. **2).** LR is always weaker than NN, which is validated by the results that $LR < NN$, $LR2GBDT < NN2GBDT$, $GBDT2LR < GBDT2NN$, and $LR + GBDT < NN + GBDT$. This is within our expectations. **3).** Almost all ensembles are better than the corresponding single model, with the only exception on `GBDT2LR` and `LR+GBDT` that are even worse than GBDT only. This indicates that boosting is better than cascading (will be described in the next part). It is noteworthy that `GBDT2LR` and `LR+GBDT` have been presented by Facebook [14] and Yandex [25], respectively. They behaved poorly was probably because Facebook mainly works for feed ads and the position feature may be not as important as with search ads.

4). Boosting is powerful that it can even further boost a non-weak model such as NN, e.g., $LR + GBDTV2 > LR$, and $NN + GBDT > NN$, and it is generally better than cascading/stacking. **5).** Lastly, `NN+GBDT` that boosts NN with GBDT turns out to be the best with 0.40% AUC gain and 2.81% RIG gain, respectively. The online A/B testing indicates that it has 1.3% click gains in online traffic. With larger training data, it can bring additional 0.5% AUC gains. Besides the A/B testing, we also have holdout flight that validates the effectiveness of `NN+GBDT`. i.e, we have the same online gain after mainstreaming into production.

4.3 Findings and Insights

The importance of Position Feature. The right treatment of position feature actually plays a critical role in prediction accuracy. First, position feature should be used in inverted form rather than the normalized one for LR, given that $LRV2 > LR$, $LR + GBDTV2 > LR + GBDT$ and $LR2GBDTV2 > LR2GBDT$. We need the inversed form because the final sigmoid will convert it back to position CTR which is empirical CTR. Second, position feature is the key factor in GBDT initialization and the boosting accuracy largely depends on the specific initialization. This is a key reason why $NN + GBDT > NN2GBDT > GBDT > LR + GBDT$. In GBDT, we have the freedom to apply any kind of transformation on the position feature before initialization, while afterward, it is never changed and never used for splitting trees to avoid the interaction among position and other features. Single GBDT ($GBDT$) and the cascaded GBDT ($LR2GBDT$ and $NN2GBDT$) are initialized with the manually-designed inverse transformation on position bias. However, it is hard to design a good transformation for a position feature, and the manual design usually leads to suboptimal accuracy. As a comparison, in boosting approach ($NN + GBDT$ and $LR + GBDT$), the transformation on position feature is automatically learned. For instance, in neutral net (shown in Figure 2), the weight of position feature to the hidden unit and the weight of that hidden unit to output can be learned together with other weights of the statistic feature during training. $NN + GBDT$ is better than $LR + GBDT$ because NN is considered better than LR.

Boosting is better than cascading. Most features in our setting are statistic features, they are updated frequently with dynamically changed value, e.g., for same $< query, ad >$ pair, the feature values are dynamic with different values at different time interval. Therefore, there might be a split-point shift issue that the split point learned at one day may not suitable for some days later. In cascading ensembles ($NN2GBDT$ and $LR2GBDT$), the split points in the first tree are learned from scratch and the split points of the same feature may vary significantly. As a comparison, in boosting

Models	Position=ML1		Position=ALL		Description
	AUC Gain	RIG Gain	AUC Gain	RIG Gain	
NN	0.00%	0.00%	0.00%	0.00%	NN with 1 hidden layer and 30 hidden units (baseline model)
LR	-1.97%	-16.14%	-1.46%	-10.01%	LR with normalized position bias
LR V2	-1.81%	-10.68%	-0.91%	-5.13%	LR with inversed position bias
GBDT2LR	0.06%	-0.17%	0.05%	0.44%	Cascade leaf index in GBDT as categorical feature to LR (used in Facebook [14])
LR+GBDT	0.12%	-1.87%	-0.33%	-1.93%	Boost LR with GBDT (used in Yandex [25])
LR2GBDT V2	0.13%	-0.14%	0.03%	0.67%	Cascade LR with inversed position bias to GBDT
GBDT	0.14%	0.36%	0.03%	0.91%	GBDT initialized with inversed position bias
LR2GBDT	0.14%	-0.27%	0.01%	0.50%	Cascade LR with normalized position bias to GBDT
GBDT2NN	0.16%	1.29%	0.04%	1.32%	Cascade GBDT to NN
LR+GBDT V2	0.24%	1.36%	0.07%	1.04%	Boost LR (inversed position bias) with GBDT
NN2GBDT	0.25%	0.15%	0.08%	0.72%	Cascade NN to GBDT
GBDT+DNN	0.25%	1.33%	0.15%	1.52%	Average NN and GBDT
NN+GBDT	0.40%	2.81%	0.15%	1.30%	Boost NN with GBDT

Table 1: Comparisons among different model ensembles. The result is ordered by the AUC gain at ML1. NN+GBDT turns to be the best, and we can see the RIG is generally consistent with the AUC.

mode ($NN+GBDT$ and $LR+GBDT$), the split point shift issue is much less serious than cascading, since the GBDT starts from the result of NN and LR and just focuses on fitting the residual. We observe this issue at A/B testing when experimenting $NN+GBDT$ and $GBDT2LR$ where NN is the baseline. During the entire 7 weeks, $NN+GBDT$ and NN show stable and consistent prediction error, while the average click probability of $GBDT2LR$ is unstable and drops significantly.

5. TRAINING OPTIMIZATIONS

The next challenge is to optimize the performance and accuracy in offline training. The detailed specific design and implementation is beyond the scope of this paper. In this section, we will share several accuracy-critical factors and optimizations that have proven effective for GBDT and DNN, respectively.

5.1 Hyper-parameter tuning in GBDT

Data size and tree number. We first show that the accuracy of GBDT improves as we increase the training data and number of trees, as shown in Figure 4. It is shown in the Figure 4a that as training data increases from 30M samples to 500M samples, AUC improves from 0.40% to 0.49%, and the RIG improves from 2.8% to 3.6%. We then fix the training data with 30M samples to evaluate the impacts of tree number. Figure 4b depicts that AUC improves from 0.29% to 0.52% and RIG improves from 2.1% to 3.2% when the number of trees increases from 100 to 2,400. RIG starts to degrade and AUC is saturated when tree number exceeds 2,000, which may be due to over-fitting. Note that accuracy gain can also been increased as we increase the number of leaf nodes (less than 400) in a single tree. This accuracy improvement is continuous but becomes smaller until over-fitting as we add more trees. We envision that more trees are required given more training data.

Bin Number and Feature Sampling. The feature value is first pre-binned [4] to reduce the number of split candidates, thus smaller bins lead to faster training. However, the number of bins would affect the final prediction accuracy. Figure 5a illustrates that 64 bins has the best accuracy and 16 bins has the worst accuracy, further increasing the bins does not improve the accuracy but with a little bit loss. We also perform stochastic boosting that randomly samples some features (or samples) to fit a tree. Figure 5b shows that we can get the best accuracy with 60% sample rate, this roughly means that we can nearly save 40% of training time [6].

[6]The saving depends on the specific training implementation.

Shrinkage Rate. The accuracy also depends on the proper hyper-parameter such as shrinkage rate. Shrinkage is a kind of tree regularization. The impacts on accuracy on ML1 position with different shrinkage (η) are shown in Figure 6. For the small training set with 27M samples (Figure 6a), there is little AUC difference for different shrinkage when tree number is less than 300. However, when the tree number increases from 300 to 2,400, the testing AUC decreases as we increase the shrinkage. As a comparison for the large training set with 470M samples (Figure 6b), AUC makes continuous improvements for large shrinkage as we increase the tree number. This indicates that a large training set can afford a larger shrinkage. One possible reason is that a large amount of shrinkage for small training data tends to cause over-fitting.

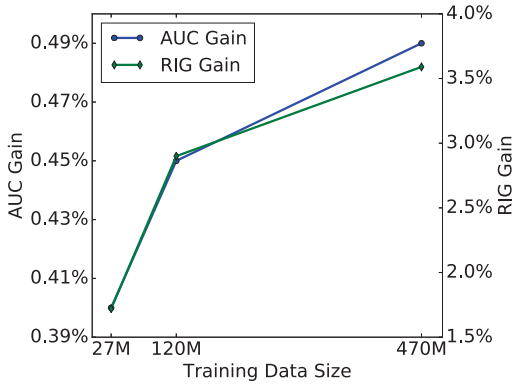
5.2 Accuracy Tuning for GBDT

Second Order Gradient. Inspired by XGBoost [4], we use second order Taylor expansion to approximate the loss function. Accordingly, the split gains and leaf scores are computed by considering the second order gradient. The difference on the split gain computing is shown in Table 2, where $g(x_i) = \partial_{f_{t-1}(x_i)} \ell(y_i, f(x_i))$ and $h(x_i) = \partial^2_{f_{t-1}(x_i)} \ell(y_i, f(x_i))$ are the first and second-order gradient on the loss function. For logloss, this second-order gradient based algorithm makes the model converge faster since the splitting gain calculation aims to reduce the global loss directly, rather than reduce the local loss of current tree that fits the pseudo residual (i.e., gradient). Figure 7 depicts the effectiveness of second-order gradient based training. Compared with the first-order method, AUC gain improves up to 0.05%. This will increase the training time by 20%-30%, since it introduces more computation in the split gain calculation.

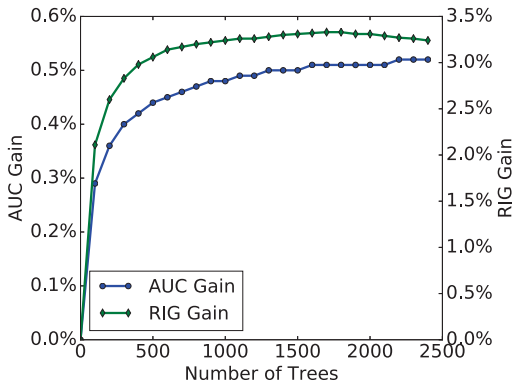
Method	Split-Gain Calculation
first-order	$gain_s = \frac{(\sum_{x_i \le s} g(x_i))^2}{\sum_{x_i \le s} 1} + \frac{(\sum_{x_i > s} g(x_i))^2}{\sum_{x_i > s} 1} - \frac{(\sum_{parent} g(x_i))^2}{\sum_{parent} 1}$
second-order	$gain_s = \frac{(\sum_{x_i \le s} g(x_i))^2}{\sum_{x_i \le s} h(x_i)} + \frac{(\sum_{x_i > s} g(x_i))^2}{\sum_{x_i > s} h(x_i)} - \frac{(\sum_{parent} g(x_i))^2}{\sum_{parent} h(x_i)}$

Table 2: Second-order gradient based split gain computation.

Negative Down-Sampling. A full day of Bing ads impression data can contain a huge amount of instances. On the one hand, more samples would achieve a better model. On the other hand, more samples will slowdown the training. Negative down-sampling [14], that keeps all positive (clicked) instances while performing uniform down-sampling for negative instances, has proven to be an effec-

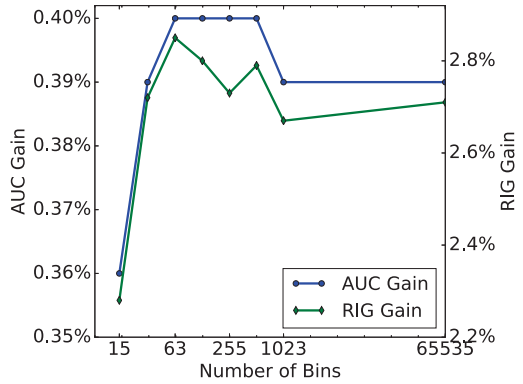


(a)

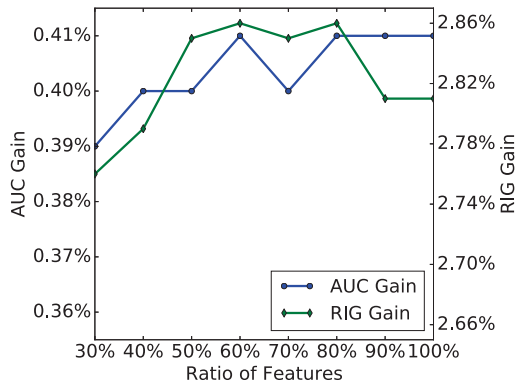


(b)

Figure 4: GBDT accuracy is improved by increasing the size of training data and the number of trees.



(a) Impacts on bin number.



(b) Impacts on feature sampling rate.

Figure 5: Impacts of hyper-parameter.

tive in speeding up the training. We re-weight the sample rather than re-calibrate the model [14] to ensure the same average CTR after down-sampling. For instance, the negative samples are re-weighted by 2 if the down-sampling rate is 50%. Experiments show that 50% sampling can save almost half the training time while the metrics are almost neutral (-0.01%/+0.02% AUC/RIG for ML1 position). Standard down-sampling does not consider the inherent imbalance in domains such as position. For instance, assume there are 40 positive and 60 negative samples at ML1 position, and 10 positive and 90 negative samples at ML4, after 50% down-sampling, the negative number becomes 30 and 45 at ML1 and ML4, respectively. Compared with the corresponding positive number, there are too few negative samples at ML1 while there are too many at ML4. In other words, the different positions should have different down-sampling rates. We have actually evaluated other sampling strategies that keep all positive/negative cases for clicked SRPV, and for non-clicked SRPV we either do `uniform down-sampling`, `SRPV-wise down-sampling` or `position-wise down-sampling`. The comparison among 4 different down-sampling methods against 120M data (after 50% down-sampling) indicates that position-wise negative down-sampling achieves the best accuracy.

Local Case-Control Sampling. We also evaluate the local case-control (LCC) [27] sampling that does down-sampling for both positive and negative instances. The sampling is different for different instances. Specifically, whether or not a sample is added depends on the absolute prediction error from a pilot model ($p(x)$ as below) which is trained on a small subset.

$$a(x,y) = |y - p(x)| = \begin{cases} 1 - p(x) & \texttt{y=1}, \\ p(x) & \texttt{y=0}. \end{cases} \tag{12}$$

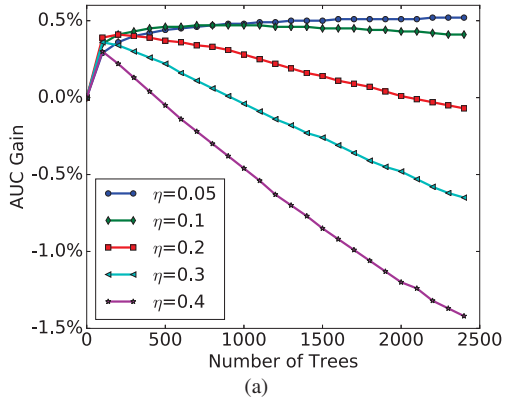
After LCC sampling, the ratio of instances, which have been learned well in NN as a pilot model, will drop and the ratio of poor learned cases will increase. GBDT then focuses more on these poorly learned cases with LCC sampling. Table 3 depicts the evaluation effectiveness of LCC sampling. It is shown that `NN+GBDT` with LCC sampling can further improve accuracy with 0.06% AUC gain and 0.23% RIG gain. When we look into the breakdown metric, we can see that most gain comes from tail traffic (0.22% AUC gain and 1.64% RIG gain), which are poorly learned part in the pilot model. There is even slight loss at head traffic (-0.03% AUC loss and -0.19% RIG loss), probably because the head traffic is significant reduced in the sampled data. In our experiments, this method removes 40%-80% training data depending on the original data set, significantly reducing training cost.

	Position=ALL		Position=ML1	
	AUC Gain	RIG Gain	AUC Gain	RIG Gain
`lcc-sampling`	0.04%	1.63%	0.06%	0.23%

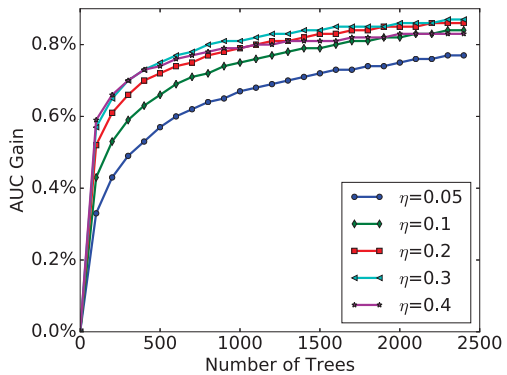
Table 3: Offline metrics for lcc-sampling.

5.3 Hyper-parameter tuning in DNN

Hidden Layers and Neuron Number. We have also evaluated the accuracy by adding more hidden layers and units. Figure 8a depicts the results of $NN + GBDT$ when the NN has a different number of hidden units, and the evaluation on different hidden layers are shown in Figure 8b. The results are relative to the baseline NN

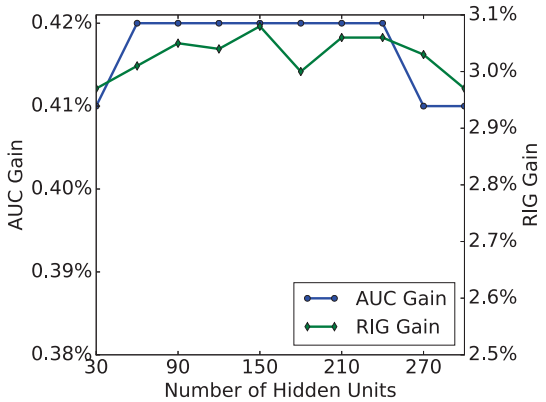


(a)

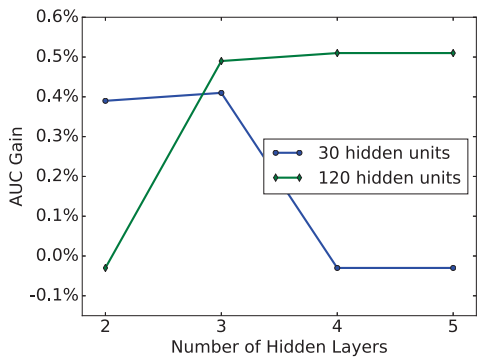


(b)

Figure 6: Impacts of learning rate on 27M training data (a), 470M training data (b).



(a)



(b)

Figure 8: Impacts of different DNN layers and units.

model with 30 hidden units. We can see that an increase in the complexity of DNN will have marginal gain, e.g., with only 0.02% extra gain when increase the units from 30 to 90. However, the AUC will not improve as we further increase the unit number, with even AUC and RIG loss when the unit number is 270. Similarly, if each hidden layer has 30 units, adding more hidden layers does not help and even cause loss; if each hidden layer has 120 units, 3 hidden layers is better than 2 hidden layers, but, adding more layers only brings marginal gain until it gets saturated.

6. RELATED WORK

Sponsored search advertising relies heavily on the accurate, scalable and quick prediction of ad click-through rates. Click prediction has received much attention from both industry and academia [10, 16]. The majority of large scale models in industry make use of logistic regression [21, 22, 14] for its scalability and online learning capability. Google [21] trains LR using an FTRL-Proximal online learning algorithm in order to increase model sparsity and memory saving. Microsoft [13] develops a Bayesian online learning algorithm for sponsored search advertising in Bing Search Engine. Yahoo Criteo [3] uses Bayesian logistic regression with hashing one-hot encoding features to predict clicks for advertising. The model updates with a new batch of data by leveraging the posterior distribution of a previously trained model as the prior for the new model. Facebook [14] combines decision trees with logistic regression. Decision trees transform each sample into the 1-of-K coding of the index of the leaf it ends up falling in each tree.

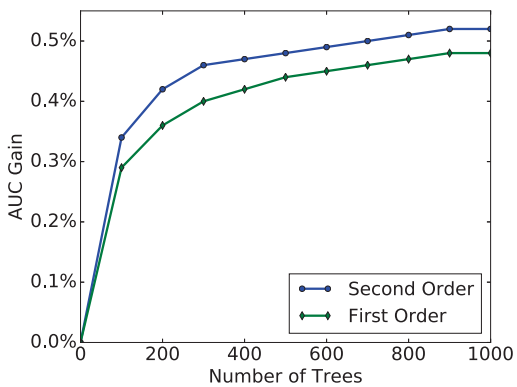


Figure 7: Effectiveness of second-order gradient.

Another trend of models for predicting click-through rate focuses on neural networks in order to improve the accuracy. Most of these works [29, 23] focus on engineering the transformation of raw features. [29] deploys factorization machines, or a sampling-based restricted Boltzmann machine or denoising autoencoder as the bottom layer of a deep neural framework in order to reduce dimensions from one-hot sparse features to dense continuous features. The deep crossing model [23] uses a single layer of the neural network as the embedding layer for each individual feature in order to avoid handcrafting combinatorial features. The output embedding is then concatenated as the input to a residual network. DeepIntent [28] uses RNNs to model the word sequence in queries and ads. On top of RNN, they propose attention based pooling to represent a sequence by a weighted sum of the vector representations of all time steps. The work [30] leverages the temporal dependency in user's behavior sequence through RNNs. However, these deep neural networks have marginal gains in real production. This is

why we adopt a shallow neural network empowered by boosting tree ensembles to capture efficiency and accuracy.

7. CONCLUSION AND FUTURE WORK

In this paper, we share our experience on designing and optimizing the model ensembles to improve ads CTR prediction in Microsoft Bing Ads. The ensemble that boosts NN with the GBDT turns out to be the best in our setting. We also share the experience in accelerating the training performance and improving the training accuracy. We believe the model ensemble is a promising direction; meanwhile, as indicated by position feature, the domain knowledge is also indispensable in the ensemble design. In the future, we will experiment different feature separation and different ensemble designs to push the limit of accuracy, e.g., we will experiment different DNN architectures such as the RNNs as well as different learning algorithms.

8. ACKNOWLEDGMENTS

We thank Pavel Serdyukov to invite this paper in the Industry Track. We are grateful the anonymous reviewers for their insightful comments. We thank Eren Manavoglu who kindly reviewed our paper and gave us valuable suggestions. Lastly, we would also like to show our gratitude to Lintao Zhang for his continued support.

9. REFERENCES

[1] E. Benjamin, O. Michael, and S. Michael. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *The American economic review*, 97(1):242–259, 2007.

[2] V. B. C., D. Vacha, G. Saikat, and S. A. C. Empirical analysis of search advertising strategies. IMC, pages 79–91, 2015.

[3] O. Chapelle, E. Manavoglu, and R. Rosales. Simple and scalable response prediction for display advertising. *ACM Trans. Intell. Syst. Technol.*, 5(4):61:1–61:34, 2014.

[4] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[5] Y. Chen, M. Kapralov, J. Canny, and D. Y. Pavlov. Factor modeling for advertisement targeting. In *NIPS*, pages 324–332. 2009.

[6] H. Cheng and E. Cantú-Paz. Personalized click prediction in sponsored search. WSDM, pages 351–360, 2010.

[7] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah. Wide & deep learning for recommender systems. 2016.

[8] C. Corinna and M. Mehryar. Auc optimization vs. error rate minimization. *NIPS*, pages 313–320, 2004.

[9] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *WSDM*, pages 87–94, 2008.

[10] D. C. Fain and J. O. Pedersen. Sponsored search: A brief history. *Bulletin of the American Society for Information Science and Technology*, 32(2):12–13, 2006.

[11] A. Galen and G. Jianfeng. Scalable training of l1-regularized log-linear models. ICML, pages 33–40, 2007.

[12] T. Graepel, J. Q. Candela, T. Borchert, and R. Herbrich. Web-scale bayesian click-through rate prediction for sponsored search advertising in Microsoft's Bing search engine. In *ICML*, 2010.

[13] T. Graepel, J. Q. Candela, T. Borchert, and R. Herbrich. Web-scale bayesian click-through rate prediction for sponsored search advertising in Microsoft's bing search engine. In *ICML*, pages 13–20, 2010.

[14] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, and J. Q. n. Candela. Practical lessons from predicting clicks on ads at facebook. ADKDD, pages 5:1–5:9, 2014.

[15] K. Inc. Welcome to kaggle competitions. `https://www.kaggle.com/competitions`, 2016.

[16] B. J. Jansen and T. Mullen. Sponsored search: an overview of the concept, history, and technology. *International Journal of Electronic Business*, 6(2):114–131, 2008.

[17] Y. Juan, D. Lafortier, and O. Chapelle. Field-aware factorization machines in a real-world online advertising system. In *WWW*, 2017.

[18] Y. Juan, Y. Zhuang, W.-S. Chin, and C.-J. Lin. Field-aware factorization machines for CTR prediction. RecSys, pages 43–50, 2016.

[19] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[20] H. B. McMahan. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In *AISTATS*, 2011.

[21] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica. Ad click prediction: a view from the trenches. In *KDD*, 2013.

[22] M. Richardson, E. Dominowska, and R. Ragno. Predicting clicks: Estimating the click-through rate for new ads. In *WWW*, 2007.

[23] Y. Shan, T. R. Hoens, J. Jiao, H. Wang, D. Yu, and J. Mao. Deep Crossing: Web-scale modeling without manually crafted combinatorial features. KDD, pages 255–262, 2016.

[24] A. Ta. Factorization machines with follow-the-regularized-leader for CTR prediction in display advertising. In *International Conference on Big Data*, pages 2889–2891, 2015.

[25] I. Trofimov, A. Kornetova, and V. Topinskiy. Using boosted trees for click-through rate prediction for sponsored search. ADKDD, pages 2:1–2:6, 2012.

[26] Wikipedia. Ensemble learning. `https://en.wikipedia.org/wiki/Ensemble_learning`, 2016.

[27] F. William and H. Trevor. Local case-control sampling: Efficient subsampling in imbalanced data sets. *Annals of statistics*, 42(5):1693, 2014.

[28] S. Zhai, K.-h. Chang, R. Zhang, and Z. M. Zhang. DeepIntent: Learning attentions for online advertising with recurrent neural networks. KDD, pages 1295–1304, 2016.

[29] W. Zhang, T. Du, and J. Wang. *Deep Learning over Multi-field Categorical Data*, pages 45–57. Springer, 2016.

[30] Y. Zhang, H. Dai, C. Xu, J. Feng, T. Wang, J. Bian, B. Wang, and T.-Y. Liu. Sequential click prediction for sponsored search with recurrent neural networks. AAAI, pages 1369–1375, 2014.