Dynamic Length Factorization Machines for CTR Prediction

Yohay Kaplan, Yair Koren, Rina Leibovits, and Oren Somekh
Yahoo Research
Haifa, Israel
{yohay,yairkoren,rina.levy,orens}@yahooinc.com

***Abstract*—Ad click-though rate prediction (pCTR) is one of the core tasks of online advertising. Driving the pCTR models of Yahoo Gemini native advertising is OFFSET - a feature enhanced collaborative-filtering based event prediction algorithm. Due to data sparsity issues OFFSET models both users and items by mapping their features into a latent space, where the resulting user vector is a non-linear function of the user feature vectors (e.g., age, gender, hour, etc.) which allows pairwise dependencies. This pairwise dependencies concept is also used by other algorithms such as the Field-aware Factorization Machines (FFM). However, both in OFFSET and in FFM, the different pairwise interactions are modeled by latent vectors of constant and equal lengths. When prediction models are used online for serving real traffic, where the total serving model size is often limited, a non uniform representation of the pairwise interactions should be considered in order to maximize the accuracy of the model while consuming the same or even less space. In this work we present a Dynamic Length Factorization Machines (DLFM) algorithm that dynamically optimizes the length of the vectors for each feature interaction during training, while not exceeding a maximal overall latent vector size. After showing good online performance of 1.46% revenue lift and a 2.15% CTR lift, serving Gemini native traffic, the DLFM was pushed into production. Since integrated into production, the DLFM has not only improved the accuracy of the model by optimizing the length of each latent space, but has also reduced the total size of the model by 25%. Although the algorithm was applied to OFFSET, we show that DLFM can be applied to any FFM-like algorithm to optimize its pairwise feature vector lengths. We also present an *Educated Model Initialization* - a novel mechanism for initializing a new model based on an existing model that has some mutual user features. Using this mechanism, we managed to reduce the training time of our models by more than 90% when compared to an equivalent model that is trained from "scratch".**

***Index Terms*—Computational advertising, Recommendation systems, Collaborative filtering, Factorization machines, Dynamic length optimization.**

I. INTRODUCTION

Launched seven years ago and operating with a yearly run-rate of many hundred of millions USD, Yahoo Gemini native marketplace[1] is one of Yahoo's largest and fastest growing businesses. With more than two billion impressions daily, and an inventory of a few hundred thousand active ads, Gemini native serves users with ads that are rendered to resemble the surrounding native content (see Figure 1 for examples of Gemini native ads on different devices). Serving native ads is considered more challenging in general, since in contrast to the search-ads marketplace, users' intent during page visits are unknown.

Fig. 1: Yahoo Gemini native ads on different devices.

In order to rank native ads for an incoming users and their specific context according to the cost per click (CPC) price type, a score (or expected revenue) is calculated by multiplying the advertiser's bid and the predicted click-through rate (pCTR) for each ad. The pCTR is calculated using models that are generated by OFFSET - a feature enhanced collaborative-filtering (CF) based event-prediction algorithm [1]. OFFSET is a one-pass incremental algorithm that updates its latent factor model for every new batch of logged data using a *gradient* based learning approach. OFFSET is implemented on the grid using *map-reduce* architecture [10], where every new logged data batch is pre-processed and parsed in parallel by many *mappers* and the ongoing training of model instances with different hyper parameters sets is done in parallel by many *reducers* to facilitate OFFSET adaptive online hyper-parameter tuning process [2].

OFFSET represents its users by their features (e.g., age, gender, geo, etc.), where each feature value (e.g., female, male, for gender) is represented by a *latent factor vector* (LFV). A user's LFV is derived from the user features' LFV by applying a non-linear function which allows for pairwise feature dependencies. Originally, this function did not distinguish between the different user features, and each one of them was allocated with the same dimension in the user vector LFV. In this work we consider an online dynamic

[1] See https://gemini.yahoo.com/advertiser/home

optimization of the user vector allocation by shortening some features' LFV and extending others, while not exceeding a maximal predefined overall LFV length. This *dynamic length factorization machines* (DLFM) algorithm changes user vector structure between training cycles to optimize model predictions accuracy. After providing good offline results, the DLFM was tested in online buckets, serving Gemini native users, and demonstrating a 1.46% revenue lift and a 2.15% CTR lift over a regular OFFSET driven click model operating with no LFV length optimization. Moreover, after DLFM was deployed in production, it showed a 25% model size reduction when compared to the baseline model. Finally, using parts of the DLFM optimization mechanism, we also present a new way to initialize models, that considerably expedites their training time. In particular, the *educated model initialization* mechanism utilizes mature models with similar feature set to "seed" new models instead of training them from "scratch" and thus saving as much as 90% of required training period to reach a certain predefined accuracy level.

All the aforementioned mechanisms require considerable additional backend resources, while none are required from the serving system, which is agnostics of their operation. This is acceptable since we are extremely sensitive to any resources surge in the Serving system, as traffic is spread globally among many servers for maintaining harsh *service-level-agreement* (SLA) requirements. However, we are almost agnostic to backend resources used to train our models, which is carried out over a couple of production grids. Moreover, as we shall demonstrate, serving resources are actually reduced due to model size reduction achieved by DLFM.

We note that the proposed algorithms which were demonstrated using OFFSET are applicable to similar *factorization machines* (FM) based algorithm [26], such as the *field-aware factorization machines* (FFM) [19][18].

The main contributions of this work are:

- The *dynamic length Factorization machines* (DLFM) algorithm - a dynamic optimization of the user vector structure which provides a better representation of each feature and each pair of features, under a maximal vector length constraint.
- The *educated model initialization* (EMI) algorithm - an algorithm for initialization of new models base on existing mature ones to accelerate the training period.
- The DLFM was evaluated online, serving real traffic and demonstrating a 1.46% revenue lift and a 2.15% CTR lift over the baseline.
- The DLFM is deployed in production, demonstrating model size reduction of 25% when compared to its baseline model size with no performance degradation.
- We show that the proposed DLFM algorithm, which was successfully applied to OFFSET, may be applied to other FM based algorithms, such as FFM.

The rest of the paper is organized as follows. In Sections II and III we provide the relevant background and related work. We set our goal in Section IV and elaborate on our approach in Section V. Section VI presents the offline and online evaluation of our solution. The size reduction achieved by applying DLFM is considered in Section VII. In Section VIII we discuss how our approach can be applied to other FM base algorithms. EMI is presented and evaluated in Section IX. Finally, we conclude and consider future work in Section X.

II. THE OFFSET ALGORITHM

Yahoo Gemini native models are driven by OFFSET (One-pass Factorization of Feature Sets): a feature enhanced collaborative-filtering (CF) based event prediction algorithm [1].

The pCTR of a given user u and an ad a according to OFFSET is given by

$$\text{pCTR}(u,a) = \sigma(\phi(u,a)) \ \in [0,1] \quad (1)$$

where $\sigma(x) = (1+e^{-x})^{-1}$ is the *Logistic sigmoid* function, and $\phi(u,a) = b + \nu_u^T \nu_a$, where $\nu_u, \nu_a \in \mathrm{I\!R}^{\mathrm{D}}$ denote the user and ad latent factor vectors respectively, and $b \in \mathrm{I\!R}$ denotes the model bias. The product $\nu_u^T \nu_a$ denotes the tendency score of user u towards ad a, where a higher score translates into a higher pCTR. Note that $\Theta = \{\nu_u, \nu_a, b\}$ are the model parameters which are learned from the logged data, as explained below.

A. OFFSET Latent Vectors

Each user u is associated with a set of F user-feature values $[u^1, u^2, ..., u^F]$ (e.g., his gender, his age, his geo, etc.), where $u^f \in N_f$ and N_f is the set of available values for feature f (e.g., if f denotes gender, then $u^f \in \{male, female, unknown\}$). Each value u^f is represented by a unique latent vector v_u^f, and the user vector ν_u is a function of the vectors $[v_u^1, v_u^2, ..., v_u^F]$. In a similar manner, each ad a is also associated with a set of A ad-features values (e.g., the ad id, the campaign id, the advertiser id, the ad category, etc.), and the vector ν_a is a function of the vectors $[v_a^1, v_a^2, ..., v_a^A]$. However the construction of the user latent vector and the ad latent vector are quite different.

a) The User Vector: The construction of the user latent vector given its different user-feature vectors is more complex in order to allow non-linear pairwise dependencies between feature pairs. The user vector $\nu_u \in \mathrm{I\!R}^{\mathrm{D}}$ is a concatenation of $\left(\binom{F}{2} + F\right)$ "blocks" of size k, i.e., $D = \left(\binom{F}{2} + F\right) \cdot k$. Each block is dedicated to either a combination of two features ($\binom{F}{2}$ of the blocks), or a single feature (the rest F blocks)[2]. Each feature vector $v_u^f \in \mathrm{I\!R}^{\mathrm{d}}$, where $d = F \cdot k$, has one block of k entries that is attributed to that specific feature only, and $(F-1)$ blocks each of size k that are overlapping with the rest $(F-1)$ features in the user vector. While constructing the user vector, each block of the feature vector is mapped to the appropriate block in the user vector. Now, let $\tilde{v}_u^f \in \mathrm{I\!R}^{\mathrm{D}}$ be an extension of v_u^f, such that the entries of v_u^f appear in d of the entries of $\tilde{v}_u^f$ according to the user vector mapping, and the rest of the entries are filled with 1's. Using this notation we

[2]For simplicity we set both, pair vectors and single vectors, to have equal dimension k, however, in practice they may be set to different dimensions k_p and k_s.

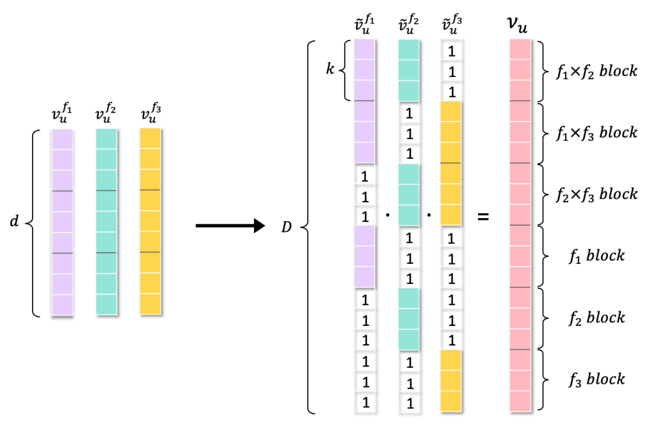


Fig. 2: Example of a user latent factor vector construction for or $F = 3,\ k = 3$.

can now formulate the user vector, which is given by $\nu_u = \Pi_{f=1}^{F} \tilde{v}_u^f$. An illustration of the user vector construction is given in Figure 2.

We end by emphasizing the advantage of presenting users by their features over the standard CF approach where each user is assigned with a unique LFV. Hence, the model includes only $O(F)$ feature LFVs instead of hundreds of millions of unique user LFVs and user *cold start* problem is avoided (see [5][9][11] and references therein).

b) The Ad Vector: In the ad space, each vector v_a^i is of length D, and the ad vector $\nu_a \in \mathrm{IR}^{\mathrm{D}}$ is a simple summation of its feature values' vectors, i.e., $\nu_a = \sum_{i=1}^{A} v_a^i$.

B. Training

To learn the model parameters Θ, OFFSET minimizes the logistic loss (LogLoss) of the training data set $\mathcal{T}$ (i.e., past skips and clicks) using one-pass *stochastic gradient descent* (SGD)

$$\underset{\Theta}{\operatorname{argmin}} \sum_{(u,a,y) \in \mathcal{T}} \mathcal{L}(u,a,y) \ ,$$

where

$$\mathcal{L}(u,a,y) = -(1-y)\log\left(1 - pCTR(u,a)\right) - y \log pCTR(u,a) + \frac{\lambda}{2} \sum_{\theta \in \Theta} \theta^2 \ ,$$

$y \in \{0,1\}$ is the click indicator for the event involving user u and ad a, and λ is the $L2$ regularization parameter. For each training event (u,a,y), OFFSET updates its relevant model parameters by the SGD step

$$\theta \leftarrow \theta - \eta(\theta) \nabla_\theta \mathcal{L}(u,a,y) \ ,$$

where $\nabla_\theta \mathcal{L}(u,a,y,t)$ is the gradient of the objective function w.r.t θ. In addition, the parameter-dependent step size is given by

$$\eta(\theta) = \eta_0 \frac{1}{\alpha + \left(\sum_{(u,a,y) \in \mathcal{T}'} |\nabla_\theta \mathcal{L}(u,a,y)|\right)^{\beta}} \ ,$$

where η_0 is the SGD initial step-size, $\alpha,\ \beta \in \mathrm{IR}^+$ are the parameters of the adaptive gradient (AdaGrad) algorithm [12], and $\mathcal{T}'$ is the set of training events seen so far. OFFSET updates its model incrementally for every new batch of logged events (e.g., 15 minutes worth of data), and periodically sends its model to the serving system.

The OFFSET algorithm also includes an adaptive online hyper-parameter tuning mechanism [2]. This mechanism takes advantage of the system parallel architecture and strives to tune OFFSET hyper-parameters (e.g., step size and AdaGrad parameters) to match the varying marketplace conditions (changed by temporal effects and trends). We note that other components of OFFSET, such as its weighted multi-value feature [6], and similarity weights used for applying "soft" recency and frequency rules[3] [3], are not presented here for the sake of brevity.

C. Serving

When a user arrives at a Yahoo *owned and operated* (O&O) or Syndication[4] site, and a Gemini native slot should be populated by ads, an auction takes place. Initially, Serving generates a list of eligible active ads for the user as well as each ad's score.

The score is a measure that attempts to rank the ads according to their potential revenue with respect to the arriving user and her context (e.g., day, hour, site, device type, etc.). In general, an ad's score is defined as

$$\mathrm{rankingScore}(u,a) = \mathrm{bid}(a) \cdot \mathrm{pCTR}(u,a) \ ,$$

where, $\mathrm{pCTR}(u,a)$ is the predicted click probability (see Eq. (1)), and $\mathrm{bid}(a)$ is the amount of money the advertiser is wiling to pay for a click on ad a. In order to calculate $pCTR(u,a)$, the serving system fetches the bias b, constructs the user vector $\nu_u = \Pi_{f=1}^{F} \tilde{v}_u^f$, constructs the ad vector $\nu_a = \sum_{i=1}^{A} v_a^i$, and applies (1) to get the click prediction, for all eligible ads.

To encourage advertiser truthfulness, the cost incurred on the winner of the auction is according to *generalized second price* (GSP) [13], which is defined as

$$\mathrm{gsp} = \frac{\mathrm{rankingScore}_2}{\mathrm{rankingScore}_1} \cdot \mathrm{bid}_1 = \frac{\mathrm{pCTR}_2}{\mathrm{pCTR}_1} \cdot \mathrm{bid}_2 \ ,$$

where indices 1 and 2 correspond to the winner of the auction and the runner up, respectively. Note that by definition $\mathrm{gsp} \leq \mathrm{bid}_1$, which means the winner will pay no more than its bid. In particular, if both ads have the same pCTR, the winner will pay the bid of the runner-up (i.e., bid_2).

III. RELATED WORK AND PRACTICE

There are few published works describing models driving web scale advertising platforms. In [24] lessons learned from experimenting with a large scale logistic regression model used for CTR prediction by Google advertising system are reported.

[3]How frequent and how recent a user may be presented with the same ad or campaign.

[4]Where Yahoo presents its ads on a third party site and shares the revenue with the site owner.

These include improvements in traditional supervised learning based on a *follow the regularized leader* like online learning [23], and the use of per-coordinate learning rates. A model that combines decision trees with logistic regression is used to drive Facebook CTR prediction and is reported on in [16]. Placing ads in a tweet stream is considered in [21], where pairwise ranking is used to train a model for CTR prediction.

Recommendation technologies are crucial for CTR prediction, and without them users will find it hard to navigate through the Internet and get what they like. In particular, *collaborative filtering* (CF) in general and specifically *matrix factorization* (MF) based approaches are leading recommendation technologies, according to which entities are represented by latent vectors and learned by users' feedback (such as ratings, clicks and purchases) [20]. MF-CF based models are used successfully for many recommendation tasks such as movie recommendation [7], music recommendation [4], ad matching [1], and much more. CF is evolving constantly, where recently it was combined with *deep learning* (DL) for embedding entities into the model [15].

Factorization machines (FM) provides a framework which captures many MF-CF architectures that were considered over recent years [26]. In particular, OFFSET may be seen as a special case of FM. OFFSET is also closely related to FFM [19][18] and FwFM [25]. However, it is fundamentally different than FFM and FwFM, since those treat the ad side as a regular field, while in OFFSET the ad side field is unique and is multiplied by the user features pair and single k dimensions LFVs. Hence, the resulting score is a sum of triplet and pair multiplications as oppose to the pair multiplications only of FFM and FwFM.

Similar ideas of feature vector allocation and model size optimization were considered (in parallel to our efforts) in [27]. In this work the *primary component analysis* (PCA) is used for size reduction in a unified framework referred to as FM^2. It is noted that while FM^2 algorithm was tested in offline using public datasets, our DLFM algorithm is embedded in a commercial advertising system and is online evaluated in Web scale serving real traffic. Moreover, while FM^2 uses a proxy (i.e., PCA) for pruning its vectors, DLFM uses LogLoss, which is the actual metric used to train the model parameters. Finally, DLFM is an online algorithm which is capable of adapting to temporal affects and trends, while FM^2 in its current form optimizes the sizes which are then used in the actual model train. It is worth mentioning that dimension reduction is also considered for deep learning models in various settings (e.g., see [29], [17], [8] and references therein).

Model size optimization is a common practice in the process of building CF-MF recommender systems, and it is done as part of the algorithm hyper parameter tuning. Accordingly, models are trained offline with increasing dimensions until the residual performance lifts are small enough. The DLFM algorithm presented here is novel since it is done online and it allows non-uniform feature structure, where each feature and feature pair establishes its "correct" dimension over time while improving prediction accuracy. Moreover, we show that DLFM is applicable to other popular FM based algorithms such as FFM. As far as we are aware, there is no work that considers such a dynamic optimization process.

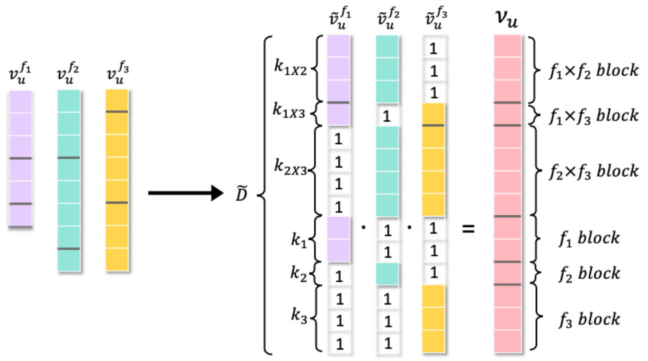


Fig. 3: An example of a nonuniform user vector allocation after applying the DLFM algorithm. Observing feature f_3, it is evident that 2 entries were removed from the block representing its interaction with f_1, while the block representing its interaction with f_2 has been expanded. In addition, the block representing f_3 alone remains unchanged.

IV. OUR GOAL

Being a major component of any web scale online advertising system, the event prediction model (e.g., ad click prediction model) and in particular its size plays a crucial role. Focusing on MF-CF models (such as OFFSET) the LFV size affects many of the system performance measures such as the training time, memory consumption, serving delay, and number of queries (or auctions) per second (QPS) it can handle. As described in Section II the size of the model is dictated by the length of the user vector $D = \left(\binom{F}{2} + F\right) \cdot k$. Hence, in order to reduce the size of the model without removing existing features, the parameter k should be reduced. Our goal is to take the optimization a step farther. As the different user features hold different informative value, using a uniform block size k to represent each of the features and their interactions in the user vector, prevents the model from reaching its full potential. Let k_{f_i} denotes the block size assigned to feature f_i in the user vector, and k_{f_i,f_j} denotes the block size assigned to the interaction of f_i and f_j in the user vector. Our goal is to dynamically optimize the values k_{f_i}, and k_{f_i,f_j} for every i, j. By allowing non uniform blocks sizes, not only that we can reduce the model size, but we would also improve the model accuracy by finding the right combination of features that provides the best user representation.

In this work we suggest a dynamic approach for optimizing the user vector allocation automatically with respect to the online data as well as a solution to the problem of adding new features to an existing allocation. An example of a user vector allocation snapshot after applying DLFM is depicted in Figure 3

V. Our Approach

A. Overview

Starting with a uniform structure, the *dynamic length factorization machines* (DLFM) algorithm optimizes the user vector structure by testing and applying small changes to it every tuning cycle (i.e., 3 hours). In practice, the algorithm can remove and/or add up to N vector entries per tuning cycle, where N is a system parameter. Each such change is automatically tested and evaluated before being applied to the model.

To evaluate which entries can be removed or added to the model, DLFM uses two main components. The first component analyzes the user and ad vectors during training, and evaluates which entries can potentially be removed, i.e., which blocks can be shortened. The second component trains potential extensions for each of the blocks, and evaluate the benefit of adding the extensions to the model, i.e., expanding a block.

To execute these two components, we take advantage of the map-reduce architecture of OFFSET, which allows training hundreds of models in parallel. Originally, the system was designed to train multiple versions of the learning model in parallel, each one with a different set of hyper-parameters [2]. At the end of each tuning cycle, all of the models are evaluated and the best resulting model is selected. Then, the system virtually duplicates the best performing model and resuming its training with multiple new hyper-parameter sets.

We support DLFM by adding to each hyper-parameter tuning cycle several new models, each one of them with a slightly different structure, and train them alongside the different hyper-parameter sets models. If one of these models achieves the best results, the training during the next cycle will continue from this model structure, with new generated variations of its hyper-parameters, and new generated variations of its user vector structure. The DLFM models tested in each cycle differ in the number of changed entries (1 to n) and the operation applied (i.e., entry addition or entry removal). As a safety mechanism, we also keep the last 10 best performing models and train them with a fixed user vector structure and a fixed hyper-parameter set. These models are evaluated with the others at the end of each tuning cycle and can be chosen as the best model. This allows us to "revert" the changes being made to the user vector structure if they do not prove themselves over time (e.g., they cause their models to diverge[5]).

B. Entry Removal Component

The entry removal component is responsible for identifying entries that potentially can be removed from the model. To identify these entries, we are using heuristics to evaluate the model's performance operating without each of its entries. The pCTR of an event given user u and an ad a is defined in Section II as $pCTR(u,a) = \sigma(b+\nu_u^T\nu_a)$, where $\nu_u, \nu_a \in \mathbb{R}^{\mathrm{D}}$ denote the user and ad latent factor vectors of the event. Based on this equation we define

$$pCTR_{-i}(u,a) = \sigma(b + \nu_u^T\nu_a - \nu_{u_i} \cdot \nu_{a_i})$$

as the pCTR of the event without the ith entry of vectors ν_u and ν_a. Respectively, we define the loss function without the ith entry as

$$\mathcal{L}_{-i}(u,a,y) = -(1-y)\log\left(1 - pCTR_{-i}(u,a)\right) - y\log pCTR_{-i}(u,a) + \frac{\lambda}{2}\sum_{\theta\in\Theta}\theta^2 \ ,$$

We can now define the accumulative loss without the ith entry over the entire training data set $\mathcal{T}$ as

$$\mathcal{L}_{total_{-i}} = \sum_{(u,a,y)\in\mathcal{T}} \mathcal{L}_{-i}(u,a,y) \ ,$$

This value represents the loss of a hypothetical model that does not contain the ith entry of the vectors, and in which the block of the features that correspond to this entry is shorten. At the end of each tuning cycle the entries for which $\mathcal{L}_{total_{-i}}$ is the lowest are considered for removal.

C. Entry Generation Component

Deciding which feature the model needs to expand is a more complicated task than deciding which entry to remove. It is not clear how to evaluate in advance the outcomes of a possible change. Moreover, even if it is clear which feature to expand, what should be the initial values of the new entries? If the algorithm initializes the entries with small random values, it would take a lot of time for them to converge and become beneficial. To solve these two problems we designed the *Greenhouse* - an independent unit that trains alongside the model and "grows" sets of potential extra entries - one set for each block, i.e., one for each feature and for each pair of features, a total of $\left(\binom{F}{2} + F\right)$ "incubated" entry sets. Each such entry set includes one entry per each feature value and each ad feature[6]. An illustration of the Greenhouse for a toy model example, is given in Figure 4.

Each entry set in the Greenhouse is trained as if it's part of the model, and its potential contribution to the model performance is measured. For each Greenhouse entry set j we define

$$pCTR_{+j}(u,a) = \sigma(b + \nu_u^T\nu_a + \nu_{u_j} \cdot \nu_{a_j})$$

as the pCTR of the event with the Greenhouse jth entry. Respectively, we define the loss function with the Greenhouse jth entry as

$$\mathcal{L}_{+j}(u,a,y) = -(1-y)\log\left(1 - pCTR_{+j}(u,a)\right) - y\log pCTR_{+j}(u,a) + \frac{\lambda}{2}\sum_{\theta\in\Theta}\theta^2 \ ,$$

[5]We declare model divergence if the absolute value of one of its vectors' entries exceeds a predefined threshold.

[6]Assuming all block are of length k the Greenhouse actually increases the model size by a factor of $(k+1)/k$.

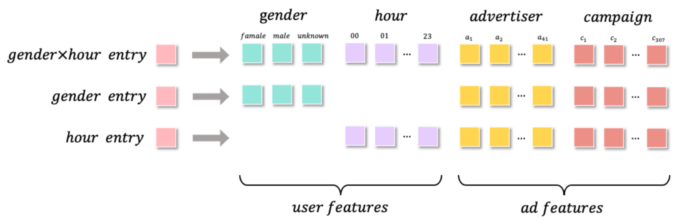


Fig. 4: Example of the Greenhouse entries for a toy model with 2 user features - gender and hour, and 2 ad features - advertiser and campaign. Each entry extends one of the user vector blocks. In this case, since there are only two user features ($F = 2$), we have only 3 entries in the Greenhouse.

Since in the generation component we are not only analyzing but also training new potential entries, the gradient of $\mathcal{L}_{+j}$ is being used for training the jth entry using SGD as described in Section II. As before, we also define the accumulative loss with the Greenhouse jth entry over the entire training data set $\mathcal{T}$ as

$$\mathcal{L}_{total_{+j}} = \sum_{(u,a,y)\in\mathcal{T}} \mathcal{L}_{+j}(u, a, y) \ ,$$

This value represents the loss of a hypothetical model that includes the jth entry set of the Greenhouse, and in which the block of the features that corresponds to this entry set is expanded. At the end of each tuning cycle the entry set for which $\mathcal{L}_{total_{+j}}$ is the lowest is considered to be added to the model. In case one of the Greenhouse entry set was added to the model, the latent vectors of all of the values of the corresponding features are expanded, and their additional values are copied from the Greenhouse. Then, the algorithm initializes the Greenhouse entry parameters, and "grows" it again from "scratch" for future use. Intuitively, the Greenhouse performs as an oracle, suggesting new mature values that are guaranteed to contribute to the model and improve its prediction accuracy.

D. Generating DLFM models

While the entry removal and entry generation components are being updated during each model training cycle, at each hyper-parameter tuning cycle three potential DLFM model types are considered:

- Removal model - a model with at least one less entry.
- Addition model - a model with at least one additional entry.
- Replacement model - a model that extends the size of one or more features at the expense of others.

Given a system parameter N, which dictates the maximal number of allowed simultaneous changes, we create $3N$ hypothetical models for evaluation, N of each type. (a) Removal model $\ell \in [1, N]$ that would remove the ℓ worst model entries, as predicted by the entry removal component; (b) Addition model $\ell \in [1, N]$ that would add the ℓ best Greenhouse entries, as predicted by the entry generation component; and (c) Replacement model $\ell \in [1, N]$ that would do both - replace ℓ old entries with ℓ new ones - as predicted by the entry removal and entry generation components. It is worth mentioning, that the length of the user vector is always bounded from above by its original uniform length D. Hence, an Addition model that suggests a user vector larger than D would be excluded from the list of potential models.

E. Resources

There are two aspects to consider regarding DLFM influence on system resources - the backend additional resources during training, and the frontend (or serving system) resources during serving. As mentioned earlier, we are less sensitive to resources surge during training, and highly sensitive to resources surge in serving time. However, since our serving system is oblivious to the DLFM algorithm any additional resources are needed for backend model training purposes only.

a) Training Resources: For brevity and readability reasons, in the following discussion, we assume that all feature blocks are of k dimensions. For each training cycle, the entry removal component performs $O(D) = O\big(\binom{F}{2} + F\big) \cdot k)$ additional operations in order to analyse the contribution of each of the user vector entries. However, this calculation can be performed in parallel to the regular model training, and therefore will not increase latency. The entry generation component requires both additional memory and processing time. Since the entry generation unit holds an entry set per each feature and each feature pair, the number of entry sets in the component is $O(\binom{F}{2} + F)$, which is $1/k$ of the original uniform model size. The number of additional operations is also $O(\binom{F}{2} + F)$ as we train and analyze each of the entries set per event[7].

Note that the algorithm components increase the memory and processing time on one hand, however, reduction of the model size decreases those resources on the other. Therefore, it is hard to evaluate the additional resources as it highly depends on the features the algorithm chooses to expand - for example, removing five entries of feature that has 3 values each, and adding 1 entry to a feature that has 100 values, eventually increases the memory size and vice versa.

b) Serving: According to Section II-C, the resulting model used by the serving system contains only the user feature vectors and ad features vectors. Let $|V|$ be the total number of vectors in the model, then the size of the model is given by $O(D \cdot |V|)$. After applying DLFM, the new length of the model is $\tilde{D} \leq D$, and therefore the memory consumption would be reduced by $\tilde{D}/D$. From processing resources perspective, for each auction (one for each incoming impression), a vector of size D would be constructed for the user and for each of the eligible ads. Then, in order to rank the ads, each of the ad vectors is multiplied (i.e., dot product) by the user vector. Each of these tasks requires $O(D)$ operations. Therefore, the reduction of the model's vector sizes to $\tilde{D}$ would reduce the processing resources by $\tilde{D}/D$ as well.

[7]The exact number of additional operations and memory depends on the number of values of the user and ad features.

VI. Performance Evaluation

In this section we report the offline and online performance of a DLFM enhanced OFFSET model. For both cases we describe the setting, define the performance metrics, and present the results. Since proprietary logged data is used for evaluating our model, it is obvious that reproducing the results by others is impossible. This caveat is common in papers describing commercial systems and we hope it does not undermine the overall contribution of this work.

A. *Offline Evaluation*

a) Setup: To evaluate offline performance we train two OFFSET models, one using DLFM to dynamically optimize the user vector structure as described in Section V and the other with a uniform user vector structure, serving as **baseline**. We train both models from “scratch” where all model parameters are randomly initialized, over several months of Gemini native logged data, which include many billions of impressions.

We use both sAUC and LogLoss metrics (defined next), to measure offline performance, where each impression is used for training the system after being applied to the performance metrics. OFFSET hyper-parameters, such as SGD step size and regularization coefficient, are determined automatically by the adaptive online tuning mechanism [2].

b) Performance metrics:
Area-under ROC curve (AUC) The AUC specifies the probability that, given two random events (one positive and one negative, e.g., click and skip), their predicted pairwise ranking is correct [14].
Stratified AUC (sAUC) The weighted average (by number of positive event, e.g., number of clicks) of the AUC of each Yahoo Gemini section[8]. This metric is used since different sections have different prior CTR bias and therefore even using the section feature alone turns out as sufficient for achieving high AUC values.
Logistic loss (LogLoss)

$$\sum_{(u,a,y)\in\mathcal{T}} -y\ln pCTR(u,a) - (1-y)\ln\left(1-pCTR(u,a)\right),$$

where $\mathcal{T}$ is a training set and $y \in \{0,1\}$ is the positive event indicator (e.g., click or skip). We note that the LogLoss metric is used to optimize OFFSET model parameters and its algorithm hyper-parameters.

c) Results: The daily LogLoss and sAUC lifts of the DLFM model over the baseline are plotted vs. time in Figure 5 , over a period of 4 months. On average, over the last 2 weeks, the DLFM model showed 0.38% LogLoss lift[9] and 0.35% sAuc lift[10] over the baseline. Since the evaluation is done over billions of impressions, the results are surely statistically significant.

[8]A section is a group of similar web pages (or sites) on Yahoo O&O and third parties’ properties, such as Yahoo! desktop Sports pages, or a specific mobile app such as Yahoo! weather for Android Smartphones.

[9]Since lower-is-better with the LogLoss metric, the lift is given by $(1 - \text{LogLoss}_{\text{DLFM}}/\text{LogLoss}_{\text{baseline}})\cdot 100$.

[10]Since higher-is-better with the sAUC metric, the lift is given by $(\text{sAUC}_{\text{DLFM}}/\text{sAUC}_{\text{baseline}} - 1)\cdot 100$.

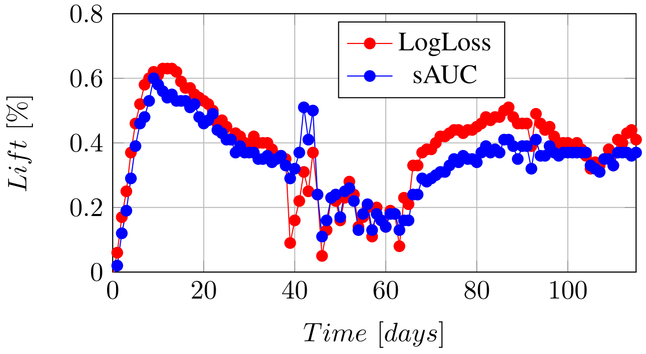


Fig. 5: Offline LogLoss and stratified AUC (sAUC) lifts of DLFM model over the baseline model vs. time.

B. *Online Evaluation*

a) Setup: To evaluate the online performance that determines whether the DLFM algorithm is pushed into the production system, we launched an online bucket serving a portion of Yahoo Gemini native traffic and measured the revenue lift in terms of the *average cost per thousand impressions* (CPM) with respect to the production model. It is noted that since bucket sizes are practically equal after proper normalization, CPM lift is actually revenue lift.

b) Results: The daily CPM lifts of the DLFM bucket when compared to the baseline bucket are presented in Figure 6. On average the online DLFM bucket showed 1.46% CPM lift and 2.15% CTR lift[11]. Such a CPM (or revenue) lift translates to many millions of USD in yearly revenue once the solution is deployed to all traffic. CTR improvement is impressive and stems from the improved model accuracy demonstrated by the offline evaluation. As with the offline results, the results here are also statistically significant since evaluation is done over many billions of impressions.

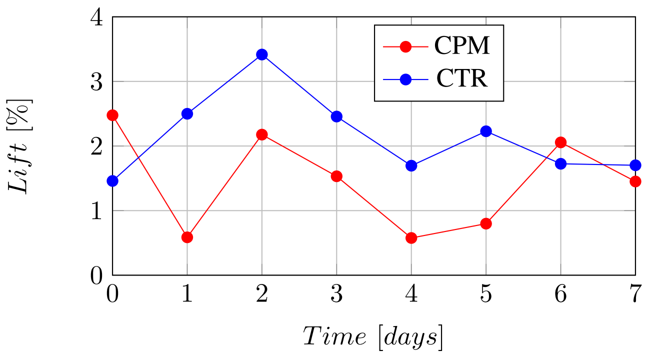


Fig. 6: Online CPM and CTR lifts of DLFM bucket over the production bucket vs. time.

[11]Since higher-is-better with CPM and CTR metrics, the lifts are calculated as the sAUC lift.

VII. Impact

Since integrated into production, DLFM has reduced the total size of the model by 25%. Starting with a uniform vector allocation, where each block is of size k and a total vector size of several hundred entries, the algorithm has continuously removed, added, and replaced the user vector entries optimizing the performance with each tuning cycle. In Table I we present a snapshot of the the user vector allocation taken from the production model several months after deploying DLFM. The model contains several types of features (a) straight forward features such as age, gender, and geo; (b) contextual features such as *page section*[12]; (c) user behavioural features that represents the user's previous interactions; and (d) user interest features that represents the user's interests on various domains (such as search and news). Each table entry $a_{i,j}$ represents the ratio between the original size of block (i, j), i.e., k, to its current length, where values smaller than 1, indicate that the block was shortened and values greater than one indicate the opposite. Examining the table it is clear that the user vector has gone through many changes reaching its current structure. Moreover, the *page section* feature holds the largest share of the user vector entries, which indicates its complexity (it has a few thousand values) and significance. This is not a surprise since other feature selection methods have already indicated its importance. It is also interesting to see that several of the blocks were reduced to zero, meaning that the interaction between the features did not contribute to the model prediction, nor improve its prediction accuracy.

VIII. Applying DLFM on FFM

FFM has been proven recently to be among the best ad click prediction algorithms [19][18]. This is due to their explicit modeling of the different interactions between the model features. In the sequel we adopt the notation of [19] and show that DLFM may be applied to FM-like algorithms such as FFM. As mentioned before, FFM does not treat the ad side differently, and its score is a sum of multiplications of feature pairs vectors. Hence, DLFM can optimize FFM by applying the entry removal and entry generation components on each pair. In the sequel we provide a formal description of this notion.

Given a data set with n feature values, and an event (x, y) where x is an n-dimensional vector indicates the event feature values, and y is the label, the pCTR is given by $\mathrm{pCTR}(x) = \sigma(\phi_{FFM}(w, x)) \in [0, 1]$, and

$$\phi_{FFM}(w, x) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} (w_{j_1,f_2} \cdot w_{j_2,f_1}) x_{j_1} \cdot x_{j_2} \ , \quad (2)$$

where j_1 represents a value of feature f_1, and j_2 represents a value of feature f_2. In addition, w_{j_1,f_2} is one of the latent vectors of feature f_1 that interacts with feature f_2, and w_{j_2,f_1} is one of the latent vectors of feature f_2 that interacts with feature f_1. For example, consider a model with 3 features - gender (G), hour (H), and advertiser (A). In this case $\phi_{FFM}(w, x)$ may be

$$w_{female,H} \cdot w_{03,G} + w_{female,A} \cdot w_{Nike,G} + w_{03,A} \cdot w_{Nike,H} \ .$$

To avoid using feature values, we use $w(x)_{f_{i_1}}$ to denote the specific value of feature f_{i_1} which was used during the event (x, y). Using this notation we can rewrite (2)

$$\phi_{FFM}(w, x) = \sum_{i_1=1}^{F} \sum_{i_2=i_1+1}^{F} w(x)_{f_{i_1},f_{i_2}} \cdot w(x)_{f_{i_2},f_{i_1}} \ , \quad (3)$$

where F is the number of features (or fields - adhering to the notation of [19]). Going back to our example, $\phi_{FFM}(w, x)$ would become

$$w(x)_{G,H} \cdot w(x)_{H,G} + w(x)_{G,A} \cdot w(x)_{A,G} + w(x)_{G,A} \cdot w(x)_{Nike,H} \ .$$

Using the same terms we used for describing Offset, we interpret the expression $w(x)_{f_{i_1},f_{i_2}} \cdot w(x)_{f_{i_2},f_{i_1}}$ as the "block" of features f_{i_1}, f_{i_2} - a multiplication of two latent vectors of size k_{i_1,i_2}, (at the beginning $k_{i_1,i_2} \equiv k, \forall i_1, i_2$). We are now ready to define $\phi_{FFM}(w, x)$ without the lth element of block $f_{i_1} \times f_{i_2}$ by

$$\begin{aligned} &\phi_{FFM}(w, x)_{-(i_1,i_2,l)} = \\ &\qquad \phi_{FFM}(w, x) - w(x)_{f_{i_1},f_{i_2}\,l} \cdot w(x)_{f_{i_2},f_{i_1}\,l} \ , \end{aligned} \quad (4)$$

where $l \in [1, k_{i_1,i_2}]$, and $, k_{i_1,i_2}$ is the current length of block $f_{i_1} \times f_{i_2}$. Using (4) we can calculate and aggregate the loss of the model without each of its elements - entry removal component. The entry generation component can be implemented in a similar way - we can train an extension $\tilde{w}(x)_{f_{i_1},f_{i_2}} \cdot \tilde{w}(x)_{f_{i_2},f_{i_1}}$ for each block $f_{i_1} \times f_{i_2}$, and evaluate the performance of the extension using

$$\begin{aligned} &\phi_{FFM}(w, x)_{+(i_1,i_2)} = \\ &\qquad \phi_{FFM}(w, x) + \tilde{w}(x)_{f_{i_1},f_{i_2}} \cdot \tilde{w}(x)_{f_{i_2},f_{i_1}} \ , \end{aligned}$$

Applying these two components we can optimize any FFM by DLFM as we optimized Offset.

IX. Educated Model Initialization

One of the main properties of an iterative machine learning algorithm is its convergence rate. In our work cycle it also dictates the time between a new idea to a functional production model. However, as new features are added to the model, the model size is growing as well as the convergence time. Recently, it has been taking several weeks worth of logged data for a model to converge, which translates to several days in real time. Testing new features and new algorithm improvements became a slow process, and the production throughput decreases. To overcome this drawback and to accelerate the algorithm convergence rate, we are taking advantage of DLFM abilities. As a new model is usually an improved versions of the previous one, we can exploit the data kept in the last model to initialize the new model, instead of initializing it with random values and start training from "scratch". In this scenario, we first inherit the previous model user vector

[12] A unique page (or site) identifier on Yahoo O&O and third parties' properties with tens of thousands of distinct values.

Feature	user interests #1	user interests #2	user behaviour #1	user behaviour #2	user behaviour #3	user interests #3	user interests #4	user interests #5	hour	page section	publisher	user interests #6	geo	user behaviour #4	age	gender	user interests #7	user interests #8
user interests #1	0.14	1.00	1.00	1.00	0.86	1.00	1.00	0.29	0.57	0.57	0.71	0.29	0.29	1.00	0.57	1.00	1.00	1.00
user interests #2		0.14	1.00	0.86	1.00	1.00	0.57	0.14	0.00	1.43	0.71	1.00	0.43	0.57	0.57	0.57	0.43	1.00
user behaviour #1			0.14	1.71	3.29	1.00	1.14	0.71	0.71	5.43	1.29	1.71	0.43	1.29	0.86	1.00	1.29	1.86
user behaviour #2				0.14	0.86	0.86	1.00	0.71	0.00	1.00	0.14	1.14	0.00	0.86	0.43	0.43	0.71	1.00
user behaviour #3					0.14	0.86	1.14	0.14	0.00	4.14	1.00	1.14	0.00	0.57	0.43	0.43	0.00	0.71
user interests #3						0.71	0.71	0.86	0.86	1.00	0.86	1.00	0.86	0.86	0.86	0.86	0.86	0.86
user interests #4							0.00	0.43	0.00	1.57	0.71	0.86	0.43	0.71	0.29	0.29	0.29	0.71
user interests #5								0.14	0.14	1.14	0.14	1.14	0.00	0.14	0.14	0.14	0.86	0.71
hour									0.00	1.00	0.00	0.43	0.29	0.29	0.14	0.00	0.29	0.00
page section										0.14	0.57	6.00	3.00	1.57	1.57	1.57	1.43	1.57
publisher											0.00	0.86	0.57	0.29	0.57	0.00	0.14	0.57
user interests #6												0.14	0.71	1.00	1.00	0.71	1.00	1.00
state													0.14	0.00	0.00	0.00	0.00	0.14
user behaviour #4														0.00	0.29	0.57	0.43	0.57
age															0.14	0.29	0.29	0.71
gender																0.14	0.86	0.86
user interests #7																	0.14	1.43
user interests #8																		0.14

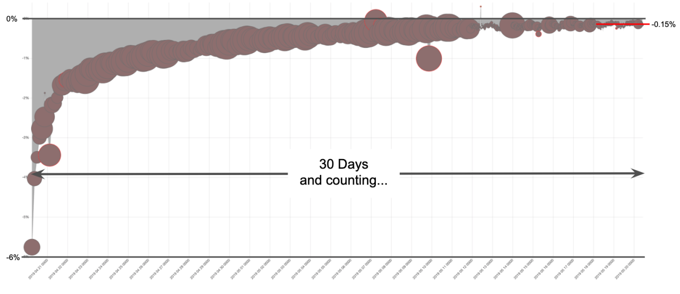
TABLE I: User vector normalized feature-allocation, several months after deploying DLFM into production, where darker colors indicate higher values.

allocation - keeping the proportions between the two models' mutual features. Then, the new model's latent vectors are initialized with the values of the previous one. In case the new block sizes are smaller than the original block size, the entry removal component is used to determine the most valuable entries of each block that should be copied to the new model. We note that while incremental training is standard in recommendation systems (see [22] for MF-CF models, and [28] for deep models), our EMI enables adding (or removing) features to (from) the new incremental model, by using the DLFM mechanisms (see Sections V-C and V-B).

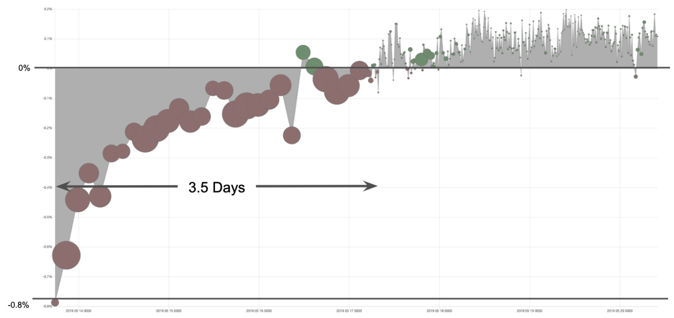
a) Results: To test the educated model initialization (EMI) mechanism, we used two "mature" models, M_{11} having 11 user features, and M_{12} having an additional user feature. Then, we created two new models $M_{12-scratch}$ and M_{12-EMI} both having 12 user features as model M_{12}. $M_{12-scratch}$ was trained from "scratch", while M_{12-EMI} was initialized based on model M_{11} using EMI. The convergence time of the models was evaluated by comparing their offline results to the offline results of model M_{12} - the baseline. In Figure 7 we present the offline LogLoss lifts of the two models over M_{12}. After 30 days, the model starting from "scratch" $M_{12-scratch}$ still does not meet model M_{12}'s results, while the model initialized with EMI M_{12-EMI} not only meets, but exceeds the result in less then 4 days worth of data. A summary of the experiment results can be found in Table II. We ran additional experiments, using different settings - increasing/decreasing the model size, adding/removing user features, as well as the combination of the above, and in all the experiments the EMI have reduced the training time by more than 90%.

X. Conclusions and Future work

Offset is a user-less model, where a user is represented by its features ,i.e., the user LFV is a combination of the



(a) A model training from "scratch"



(b) A model initialized with EMI

Fig. 7: Offline LogLoss lifts of the two models over the baseline. The brown circles indicate negative lifts, while the green circles indicates positive lifts. The circles sizes are proportional to the number of impressions used to evaluate the presented lifts.

user features LFV's. Clearly, different user features possess different informative value, therefore an accurate and delicate construction of the user vector may be highly beneficial. In this paper we described the implementation of such a dynamic approach named DLFM, where during each training cycle

Feature	from "scratch"	using EMI
convergence nominal time	45 days	4 days
convergence real time	more than 2 weeks	2 days
initial LogLoss over baseline	-6%	-0.8%

TABLE II: A comparison between a model training from "scratch" (model parameters are initialized by random values) and a model initialized with the *educated model initialization* (EMI) algorithm.

we train and evaluate several models, each using a slightly different vector allocation. The new user vector allocation is generated using metrics we design in order to detect "weak" entries we would like to remove, and "deprived" features we would like to enhance. Offline and online evaluations demonstrated the benefit of DLFM over the uniform user vector structure that has been used so far. In addition, the user vector structure, which every model outputs after each training cycle, can be of great interest by itself, as it reveals the balance between user features, and can play a significant role in the evaluation of existing and potential user features. Moreover, leveraging the DLFM functionality for developing a new initialization mechanism yielded a great reduction in training time. In particular, the EMI reduces the training time by more than 90%, providing a quick and easy way to apply changes to the production model, and hence significantly improving the productivity of the research team supporting the modeling processes.

REFERENCES

[1] M. Aharon, N. Aizenberg, E. Bortnikov, R. Lempel, R. Adadi, T. Benyamini, L. Levin, R. Roth, and O. Serfaty. Off-set: one-pass factorization of feature sets for online recommendation in persistent cold start settings. In Proc. RecSys'2013.

[2] M. Aharon, A. Kagian, and O. Somekh. Adaptive online hyper-parameters tuning for ad event-prediction models. In Proc. WWW'2017 Companion.

[3] M. Aharon, Y. Kaplan, R. Levy, O. Somekh, A. Blanc, N. Eshel, A. Shahar, A. Singer, and A. Zlotnik. Soft frequency capping for improved ad click prediction in yahoo gemini native. In Proc. CIKM'2019.

[4] N. Aizenberg, Y. Koren, and O. Somekh. Build your own music recommender by modeling internet radio streams. In Proc. WWW'2012.

[5] O. Anava, S. Golan, N. Golbandi, Z. Karnin, R. Lempel, O. Rokhlenko, and O. Somekh. Budget-constrained item cold-start handling in collaborative filtering recommenders via optimal design. In Proc. WWW'2015.

[6] M. Arian, E. Abutbul, M. Aharon, Y. Koren, O. Somekh, and R. Stram. Feature enhancement via user similarities networks for improved click prediction in yahoo gemini native. In Proc. CIKM'2019.

[7] R. M Bell and Y. Koren. Lessons from the netflix prize challenge. Acm SIGKDD Explorations Newsletter, 9(2):75–79, 2007.

[8] W. Cheng, Y. Shen, and L. Huang. Differentiable neural input search for recommender systems. arXiv preprint arXiv:2006.04466, 2020.

[9] D. Cohen, M. Aharon, Y. Koren, O. Somekh, and R. Nissim. Expediting exploration by attribute-to-feature mapping for cold-start recommendations. In Proc. RecSys'2017.

[10] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. Communications of the ACM, 51(1):107–113, 2008.

[11] D. Drachsler-Cohen, O. Somekh, S. Golan, M. Aharon, O. Anava, and N. Avigdor-Elgrabli. ExcUseMe: asking users to help in item cold-start recommendations. Proc. RecSys'2015.

[12] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. The Journal of Machine Learning Research, pages 2121–2159, 2011.

[13] B. Edelman, M. Ostrovsky, and M. Schwarz. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. American economic review, 97(1):242–259, 2007.

[14] T. Fawcett. An introduction to ROC analysis. Pattern recognition letters, 27(8):861–874, 2006.

[15] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.S. Chua. Neural collaborative filtering. In Proc. WWW'2017.

[16] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, et al. Practical lessons from predicting clicks on ads at facebook. In Proc. of the Eighth International Workshop on Data Mining for Online Advertising, 2014.

[17] M. R. Joglekar, C. Li, M. Chen, T. Xu, X. Wang, J.K. Adams, P. Khaitan, J. Liu, and Q.V. Le. Neural input search for large scale recommendation models. In Proc. KDD'2020.

[18] Y. Juan, D. Lefortier, and O. Chapelle. Field-aware factorization machines in a real-world online advertising system. In Proc. WWW'2017 Companion.

[19] Y.Juan, Y. Zhuang, W.S. Chin, and C.J. Lin. Field-aware factorization machines for ctr prediction. In Proc. RecSys'2016.

[20] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. Computer, 42(8):30–37, 2009.

[21] C. Li, Y. Lu, Q. Mei, D. Wang, and S. Pandey. Click-through prediction for advertising in twitter timeline. In Proc. KDD'2015.

[22] X. Luo, Y. Xia, and Q. Zhu. Incremental collaborative filtering recommender based on regularized matrix factorization. Knowledge-Based Systems, 27:271–280, 2012.

[23] B. McMahan. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In Proc. of the International Conference on Artificial Intelligence and Statistics, 2011.

[24] H.B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, et al. Ad click prediction: a view from the trenches. In Proc. KDD'2013.

[25] J. Pan, J. Xu, A.L. Ruiz, W. Zhao, S. Pan, Y. Sun, and Q. Lu. Field-weighted factorization machines for click-through rate prediction in display advertising. In Proc. WWW'2018.

[26] S. Rendle. Factorization machines. In Proc. IEEE International Conference on Data Mining, 2010.

[27] Y.Sun, J. Pan, A. Zhang, and A. Flores. FM^2: Field-matrixed factorization machines for recommender systems. In Proc. WWW'2011.

[28] Y. Wang, H. Guo, R. Tang, Z. Liu, and X. He. A practical incremental method to train deep ctr models. arXiv preprint arXiv:2009.02147, 2020.

[29] X. Zhao, C. Wang, M. Chen, X. Zheng, X. Liu, and J. Tang. Autoemb: Automated embedding dimensionality search in streaming recommendations. arXiv preprint arXiv:2002.11252, 2020.