Online Meta-Learning for Model Update Aggregation in Federated Learning for Click-Through Rate Prediction

Xianghang Liu
xianghang.liu@huawei.com
Huawei London Research Centre
London, UK

Bartłomiej Twardowski
bartlomiej.twardowski@huawei.com
Huawei Ireland Research Centre
Dublin, Ireland

Tri Kurniawan Wijaya
tri.kurniawan.wijaya@huawei.com
Huawei Ireland Research Centre
Dublin, Ireland

ABSTRACT

In Federated Learning (FL) of click-through rate (CTR) prediction, users' data is not shared for privacy protection. The learning is performed by training locally on client devices and communicating only model changes to the server. There are two main challenges: (i) the client heterogeneity, making FL algorithms that use the weighted averaging to aggregate model updates from the clients have slow progress and unsatisfactory learning results; and (ii) the difficulty of tuning the server learning rate with trial-and-error methodology due to the big computation time and resources needed for each experiment. To address these challenges, we propose a simple online meta-learning method to learn a strategy of aggregating the model updates, which adaptively weighs the importance of the clients based on their attributes and adjust the step sizes of the update. We perform extensive evaluations on public datasets. Our method significantly outperforms the state-of-the-art in both the speed of convergence and the quality of the final learning results.

KEYWORDS

federated learning, meta-learning, click-through rate prediction

1 INTRODUCTION

Federated Learning (FL) [19] is a machine learning paradigm where user data is not shared for privacy protection. The model is trained via arranged communications between the server and many system clients. In each communication round, the selected clients run local optimization algorithms for multiple steps and send the local model updates back to the server. The server aggregates the received local updates and then applies a gradient-based server optimizer, taking the aggregated update as the gradient input. In recent years, deep neural network based models have proven to be successful in click-through rate (CTR) prediction [24, 28]. Based on user, item and context features, they predict CTR without tedious crafting of input features or higher-order interactions between them.

In this work, we focus on FL application for CTR prediction models. Specifically, we propose an algorithm to learn the aggregation strategy of the local model updates in FL CTR models, where the aggregation strategy is adjusted according to the feedback during the learning process. There are two main motivations for our work described below.

Weighting of local updates and client heterogeneity. Weighted averaging is a commonly-used strategy for local update aggregation in FL. When all clients are homogeneous, taking the average over local updates gives good results [16]. In practice, however, the clients are often heterogeneous. This client heterogeneity is a major challenge in FL for CTR prediction models and recommender systems (RecSys) in general, because it causes inconsistencies between the client and the global objectives; it slows down the training progress and often leads to suboptimal final results. This issue is also known as *client drift* [14].

To obtain faster learning progress and better final solutions, a good client weighting strategy for update aggregation should be (1) aware of the client heterogeneity: the importance weight of a client should depend on its attributes; (2) adaptive: at different phases of the learning process, the strategy may focus on different sets of clients; (3) parameter-wise: not all parameters behave in the same way during a training process, the strategy should support different learning dynamics for them. For example, in FL of deep CTR models, there are parameters for sparse feature embeddings that are only updated by a small set of clients. On the contrary, dense fully-connected layers are updated by all clients in each round. The optimal weighting strategies for these two sets of parameters are likely to be different.

Server learning rate tuning. Learning rate is arguably one of the most important hyper-parameters for gradient-based learning algorithms in centralised machine learning [2]. Likewise, server learning rate, used by the server optimizer to control the step-size of the aggregated update, has an important role in FL [3, 22, 26]. Similar to the case of client weighting, it is desirable that the server learning rate schedule is *adaptive* and *parameter-wise*.

In this paper, we propose an online meta-learning method to learn a strategy to aggregate the local model updates in federated learning of CTR prediction models, which has the advantages of being aware of client heterogeneity, adaptive to learning process, and parameter-wise. Our contributions include:

- Meta-learning formulation of the aggregation problem in FL to compute the gradients of meta-parameters without the need of an extra dataset.
- A new online meta-learning method (MetaUA) that jointly learns the target CTR prediction model and the meta-aggregation model.
- Extensive experimental evaluations, including: comparison to other methods on four publicly available datasets, ablation study, analysis of clients' attributes selection, and method robustness to varying percentages of participating clients and meta-learning rate. MetaUA outperforms state-of-the-art methods in both the speed of convergence and the final results in few different settings.

2 RELATED WORK

The first FL algorithm FedAvg was proposed [19]. Since then, there have been many FL works dedicated to RecSys [8, 18, 21]. While

Conference'17, July 2017, Washington, DC, USA
2022. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

most of these works focus on learning generalized matrix factorization models [10], our work is agnostic to model architectures.

Client heterogeneity is recognized as one of the key challenges in FL [13, 26]. A few approaches have been proposed to address the client drift problem caused by heterogeneity: FedNova [27] normalizes the local updates before averaging to eliminate the objective inconsistency; SCAFFOLD [14] uses control variates to correct the drift in the local updates; FedProx [15] adds a proximal term in the local objectives to stabilize the learning.

On the aggregation of local model updates, FedAvg uses a weighted average by the number of samples received from the clients. Inspired by the attention mechanism [11, 21, 25] use the Euclidean distance between local and global models as the importance weights of clients. In a similar spirit, [5] uses the divergence of local models to the global model as the importance of clients, but the importance is used in client sampling instead of aggregating updates. A similar work in client sampling with priority is discussed in [9], where a local loss value is used as client importance. On the adaption and scheduling of the learning rates in FL: [22] proposes adaptive optimization methods, such as Adagrad, Adam and YOGI, at the server side; [3] proposes a heuristic rule to adjust server and local learning rates based on loss values. All these existing works on local model update aggregation and learning rate adaption are all based on manually engineered strategies, which often require a great amount of efforts in both designing and experimentation.

Our proposed algorithm belongs to the class of online meta-learning [7]. In non-FL settings, ideas have be explored in the learning rate adaption and training example weighting: gradient-based methods to learn the learning rate are proposed in [1]. It is extended by [20] from the next step objective to a long-horizon one. On training example weighting, Meta-weight-net [23] learns the weight of training examples to address the bias and noise in the training data. In this thread, [32] proposes a method to learn without the need of a reward dataset and [31] speeds up the meta-gradient computation with a faster layer-wise approximation. In FL, [4] applies MAML [6] and Meta-SGD [17] to learn the initial model weights for local training on each client. However, the idea of using meta-learning to adapt the learning rate or client weights is yet to be explored in FL.

3 PROBLEM FORMULATION

CTR prediction can be formulated as a supervised learning problem: given the input feature and click/no click label pairs $\{x, y\}$, the goal is to learn a function $h(x; w)$ parameterized by w, which minimizes the loss between the prediction and the ground truth label $\ell(h(x; w), y)$. At communication round t of FL, a set of clients S^t are selected for a participation, to whom a copy of the global model w is distributed. Each client $k \in S^t$ trains the model using their local data D_k to generate a local model w_k^t: $w_k^t = \text{opt-local}(w^t; D_k)$. opt-local$(\cdot)$ is the local optimization of $\ell(\cdot)$ initiated using weights w^t on dataset D_k, which is often several epochs of SGD. The clients send back to the server the model updates $\Delta W^t := \{\Delta w_k^t, k \in S^t\}$, where $\Delta w_k^t := w_k^t - w^t$. These local updates are then aggregated at the server to get the update on the global model $\Delta w^t = \text{agg}(\Delta W^t, Z^t)$. where $Z^t := \{z_i, i \in S\}$ are the non-sensitive client attributes and agg$(\cdot)$ is the aggregation function which takes the local updates and the client attributes as input and outputs the aggregated update. The aggregated update Δw^t is then applied to the global model w^t by performing one step of server optimization, i.e. opt-server$(\cdot)$, to get the new global model $w^{t+1} = \text{opt-server}(w^t, \Delta w^t)$.

Many existing FL algorithms can be formulated under this framework. A few examples are:

► FedAvg [19] $\text{agg}(\Delta W^t, Z^t) = \frac{1}{n^t}\sum_{k\in S} n_k^t \cdot \Delta w_k^t$, and $\text{opt-server}(w, \Delta w) = w^t + \Delta w^t$, where $z_k^t = n_k$, for $k \in S^t$, n_k is the number of samples for client k, and $n^t = \sum_{k\in S^t} n_k$.

► FedNova[27] $\text{agg}(\Delta W^t, Z^t) = \big(\sum_{k\in S^t} \tau_k \cdot \frac{n_k}{n^t}\big) \cdot \sum_{k\in S} \frac{n_k}{n^t} \cdot \frac{1}{\tau_k} \cdot \Delta w_k^t$, where $z_k^t = [n_k, \tau_k]$, τ_k is the number of local gradient descent steps on client k. The function opt-server$(\cdot)$ is the same as that of FedAvg.

► FedAdam and FedAdagrad[22] These two methods have the same aggregation function as FedAvg. The server optimization method maintains first and second order momentum, m^t and M^t: $m^t = \beta_1 \cdot m^{t-1} + (1-\beta_1) \cdot \Delta w^t$, and $M^t = M^{t-1} + (\Delta w^t)^2$, for FedAdagrad, $M^t = \beta_2 \cdot M^{t-1} + (1-\beta_2) \cdot (\Delta w^t)^2$, for FedAdam, The optimization step is performed as: $w^{t+1} = w^t + \gamma_s \cdot \frac{m^t}{\sqrt{M^t}+\epsilon}$, where $\gamma_s, \beta_1, \beta_2$ and ϵ are all hyper-parameters. γ_s is generally known as server learning rate.

4 ONLINE META LEARNING FOR LOCAL UPDATE AGGREGATION

To have a parameter-wise aggregation model, we will first define a partition of the weight indices $\mathcal{P}$, s.t. $\bigcup_{A\in\mathcal{P}} A$ is all the weight indices and $A_1 \cap A_2 = \emptyset$, for any pair of $A_1, A_2 \in \mathcal{P}, A_1 \neq A_2$. The most common partition is layerwise, where each $A \in \mathcal{P}$ is the set of weight indices of a network layer. We will use $[\cdot]$ to denote the indexing operation on the weights.

Meta model and its parameters Our learnable aggregation model agg$(\cdot)$ is defined as:

$$\Delta w^t[A] = \text{agg}(\Delta W^t[A], Z^t[A]; \theta[A]) := \theta_s[A] \cdot \sum_{k\in S^t} \alpha_k[A] \cdot \Delta w_k[A],$$
$$\text{where } \alpha_k[A] = \text{softmax}(f_\alpha(z_k^t[A]; \theta_\alpha[A])), \forall A \in \mathcal{P}.$$

$\theta = \{\theta_s, \theta_\alpha\}$ are the meta parameters to be learned. Each element of θ_s is between 0 and 1. It is the scaling on the server learning rate for step-size adaption during the learning process. $f_\alpha(z; \theta_\alpha)$ is the client weighting function parameterized by θ_α, which gives the importance of the clients based on their non-sensitive attribute(s) z. f_α should be differentiable in θ_α. The selection of f_α and z_k^t will be detailed in Sections 4 and 6.

The output of the meta model is the aggregated update $\Delta w^t[A], A \in \mathcal{P}$, which is applied to get the new weight:

$w^{t+1}[A] = \text{opt-server}(w^t[A], \Delta w^t[A]), \forall A \in \mathcal{P}$. This formulation allows the flexibility of aggregating different subsets of weight updates differently. We will omit this indexing in the rest part of the paper for readability.

Meta loss. Each local dataset D_k is first split into the *support* dataset $D_k^{(s)}$ for local training, and the *query* dataset $D_k^{(q)}$ for meta loss evaluation. The meta loss function is then defined in a *delayed* way, i.e. the loss for the meta parameters at communication round $t-1$, θ^{t-1} will be evaluated at the round t:

$$L^t(\theta^{t-1}) = \sum_{k\in S^t} L^t(\theta^{t-1}; D_k^{(q)}), \tag{1}$$

where each $L^t(\theta^{t-1}; D_k^{(q)}) := \sum_{\{x,y\}\in D_k^{(q)}} \ell\big(h(x; w^t); y\big)$, abbreviated as $L_k^t(\theta^{t-1})$. The motivation is that, at the current round t, we evaluate the quality of our aggregation strategy performed at the previous round $t-1$ and adjust it accordingly. One advantage of (1) that it is an unbiased estimation of the training loss of w^t from training examples on S^t. Another advantage is that both its evaluation and optimization is done *online*; this means that no extra datasets are needed nor are additional communication rounds required.

Meta optimization We use gradient descent to optimize the meta parameters. The computation of the meta gradient is one of the main contributions of this paper. This section gives the description on the computation of $\frac{\partial L^t}{\partial \theta^{t-1}}(\theta^{t-1})$. We will slightly abuse the notation by using θ^{t-1} for both the variable and its value, which can be distinguished by the context.

Similar to the meta loss, its gradient is also computed in a *delayed* way: the gradient of θ^{t-1} is computed at round t. To do this, we will need to store the model weight, local model updates and clients' attributes of the previous round at the server, so that we can assume that at round $t > 1$, we have $w^{t-1}, \Delta w_k^{t-1}$ and $z_k^{t-1}, \forall k \in S^{t-1}$ avaialble.

By the chain rule, we have

$$\frac{\partial L^t}{\partial \theta^{t-1}}(\theta^{t-1}) = \frac{\partial L^t}{\partial w^t}(w^t) \cdot \frac{\partial w^t}{\partial \theta^{t-1}}(\theta^{t-1}). \tag{2}$$

The first part of (2) RHS, denoted as g^t, is the gradient of L^t w.r.t w^t evaluated at w^t on the training data of the clients in S^t, i.e. $g^t = \sum_{k\in S^t} \underbrace{\frac{\partial L_k^t}{\partial w^t}(w^t)}_{g_k^t :=}$. Each g_k^t is evaluated locally on each client $k \in S^t$ as

$$g_k^t = \sum_{x,y\in D_k^{(q)}} \frac{\partial \ell}{\partial w^t}\big(f(x, w^t), y\big), \tag{3}$$

and then sent back to the server. The second part of (2) RHS is the Jacobian matrix $\partial w^t / \partial \theta^{t-1}$ evaluated at θ^{t-1}. The dependency of w^t on θ^{t-1} is through Δw^t as the composite of the two functions $\text{agg}(\cdot)$ and $\text{opt-server}(\cdot)$. Both of these two functions are differentiable, so we can also apply the chain rule: $\frac{\partial w^t}{\partial \theta^{t-1}} = \frac{\partial w^t}{\partial \Delta w^{t-1}} \cdot \frac{\partial \Delta w^{t-1}}{\partial \theta^{t-1}}$. In the implementation, there is no need to compute the full Jacobian matrices. Instead, since g^t is not a function, we move it into the second paritial derivative in Eq. (2), which becomes:

$$\frac{\partial L^t}{\partial \theta^{t-1}}(\theta^{t-1}) = \frac{\partial (g^t \cdot w^t)}{\partial \theta^{t-1}}(\theta^{t-1}). \tag{4}$$

The inner product $g^t \cdot w^t$ is a scalar-output linear function of w^t. Its gradient w.r.t θ^{t-1} can easily be computed through back propagation provided by most automatic differentiation packages. The procedure of gradient computation in MetaUA in two FL rounds is presented in Fig. 1.

MetaUA Algorithm The proposed algorithm of Online **Meta**-Learning for **U**pdate **A**ggregation (**MetaUA**) is given in Algorithm 1, where all computations follow previously described derivations. The key difference between it and the other FL algorithms is that, at each communication round, the meta parameters will be updated by a step of gradient descent, which are then used to aggregate local updates to form a new version of the server's model.

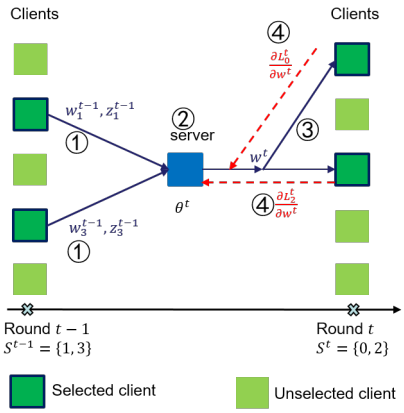

Figure 1: Overview of the gradient computation procedure in MetaUA: ① local model updates and attributes are sent from selected clients; ② local updates are aggregated using meta model; ③ new model is distributed to the selected clients; ④ server receives the gradients from the selected clients and computes the gradients for the meta-parameters.

Algorithm 1: Online Meta-Learning for Update Aggregation (**MetaUA**)

Input: Randomly initialized weight w^1
Output: w after the learning is complete
Data: An universe of clients I, for each client $k \in I$, an associated local dataset $D_k = D_k^{(s)} \cup D_k^{(q)}$

for *communication round* $t = 1, 2, \ldots, T$ **do**
  Randomly select a set of clients S^t
  for *each client* k *in* S^t **do**
    $\Delta w_k^t \leftarrow \text{opt-local}(w^t, D_k^{(s)}) - w^t$ `// local training`
    compute g_k^t with equation (3)
    set client attributes to z_k^t
  if $t > 1$ **then**
    `/* backward pass of update aggregation */`
    $g^t \leftarrow \sum_{k\in S^t} g_k^t$
    compute $\frac{\partial L^t}{\partial \theta^{t-1}}(\theta^{t-1})$ with equation (4)
    $\theta^t \leftarrow \theta^{t-1} - \gamma_{meta} \cdot \frac{\partial L^t}{\partial \theta^{t-1}}(\theta^{t-1})$
  `/* forward pass of update aggregation */`
  $\Delta w^t \leftarrow \text{agg}(\Delta W^t, Z^t; \theta^t)$
  $w^{t+1} \leftarrow \text{opt-server}(w^t, \Delta w^t)$

Server optimizer. Our algorithm is flexible in the selection of the server optimizer. We run experiments on SGD, FedAdagrad and FedAdam, in which FedAdagrad has the best results. We thus choose to use FedAdagrad in all the experiments in this paper.

Client weighting model. The main consideration in choosing f_α is that it will run for all the selected clients in both forward and backward pass, therefore, the use of a simple architecture to keep the additional computations small should be favored. In our experiments, we found that using a simple linear model for f_α is enough to get good results.

For client attributes, a good candidate should: 1) not reveal user privacy; 2) capture the client heterogeneity; 3) introduce little or no extra computation. Inspired by [23] and [9], we choose the local

loss $z_k^t = \frac{1}{n_k} \sum_{\{x,y\} \in D_k} \ell(f(x; w^t); y)$. Intuitively, the local loss measures how well the current model fits the client's local data. A higher local loss than others could be caused by either system or data heterogeneity. Some possible reasons are: the client has lagged behind others in learning because it has not actively participated in FL, or less local training steps have been performed on it because it has less powerful hardware, or the client has examples that the current model fits poorly, which could be examples novel to the model, ambiguous examples near the decision boundary or mis-labelled examples. We have also considered including: number of samples,norm of the weight updates, number of unique features, etc. Experimental results are provided in section 6.

γ_{meta} is the learning rate on the meta-parameters. We find that our method is generally robust to it. In our experiments, we fix it to be 0.1, which has reasonably good results under all the settings.

5 COST ANALYSIS

Compared to other FL methods there are some extra costs incurred from MetaUA at each training round.

Communication To compute g^t, we need to transfer the gradient of the local loss w.r.t the model parameter to the server for all the selected clients, i.e. $g_k^t, k \in S^t$. This will double the communication cost from the clients to the server. A few methods can be explored to reduce the communication: compressing the model gradient before sending or sub-sampling the clients in gradient evaluation.

Computation On the client side, the extra computation needed is for $g_k^t, k \in S^t$. This requires one additional pass over the local data. We consider the increase in computation to be marginal, as each selected client already have to perform several epochs of training in FL round. On the server side, an extra computation is needed for: the meta aggregation operator, the opt-server, the inner product in (4) and a backward pass on them for the gradient computation. The aggregation operator involves shallow networks to be applied as many times as the number of selected clients in an FL round, while other operations are just a constant number of simple operations. The impact of this extra computation cost on the server side is not significant, because the server is usually a high-performance cluster.

Storage Since our meta gradient evaluation is delayed, we need to store the model updates received from the selected clients for a single communication round. In practice, the user population is quite large, so additional memory resources have to be allocated for that.

6 EXPERIMENTS

We evaluated a proposed method for a click-through rate (CTR) prediction task in FL setting that can be used in real-world scenario. A series of experiments are designed and conducted to present methods properties.

Datasets In our experiments we use four well-know datasets: MovieLens-1M, Tmall, Yelp, and Amazon-Cds. Their main statistics are presented in Table 1. Input features can be related to: user, item or context of the event. Data of each user is taken as a client's partition in all federated learning experiments. The last 10% of client's data (based on a timestamp) is taken for a validation.

Implementation Details In all experiments we use DCNv2 [28] prediction model with binary cross entropy as a loss function and

Dataset	# Examples	# Features	Vocab Size	# Users	# Items	Density
MovieLens-1M	739012	7	13196	6040	3668	3.34
Tmall	1899378	4	44239	22284	17705	0.48
Yelp	530124	9	34462	22128	12232	0.20
Amazon-CDs	890824	3	31985	15592	16184	0.35

Table 1: Dataset statistics

		AUC			Logloss		
Dataset	Round	FedAvg	FedAdagrad	MetaUA	FedAvg	FedAdagrad	MetaUA
ML-1M	20	0.648	0.809	**0.814**	0.594	0.533	**0.500**
	50	0.673	0.842	**0.850**	0.592	0.499	**0.432**
	100	0.811	0.864	**0.867**	0.492	0.465	**0.413**
	150	0.811	0.870	**0.875**	0.490	0.452	**0.403**
	200	0.812	0.874	**0.880**	0.490	0.450	**0.393**
Tmall	20	0.704	0.775	**0.779**	0.338	0.308	**0.301**
	50	0.709	0.781	**0.786**	0.331	0.308	**0.301**
	100	0.711	0.786	**0.794**	0.326	0.308	**0.300**
	150	0.713	0.789	**0.802**	0.322	0.308	**0.293**
	200	0.715	0.792	**0.807**	0.319	0.308	**0.292**
Amzn	20	0.588	0.605	**0.640**	0.693	0.670	**0.662**
	50	0.599	0.668	**0.808**	0.693	0.670	**0.541**
	100	0.600	0.899	**0.919**	0.693	0.434	**0.335**
	150	0.601	**0.940**	0.937	0.693	0.320	**0.286**
	200	0.604	**0.951**	0.944	0.693	0.279	**0.269**
Yelp	20	0.579	**0.709**	0.708	0.692	0.653	**0.631**
	50	0.595	0.745	**0.758**	0.686	0.653	**0.596**
	100	0.603	0.762	**0.781**	0.682	0.652	**0.576**
	150	0.607	0.770	**0.794**	0.680	0.647	**0.564**
	200	0.611	0.774	**0.801**	0.678	0.641	**0.559**

Table 2: Performance evaluation of FedAvg, FedAdagrad and MetaUA over different datasets. For a comparison logloss and AUC are presented for different rounds of federated learning. Best values are in bold.

dimension for all embeddings set to 4. For a central training experiments Adam optimizer is used for ten epochs with $lr = 0.0001$, weight decay 0.0001, batch size of 256. In FL settings we use SGD as a local optimizer with $lr = 0.01$ with batch size of 15 and three epochs done in each of 200 communication rounds. In proposed method, meta-parameters are optimized with SGD with $lr = 2.0$. We use layer-wise approach where each parts of the network have dedicated meta-parameters. Implementation is done using PyTorch library.

Comparison with other methods The results for comparison of MetaUA with FedAvg and FedAdagrad are presented in Table 2. Both adaptive methods are better than simple FedAvg, they converge faster, and the training process is more stable. MetaUA got slightly better final AUC than FedAdagrad or very competitive (AmazonCDs worse only by 0.007). However, if we consider for those results logloss value during the train the difference is big. On average MetaUA gets 8.5% better logloss. This margin is noticeable in FL training curves in Figure 2.

Selection of client's attributes We evaluated a few different information that can be shared to a server from clients during a FL training. Following desired characteristic described in section 12 we used: z_1 - number of samples [19], z_2 - local loss value [23], z_3 - gradient norm, z_4 - ratio of loss value after local SGD train to local loss, z_5 - positive class probability, and z_6 - number of unique features in local dataset [12, 29, 30]. Each of the variables grasps a different aspect of a client's local dataset and local training

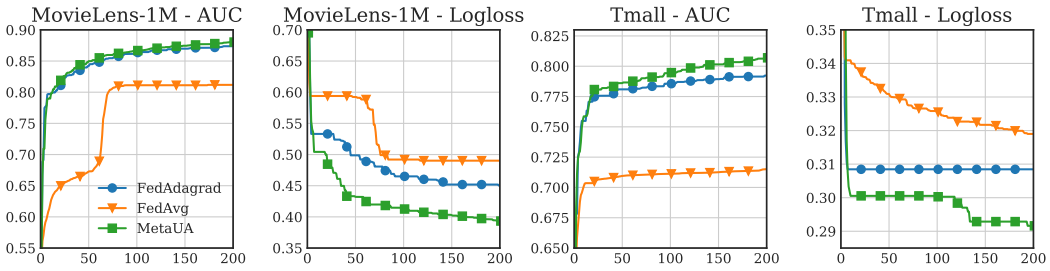

Figure 2: AUC and Logloss values during the federated train with methods FedAvg, FedAdaptive and MetaUA for MovieLens-1M and Tmall datasets.

	AUC					Logloss				
rounds / method	20	50	100	150	200	20	50	100	150	200
None	0.640	0.761	0.911	0.933	0.943	**0.661**	0.606	0.362	0.298	0.269
z_1	0.700	0.897	0.936	0.949	0.956	0.685	0.399	0.300	0.261	0.241
z_2	0.629	0.637	0.887	0.944	**0.964**	0.667	0.667	0.405	0.275	**0.206**
z_3	0.595	0.690	0.825	0.871	0.894	0.693	0.693	0.676	0.573	0.498
z_4	0.611	0.641	0.815	0.881	0.922	0.679	0.679	0.540	0.419	0.335
z_5	0.637	0.650	0.887	0.926	0.944	0.666	0.666	0.404	0.320	0.275
z_6	**0.712**	**0.899**	**0.937**	**0.950**	0.957	0.684	**0.391**	**0.296**	**0.257**	0.239
All	0.592	0.728	0.808	0.845	0.864	0.691	0.664	0.664	0.664	0.649

Table 3: Performance of MetaUA evaluated with different set of client's attributes Z on Amazon-CDs dataset.

process during the FL communication round. Sampled clients Z distributions during FL train for MoveieLens-1M and Amazon-CDs datasets are presented in Figure 3. The smallest variation presents z_3. However, this is the only one dependent on FL round as well as the layer of the network (computed layer-wise, on plots we see aggregated values).

Table 3 present the results for Amazon-CDs dataset. The best final logloss value is achieved only using z_2 (local loss), which happens to be as well a good proxy for client importance in previous work [23]. Up to some point in FL training, using only θ_s is also beneficial. z_6 (unique features num.) and z_2 are second good choices. Surprisingly, taking all variables does not yield the best results. For evaluated datasets it seems like simpler choices and models works better with an online meta-learning. In our experiments we evaluated different models, i.e. MLP network for $f_\alpha(Z;\theta_\alpha)$. However, the results for simple linear model were better.

Ablation study In order to evaluate contribution of different elements of MetaUA method we performed ablation study where each part of the approach is disabled. Four versions are evaluated in very same setting: only server learning rate meta-learning, client weighting with meta-learning, both at the same time and no adaptation at all. Comparison is presented in Table 4 for MovieLens-1M dataset. While the model is optimized toward the best logloss value, here application of whole MetaUA gives the best advantage. For a longer FL training period, at round 400, adaptation of server learning rate and client weighting yield similar outcome of 0.413. Only the application both at once lower the results to 0.382. AUC seems also beneficial when all elements of MetaUA are used.

Different fraction of clients To present robustness of MetaUA towards different fraction of client sampled during FL training rounds, we lowered this value to 5% and 1%. Comparison with FedAdagrad

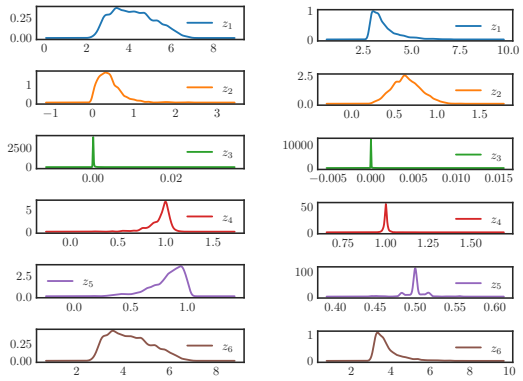

Figure 3: Distribution of Z variables for datasets Movielens-1M (left) and Amazon-CDs (right). Snapshot of sampled values are taken during round number 20.

	AUC				Logloss			
FL round	50	100	200	400	50	100	200	400
no adjustment	0.841	0.864	0.878	0.882	0.502	0.461	0.436	0.429
client weighting	0.844	0.862	**0.880**	0.885	0.454	0.439	0.417	0.413
server lr	0.846	0.865	0.877	0.881	0.484	0.441	0.415	0.413
client weighting + server lr	**0.850**	**0.867**	**0.880**	**0.889**	**0.432**	**0.413**	**0.393**	**0.382**

Table 4: Ablation study

is presented in Figure 4. We presented a longer training as with fewer clients during training more rounds can be necessary for the convergence. While fraction get smaller, MetaUA still wins with FedAdagrad with a big margin. For MovieLens-1M, the difference is clear. Only FedAdagrad with fraction 5% outperforms 10% with more than 350 rounds. For Amazon-CDs the best at final round got MetaUA with 5% clients sampled in each round.

Robusness to hyper-params - selection of meta lr MetaUA is helping to solve its main task – adaptive optimization during FL on the server side. However, as any other meta-learning method, it introduces its own hyper-parameters at the meta-level. As long as those are easy to find the whole process is applicable. For MetaUA the meta-learning rate is the one most crucial. In all experiments we used fixed value of 2.0, without doing extensive hyper-parameter search each time. Thus, in Figure 5 we can see the performance for different values of the meta-learning rate on MovieLens-1M dataset

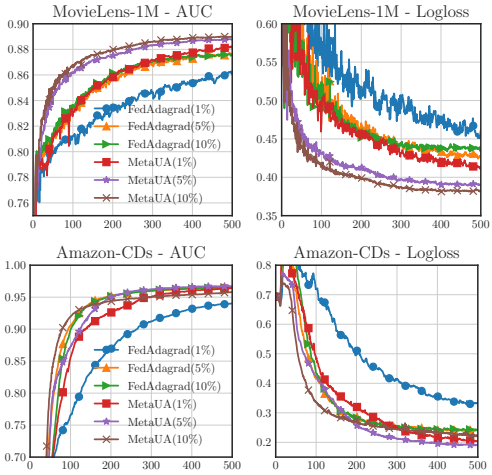

Figure 4: Comparison of MetaUA and FedAdagrad training curves when different fraction of clients is used during FL rounds.

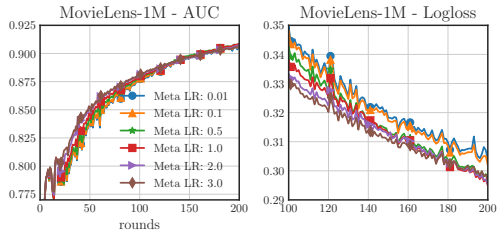

Figure 5: AUC and Logloss values during FL training on MovieLens-1M dataset for a various meta-learning rate value. Logloss (right) plot is zoomed-in after 100 rounds to better present the difference.

in order to present the robustness of our approach. It is easy to find the best performing meta-learning. Here, for values bigger than one almost the same logloss at round 200 is received.

7 CONCLUSIONS

This paper presents a method of learning adaptive models for client weighting and server learning rate adaption in the aggregation of local model updates for federated deep CTR models. In our experiments on public benchmarks, our method shows better performance compared to the state-of-the-art methods. The method is adjusted in an online way fitted in FL communication rounds, without a need of additional heavy communication from the clients. As future work, one important direction it to improve the privacy protection of our method, for example, by introducing a random noise to both model updates and gradients communicated back to the server. Another extension of a proposed MetaUA method is to learn the sample weighting for local trains in the FL settings.

REFERENCES

[1] A. Gunes Baydin et al. "Online Learning Rate Adaptation with Hypergradient Descent". In "International Conference on Learning Representations", 2018.

[2] Yoshua Bengio. Practical Recommendations for Gradient-Based Training of Deep Architectures, pp. 437–478. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[3] Zachary Charles et al. "On the outsized importance of learning rates in local update methods". 2020.

[4] Fei Chen et al. "Federated Meta-Learning with Fast Convergence and Efficient Communication", 2019.

[5] Zihan Chen et al. "Dynamic Attention-based Communication-Efficient Federated Learning", 2021.

[6] Chelsea Finn et al. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In doina Precup et al., Editors, "Proceedings of the 34th International Conference on Machine Learning", vol. 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135. PMLR, 2017.

[7] Chelsea Finn et al. "Online Meta-Learning". In Kamalika Chaudhuri et al., Editors, "Proceedings of the 36th International Conference on Machine Learning", vol. 97 of *Proceedings of Machine Learning Research*, pp. 1920–1930. PMLR, 2019.

[8] Adrian Flanagan et al. "Federated Multi-view Matrix Factorization for Personalized Recommendations". 2020.

[9] Jack Goetz et al. "Active Federated Learning". 2019.

[10] Xiangnan He et al. "Neural Collaborative Filtering". In "Proceedings of the 26th International Conference on World Wide Web", WWW '17, p. 173–182. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 2017.

[11] Shaoxiong Ji et al. "Learning Private Neural Language Modeling with Attentive Aggregation". In "2019 International Joint Conference on Neural Networks (IJCNN)", pp. 1–8. 2019.

[12] Hadi Samer Jomaa et al. "Hyp-RL : Hyperparameter Optimization by Reinforcement Learning". 2019.

[13] Peter Kairouz et al. "Advances and open problems in federated learning. CoRR". 2019.

[14] Sai Praneeth Karimireddy et al. "SCAFFOLD: Stochastic Controlled Averaging for Federated Learning". In Hal Daumé III et al., Editors, "Proceedings of the 37th International Conference on Machine Learning", vol. 119 of *Proceedings of Machine Learning Research*, pp. 5132–5143. PMLR, 2020.

[15] Tian Li et al. "Federated Optimization in Heterogeneous Networks". In I. Dhillon et al., Editors, "Proceedings of Machine Learning and Systems", vol. 2, pp. 429–450. 2020.

[16] Xiang Li et al. "On the convergence of fedavg on non-iid data". 2019.

[17] Zhenguo Li et al. "Meta-SGD: Learning to Learn Quickly for Few-Shot Learning", 2017.

[18] Yujie Lin et al. Meta Matrix Factorization for Federated Rating Predictions, p. 981–990. Association for Computing Machinery, New York, NY, USA, 2020.

[19] Brendan McMahan et al. "Communication-efficient learning of deep networks from decentralized data". In "Artificial intelligence and statistics", pp. 1273–1282. PMLR, 2017.

[20] Paul Micaelli et al. "Non-greedy Gradient-based Hyperparameter Optimization Over Long Horizons". 2020.

[21] Khalil Muhammad et al. "FedFast: Going Beyond Average for Faster Training of Federated Recommender Systems". In "Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining", KDD '20, p. 1234–1242. Association for Computing Machinery, New York, NY, USA, 2020.

[22] Sashank J. Reddi et al. "Adaptive Federated Optimization". In "International Conference on Learning Representations", 2021.

[23] Jun Shu et al. "Meta-Weight-Net: Learning an Explicit Mapping For Sample Weighting". In "NeurIPS", pp. 1917–1928. 2019.

[24] Weiping Song et al. "AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks". In "Proceedings of the 28th ACM International Conference on Information and Knowledge Management", CIKM '19, p. 1161–1170. Association for Computing Machinery, New York, NY, USA, 2019.

[25] Ashish Vaswani et al. "Attention is All you Need". In I. Guyon et al., Editors, "Advances in Neural Information Processing Systems", vol. 30. Curran Associates, Inc., 2017.

[26] Jianyu Wang et al. "A Field Guide to Federated Optimization", 2021.

[27] Jianyu Wang et al. "Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization". In H. Larochelle et al., Editors, "Advances in Neural Information Processing Systems", vol. 33, pp. 7611–7623. Curran Associates, Inc., 2020.

[28] Ruoxi Wang et al. "DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems". In "Proceedings of the Web Conference 2021", pp. 1785–1797. 2021.

[29] Martin Wistuba et al. "Two-Stage Transfer Surrogate Model for Automatic Hyperparameter Optimization". In "ECML/PKDD", 2016.

[30] Martin Wistuba et al. "Scalable Gaussian process-based transfer surrogates for hyperparameter optimization". 2017.

[31] Youjiang Xu et al. "Faster Meta Update Strategy for Noise-Robust Deep Learning". In "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)", pp. 144–153. 2021.

[32] Zizhao Zhang et al. "Learning Fast Sample Re-Weighting Without Reward Data". In "Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)", pp. 725–734. 2021.