Field-aware Factorization Machines in a Real-world Online Advertising System

Yuchin Juan*
Criteo Research
Palo Alto, CA
yc.juan@criteo.com

Damien Lefortier*
Facebook
London, UK
dlefortier@fb.com

Olivier Chapelle
Google
Mountain View, CA
chapelle@google.com

ABSTRACT

Predicting user response is one of the core machine learning tasks in computational advertising. Field-aware Factorization Machines (FFM) have recently been established as a state-of-the-art method for that problem and in particular won two Kaggle challenges. This paper presents some results from implementing this method in a production system that predicts click-through and conversion rates for display advertising and shows that this method it is not only effective to win challenges but is also valuable in a real-world prediction system. We also discuss some specific challenges and solutions to reduce the training time, namely the use of an innovative seeding algorithm and a distributed learning mechanism.

1. INTRODUCTION

Online advertising is a major business for Internet companies and one of the core problem in that field is to be able to match the right advertisement to the right user at the right time. Accurate click-through rate prediction is essential for solving that problem and has been the topic of extensive research, both for search advertising [11, 20] and display advertising [5, 14]. Performance based advertisers measure the performance of their campaigns not only with respect to clicks, but also to conversions – defined as a user action on the website such a purchase – and specific machine learning models have been developed for conversion prediction [15, 23, 3, 26].

A prominent model for these prediction problems is logistic regression with cross-features [20, 5]. When all cross-features are added, the resulting model is equivalent to a polynomial kernel of degree 2 [2]. A Kaggle challenge was hosted by Criteo in 2014 to compare CTR prediction algorithms.[1] Logistic regression with cross-features was indeed quite successful in that competition: the 3rd place winner solution was based on this technique [24]. But the winning solution is a variant of factorization machines [22] called *Field-aware Factorization Machines* (FFM) [14]. The impressive performance of FFM prompted us to implement it and test it as part of our production system.

FFM.

Consider the case of categorical features – most features in ad systems are either categorical or can be made categorical through discretization. Let F be the number of features (or fields) and $v_1, \ldots, v_F$ be the values of these features for a given example. The FFM prediction on this example can be written as:[2]

$$\sum_{f_1=1}^{F} \sum_{f_2=f_1+1}^{F} \mathbf{w}_{i_1} \cdot \mathbf{w}_{i_2}, \quad \text{where } i_1 = \Phi(v_{f_1}, f_1, f_2),\ i_2 = \Phi(v_{f_2}, f_2, f_1), \quad (1)$$

with $\mathbf{w} \in \mathbb{R}^{d \times k}$ the weight matrix and $\mathbf{w}_i \in \mathbb{R}^k$ denotes the embedding of the i-th entry. The mapping $\Phi(v, f_1, f_2)$ maps a value v of feature f_1 in the context of feature f_2 to an index from 1 to d. This may be any hash function or based on dictionary. In the latter case, d will be equal to $F \times \sum_{f=1}^{F} c_f$, with c_f the cardinality of the f-th feature.

In regular factorization machines, there is a unique embedding for a given feature value; in other words, the indices in (1) for FM are $i_1 = \Phi(v_{f_1}, f_1)$ and $i_2 = \Phi(v_{f_2}, f_2)$. But in *field-aware* FM, there is a different embedding depending on the other feature of the dot product. As argued in [14] this gives additional modeling flexibility.

Related work.

A similar effort to ours has been reported by AdRoll in a blog post:[3] the author reports substantial gains after deploying FMs in their CTR prediction system. Google [20] and Facebook [12] may not use FMs but have reported some specific challenges they encountered in productionizing their large scale CTR prediction system, which are related to the challenges for productionizing FFM. Factorisation Machine supported Neural Network (FNN) and Sampling-based Neural Network (SNN) [28] are two learning algorithms related to FMs that have also been applied a CTR prediction task. They are both deep neural networks but differ in their embedding layer: SNN uses a regular embedding layer while

*Contributed equally to this work.

[1] https://www.kaggle.com/c/criteo-display-ad-challenge

[2] The prediction here is specific to categorical features while [14] handles the more general case of continuous features.

[3] http://tech.adroll.com/blog/data-science/2015/08/25/factorization-machines.html

©2017 International World Wide Web Conference Committee (IW3C2), published under Creative Commons CC BY 4.0 License.
WWW'17 Companion, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4914-7/17/04.
http://dx.doi.org/10.1145/3041021.3054185

FNN is initialized with the result of a factorization machine. The recent interest in factorization machines have led to the development of distributed solvers [18] for these techniques. Finally a hierarchical version of factorization machines has been introduced in [21].

Even though FFM have been shown to be a state-of-the-art method for computational advertising by winning two Kaggle challenges, it is still unclear if they are well suited in a production environment. The Netflix challenge is a reminder that a production system has some specific set of constraints and goals that differ from the ones of an academic competition: ultimately Netflix decided not use the winning solution.[4]

This paper discusses our attempt at implementing FFM in a production system that predicts click-through and conversion rates on display advertisements. Section 2 presents offline and online (A/B test) results and provides some insights on the benefits of this method over standard logistic regression as well as the challenges for using FFM in a production system. These positive results further led us to address one of the main bottleneck encountered with our FFM implementation: training speed. Section 3 investigates how to train FFM in a distributed environment. And Section 4 offers an innovative model seeding procedure to further solve that problem, resulting in a more accurate model with a shorter training time and using less computation resources. Finally Section 5 presents conclusions and future work.

2. FFM IN A PRODUCTION SYSTEM

In this Section, we describe how we use FFM in our production system, present our offline and online results, and discuss the benefits and challenges of using FFM in such a setting.

2.1 Baseline

As discussed in Section 1, state-of-the-art advertising systems are based on click-through rate (CTR) and conversion rate (CR) prediction models. In this paper, we consider both CTR and CR prediction models used for bidding in real-time auctions (see, e.g., [5, 26]). To predict the probability of a sale given a display, we use a multiplicative model between a model of the probability of a click given a display and a model of the probability of a sale given a click, as discussed in [3]. So, in the rest of the paper we call these two models CTR and CR.

Our baseline system for training these models is based on previous work [1, 5, 26]. Indeed, following [1, 5], we use the hashing trick [27] to reduce the dimensionality of our data and to thus reduce the number of parameters to fit. We use logistic regression (LR) with cross-features fitted with L-BFGS warm-started using SGD [1, 5]. Following [26], we also use cost-sensitive learning for the CR model and weight each sale depending on the value of the sale for the advertiser, as this was shown to increase the performance for the CR model both offline and online. We use Hadoop AllReduce for distributing the learning of our models [1].

Below, we investigate the usage of FFM instead of LR for training our CTR and CR prediction models. We still use the hashing trick. So, the mapping $\Phi(v, f_1, f_2)$ in (1) is based on a hash which a fixed hashing space (of the order of tens of millions).

[4]http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html

2.2 Offline comparison

We now present results comparing FFM to the state-of-the-art baseline on an offline dataset.

Offline metrics.

We use two offline metrics. First, we use the normalized log loss (NLL). This metric shows the relative improvement in log loss (LL) of the model to be evaluated versus a baseline predictor, in our case the average empirical CTR or CR of the dataset, similar to the normalization in [12, 16, 26]. This metric is defined formally for any prediction p as follows, where we denote $\bar{p}$ the best constant predictor on the test set and N the number of impressions in our dataset.

$$\mathbf{LL}(p) = -\sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \qquad (2)$$

$$\mathbf{NLL}(p) = \frac{\mathrm{LL}(\bar{p}) - \mathrm{LL}(p)}{\mathrm{LL}(\bar{p})} \qquad (3)$$

We also use the *Utility*[5] metric [4, 26], which allows to model offline the potential change in profit due to a prediction model change. Since the observed profit in historical data is fixed, this metric makes the assumption that the display costs are determined by the highest second bids coming from a second price auction and that they are generated according to a distribution conditioned on the observed display cost. This metric is defined as follows, where v_i is the reward of the i^{th} impression.

$$\mathbf{Utility} = \sum_i \int_0^{p(\mathbf{x}_i)v_i} (y_i \cdot v_i - \tilde{c}) \Pr(\tilde{c} \mid c_i) d\tilde{c} \qquad (4)$$

The distribution $\Pr(\tilde{c} \mid c)$ specifies what could have been the second price instead of the observed cost c; [4] suggests a Gamma distribution with $\alpha = \beta c + 1$ and free parameter β. The motivation for selecting this distribution is that it interpolates nicely between two limit distributions: a Dirac distribution centered at c (as $\beta \to +\infty$) and a uniform distribution (as $\beta \to 0$). It can be shown that the utility with a uniform distribution is equivalent to a weighted squared error [13].

Experimental setup.

We use internal data from Criteo to do our experiments. Note, however, that, as discussed in Section 1, FFM have been shown to be better than existing methods on many public data sets already [14]. Moreover, the goal of this section is to show we can improve upon our baseline using FFM in a real-world online advertising system, which uses its own data. We need offline experiments to ensure that FFM are performing well in our system (both in terms of predictive performance and scalability) and for parameter tuning, before we can perform a live experiment (A/B test).

We use a variant of progressive validation, similar to [20], for our experiments. The day following the training period serves as a validation set. As shown on Figure 1 below,

[5]This metric is called *expected* Utility in [4], but we refer to it as Utility in this paper.

the process is repeated N times, shifting the learning period (indicated by "tr") by 1 day at each step. The final results are the average metrics over all the test sets (indicated by "te").

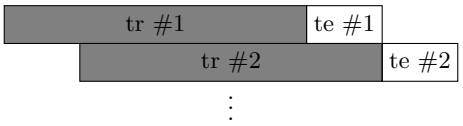

Figure 1: Progressive validation

Parameter tuning is done on a separate temporal slice of data from the data of 1 used for the final experiments. Following [14] the following parameters are tuned: the regularization parameter, the learning rate, and the number of latent factors. We use early-stopping to avoid over-fitting.

Below and in the rest of the paper, we use confidence intervals computed using bootstraps [10] at the 90% level. Finally the learning of FFMs is multi-threaded as in [14] to reduce the learning time.

Latency & memory consumption.

One potential drawback of using FFM in a production system is that they require more CPU time for inference [14]. This may lead to increased latency online when responding to bid requests and therefore come more timeouts. FFM also require more memory for storing the model as the number of latent factors and/or the number of fields increase, which may lead to a much larger memory consumption than LR.

To solve the memory issue, we propose to reduce the size of the hashing space of FFM models (compared to our baseline) so that FFM models have the same size as the LR models (the exact value depends on the number of fields and on the number of latent factors). Note that if we had not reduced the size of the hashing space, but kept it constant, the size of FFM models would be more than a *100 times* larger than our baseline, which would make it impractical. Therefore, in the results below, FFM and LR models have the same number of parameters (unlike in [14]).

To solve the latency issue, we propose to reduce the number of latent factors as much as possible without significantly degrading the performance of FFM. Using these two solutions, FFM and LR consume the same amount of memory and we can limit the impact on latency to handle the requirement of our production system, as we will see below.

Offline results.

We compare LR and FFM on our CTR and CR prediction tasks in terms of NLL (Table 1) and Utility (Table 2). FFM achieves significantly better results with a large effect compared to LR, both in terms of NLL and of Utility for our CTR model, thus confirming the results from [14] on our data. We also observe large gains on our CR model, thus extending the results from [14] to CR models on all our offline metrics.

We also observe that the improvements are even larger on small advertisers, which represent a significant portion of our traffic, for both our CTR and CR models on all metrics. Our hypothesis to explain these results has to do with sparse data and unobserved cross-features: LR is unable to predict the value associated with a cross-feature that is not part of the training data; on the other hand, FFMs are able to better generalize through their latent representation (see detailed explanation and example in [14, Section 2]). For large advertisers LR has enough data to learn a good model, but for small advertisers FFMs handle this data sparsity issue better than LR.

During the tuning of the hyper-parameters, we observed very similar results as in [14] in terms of performance w.r.t each hyper-parameter. The most important parameter is the number of epochs and we use early-stopping to automatically tune it.

We also investigated the prediction time of FFM compared to the baseline model, which is expected to increase despite the fact that we constrained our FFM models to be of the same size of the baseline. This is because the number of operations to compute the prediction (1) is $O(F^2k)$ while LR with all cross-features requires only $O(F^2)$ operations. We observed that the slowdown of FFM is indeed proportional to the number of latent factors k. It turns out that $k = 2$ is a good trade-off: it hardly degrades the accuracy results compared to the results above which were obtained with 4 latent factors (0.1% in NLL) and a 2x in prediction time in our system is acceptable since prediction is is not the most time-consuming part of processing a request (compared to extracting raw features, pre-processing them, etc.).

2.3 Online comparison

As the offline results were quite promising, we decided to run an A/B test using FFM for both CTR and CR predictions models. Although FFM require more time for the inference (see above), we did not observe any significant impact on our timeouts while serving live traffic. So, we were able to A/B test FFM on a large portion of our live traffic. This A/B test served ∼5B displays (∼2.5B for each population).

During the A/B test, we ensured that both the baseline model and FFM were refreshed online *synchronously* since different refresh rates might bias the results. Even with multi-threading, the learning time of FFM is indeed much higher than for our distributed optimization baseline. In Section 3 and 4, we will see how to reduce this learning time, but now we focus only on the performance improvements we can get online with FFM. The results we obtained are the following.

Results are shown in Table 3. We observed an increase in the number of displays (+4.59%), while the overall display cost stayed almost constant. We observed less clicks, but more conversions leading to more advertiser value for the same cost. Our change therefore resulted in a significant positive impact: +0.97% of Return On Investment (ROI), that is of advertiser value over cost, which is substantial. We also observed that the improvements were even larger on small advertisers (defined as the ones with less than 30 sales per day), which represent a significant portion of our traffic. On small advertisers, we observed an increase in the number of displays (+4.85%), while the overall display cost stayed almost constant too. We also observed less clicks, but even more conversions leading to even more advertiser value for the same cost and to +2.61% of Return On Investment (ROI), which is remarkable.

This confirms our offline results and shows that one of the strengths of FFM is indeed their ability to generalize better than logistic regression through their use of a latent representation.

Table 1: Offline relative comparison between Logistic Regression (baseline) and FFM on our CTR and CR prediction tasks in terms of the NLL metric (3). We present results on all advertisers and on small advertisers – defined as advertisers with less than 30 sales per day on average. NLL of the CTR (resp. CR) model is the NLL of the probability of a click (resp. sale) given a display. Statistical significance is indicated by ▲.

Prediction model with FFM	NLL on all advertisers	NLL on small advertisers
CTR	+3.71%▲	+5.9%▲
CTR + CR	+1.21%▲	+6.2%▲

Table 2: Offline relative comparison between Logistic Regression (baseline) and FFM on our CTR and CR prediction tasks in terms of Utility metrics (4). We report the Utility of our model for the expected number of sales given display, which uses our CTR and CR models as sub-models. Statistical significance is indicated by ▲.

Prediction model with FFM	Utility$_{\beta=10}$ on all advertisers	Utility$_{\beta=10}$ on small advertisers	Utility$_{\beta=1000}$ on all advertisers	Utility$_{\beta=1000}$ on small advertisers
CTR	+6.29%▲	+9.70%▲	+2.22%▲	+4.39%▲
CTR + CR	+11.42%▲	+38.44%▲	+5.43%▲	+18.34%▲

2.4 Discussion

Our positive online results motivate us to use FFM in production instead of LR. To do so, the code change is rather small if SGD is already available. However, there are a few challenges to keep in mind when using FFM instead of LR in a production system.

The main concern with rolling out FFM is the learning time, which is much higher than the baseline as discussed before. This means that our models would be refreshed less often with FFM, at the cost of reducing the performance of the system. All our offline experiments to improve our models would also take much longer. This is *not acceptable* and we will discuss in the next two sections how to tackle this problem to handle the scale of a large production system, in particular by distributing the learning on multiple machines.

There are also other challenges. Above, we discussed the memory consumption and prediction latency issues and we showed how to manage them. Another potential problem is the non-convexity of the objective function of FFM, which may lead to some instability in the performance of FFM due to local minimums. To investigate this, we learned multiple FFMs on the same dataset initialized with random weights as in [14]. We observed that all the models have similar performance (±0.05% of NLL) despite the different initializations. The local minimum issue is thus not a major concern.

We also saw above that the number of hyper-parameters in FFM is larger than for LR with the addition of the learning rate (as we use L-BFGS for training our LR models) and of the number of latent factors, while we only had the regularization parameter to tune for LR. This means that tuning takes more time when improving our models. However, and as discussed in [14], this is not a major problem for multiple reasons. First, the performance is not very sensitive to the number of latent factors and to the regularization parameter, while a good value for the learning rate is easy to find. We also found the performance of FFM to be stable over time w.r.t to the hyper-parameters (no need for constant re-tuning).

As we have not been able to find a satisfying regularizer for FFM, we use early-stopping to avoid over-fitting [14]—the only solution we have. So, some monitoring should also be added to ensure that we are not under-fitting or over-fitting despite using early stopping (e.g., if the small amount of data used for testing and deciding when to stop is not representative).

Note finally that for efficient regression testing [16], we need to fix the seed used for randomizing the initial weights [14].

3. A SIMPLE DISTRIBUTED SETTING

In the previous section, we discussed that the training time of FFM is too slow to meet our production requirement, even after applying the parallelization approach mentioned in [14] on a multi-core machine.

To get more speed-up, a natural option is to train FFM on a distributed system. Generally speaking, for sequential algorithms such as SGD or dual coordinate descent, the convergence of their parallelization depends on how often each worker can access the model. In shared-memory systems, because each thread can access the model in real-time, it is possible that the convergence remains the same, as shown in [14]. However, in distributed systems, where we need to use the network for communication, we can no longer share the model among machines in real-time (due to network overhead). There are two main ways of distributing a stochastic gradient algorithm, *synchronously* and *asynchronously*. In both cases, each machine has a subset of the data and its own local model and it updates the global model after a batch of data points has been processed. The asynchronous training is often referred to as the *parameter server* approach [17, 18, 7]: some machines are dedicated to storing the global model and the workers are continuously reading and updating that model with their local model. The synchronous training on the other hand is referred to as *iterative parameter mixing* (IPM) [19, 29, 1]: all the models are averaged after a certain amount of data has been processed (e.g. every epoch). From an engineering point of view, simplicity is one of the most important factors we consider when choosing an algorithm. A complicated algorithm requires more time for development, is harder to maintain, and is more likely to introduce bugs. Therefore, in practice, if a simpler algorithm can solve our problem, we would not go for a more

Table 3: Online relative comparison between Logistic Regression (baseline) and FFM on our CTR and CR prediction models in terms of Return On Investment (ROI), i.e. advertiser value over cost, during our A/B test. Statistical significance is indicated by ▲.

Prediction model with FFM	ROI on all advertisers	ROI on small advertisers
CTR + CR	+0.97%▲	+2.61%▲

Algorithm 1 Iterative Parameter Mixing (IPM) for AdaGrad

Split m data points across k machines
Initialize $\boldsymbol{w}$
Initialize $G_i \leftarrow I \quad \forall i \in \{1, \cdots, k\}$
for $t \in \{1, \cdots, T\}$ **do** ▷ T: number of epochs
  Let $\boldsymbol{w}_i \leftarrow \boldsymbol{w} \quad \forall i \in \{1, \cdots, k\}$
  for $i \in \{1, \cdots, k\}$ *parallel* **do**
    for each data point **do**
      Calculate the gradient $\boldsymbol{g}$
      Update G_i: $G_i \leftarrow G_i + \text{diag}(\boldsymbol{g}\boldsymbol{g}^T)$
      Update $\boldsymbol{w}_i$: $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i - \eta G_i^{-1/2}\boldsymbol{g}$
  $\boldsymbol{w} \leftarrow \sum_{i=1}^{k} \boldsymbol{w}_i / k$

complicated one. As we will see, with IPM we are already be able to speed-up the training time 12x with 32 machines. This already meets our requirement, so we do not investigate the parameter server approach in this paper. IPM for the AdaGrad learning algorithm [9] is described in Algorithm 1.

The speed-up of a distributed algorithm can be modeled by the following equation

$$\text{speed-up} = \#\text{machines} \times \frac{\#\text{epochs with multiple machines}}{\#\text{epochs with one machine}}$$

This equation is based on two assumptions:

1. Each machine finishes the computation at almost the same time.
2. The communication cost among machines is negligible.

In our case, both assumptions indeed hold. The first assumption holds, because we equally distribute the training data to all machines and make sure that each machine has similar computing power. The second one holds because IPM only requires synchronization at the end of each epoch, making the synchronization time much less than computation time.

The "real" distributed algorithm is embedded in our internal system and run on our internal datasets, therefore we are not able to release it. For experimental reproducibility, in this paper we use multi-threading to simulate machines, and use the dataset obtained from Criteo's CTR Prediction Challenge described in Section 1. This simulation is close to reality because the speed-up of a distributed algorithm depends only on the number of machines used and on the slow down in convergence, which can be exactly simulated by multi-threading.

If we directly apply IPM, then the convergence gets slower and slower when we keep adding machines. The experiment result is shown in Table 4. Suppose we use 32 machines instead of 1, although the computation is 32 times faster, it also needs 20 times more epochs. Therefore, the speed-up is only $32/(157/8) \approx 1.6$. A natural way to make the convergence faster is to increase the learning rate η. Though increasing the learning rate indeed makes the algorithm converge faster, it also make the log loss worse. This result is shown in Table 5a.

# machines	#epochs	log loss
1	8	0.44552
2	15	0.44548
4	29	0.44549
8	47	0.44560
16	100	0.44554
32	157	0.44585

Table 4: The number of epochs required to reach the best log loss with different number of machines. Algorithm 1 is applied. The learning rate η is 0.2.

η	#epochs	log loss
0.2	157	0.44585
0.5	70	0.44569
1.0	37	0.44590
2.0	26	0.44622
3.0	21	0.44654
4.0	19	0.44688
5.0	18	0.44721

(a) Algorithm 1

η	#epochs	log loss
0.2	200	0.44819
0.5	130	0.44600
1.0	55	0.44578
2.0	31	0.44565
3.0	22	0.44577
4.0	18	0.44592
5.0	16	0.44608

(b) Algorithm 2

Table 5: With 32 machines, the number of epochs required to reach the best log loss with different learning rates.

We propose the following approach to solve this issue. Remember that, following [14], we use AdaGrad [9] to boost the performance of SGD. AdaGrad records the squared gradient sum (G) to dynamically adjust the learning rate for each dimension. In Algorithm 1, G is not synchronized among machines. It may make G on each machine very small and make the effective learning rate too large. Based on an idea similar to [1], we aggregate G among each machine at the end of each epoch. This new algorithm is described in Algorithm 2. The experiment result is shown in Table 5b. The log loss is much better when a large learning rate is used.

Under this setting, if we choose $\eta = 3.0$, the speed-up we can achieve is $32 \times (8/22) \approx 12$. Indeed, after we applied this setting in our system, we observe a similar speed-up, which enables us to train a model as fast as our current system.

4. WARM-START

As described in Section 2, we regularly re-train models. In Figure 1, suppose each training set contains several days of data, and we move a few hours forward at each step,

Algorithm 2 Improved IPM for AdaGrad

Spread m data points into k machines
Initialize $\boldsymbol{w}$
Initialize $G \leftarrow I$
for $t \in \{1, \cdots, T\}$ **do** ▷ T: number of epochs
  Let $\boldsymbol{w}_i \leftarrow \boldsymbol{w} \quad \forall i \in \{1, \cdots, k\}$
  Let $G_i \leftarrow G \quad \forall i \in \{1, \cdots, k\}$
  for $i \in \{1, \cdots, k\}$ *parallel* **do**
    for each data point **do**
      Calculate the gradient $\boldsymbol{g}$
      Update G_i: $G_i \leftarrow G_i + \text{diag}(\boldsymbol{g}\boldsymbol{g}^T)$
      Update $\boldsymbol{w}_i$: $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i - \eta G_i^{-1/2}\boldsymbol{g}$
  $\boldsymbol{w} \leftarrow \sum_{i=1}^{k} \boldsymbol{w}_i / k$
  $G \leftarrow \sum_{i=1}^{k} G_i$

Algorithm 3 A naive warm-start

Require: an initial model $\mathbf{w}_0$
  $\mathbf{w} \leftarrow \mathbf{w}_0$
  calculate the validation loss L_0
  for $t \in \{1, \ldots, T\}$ **do**
    update $\mathbf{w}$
    $\mathbf{w}_t \leftarrow \mathbf{w}$
    calculate the validation loss L_t
    if $L_t > L_{t-1}$ **then**
      return $\mathbf{w}_{t-1}$

there will be a large amount of overlap between training sets #1 and #2. This means that the model obtained from #1 may be very similar to one obtained from #2. For logistic regression, the training time to obtain model #2 can be significantly reduced by initializing model #2 with model #1. This technique is known as warm-start [6, 25, 8].

For logistic regression, a convex optimization problem, the model will eventually converge to the global optimum no matter warm-start is used or not.[6] Warm-start only influences the convergence speed. However, this is not the case for FFM. To explain why, we first review an undesired property of FFM that has been investigated in [14] – we do not have a good regularization method for FFM, and hence need to rely on early-stopping to prevent over-fitting. We visualize this property in Figure 2. To obtain the best test accuracy, the number of epochs must be carefully selected – with insufficient epochs, the model can be under-fitting; on the other hand, with too many epochs, the model can be over-fitting. To determine the best number of epochs, we usually use a validation set to monitor the model performance at each epoch. Once the validation loss goes up, we stop the training process. We define three phases to indicate the "maturity" of the model.

- Pre-mature: the model is trained with too few epochs
- Mature: the model is trained with enough epochs
- Post-mature: the model is trained with too many epochs

The use of early stopping, however, makes warm-start difficult to be applied. If we seed a mature model to the next step and keep training, then the new model can be post-mature. This problem can be demonstrated in the following

[6]Assuming an appropriate optimization method and a tight stopping criteria are applied.

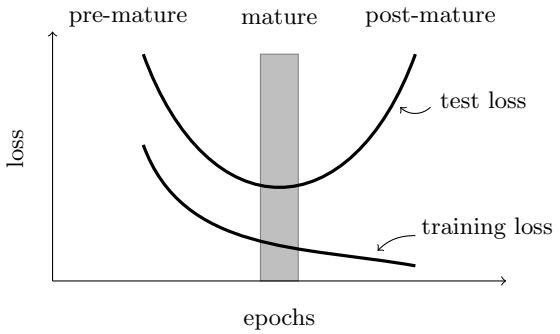

Figure 2: An illustration of over-fitting problem.

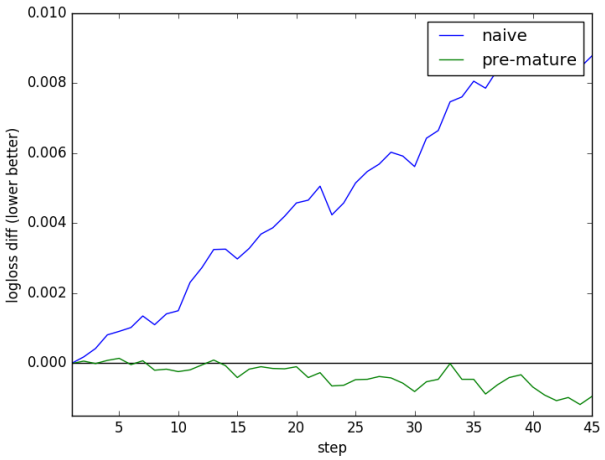

Figure 3: The test log loss of using FFM with different seeding approaches. The y-axis is the difference of log loss compared with the baseline (FFM without warm-start).

experiment. We again use Criteo's CTR Prediction Challenge dataset for reproducibility. We split the data set into 90 blocks, and at each step, 44 blocks are used for training, 1 block for validation, and 1 block for test. Therefore, the entire experiment starts from the 46th block (as test set), moves one block forward at each step, and ends at the 90th block (as test set). The validation set is used to determine the number of epochs. We first compare a *baseline* setting, which did not use any warm-start approach, with a *naive* warm-start described in Algorithm 3, which simply seeds the model obtained in the end of each step into the next step.

The experiment result shown in Figure 3 indicates that the post-mature problem indeed happens seriously – the test accuracy is getting worse and worse when the experiments move forward. Again note that the goal of a warm-start technique is to reduce training time while keep the same predictability of the model. Clearly, by using a naive warm-start for FFM, this goal is not achieved.

In this paper, we propose a new warm-start approach named *pre-mature* warm-start. The idea is that instead of seeding a mature model to the next step, a pre-mature model is used as the seed. At each step, since the new model is initialized with a pre-mature model, it may be able to learn

Algorithm 4 Our proposed "pre-mature" warm-start

Require: an initial model $\mathbf{w}_{-1}$
  $\mathbf{w} \leftarrow \mathbf{w}_0 \leftarrow \mathbf{w}_{-1}$
  calculate the validation loss L_0
  for $t \in \{1, \ldots, T\}$ **do**
    update $\mathbf{w}$
    $\mathbf{w}_t \leftarrow \mathbf{w}$
    calculate the validation loss L_t
    if $L_t > L_{t-1}$ **then**
      return $(\mathbf{w}_{t-1}, \mathbf{w}_{t-2})$

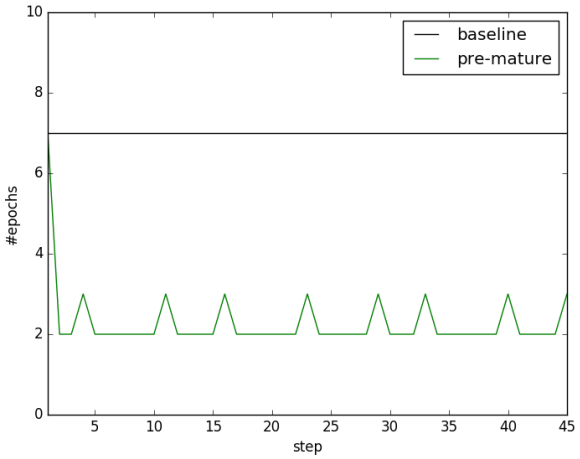

Figure 4: Number of epochs used in eash step. Both settings use 44 blocks of training data.

from the new data without over-fitting to the old data. For example, if the mature model comes at the 6th epoch, then this model will be used for prediction, but the model obtained at the 5th epoch will be seeded to the next step. The algorithm of pre-mature warm-start is described in Algorithm 4. Here, $\mathbf{w}_{t-1}$ is used for prediction and $\mathbf{w}_{t-2}$ is seeded.

Offline results.

The experiment results in Figure 3 and 4 show that with pre-mature warm-start, the test performance is not worse than the baseline any more, and the number of epochs required is significantly reduced.

It is noteworthy that the log loss of FFM with warm-start is getting lower as the experiment moves forward. This suggests that FFM may have some ability to remember the information learnt in the past. Inspired by this observation, we tried reducing the size of training set. Figure 5 shows the comparison among different training sizes with pre-mature warm-start. We see that after sufficient number of steps, pre-mature with only 4 blocks of training set is still better than the baseline using 44 blocks. By using smaller training set, the training becomes much faster. The comparison of training time is shown in Table 6. If we use 4 blocks for training, then it is 20 times faster than the baseline.

An extreme case is to reduce the size of training set to only one block. In this case, because there is no overlap between two consecutive steps, we do not have to use pre-

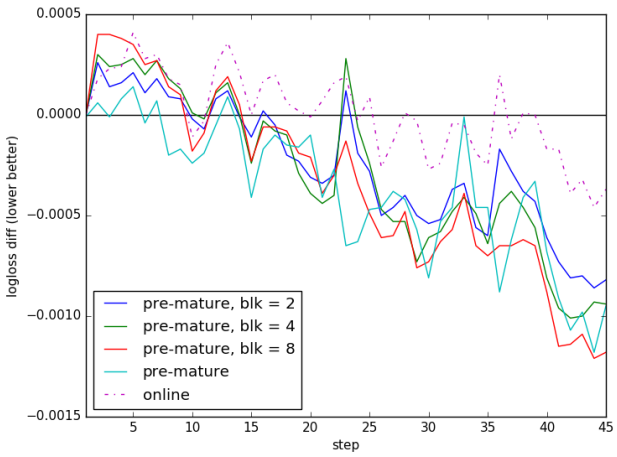

Figure 5: The log loss difference between baseline and different warm-start approaches and training sizes. The baseline (without warm-start) and pre-mature use 44 blocks as training data at each step. Note that we change the training size from the second step. For the first step, all settings use 44 previous blocks as training set, so the log loss are the same.

	#epochs	time / epoch	total time
baseline	315	236s	20.6hr
pre-mature	103	236s	6.8hr
pre-mature (8 blks)	115	47s	1.5hr
pre-mature (4 blks)	128	26s	0.9hr
pre-mature (2 blks)	136	13s	0.5hr

Table 6: Total number of epochs, average time per epoch, and total training time of the entire experiment. Both baseline and pre-mature use 44 blocks as training data.

mature warm-start. (The purpose of pre-mature warm-start is to prevent over-fitting old data.) We illustrate this setting in the following figure, and refer to it as *online*.

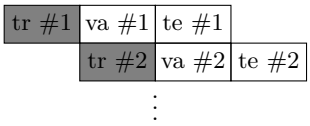

Figure 5 shows FFM still can memorize the information under this setting, as it still out-performs the baseline. However, we do not use this setting because of two reasons. First, our proposed approach can achieve better log loss. Second, conceptually, if we only use a very small portion of data for training, the model can be very sensitive to the quality of this small set. Indeed, for example, at the 36th epoch in Figure 5, we see that online is worse than baseline while our proposed approach still out-performs baseline.

Discussion.

We have proposed two different ways to reduce training time. Distributed learning reduces the training time by adding more machines, but at the same time also *increases* the amount of computation. (In our previous experiments, when 32 machines are used, we needed roughly 3 times more epochs.) On the other hand, warm-start reduces

the training time by initializing a model wisely and require less training epochs, which means the amount of computation is *decreased*. In a sense, warm-start seems to be a better approach than distributed learning. However, we cannot completely replace distributed learning with warm-start, because sometimes a cold-start is required, which means we need to train an entirely new model. In practice, this can happen when the code is updated or the system encounters unexpected error. In the cold-start scenario, we still need to rely on distributed learning to make sure we can learn the model on time.

5. CONCLUSION

In this paper, we showed that Field-aware Factorization Machines can be successfully deployed in large scale advertising system, and that it significantly improves business metrics, in particular for small advertisers. One of the strengths of FFM is indeed their ability to generalize better than logistic regression through their use of a latent representation.

Further, we proposed two ways to make training FFM faster: distributed learning and warm-start. The code for the experiments in Section 3 and 4 is available online.[7] As future works, we plan to try our warm start method on our other non-convex problems that are difficult to regularize, such as a deep neural network.

6. REFERENCES

[1] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford. A reliable effective terascale linear learning system. *The Journal of Machine Learning Research*, 15(1):1111–1133, 2014.

[2] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin. Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research*, 11(Apr):1471–1490, 2010.

[3] O. Chapelle. Modeling delayed feedback in display advertising. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1097–1105. ACM, 2014.

[4] O. Chapelle. Offline evaluation of response prediction in online advertising auctions. In *Proceedings of the 24th International Conference on World Wide Web Companion*, pages 919–922. International World Wide Web Conferences Steering Committee, 2015.

[5] O. Chapelle, E. Manavoglu, and R. Rosales. Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(4):61, 2014.

[6] B.-Y. Chu, C.-H. Ho, C.-H. Tsai, C.-Y. Lin, and C.-J. Lin. Warm start for parameter selection of linear classifiers. In *KDD*, 2015.

[7] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *NIPS*, 2012.

[8] D. DeCoste and K. Wagstaff. Alpha seeding for support vector machines. In *KDD*, 2000.

[7] `https://www.csie.ntu.edu.tw/~r01922136/ffmpaper2/exp/ffmpaper2-exp.tar`

[9] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12:2121–2159, 2011.

[10] B. Efron and R. J. Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[11] T. Graepel, J. Q. Candela, T. Borchert, and R. Herbrich. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 13–20, 2010.

[12] 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 *Proceedings of 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1–9. ACM, 2014.

[13] P. Hummel and R. P. McAfee. Loss functions for predicted click through rates in auctions for online advertising. *Preprint, Google Inc*, 2013.

[14] Y. Juan, Y. Zhaung, W.-S. Chin, and C.-J. Lin. Field-aware factorization machines for CTR prediction. In *RecSys*, 2016.

[15] K.-c. Lee, B. Orten, A. Dasdan, and W. Li. Estimating conversion rate in display advertising from past erformance data. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 768–776. ACM, 2012.

[16] D. Lefortier, A. Truchet, and M. de Rijke. Sources of variability in large-scale machine learning systems. In *Machine Learning Systems (NIPS 2015 Workshop)*, 2015.

[17] M. Li, D. G. Andersen, A. Smola, and K. Yu. Communication efficient distributed machine learning with the parameter server. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 19–27, Cambridge, MA, USA, 2014. MIT Press.

[18] M. Li, Z. Liu, A. J. Smola, and Y.-X. Wang. Difacto: Distributed factorization machines. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 377–386. ACM, 2016.

[19] R. McDonald, K. Hall, and G. Mann. Distributed training strategies for the structured perceptron. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 456–464, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[20] 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 *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1222–1230. ACM, 2013.

[21] R. J. Oentaryo, E.-P. Lim, J.-W. Low, D. Lo, and M. Finegold. Predicting response in mobile advertising with hierarchical importance-aware factorization machine. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 123–132. ACM, 2014.

[22] S. Rendle. Factorization machines with libFM. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57, 2012.

[23] R. Rosales, H. Cheng, and E. Manavoglu. Post-click conversion modeling and analysis for non-guaranteed delivery display advertising. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 293–302. ACM, 2012.

[24] G. Song. Criteo display advertising challenge. Available at `https://www.kaggle.com/c/criteo-display-ad-challenge/forums/t/10547/document-and-code-for-the-3rd-place-finish`, 2014.

[25] C.-H. Tsai, C.-Y. Lin, and C.-J. Lin. Incremental and decremental training for linear classification. In *KDD*, 2014.

[26] F. Vasile, D. Lefortier, and O. Chapelle. Cost-sensitive learning for utility optimization in online advertising auctions. *arXiv preprint arXiv:1603.03713*, 2016.

[27] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120. ACM, 2009.

[28] W. Zhang, T. Du, and J. Wang. Deep learning over multi-field categorical data. In *European Conference on Information Retrieval*, pages 45–57. Springer, 2016.

[29] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. Parallelized stochastic gradient descent. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2595–2603. Curran Associates, Inc., 2010.