Privileged Features Distillation at Taobao Recommendations

Chen Xu*, Quan Li*, Junfeng Ge, Jinyang Gao, Xiaoyong Yang, Changhua Pei, Fei Sun, Jian Wu, Hanxiao Sun, and Wenwu Ou
Alibaba Group, Beijing, China
{chaos.xc,liquan.quanli,beili.gjf,jinyang.gjy,xiaoyong.yxy,changhua.pch,ofey.sf,joshuawu.wujian}@alibaba-inc.com

ABSTRACT

Features play an important role in the prediction tasks of e-commerce recommendations. To guarantee the consistency of off-line training and on-line serving, we usually utilize the same features that are both available. However, the consistency in turn neglects some discriminative features. For example, when estimating the conversion rate (CVR), i.e., the probability that a user would purchase the item if she clicked it, features like dwell time on the item detailed page are informative. However, CVR prediction should be conducted for on-line ranking before the click happens. Thus we cannot get such post-event features during serving.

We define the features that are discriminative but only available during training as the privileged features. Inspired by the distillation techniques which bridge the gap between training and inference, in this work, we propose privileged features distillation (PFD). We train two models, i.e., a student model that is the same as the original one and a teacher model that additionally utilizes the privileged features. Knowledge distilled from the more accurate teacher is transferred to the student, which helps to improve its prediction accuracy. During serving, only the student part is extracted and it relies on no privileged features. We conduct experiments on two fundamental prediction tasks at Taobao recommendations, i.e., click-through rate (CTR) at coarse-grained ranking and CVR at fine-grained ranking. By distilling the interacted features that are prohibited during serving for CTR and the post-event features for CVR, we achieve significant improvements over their strong baselines. During the on-line A/B tests, the click metric is improved by **+5.0%** in the CTR task. And the conversion metric is improved by **+2.3%** in the CVR task. Besides, by addressing several issues of training PFD, we obtain comparable training speed as the baselines without any distillation.

CCS CONCEPTS

• **Information systems → Information retrieval**; • **Computing methodologies → Neural networks**.

*Both authors contributed equally to this work.

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403309

KEYWORDS

Privileged Features, Distillation, E-commerce Recommendations, CTR, CVR

ACM Reference Format:
Chen Xu*, Quan Li*, Junfeng Ge, Jinyang Gao, Xiaoyong Yang, Changhua Pei, Fei Sun, Jian Wu, Hanxiao Sun, and Wenwu Ou. 2020. Privileged Features Distillation at Taobao Recommendations. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394486.3403309

1 INTRODUCTION

In recent years, deep neural networks (DNNs) [5, 6, 11, 22, 29, 41] have achieved very promising results in the prediction tasks of recommendations. However, most of these works focus on the model aspect. While there are limited works except [5, 6] paid attention to the feature aspect in the input, which essentially determine the upper-bound of the model performance. In this work, we also focus on the feature aspect, especially the features in e-commerce recommendations.

To ensure the consistency of off-line training and on-line serving, we usually use the same features that are both available in the two environments in real applications. However, a bunch of discriminative features, which are only available at training time, are thus ignored. Taking conversion rate (CVR) prediction in e-commerce recommendations as an example, here we aim to estimate the probability that the user would purchase the item if she clicked it. Features describing user behaviors in the clicked detail page, e.g., the dwell time on the whole page, can be rather helpful. However, these features cannot be utilized for on-line CVR prediction in recommendations, because it has to be done before any click happens. Although such post-event features can indeed be recorded for off-line training. In consistent with the learning using privileged information [35, 36], here we define the features that are discriminative for prediction tasks but only available at training time, as the *privileged features*.

A straightforward way to utilize the privileged features is multi-task learning [32], i.e., predicting each feature with an additional task. However, in the multi-task learning, each task does not necessarily satisfy a no-harm guarantee (i.e., privileged features can harm the learning of the original model). More importantly, the no-harm guarantee will very likely be violated since estimating the privileged features might be even more challenging than the original problem [21]. From the practical point of view, when using dozens of privileged features at once, it would be a challenge to tune all the tasks.

Inspired by learning using privileged information (LUPI) [25], here we propose privileged features distillation (**PFD**) to take advantage of such features. We train two models, i.e., a student and a

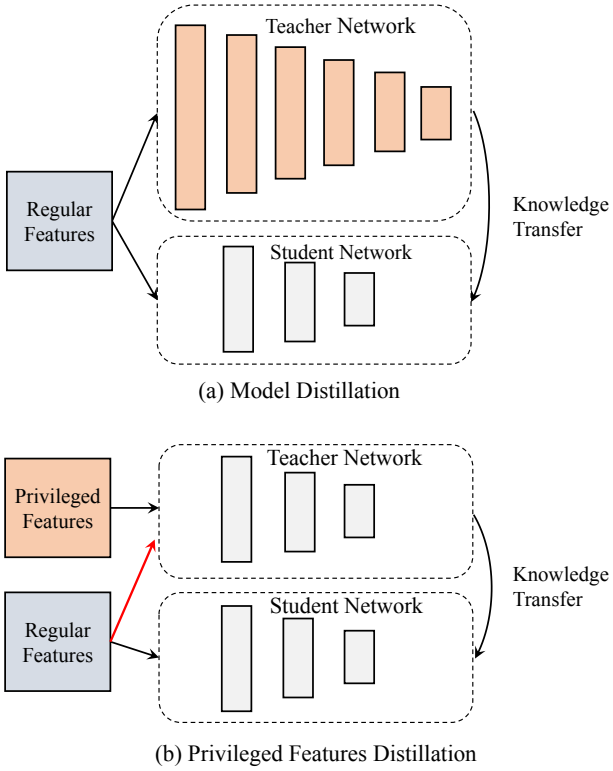


Figure 1: Illustration of model distillation (MD) [14] and privileged features distillation (PFD) proposed in this work. In MD, the knowledge is distilled from the more complex model. While in PFD, the knowledge is distilled from *both* the privileged *and* the regular features. PFD also differs from the original learning using privileged information (LUPI) [25], where the teacher *only* processes the privileged features.

teacher model. The student model is the same as the original one, which processes the features that are both available for off-line training and on-line serving. The teacher model processes all features, which include the privileged ones. Knowledge distilled from the teacher, i.e., the soft labels in this work, is then used to supervise the training of the student in addition to the original hard labels, i.e., $\{0, 1\}$, which additionally improves its performance. During on-line serving, only the student part is extracted, which relies on no privileged features as the input and guarantees the consistency with training. Compare with MTL, PFD mainly has two advantages. On the one hand, the privileged features are combined in a more appropriate way for the prediction task. Generally, adding more privileged features will lead to more accurate predictions. On the other hand, PFD only introduces one extra distillation loss no matter what the number of privileged features is, which is much easier to balance.

PFD is different from the commonly used model distillation (MD) [3, 14]. In MD, both the teacher and the student process the same inputs. And the teacher uses models with more capacity than the student. For example, the teachers can use deeper networks to instruct the shallower students [20, 40]. Whereas in PFD, the teacher and the student use the same models but differ in the inputs. PFD is also different from the original LUPI [25], where the teacher network in PFD additionally processes the regular features. Figure 1 gives an illustration on the differences.

In this work, we apply PFD to Taobao recommendations. We conduct experiments on two fundamental prediction tasks by utilizing the corresponding privileged features. The contributions of this paper are four-fold:

- We identify the privileged features existing at Taobao recommendations and propose PFD to leverage them. Compared with MTL to predict each privileged feature independently, PFD unifies all of them and provides a one-stop solution.
- Different from the traditional LUPI, the teacher of PFD additionally utilizes the regular features, which instructs the student much better. PFD is in complementary to MD. By combining both of them, i.e., PFD+MD, we can achieve further improvements.
- We train the teacher and the student synchronously by sharing common input components. Compared to training them asynchronously with independent components as done in tradition, such training manner achieves even or better performance, while the cost of time is much reduced. Thus the technique is adoptable in online learning, where real-time computation is in high demand.
- We conduct experiments on two fundamental prediction tasks at Taobao recommendations, i.e., CTR prediction at coarse-grained ranking and CVR prediction at fine-grained ranking. By distilling the *interacted features* that are prohibited due to efficiency requirement for CTR at coarse-grained ranking and the *post-event features* for CVR as introduced above, we achieve significant improvements over their strong baselines. During the on-line A/B tests, the click metric is improved by **+5.0%** in the CTR task. And the conversion metric is improved by **+2.3%** in the CVR task.

2 RELATED DISTILLATION TECHNIQUES

Before giving detailed description of our PFD, we will firstly introduce the distillation techniques [3, 14]. Overall, the techniques are aiming to help the non-convex student models to train better. For model distillation, we can typically write the objective function as follows:

$$\min_{\boldsymbol{W}_s} (1-\lambda) * L_s\left(\boldsymbol{y}, f_s(\boldsymbol{X}; \boldsymbol{W}_s)\right) + \lambda * L_d\left(f_t(\boldsymbol{X}; \boldsymbol{W}_t), f_s(\boldsymbol{X}; \boldsymbol{W}_s)\right), \quad (1)$$

where f_t and f_s are the teacher model and the student model, respectively. L_s denotes the student pure loss with the known hard labels $\boldsymbol{y}$ and L_d denotes its loss with the soft labels produced by the teacher. $\lambda \in [0, 1]$ is the hyper-parameter to balance the two losses. Compared with the original function that minimizes L_s alone, we are expecting that the additional loss L_d in Eq.(1) will help to train $\boldsymbol{W}_s$ better by distilling the knowledge from the teacher. In the work of [30], Pereyra et. al. regard the distillation loss as regularization on the student model. When training f_s alone by minimizing L_s, it is prone to get overconfident predictions, which overfit the training set [33]. By adding the distillation loss, f_s will also approximate

the soft predictions from f_t. By softening the outputs, f_s is more likely to achieve better generalization performance.

Typically, the teacher model is more powerful than the student model. Teachers can be the ensembles of several models [3, 14, 39], or DNNs with more neurons [34], more layers [20, 40], or even broader numerical precisions [28] than students. There are also some exceptions, e.g., in the work of [1], both of the two models are using the same structure and learned from each other, with difference only in the initialization and the orders to process the training data.

As indicated in Eq.(1), the parameter $\boldsymbol{W}_t$ of the teacher is fixed across the minimization. We can generally divide the distillation technique into two steps: firstly train the teacher with the known labels $\boldsymbol{y}$, then train the student by minimizing Eq.(1). In some applications, the models could take rather long time to converge, thus it is impractical to wait for the teacher to be ready as Eq.(1). Instead, some works try to train the teacher and the student synchronously [1, 39, 40]. Besides distilling from the final output as Eq.(1), it is possible to distill from the middle layer, e.g., Romero et al. [31] try to distill the intermediate feature maps, which help to train a deeper and thinner network.

In addition to distilling knowledge from more complex models, Lopez-Paz et al. [25] propose to distill knowledge from privileged information $\boldsymbol{X}^*$, which is also known as learning using privileged information (LUPI). The loss function then becomes:

$$\min_{\boldsymbol{W}_s} (1-\lambda) * L_s\left(\boldsymbol{y}, f(\boldsymbol{X}; \boldsymbol{W}_s)\right) + \lambda * L_d\left(f(\boldsymbol{X}^*; \boldsymbol{W}_t), f(\boldsymbol{X}; \boldsymbol{W}_s)\right). \quad (2)$$

In the work of [38], Wang et al. apply LUPI to image tag recommendation. Besides the teacher and the student, they additionally learn a discriminator, which ensures the student to learn the true data distribution at the equilibrium faster. Chen et al. [4] apply LUPI to review-based recommendation. They also utilize adversarial training to select informative reviews. Although achieving better performance, both of the works are only validated on relatively small datasets. It still remains unknown whether these techniques can reach equilibrium of the min-max game in industry-scale datasets.

3 PRIVILEGED FEATURES AT TAOBAO RECOMMENDATIONS

To better understand the privileged features exploited in this work, we firstly give an overview of Taobao recommendations in Figure 2. As usually done in industry recommendations [6, 24], we adopt the cascaded learning framework. There are overall three stages to select/rank the items before presenting to the user, i.e., candidate generation, coarse-grained ranking, and fine-grained ranking. To make a trade-off between efficiency and accuracy, more complex and effective model is adopted as the cascaded stage goes forward, with the expense of higher latency to scoring the items. In the candidate generation stage, we choose around 10^5 items that are most likely to be clicked or purchased by a user from the huge-scale corpus. Generally, the candidate generation is mixed from several sources, e.g., collaborative filtering [9], the DNN models [6], etc. After the candidate generation, we adopt two stages for ranking, where PFD is applied in this work.

In the coarse-grained ranking stage, we are mainly to estimate the CTRs of all items selected by the candidate generation

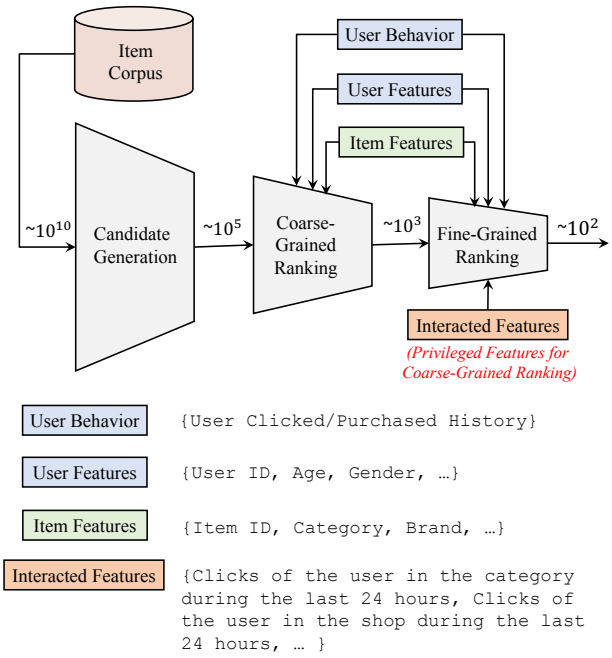


Figure 2: Overview of Taobao recommendations. We adopt a cascaded learning framework to select/rank items. At coarse-grained ranking, the interacted features, although being discriminative, are prohibited as they greatly increase the latency at serving. Some representative features are illustrated in the lower part.

stage, which are then used to select the top-k highest ranked items for the next stage. The inputs of the prediction model mainly consist of three parts. The first part consists of the user behavior, which records the history of her clicked/purchased items. As the user behavior is in sequential, RNNs [13, 15] or self-attention[19, 37] is usually adopted to model the user's long short-term interests. The second part consists of the user features, e.g., user id, age, gender, etc. and the third part consists of the item features, e.g., item id, category, brand, etc. Across this work, all features are transformed into categorical type and we learn an embedding for each one[1].

In the coarse-grained ranking stage, the complexity of the prediction model is strictly restricted, in order to grade tens of thousands of candidates in milliseconds. Here we utilize the inner product model [17] to measure the item scores:

$$f\left(\boldsymbol{X}^u, \boldsymbol{X}^i; \boldsymbol{W}^u, \boldsymbol{W}^i\right) \triangleq \left\langle \boldsymbol{\Phi}_{\boldsymbol{W}^u}\left(\boldsymbol{X}^u\right), \boldsymbol{\Phi}_{\boldsymbol{W}^i}\left(\boldsymbol{X}^i\right) \right\rangle, \quad (3)$$

where the superscript u and i denote the user and item, respectively. $\boldsymbol{X}^u$ denotes a combination of user behavior and user features. $\boldsymbol{\Phi}_{\boldsymbol{W}}(\cdot)$ represents the non-linear mapping with learned parameter $\boldsymbol{W}$. $\langle \cdot, \cdot \rangle$ is the inner product operation. As the user side and the item side are separated in Eq.(3), during serving, we can compute the mappings $\boldsymbol{\Phi}_{\boldsymbol{W}^i}(\cdot)$ of all items off-line in advance[2]. When a request comes, we only need to execute *one* forward pass to get the user mapping $\Phi_{\boldsymbol{W}^u}(\boldsymbol{X}^u)$ and compute its inner product with all

[1] Numerical features are discretized with pre-defined boundaries.
[2] In order to capture the real-time user preference, the user mappings cannot be stored.

Figure 3: Illustrative features describing the user behavior in the detailed page of the clicked item. Including the dwell time that is not shown, these features are rather informative for CVR prediction. However, during serving, we need to use CVR to rank all candidate items as shown in left sub-figure before any item being clicked. We thus denote these features as the privileged features for CVR prediction.

candidates, which is extremely efficient. For more details, see the illustration in Figure 4.

As shown in Figure 2, the coarse-grained ranking does not utilize any interacted features, e.g., clicks of the user in the item category during the last 24 hours, clicks of the user in the item shop during the last 24 hours, etc. As verified by the experiments below, adding these features can largely enhance the prediction performance. However, it in turn greatly increases the latency during serving, since the interacted features are depending on the user and the specific item. In other words, the features vary with different items or users. If putting them either at the item or the user side of Eq.(3), the inference of the mappings $\Phi_{\boldsymbol{W}}(\cdot)$ need to be executed as many times as the number of candidates, i.e., 10^5 here. Generally, the non-linear mapping $\Phi_{\boldsymbol{W}}(\cdot)$ costs several orders more computation than the simple inner product operation. It is thus unpractical to use the interacted features during serving. Here we regard them as the *privileged features* for CTR prediction at coarse-grained ranking.

In the fine-grained ranking stage, besides estimating the CTR as done in the coarse-grained ranking, we need to estimate the CVR for all candidates, i.e., the probability that the user would purchase the item if she clicked it. In e-commerce recommendations, one of the main aims is to maximize the Gross Merchandise Volume (GMV), which can be decomposed into CTR × CVR × Price. Once estimating the CTR and CVR for all items, we can rank them by the expected GMVs to maximize them. Under the definition of CVR, it is obvious that user behaviors on the detailed page of the clicked item, e.g., dwell time, whether viewing the comment or not, whether communicating with the seller or not, etc., can be rather helpful for the prediction. However, CVR needs to be estimated for ranking before any future click happens. Features describing user behaviors on the detailed page are not available during inference. Here we thus denote these features as the *privileged features* for CVR prediction. To better understand that, we give an illustration in Figure 3.

Algorithm 1 Minimizing the student, distillation, and teacher loss in Eq.(5) synchronously with SGD.

Input: Hyper-parameter λ, swapping step k, and learning rate η

Initialize $(\boldsymbol{W}_s, \boldsymbol{W}_t)$ and let $i = 0$.
while not converged **do**
  Get training data $(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{X}^*)$.
  if $i < k$ **then**
    $\boldsymbol{W}_s = \boldsymbol{W}_s - \eta \nabla_{\boldsymbol{W}_s} L_s$.
  else
    $\boldsymbol{W}_s = \boldsymbol{W}_s - \eta \nabla_{\boldsymbol{W}_s} \{(1-\lambda) * L_s + \lambda * L_d\}$.
  end if
  $\boldsymbol{W}_t = \boldsymbol{W}_t - \eta \nabla_{\boldsymbol{W}_t} L_t$. //No distillation loss L_d
  Update $i = i + 1$
end while

Output: $(\boldsymbol{W}_s, \boldsymbol{W}_t)$

4 PRIVILEGED FEATURE DISTILLATION

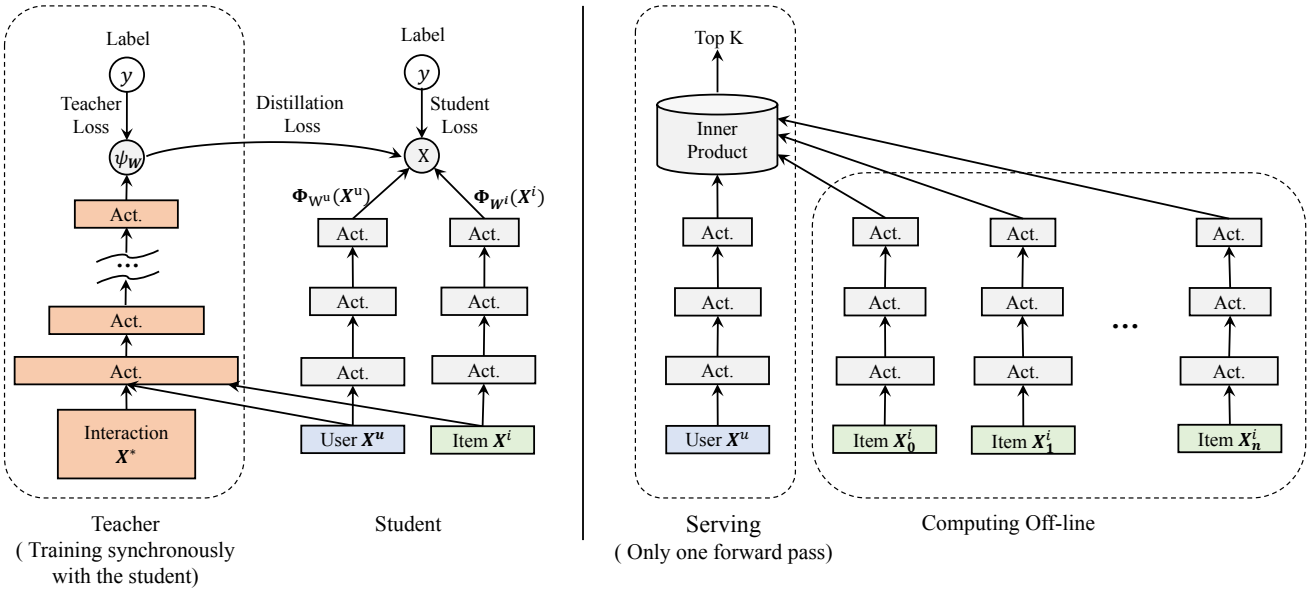
In the original LUPI as Eq.(2), the teacher relies on the privileged information $\boldsymbol{X}^*$. Although being informative, the privileged features in this work only partially describe the user's preference. The performance of which using these features can even be inferior to that of which using regular features. Besides, the predictions based on privileged features can sometimes be misleading. For example, it generally takes more time for customers to decide on expensive items, while the conversion rate of these items is rather low. When conducting CVR estimation, the teacher of LUPI makes predictions relying on the privileged features, e.g., dwell-time, but not considering the regular features, e.g., item price, which may result in false positive predictions on the expensive items. To alleviate that, we additionally feed the regular features to the teacher model. The original function of Eq.(2) is thus modified as follows:

$$\min_{\boldsymbol{W}_s} (1-\lambda) * L_s\left(\boldsymbol{y}, f(\boldsymbol{X}; \boldsymbol{W}_s)\right) + \lambda * L_d\left(f(\boldsymbol{X}, \boldsymbol{X}^*; \boldsymbol{W}_t), f(\boldsymbol{X}; \boldsymbol{W}_s)\right). \quad (4)$$

Generally, adding more information, e.g., more features, will lead to more accurate predictions. The teacher $f(\boldsymbol{X}, \boldsymbol{X}^*; \boldsymbol{W}_t)$ here is thus expected to be stronger than the student $f(\boldsymbol{X}; \boldsymbol{W}_s)$ or the teacher of LUPI $f(\boldsymbol{X}^*; \boldsymbol{W}_t)$. In the above scenario, by taking both the privileged and the regular features into consideration, the dwell time feature instead can be used to distinguish the extent of preference on different expensive items. The teacher is thus more knowledgeable to instruct the student rather than mislead it. As verified by the experiments below, adding regular features to the teacher is non-trivial and it greatly improves the performance of LUPI. Thereafter we denote this technique as PFD to distinguish it from LUPI.

As Eq.(4) indicates, the teacher $f(\boldsymbol{X}, \boldsymbol{X}^*; \boldsymbol{W}_t)$ is trained in advance. However, it takes a long time to train the teacher model alone in our applications. It is thus quite unpractical to apply distillation as Eq.(4). A more plausible way is to train the teacher and the student synchronously as done in [1, 39, 40]. The objective function is then modified as follows:

$$\min_{\boldsymbol{W}_s, \boldsymbol{W}_t} (1-\lambda) * L_s\left(\boldsymbol{y}, f(\boldsymbol{X}; \boldsymbol{W}_s)\right) + \lambda * L_d\left(f(\boldsymbol{X}, \boldsymbol{X}^*; \boldsymbol{W}_t), f(\boldsymbol{X}; \boldsymbol{W}_s)\right) + L_t\left(\boldsymbol{y}, f(\boldsymbol{X}, \boldsymbol{X}^*; \boldsymbol{W}_t)\right). \quad (5)$$



(a) Training with PFD+MD (b) Serving with the Student

Figure 4: Illustration of training the inner-product model with PFD+MD and (b) its deployment during serving. At the training time, the privileged features, i.e., interaction X^* between X^u and X^i, and the more complex DNN model together form a strong teacher to instruct the student. During serving, we compute the mappings $\Phi_{W^i}(\cdot)$ of all items off-line in advance. When a request comes, we only need to execute one forward pass to derive the user mapping $\Phi_{W^u}(X^u)$.

Although saving time, synchronous training can be un-stable. In the early stage when the teacher is not well-trained, the distillation loss L_d may distract the student and slow the training down. Here we alleviate it by adopting a warm up scheme. We set λ of Eq.(5) to 0 in the early stage and fix it to the pre-defined value thereafter, with the swapping step being a hyper-parameter. In our huge scale datasets, we find that such simple scheme works well. Different from mutual learning [39], we only enable the student to learn from the teacher here. Otherwise the teacher will co-adapt with the student, which degenerates its performance. When computing the gradient with respect to the teacher parameters $\boldsymbol{W}_t$, we thus omit the distillation loss L_d. The update with SGD is illustrated in Algorithm 1.

Across this work, all models are trained in the parameter sever systems [8], where all parameters are stored in the servers and most computation is executed in the workers. The training speed is mainly depending on the computation load in the workers and the communication volume between the workers and the servers. As indicated in Eq.(5), we train the teacher and the student together. The number of parameters and the computation are roughly doubled. Training using PFD can thus be much slower than training on the student alone, which is unpractical in industry. Especially for on-line learning where real-time computation is in high demand, adopting distillation can add much burden. Here we alleviate that by sharing all common input components in the teacher and the student. Since the embeddings of all features occupy most of the storage in the severs[3], the communication volume is almost halved by sharing. The computation can also be reduced by sharing the components of processing the user clicked/purchased behavior, which is known to be costly. As verified by the experiments below, we can achieve even or better performance by sharing. Besides, we increase only a little extra time compared to training the student alone, which makes PFD adoptable for online learning.

Extension: PFD+MD. As illustrated in Figure 1, PFD distills knowledge from the privileged features. In comparison, MD distills knowledge from the more complex teacher model. The two distillation techniques are complementary. A natural extension is to combine them by forming a more accurate teacher to instruct the student.

In the CTR prediction at coarse-grained ranking, as Eq.(3) shows, we use the inner product model to increase the efficiency during serving. In fact, the inner product model can be regarded as the generalized matrix factorization [6]. Although we are using *non-linear* mapping $\boldsymbol{\Phi}_{\boldsymbol{W}}(\cdot)$ to transform the user and item inputs, the model capacity is intrinsically limited by the *bi-linear* structure at the inner product operation. DNNs, with the capacity to approximate any function [7, 16], are considered as a substitution for the inner product model in the teacher. In fact, as proved in Theorem 1 of [23], the product operation can be approximated arbitrarily well by a two-layers neural network with only 4 neurons in the hidden layer. Thus the performance of using DNN is supposed to be lower-bounded by that of using the inner-product model.

In PFD+MD, we thus adopt the DNN model as the teacher network. In fact, the teacher model here is the same as the model used for CTR prediction at fine-grained ranking. PFD+MD in this task can be regarded as distilling knowledge from the fine-grained ranking to improve the coarse-grained ranking. For better illustration,

[3]For the student model alone, all embeddings take up to 150 Gigabytes.

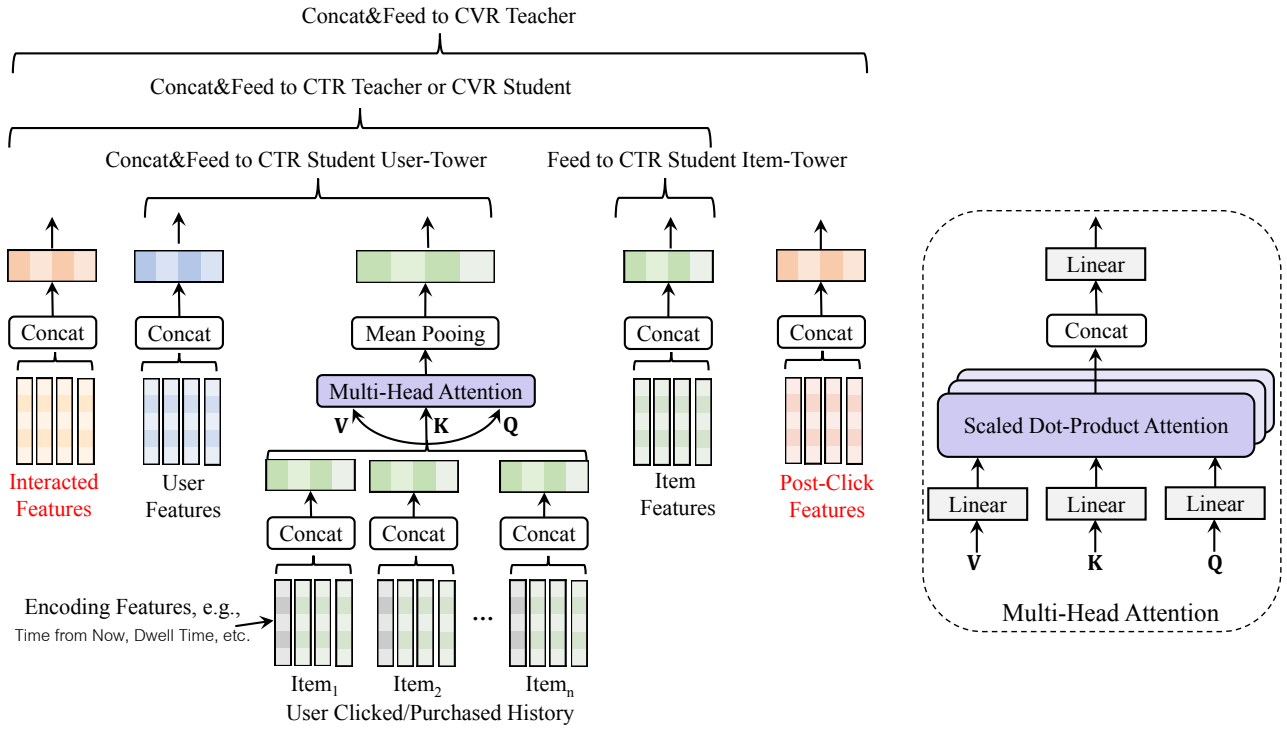


Figure 5: Illustration of input components used for CTR at coarse-grained ranking and CVR at fine-grained ranking. We adopt the multi-head self-attention structure [37] to model the user clicked/purchased history. The common input components between the teacher and the student are shared during training.

we give the whole framework in Figure 4. During serving, we extract the student part only, which relies on no privileged features. As the mappings $\Phi_{W^i}(X^i)$ of all items are independent of the users, we compute them off-line in advance. When a request comes, the user mapping $\Phi_{W^u}(X^u)$ is firstly computed. After that, we compute its inner-product with the mappings of all items produced from the candidate generation stage. The top-k highest scored items are then chosen and fed to the fine-grained ranking. On the whole, we only execute one forward pass to derive the user mapping and conduct efficient inner product operations between the user and all candidates, which are rather friendly in the aspect of computation.

5 EXPERIMENTS

In this section, we conduct experiments at Taobao recommendations with the aim of answering the following research questions:

- **RQ1:** What is the performance of PFD on the tasks of CTR at coarse-grained ranking and CVR at fine-grained ranking?
- **RQ2:** Compared to individual PFD, can we achieve additional improvements by combining PFD with MD?
- **RQ3:** Is PFD sensitive to the hype-parameter λ in Eq.(5)?
- **RQ4:** What is the effect of training the teacher and student synchronously by sharing common input components?

5.1 Experimental Settings

To better understand the network structure, we give an illustration of all input components used in this work in Figure 5. As mentioned earlier, all features are transformed into categorical type and we learn an embedding for each one. The entities, e.g., user and item, are then represented by the concatenations of their corresponding feature embeddings. Here we adopt the self-attention structure [37] to model the user clicked/purchased history. Let $\mathbf{V} \in \mathbb{R}^{n \times k}$ denote the input values of the sequence, with n being the combined embedding dimension and k being the sequence length. The transformed values $\tilde{\mathbf{V}}$ are then derived as weighted combinations of input $\mathbf{V}$, i.e., $\tilde{\mathbf{V}} = \text{Softmax}\left(\mathbf{Q}^T\mathbf{K}\right)\mathbf{V}$, where the queries $\mathbf{Q}$ and the keys $\mathbf{K}$ are the same as the values $\mathbf{V}$. To improve the effective resolution of combinations, the multi-head structure is adopted, where $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are projected linearly into several subspaces and the transformations are executed, respectively. Note that we also add feed-forward network, skip connections [12], layer normalization [2] to the attention mechanism as done in the original work [37]. For simplicity, we omit the details here. The only difference with the original self-attention is that we do not adopt position encodings in the input. We instead insert several extra features describing the user behavior on the particular item, e.g., the clicked/purchased time from now, the dwell time on the item, etc. Besides encoding the relative positions of the items, the extra features also reflect the importance of the items for future predictions. According to our experiments, adding these features can greatly improve the performance.

After deriving $\tilde{\mathbf{V}}$ with one-layer's self-attention, we adopt the mean pooling operation over the sequence length k. Here we set

Table 1: CTR dataset description at coarse-grained ranking of Taobao recommendations.

Datasets	# Users	#Items	#Clicks	# Impressions
1 Day	9.35×10^7	2.67×10^7	5.03×10^8	1.09×10^{10}
10 Days	2.88×10^8	4.45×10^7	4.57×10^9	9.90×10^{10}

Table 2: Testing AUC of different methods for CTR at coarse-grained ranking. We do not include MTL as it is too cumbersome to predict dozens of privileged features. Due to the huge training cost, we only compare PFD+MD with the baseline in the dataset of 10 days.

Methods	Dataset of 1 Day		Dataset of 10 Days	
	Student	Teacher	Student	Teacher
Baseline	0.6625	–	0.7042	–
LUPI [25]	0.6637	0.6687	–	–
MD [14]	0.6704	0.6892	–	–
PFD	0.6712	0.6921	–	–
PFD+MD	**0.6745**	0.7110	**0.7160**	0.7411

k to 50, the number of heads to 4, and the subspace dimension to 32. As illustrated in Figure 5, the input components are fed to the corresponding teacher and student networks, which consists of several fully-connected layers. We use LeakyReLU [26] as the activation and insert batch normalization [18] before it. The models are trained in the parameter servers with the asynchronous Adagrad optimizer [10]. In the first one million steps, the learning rate is increased linearly to the pre-defined value 0.01, which is then kept fixed. We set the batch size to 1024 and the number of epoch to 1. We train the teacher and student synchronously by sharing common input components. Unless stated otherwise, λ is set to 0.5 and the swapping step k in Algorithm 1 is set to 10^6.

As the labels are in 1 or 0, i.e., whether the users clicked/purchased the item or not, we use the log-loss for both the teacher and the student, i.e.,

$$L_{t/s} \triangleq \frac{1}{N}\sum_{i=1}^{N}\left(y_i \log\left(1+e^{-f_{t/s,i}}\right)+(1-y_i)\log\left(1+e^{f_{t/s,i}}\right)\right), \quad (6)$$

where $f_{t/s,i}$ denotes the output of the i-th sample from the teacher or student model. For the distillation loss L_d, we use the cross entropy, i.e., by replacing y_i in the above equation with $1/(1+e^{-f_{t,i}})$. Here we measure the performance of models with the widely-used areas under the curve (AUC) in the next-day held-out data.

5.2 CTR at Coarse-grained Ranking (RQ1-2)

Across this work, we conduct experiments using the traffic logs of the Guess You Like scenario in the front page of Taobao app. The dataset description of CTR at coarse-grained ranking is summarized in Table 1. For the inner-product model, the user mapping $\mathbf{\Phi}_{\boldsymbol{W}^u}(\cdot)$ and the item mapping $\mathbf{\Phi}_{\boldsymbol{W}^i}(\cdot)$ in Eq.(3) are formulated as: Input->512-> Act.->256->Act.->128->ℓ_2-normalize. When executing inner-product between $\mathbf{\Phi}_{\boldsymbol{W}^u}(\cdot)$ and $\mathbf{\Phi}_{\boldsymbol{W}^i}(\cdot)$, we additionally multiply it with a scalar, i.e., 5, in compensation for the shrinkage of value after normalization. In LUPI [25], MD [14], and PFD+MD, the teacher networks use 3-layers' MLP, with the number of hidden neurons being 512, 256, and 128, respectively. In PFD, we use the inner-product model for both the teacher and the student, where the privileged features are put at the user side of the teacher. The teacher and the student of PFD (and PFD+MD) share all common input components except user id. We do not include MTL as it is too cumbersome to predict dozens of privileged features.

Off-line and on-line performance. The testing AUC of all compared distillation techniques are shown in the left part of Table 2. Compared the teacher of PFD with the baseline, we confirm the effectiveness of the interacted features. By distilling knowledge from these features, we improve the testing AUC from 0.6625 to 0.6712. Although both utilizing the same privileged features, PFD is superior to LUPI. This is because that the teacher of PFD additionally processes the regular features, which results in more accurate predictions to instruct the student than using privileged features alone as done in LUPI (0.6921 v.s. 0.6687). Similarly, we achieve further improvements with PFD+MD by forming a more accurate teacher. In order to validate whether the superiority of PFD+MD could still holds when training longer with more data. We conduct experiments on the traffic logs of 10 days. Due to the huge training cost, we only compare PFD+MD with the baseline. As shown in the right part of Table 2, the student of PFD+MD surpasses the baseline AUC with a margin, i.e., +0.0118. We also conduct on-line A/B tests to validate its effectiveness. Compared with the baseline, PFD+MD consistently improves the click metric by +**5.0**%. And we have **fully deployed** the technique into production.

Cost of directly utilizing interacted features. As discussed earlier, the interacted features are prohibited for the inner-product model during inference. With such features, we need to compute the mappings $\Phi_{\boldsymbol{W}}(\cdot)$ as many times as the number of candidates, i.e., 10^5. In contrast, without such features, we only need to compute the mapping once and execute its inner product with all candidates efficiently. Suppose that the input to $\mathbf{\Phi}_{\boldsymbol{W}}(\cdot)$ has 1024 dimensions. Theoretically, we need $1024\times512+512\times256+256\times128 \approx 6.9\times10^5$ fused multiply-add flops to get one mapping. In comparison, executing one inner product operation needs 128 flops, which is $\sim$ 5400× less. We also conduct simulated experiments in the personal computer. We repeat 10^5 times to simulate the mapping inference and the inner product operation, which totally costs 89.695s and 0.108s, respectively. Executing the mapping is $\sim$ 830× slower than the inner product operation.

Table 3: CVR dataset description at fine-grained ranking of Taobao recommendations

Datasets	#Users	#Items	#Purchases	#Clicks
30 Days	2.78×10^8	3.74×10^7	6.97×10^7	1.40×10^{10}
60 Days	3.36×10^8	5.27×10^7	1.32×10^8	2.71×10^{10}

5.3 CVR at fine-grained ranking (RQ1-2)

In the above CTR prediction, we use the traffic logs of all impressions. While in the CVR prediction, we extract the traffic logs of all clicks. The datasets are summarized in Table 3. We adopt 3-layers' MLP for the baseline and all of the student networks, with the number of hidden neurons being 512, 256, and 128, respectively. The

Table 4: Testing AUC of different methods for CVR prediction at fine-grained ranking.

Methods	Dataset of 30days		Dataset of 60days	
	Student	Teacher	Student	Teacher
Baseline	0.9040	–	0.9082	–
MTL [32]	0.9045	–	0.9077	–
LUPI [25]	0.8965	0.9651	0.9003	0.9659
MD [14]	0.9052	0.9058	0.9093	0.9103
PFD	**0.9084**	0.9901	0.9135	0.9923
PFD+MD	0.9082	0.9911	**0.9138**	0.9929

Table 5: Students' testing AUC of different hyper-parameters λ in the 1 day's dataset of CTR. The superscript $^{+}/^{-}$ indicates the highest/lowest AUC of each method among all chosen hyper-parameters.

λ	0.1	0.3	0.5	0.7	0.9
LUPI [25]	0.6648^{+}	0.6640	0.6637	0.6631	0.6624^{-}
MD [14]	0.6695^{-}	0.6697	0.6704	0.6706^{+}	0.6700
PFD	0.6711	0.6709	0.6712^{+}	0.6700	0.6696^{-}
PFD+MD	0.6741	0.6740	0.6745	0.6747^{+}	0.6739^{-}

Table 6: Students' testing AUC of different hyper-parameters λ in the 30 days' dataset of CVR.

λ	0.1	0.3	0.5	0.7	0.9
LUPI [25]	0.9024^{+}	0.8998	0.8965	0.8876	0.8613^{-}
MD [14]	0.9047^{-}	0.9054^{+}	0.9052	0.9050	0.9049
PFD	0.9081^{-}	0.9082	0.9084^{+}	0.9082	0.9082
PFD+MD	0.9081	0.9085^{+}	0.9082	0.9083	0.9080^{-}

teacher networks of PFD and LUPI also have the same structure as their students. In MD and PFD+MD, their teacher networks are expanded to 7-layers' MLP, with the number of hidden neurons being 8192, 4096, 2048, 1024, 512, 256, and 128, respectively. We also compare with MTL. Here we adopt the hard parameter sharing version of MTL [32], where all tasks share first hidden layer and independently predict each task with 2-layers' MLP. We adopt the mean squared error for the auxiliary continuous regression tasks, e.g., predicting the dwell time, and the log-loss for the auxiliary binary prediction tasks, e.g., predicting whether viewing the user comments or not. The hyper-parameters are chosen empirically from $\{0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2\}$ with the principle of keeping all auxiliary losses balanced.

Off-line and on-line performance. The results of all testing methods are shown in Table 4. Among them, LUPI gets the worst performance, although the testing AUC of its teacher is rather high. The reason of the inferiority has been discussed in Section 4. The experiments also confirm the positive effects of adding regular features to the teacher model, where PFD improves the baseline by +0.0044 AUC in the dataset of 30 days and +0.0053 AUC in the dataset of 60 days, respectively. Compared with PFD, PFD+MD has no distinct superiority. This is mainly due to that the improvement of MD over the baseline is moderately small. In practice, PFD is thus preferred over PFD+MD as costing much less computation resources. We conduct on-line A/B tests to validate the effectiveness of PFD. Compared with the baseline, PFD steadily improves the conversion metric by **+2.3%** over a long period of time.

Table 7: Effects of training PFD+MD in different manners in the 1 day's dataset of CTR. Ind&Async denotes that the teacher and the student are trained asynchronously with independent input components. Share&Sync denotes that the teacher and the student are trained synchronously with shared common input components. The superscript * means that all common input components are shared except user id. We also record the wall-clock time in hours in the forth column.

	Student	Teacher	Time	Relative
Baseline	0.6625	–	9.24 h	0%
Ind&Async	0.6751	0.7112	18.43 h	+99.5%
Ind&Sync	0.6748	0.7112	14.32 h	+55.0%
Share&Sync	0.6717	0.7108	9.51 h	+2.9%
Share*&Sync	0.6745	0.7110	10.29 h	+11.4%

Table 8: Effects of training PFD+MD in different manners in the 30 days' dataset of CVR. The row with superscript † is the result of PFD.

	Student	Teacher	Time	Relative
Baseline	0.9040	–	12.22 h	0%
Ind&Async	0.9067	0.9887	26.85 h	+119.7%
Ind&Sync	0.9069	0.9887	20.56 h	+67.4%
Share&Sync	0.9082	0.9911	14.97 h	+22.5%
Share&Sync†	0.9084	0.9901	12.67 h	+3.6%

5.4 Ablation Study (RQ3-4)

Sensitivity of Hyper-parameter. In the above experiments, we fix the the hyper-parameter λ at 0.5 for all distillation techniques. Here we test the sensitivity of λ. The results of CTR prediction are shown in Table 5. Most of the distillation techniques surpass the baseline, i.e., 0.6625, among all chosen hyper-parameters, except LUPI with $\lambda = 0.9$. MD, PFD, and PFD+MD are all robust to varying λ. Even their worst results improve the baseline by margins. We also conduct experiments in the CVR dataset. As shown in Table 6, LUPI narrows the gap with the baseline, i.e., 0.9040 as λ decreases, while it is still inferior to the baseline. MD, PFD, and PFD+MD are again robust to varying λ in this huge-scale dataset.

Effects of Training Manner. In the above experiments, the teacher and the student are trained synchronously by sharing common input components. Here we test the effects of such training manner. The results of CTR prediction are shown in Table 7. Training the teacher and the student synchronously achieves almost the same performance as training asynchronously (Ind&Sync v.s. Ind&Async). When training the two models by sharing all common input components, the performance of the student degenerates. As the privileged features, i.e., the interacted features between the user and her clicked/purchased items, reflect the user's personal

interests, we allocate an independent user id embedding to the student, in order to absorb the extra preferences distilled from the privileged features. The result after such modification is shown in the row of Share*&Sync, where the performance degeneration is much alleviated and only a little extra wall-clock time is introduced. We also test the effects of different training manners in the CVR dataset. The results are shown in Table 8. As indicated in the rows of Share&Sync and Ind&Sync, the teacher can be additionally improved by sharing common input components. Consequently, the student is improved by distilling knowledge from the more accurate teacher. In the last row of Table 8, we report the result of PFD. Compared with the baseline, it adds almost no extra wall-clock time while achieves similar performance to PFD+MD in the row of Share&Sync. Overall, by adopting PFD+MD for CTR and PFD for CVR, we can achieve much better performance while add no burden to training time. Therefore, the technique is even adoptable for online learning on the streaming data [27], where real-time computation is in high demand.

6 CONCLUSION

In this work, we identify the privileged features existing at Taobao recommendations, i.e., the interacted features for CTR at coarse-grained ranking and the post-event features for CVR at fine-grained ranking. And we propose PFD to leverage them. Different from the traditional LUPI, PFD additionally processes the regular features in the teacher, which is shown to be the core for its success. We also propose PFD+MD to utilize the complementary feature and model capacities to better instruct the student. And it achieves further improvements. The effectiveness is validated on two fundamental prediction tasks at Taobao recommendations, where the baselines are greatly improved by the proposed distillation techniques. During the on-line A/B tests, the click metric is improved by **+5.0%** in the CTR task. And the conversion metric is improved by **+2.3%** in the CVR task. We also address several issues of training PFD, which lead to comparable training speed as the baselines without any distillation.

REFERENCES

[1] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E Dahl, and Geoffrey E Hinton. 2018. Large scale distributed neural network training through online distillation. In *ICLR*.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv:1607.06450* (2016).

[3] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *SIGKDD*. ACM, 535–541.

[4] Xu Chen, Yongfeng Zhang, Hongteng Xu, Zheng Qin, and Hongyuan Zha. 2018. Adversarial distillation for efficient recommendation with external knowledge. *ACM Transactions on Information Systems* 37, 1 (2018), 1–28.

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

[6] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *RecSys*. ACM, 191–198.

[7] George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* 2, 4 (1989), 303–314.

[8] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large scale distributed deep networks. In *NeurIPS*. 1223–1231.

[9] Mukund Deshpande and George Karypis. 2004. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems* 22, 1 (2004), 143–177.

[10] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.

[11] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A factorization-machine based neural network for CTR prediction. In *AAAI*. 1725–1731.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.

[13] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In *ICLR*.

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

[15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[16] Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural Networks* 4, 2 (1991), 251–257.

[17] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*. ACM, 2333–2338.

[18] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*. 448–456.

[19] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *ICDM*. IEEE, 197–206.

[20] Yoon Kim and Alexander M. Rush. 2016. Sequence-Level Knowledge Distillation. In *EMNLP*. 1317–1327.

[21] John Lambert, Ozan Sener, and Silvio Savarese. 2018. Deep learning under privileged information using heteroscedastic dropout. In *CVPR*. 8886–8895.

[22] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xDeepFM: Combining explicit and implicit feature interactions for recommender systems. In *SIGKDD*. ACM, NY, USA, 1754–1763.

[23] Henry W Lin, Max Tegmark, and David Rolnick. 2017. Why does deep and cheap learning work so well? *Journal of Statistical Physics* 168, 6 (2017), 1223–1247.

[24] Shichen Liu, Fei Xiao, Wenwu Ou, and Luo Si. 2017. Cascade ranking for operational e-commerce search. In *SIGKDD*. ACM, 1557–1565.

[25] David Lopez-Paz, Léon Bottou, Bernhard Schölkopf, and Vladimir Vapnik. 2016. Unifying distillation and privileged information. In *ICLR*.

[26] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, Vol. 30. 3.

[27] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. 2013. ftrl. In *SIGKDD*. 1222–1230.

[28] Asit Mishra and Debbie Marr. 2018. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. In *ICLR*.

[29] Yabo Ni, Dan Ou, Shichen Liu, Xiang Li, Wenwu Ou, Anxiang Zeng, and Luo Si. 2018. Perceive your users in depth: Learning universal user representations from multiple e-commerce tasks. In *SIGKDD*. ACM, 596–605.

[30] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. 2017. Regularizing neural networks by penalizing confident output distributions. *arXiv:1701.06548* (2017).

[31] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2015. Fitnets: Hints for thin deep nets. In *ICLR*.

[32] Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv:1706.05098* (2017).

[33] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *CVPR*. 2818–2826.

[34] Jiaxi Tang and Ke Wang. 2018. Ranking distillation: Learning compact ranking models with high performance for recommender system. In *SIGKDD*. ACM, 2289–2298.

[35] Vladimir Vapnik and Rauf Izmailov. 2015. Learning using privileged information: similarity control and knowledge transfer. *Journal of Machine Learning Research* 16, 2023-2049 (2015), 2.

[36] Vladimir Vapnik and Akshay Vashist. 2009. A new learning paradigm: Learning using privileged information. *Neural Networks* 22, 5-6 (2009), 544–557.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*. 5998–6008.

[38] Xiaojie Wang, Rui Zhang, Yu Sun, and Jianzhong Qi. 2018. Kdgan: Knowledge distillation with generative adversarial networks. In *NeurIPS*. 775–786.

[39] Ying Zhang, Tao Xiang, Timothy M Hospedales, and Huchuan Lu. 2018. Deep mutual learning. In *CVPR*. 4320–4328.

[40] Guorui Zhou, Ying Fan, Runpeng Cui, Weijie Bian, Xiaoqiang Zhu, and Kun Gai. 2018. Rocket launching: A universal and efficient framework for training well-performing light net. In *AAAI*.

[41] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *SIGKDD*. ACM, 1059–1068.