Prototype Feature Extraction for Multi-task Learning

Shen Xin∗
Nanyang Technological University
Singapore
sxstar@zju.edu.cn

Yuhang Jiao∗
Kuaishou Technology
China
jiaoyuhang@email.cufe.edu.cn

Cheng Long
Nanyang Technological University
Singapore
c.long@ntu.edu.sg

Yuguang Wang
Nanyang Technological University
Singapore
WANG1520@e.ntu.edu.sg

Xiaowei Wang
Kuaishou Technology
China
wangxiaowei03@kuaishou.com

Sen Yang
Kuaishou Technology
China

Ji Liu
Kuaishou Technology
USA
jiliu@kwai.com

Jie Zhang
Nanyang Technological University
Singapore
ZhangJ@ntu.edu.sg

ABSTRACT
Multi-task learning (MTL) has been widely utilized in various industrial scenarios, such as recommender systems and search engines. MTL can improve learning efficiency and prediction accuracy by exploiting commonalities and differences across tasks. However, MTL is sensitive to relationships among tasks and may have performance degradation in real-world applications, because existing neural-based MTL models often share the same network structures and original input features. To address this issue, we propose a novel multi-task learning model based on Prototype Feature Extraction (PFE) to balance task-specific objectives and inter-task relationships. PFE is a novel component to disentangle features for multiple tasks. To better extract features from original inputs before gating networks, we introduce a new concept, namely prototype feature center, to disentangle features for multiple tasks. The extracted prototype features fuse various features from different tasks to better learn inter-task relationships. PFE updates prototype feature centers and prototype features iteratively. Our model utilizes the learned prototype features and task-specific experts for MTL. We implement PFE on two public datasets. Empirical results show that PFE outperforms state-of-the-art MTL models by extracting prototype features. Furthermore, we deploy PFE in a real-world recommender system (one of the world’s top-tier short video sharing platforms) to showcase that PFE can be widely applied in industrial scenarios.

CCS CONCEPTS
• Computing methodologies → Multi-task learning; Neural networks; • Information systems → Recommender systems.

∗Both Shen Xin and Yuhang Jiao are corresponding authors.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
WWW ‘22, April 25–29, 2022, Virtual Event, Lyon, France
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9096-5/22/04...
https://doi.org/10.1145/3485447.3512119

KEYWORDS
Multi-task Learning, Recommender System, Neural Network

ACM Reference Format:

1 INTRODUCTION
With the rapid development of Deep Neural Networks (DNNs), numerous real-world applications achieve better performance by introducing DNN models. For example, in e-commerce, we can utilize DNN to capture user preference and item attractiveness based on user profiles, item profiles, and user-item interactions. Obtained user embeddings and item embeddings can be used for various prediction tasks, such as Click-Through Rate (CTR) prediction, Conversion Rate (CVR) prediction, etc. In real-world applications, we often have some related tasks in one industrial scenario. Multi-Task Learning (MTL) strategy is widely used to train a general network for these related tasks simultaneously.

Several DNN-based MTL models have been proposed [4]. They improve learning efficiency by sharing one single network and parameters. Theoretically, knowledge learned from one task can help other related tasks be learned better. Besides, MTL can also improve generalization and representation transferability.

However, in real-world applications, MTL is sensitive to relationships among tasks and may have performance degradation, when some tasks are not strongly correlated, or are even contradictory. Prior studies investigate task differences in MTL and address the negative transfer problem. For example, some works [14][24][28] add many model parameters for each task to accommodate task differences and learn linear combinations for different tasks. But these methods are rarely applied in real applications because a large number of additional parameters cause low model robustness and high time complexity. They do not measure task differences explicitly. And using static weights for all tasks harms predictions because of the inherent conflicts from task differences.

More recently, a general model, called Multi-gate Mixture-of-Experts (MMoE) [23], is proposed to solve the negative transfer problem by introducing gating networks to adjust parameterization
We introduce prototype feature centers to better extract prototype features. Some tools provide templates with special effects. These also improve user experience not only for video consumers but also for video producers. Specifically, in the short video recording stage, some tasks will be completed, namely 1) clicking the template, 2) recording the video, and 3) saving the work. Therefore, when designing a system to recommend templates to producers, it is important to consider the three tasks of clicking, recording and saving simultaneously. To meet these business needs, we extend and deploy the proposed PFE on this novel multi-task recommender scenario. Both online and offline results show that PFE can successfully improve the key performance indicators of the template recommendation business.

We summarize the major contributions of this paper as follows:

1. We propose a novel Multi-Task Learning (MTL) model, namely Prototype Feature Extraction (PFE), to obtain disentangled prototype features of each sample as shared information to capture complicated task correlations explicitly. PFE learns prototype features and prototype feature centers iteratively and combines extracted prototype features and task-specific experts for multiple tasks.
2. We study PFE on two public datasets, the AliExpress dataset and the US Census-Income dataset. Experimental results demonstrate that PFE outperforms corresponding single-task models and state-of-the-art MTL models.
3. We extend and deploy PFE on a novel multi-task recommender scenario in one of the world’s largest short video sharing platforms. Both online and offline results show the effectiveness of our method.

2 RELATED WORK

2.1 Multi-task Learning

MTL is a commonly used solution when there are multiple related tasks. A single-task model only learns features associated with one task, which misses and discards the information of other tasks. On the contrary, MTL allows knowledge sharing and transferring.

Hard parameter sharing is an early MTL model, which shares hidden layers among all tasks and reduces the risk of overfitting [6]. Differing from the hard parameter sharing model, soft parameter sharing networks let each task have its own model with its own parameters and use different regularizations, e.g., $l_2$ norm [14] and trace norm [36], to encourage the parameters to be similar. To solve task conflicts, some models, e.g., cross-stitch network [24] and sluice network [28], allow task-specific networks to leverage the knowledge of other tasks by learning a linear combination of the outputs of the previous layers. These models all use static weights for all samples.

Gate structures and attention mechanisms are widely studied and applied in MTL for information fusion. The original Mixture-of-Experts (MoE) model shares some experts at the bottom hidden layers across tasks and combines the experts by a gating network [17]. Deep MoE [15] and Sparsely-Gated MoE [29] regard MoE as a basic network unit and stack them in DNNs. Recently, MMoE is proposed to extend MoE to use multiple gates for every task to learn different fusing weights for a combination of shared experts, which
explicitly learns to model task relationships from data [23]. MMoE is widely applied in real-world applications nowadays. Based on MMoE, Progressive Layered Extraction (PLE) separates task-sharing and task-specific parameters explicitly [31]. Though these models are designed to better learn the adaptive weights for experts, they all use the original input data from samples and still face the negative transfer problem.

MTL models have been successfully used across various applications of machine learning problems, such as natural language processing [11], speech recognition [13], and computer vision [16]. Besides, an increasing number of the global large internet companies are applying MTL strategy in various real-world scenarios. For example, Google proposed a multilingual neural machine translation system to enable zero-shot translation [18]. Microsoft developed a DNN-based MTL model for representation learning, focusing on semantic classification and semantic information retrieval [21]. Alibaba utilized MTL to make sale-related predictions and traffic allocation for online promotions [32][35], to solve cold-start problems in food delivery recommendations [33] and to obtain user representations based on user behavior sequences for recommendations [25].

3 PROTOTYPE FEATURE EXTRACTION FOR MULTI-TASK LEARNING

3.1 Preliminaries

Previous MTL models feed original features to experts. Simply concatenating different types of features may lead to a negative transfer problem. To better learn the complicated task correlations, we introduce a model to extract prototype features from various original features. The disentangled prototype features help to improve predictions of networks. We first briefly introduce MMoE [23] and PLE [31], which are the basis of our work. The original MoE model can be formulated as:

\[ y = \sum_{i=1}^{n} g(x)_{i} f_{i}(x) \]  

where \( n \) is the number of experts and an expert is a group of bottom networks, \( g(x)_{i} \) indicates the probability or weight for expert \( f_{i} \) and \( \sum_{i=1}^{n} g(x)_{i} = 1 \). In this model, \( f(x) \) are \( n \) expert networks and \( g(x) \) is a gating network ensembling results from all experts.

Multi-gate Mixture-of-Experts (MMoE) improves MoE by adding a gating network \( g^{k} \) for each task \( k \). We define \( g^{k} \) as the network of the \( k \)-th task. The output of task \( k \) is:

\[ y_{k} = g^{k}(f^{k}(x)) \]  

where

\[ f^{k}(x) = \sum_{i=1}^{n} g^{k}(x)_{i} f_{i}(x) \]  

and \( f_{i}(x) \) is the output of the \( i \)-th shared expert.

The weighting function calculates the weight vector of task \( k \) through linear transformation and a softmax layer as:

\[ g^{k}(x) = \text{softmax}(W_{g}^{k} x) \]  

where \( W_{g}^{k} \in \mathbb{R}^{n \times d} \). Here, \( n \) indicates the number of experts and \( d \) indicates the feature dimension.

PLE improves MMoE by separating experts into shared experts and task-specific experts. As a result, the parameter matrix becomes \( W_{g}^{k} \in \mathbb{R}^{(M_{s} + M_{k}) \times d} \). Here, \( M_{s} \) is the number of shared experts and \( M_{k} \) is the number of task \( k \)’s specific experts.

3.2 Prototype Feature Extraction

The existing models for MTL from a hard parameter sharing network to attention mechanism-based models, e.g., MMoE and PLE, all try to solve the negative transfer problem. However, they focus on separating the bottom sharing hidden layer and do not distinguish different types of original input features, because they apply attention mechanisms after experts and simply concatenate input data for multiple tasks when training experts.

In fact, the negative transfer problem not only happens on the bottom layer but is also a result of the input features. For instance, on a popular public MTL dataset, the UCI census-income, when considering the tasks of income prediction and marital status prediction, we can intuitively partition input features into three groups, namely 1) income-related features, e.g., occupation, 2) family-related features, e.g., family relationship, and 3) shared features, e.g., age. Under these circumstances, mixing income-related features and family-related features may lead to the negative transfer problem.

Motivated by some feature extraction strategies applied in graph convolution networks [22][21][3], we propose a novel prototype feature extraction for MTL. The notations and problem formulation are as follows.

We define \( x \) as an input of one sample, \( a_{i} \in \mathbb{R}^{d} \) as the output of the \( i \)-th shared expert, where \( d \) is the number of dimensions of the features obtained by experts. We concatenate \( a_{i} \) into a feature vector \( a \in \mathbb{R}^{d_{in}} \), where \( d_{in} \) is the number of dimensions of PFE’s input features. PFE is a layer which is fed with a sample \( a \) as its input and outputs prototype features for this sample:

\[ z = h(a) \]  

The output of PFE, \( z \in \mathbb{R}^{d_{out}} \), are the disentangled prototype features learned by the layer. We have \( d_{in} = d_{out} = d * M_{s} \), where \( M_{s} \) is the number of shared experts.
To extract features, we propose a concept of a prototype feature center. Suppose that there are \( L \) latent groups of features. We introduce \( L \) prototype feature centers to disentangle the feature space. We define \( \{c_i\}_{i=1}^{L} \) as the prototype feature centers, where \( c_i \in \mathbb{R}^{d_a} \).

The problem turns to be one of learning \( \{c_i\}_{i=1}^{L} \) and assigning \( a \) to these centers.

Motivated by an Expectation-Maximization (EM) algorithm for clustering, we propose a two-step algorithm to solve this problem. Let \( p_{i,j} \) be the probability that the output of the \( i \)-th shared expert belongs to the \( j \)-th prototype feature cluster, where \( i \in \{1, ..., M_b\} \) and \( j \in \{1, ..., L\} \). \( p_{i,j} \) can also be treated as the probability that we should use the output of the \( i \)-th shared expert to construct the \( j \)-th prototype feature center. Based on the assumption that \( c_i \) is independent of each sample, we initialize \( c_j \) as \( L \)-dimension network parameters randomly. We update \( p \) and \( c \) iteratively as follows:

\[
\begin{align*}
    p_{i,j}^{(t)} &= \frac{\exp\left(\frac{a_i^T}{\|a_i\|^2} c_j^{(t-1)}\right)}{\sum_{j'=1}^{L} \exp\left(\frac{a_i^T}{\|a_i\|^2} c_{j'}^{(t-1)}\right)} \\
    c_j^{(t)} &= \frac{c_j^{(t-1)} + \sum_{i=1}^{M_b} p_{i,j}^{(t)} a_i}{\|c_j^{(t-1)} + \sum_{i=1}^{M_b} p_{i,j}^{(t)} a_i\|_2}
\end{align*}
\]

where \( t \in \{1, ..., T\} \) indicates the iteration. We regard \( c \) as the prototype feature center to represent different feature spaces. The outputted prototype feature centers should directly relate to the disentangled shared experts. Thus, we formulate the output of PFE as follows:

\[
z = \text{concat}(c_1, c_2, ..., c_L)
\]

where

\[
c_j = \sum_{i=1}^{M_b} p_{i,j}^{(T)} a_i
\]

We summarize the two-step training operations for PFE in Appendix A.

### 3.3 General Multi-task Learning Network with Prototype Feature Extraction

To address the negative transfer problem, we propose a general multi-task network structure with prototype feature extraction to capture complicated task correlations based on both experts and input features. Figure 2 shows the overall network structure of our model. There are \( M_A \) experts for Task A, \( M_B \) experts for Task B, \( M_S \) shared experts and \( L \) prototype feature centers. \( E(A,i) \), \( E(B,i) \), \( E(S,i) \) and \( C_i \) indicate the \( i \)-th specific expert of Task A, the \( i \)-th specific expert of Task B, the \( i \)-th shared expert and the \( i \)-th prototype feature center, respectively.

We divide the general MTL network into two parts, namely a bottom module and a top module. The bottom module has some task-specific tower networks to output the results for each task. We combine prototype feature extraction and task-specific experts in the bottom module. Task-specific experts learn particular patterns for each task and PFE learns shared patterns for all tasks.

The tower network for each task discovers the knowledge learned from both PFE and its own specific experts. In this way, the extracted prototype features, which are the outputs of PFE, are utilized and affected by all tasks; and task-specific experts are utilized and affected by its corresponding task.

We use a gating network for each task to combine the prototype features learned by PFE and specific experts. The output of the gating network for the \( k \)-th task is defined as follows:

\[
g^k(x) = \text{softmax}(W^k x) g^k(x)
\]

where \( x \in \mathbb{R}^d \) is the input representation indicating one extracted prototype feature or one output of a task-specific expert; \( W^k \in \mathbb{R}^{(M_S + L) \times d} \) is the weighting parameter matrix for \( M_k \) task-specific experts and \( L \) extracted prototype features respectively; and \( g^k(x) \) is the concatenation of \( M_k \) task-specific experts and \( L \) prototype features, which is defined as follows:

\[
g^k(x) = [E_{(k,1)}^{T}, ..., E_{(k,M_k)}^{T}] c_1, c_2, ..., c_L]^{T}
\]

Finally, we obtain the output of the \( k \)-th task as follows:

\[
y_k(x) = q^k(g^k(x))
\]

where \( q^k \) represents the tower network of the \( k \)-th task.
We conduct experiments on two public datasets, namely the AliExpress dataset (AE) and the US Census-Income dataset (UCI), both of which are publicly available and evaluated widely.

AE is collected from traffic logs of AliExpress search system. There are 3.6 million users, 26.5 million items, 2.3 billion interactions, including click and purchase. The meanings of features are omitted for the reason of data security. There are 5 representative countries, namely Russia (RU), Spain (ES), French (FR), Netherlands (NL) and America (US). In each country, Task 1 is to predict whether the income exceeds $50k and Task 2 is to predict the marital status. In our experiments, we treat the marital status task (Task 2) as the auxiliary task in both groups, while the income task and the education task are regarded as the main task. UCI is not a good dataset for comparisons, as most previous works report on UCI, so we still keep this dataset.

### 4 EXPERIMENTS AND RESULTS

#### 4.1 Data Description

We conduct experiments on two public datasets, namely the AliExpress dataset (AE) and the US Census-Income dataset (UCI), both of which are publicly available and evaluated widely.

AE is collected from traffic logs of AliExpress search system. There are 3.6 million users, 26.5 million items, 2.3 billion interactions, including click and purchase. The meanings of features are omitted for the reason of data security. There are 5 representative countries, namely Russia (RU), Spain (ES), French (FR), Netherlands (NL) and America (US). In each country, Task 1 is to predict whether the income exceeds $50k and Task 2 is to predict the marital status. In our experiments, we treat the marital status task (Task 2) as the auxiliary task in both groups, while the income task and the education task are regarded as the main task. UCI is not a good dataset for comparisons, as most previous works report on UCI, so we still keep this dataset.

#### 4.2 Compared Algorithms and Experimental Settings

**4.2.1 Compared Algorithms.** To verify the performance of our algorithm, we compare with some baseline methods, including the shared-bottom model and the One-gate Mixture-of-Experts (OMoE) model. We implement some state-of-the-art methods including Multi-gate Mixture-of-Experts (MMoE) [23] and Progressive Layered Extraction (PLE) [31] for comparisons. Besides, we also train single-task models for all three tasks separately, i.e., income prediction, education prediction and marital status prediction, to check whether our PFE can solve the negative transfer problem. Moreover, we conduct ablation studies on all datasets. PFE without prototype features (PFE-ouP) uses traditional experts to directly learn from original features. Comparisons between PFE and PFE-ouP can demonstrate the effectiveness of the proposed model.

**4.2.2 Experimental Settings.** For a single expert, we set the hidden dimension as 256. We use 3 layers (256, 16, 1) of Multi-Layer Perceptron (MLP) as a tower network to predict each task. For the shared-bottom model, we implement the shared-bottom network with one expert. For MMoE, we use 8 experts. The parameters of PLE model are set to be the same as MMoE (4 shared experts and 4 task-specific experts) for a fair comparison. For PFE, we set $M_S = L = 4$ and use 4 task-specific experts for each task. We use ReLU for expert activation in all these four methods. All models are optimized using mini-batch Adam Optimizer [19] with batch size 1024. The learning rate is set as 0.0001 and the number of training epochs for each method are hyperparameters optimized by grid search. For feature preprocessing, we use the one-hot encoding method, which is widely used in MMoE public codes, for consistency and fairness.

**4.2.3 Evaluation Metrics.** We take Area Under the Curve (AUC) score as our evaluation metric. For the main task, Task 1, we report the best AUC and mean AUC of all experiments for comparisons. For Task 2, we report the AUC score when Task 1 performs the best and the mean AUC score for all experiments. To compare the performance of each multi-task learning model on Task 2, we also record the mean AUC of Task 2 in all experiments.

#### 4.3 Experimental Results and Analysis

We run each model for 100 times and report the experimental results among these runnings.

**4.3.1 The AE Dataset.** The results are shown in Table 1. Each country is regarded as an independent experiment. In all 5 countries, our PFE achieves the highest AUC among all compared algorithms in terms of the best AUC and the mean AUC of the main task. For example, in terms of the best AUC of the CTCVR prediction in the RU sub-dataset, PFE outperforms the other two state-of-the-art MTL methods, MMoE and PLE, by 1.5% and 1.9%, respectively.

PFE outperforms single-task models among all scenarios. Especially when the main tasks achieve the best results, the auxiliary tasks still defeat single-task models. However, other MTL models...
suffer from the negative transfer problem. For example, in the FR sub-dataset, the AUC of CTR for the single task is higher than OMoE, MMoE and PLE, while our PFE outperforms the single task by 0.5%.

The shared-bottom method has high requirements of task correlation and works better for tasks with high correlation due to a large number of shared parts. However, its performance among all compared methods is the worst due to the serious negative transfer problem. Even it achieves slightly better results on the auxiliary task of ES and US (i.e., 71.51% and 68.29%), its performance on the main task is much worse (i.e., 81.35% and 75.65%), which demonstrates the auxiliary task harms the learning on the main task in the shared-bottom method.

OMoE uses the MoE structure to improve decision-making ability, which provides better explanatory ability than the shared-bottom model. But with the single gating mechanism, it is difficult for the experts to pass information to the lower layer.

MMoE and PLE are proposed to improve MoE by better learning the task-specific information. Compared to the shared-bottom model, both MMoE and PLE have improvement. However, compared to the single-task models, they both face the negative transfer problem. Specifically, when MMoE and PLE achieve the best results for the main task on FR and US, their results for the corresponding auxiliary task are both worse than the single-task model.

PFE-woP is regarded as ablation studies. Compared with PFE-woP, our model improves by 1.94%, 1.06%, 0.77%, 1.88% and 2.66% for five countries respectively. These results demonstrate that our extracted prototype features can improve predictions.

4.3.2 The UCI Dataset. The results are shown in Table 2 and Table 3. In the two groups of experiments, our PFE achieves the highest AUC among all compared algorithms in terms of both the best AUC and the mean AUC. PFE has improvement over MMoE and PLE. MMoE and PLE have similar prediction accuracy, with improvement over the shared-bottom model. OMoE outperforms the baseline method, the shared-bottom model, and is weaker than MMoE and PLE.

PFE outperforms all other models, especially in terms of best AUCs. The experimental results show that when these models have the best performance on Task 1, only PFE outperforms single-task models on Task 2, which demonstrates that PFE can train a better model for all tasks by utilizing disentangled prototype features. For ablation studies, PFE outperforms PFE-woP among both groups.

Considering most previous works report on the popular dataset, UCI, we test PFE on it. Because AUCs are high for all three tasks, especially for the marital status prediction task, we also use the relative difference instead of the absolute difference when making comparisons. Considering the mean AUC results of the marital status prediction task for group 1, MMoE improves OMoE from 0.9832 to 0.9837 by 0.0005 and PFE improves MMoE from 0.9837 to 0.9854 by 0.0017, we can conclude that compared to the improvement from OMoE to MMoE, the improvement made by our PFE from the MMoE is more significant.

In addition, to figure out the influence of iterations, we explore the performance of PFE on UCI dataset at different iterations for the test of hyper-parameter sensitivity. We attach this experiment in Appendix C.

Table 2: Performance on the first group of UCI (%)

<table>
<thead>
<tr>
<th></th>
<th>AUC/Income</th>
<th>AUC/Marital Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Mean w/Best Edu.</td>
</tr>
<tr>
<td>Single-task</td>
<td>93.42</td>
<td>93.09</td>
</tr>
<tr>
<td>Shared-bottom</td>
<td>93.04</td>
<td>92.68</td>
</tr>
<tr>
<td>OMoE</td>
<td>93.27</td>
<td>92.91</td>
</tr>
<tr>
<td>MMoE</td>
<td>93.52</td>
<td>93.23</td>
</tr>
<tr>
<td>PLE</td>
<td>93.56</td>
<td>93.17</td>
</tr>
<tr>
<td>PFE-woP</td>
<td>93.45</td>
<td>93.20</td>
</tr>
<tr>
<td>PFE</td>
<td>94.02</td>
<td>93.49</td>
</tr>
</tbody>
</table>

Table 3: Performance on the second group of UCI (%)

<table>
<thead>
<tr>
<th></th>
<th>AUC/Education</th>
<th>AUC/Marital Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Mean w/Best Edu.</td>
</tr>
<tr>
<td>Single-task</td>
<td>88.39</td>
<td>88.07</td>
</tr>
<tr>
<td>Shared-bottom</td>
<td>88.26</td>
<td>87.93</td>
</tr>
<tr>
<td>OMoE</td>
<td>88.42</td>
<td>88.21</td>
</tr>
<tr>
<td>MMoE</td>
<td>88.54</td>
<td>88.32</td>
</tr>
<tr>
<td>PLE</td>
<td>88.52</td>
<td>88.38</td>
</tr>
<tr>
<td>PFE-woP</td>
<td>88.45</td>
<td>88.16</td>
</tr>
<tr>
<td>PFE</td>
<td>89.00</td>
<td>88.51</td>
</tr>
</tbody>
</table>

5 REAL-WORLD INDUSTRIAL APPLICATIONS

5.1 Backgrounds and Problems

In short video and other User Generated Content (UGC) recommendation fields, there exist two types of roles: producer and consumer. Producers aim to make new contents and consumers spend plenty of time looking for something they prefer. For a long time, it has been the keystone to improve recommended results for consumers. However, encouraging producers to create high-quality content is as much important as recommending to consumers. These two together promote the development of the ecosystem.

Camera tools are widely used in leading short video sharing platforms nowadays. These powerful tools can help producers take pictures or videos, then add special effects with internal templates, e.g., stickers, magic faces, Augmented Reality (AR) effects, etc. In such a scenario, video sharing platforms need to recommend internal templates immediately when producers open camera tools. In general, producers would click a template, take a picture or record a video and save it. It is worth noting that among all templates exposed to a producer, only the template clicked by the producer could be subsequently used for recording. Similarly, only the template used for recording has an opportunity to be saved.

Therefore, for a given producer, if we want to predict the template that the producer is most likely to save, we need to consider whether this template will be clicked by the producer, whether it will be used for recording after the producer’s clicking, and whether the picture or the video using this template will be saved after the producer’s recording. The problem becomes predicting Click-Through Rate (CTR), Record-Through Rate (RTR) and Save-Through Rate (STR) of each template for a given producer. From the perspective of producers as a whole, we pay attention to the following three
The modern recommender system [12] includes two stages, namely
metrics:

\[
CTR = \frac{\sum_{i=1}^{N_u} \mathbb{1}(\text{click}_{u_i} = 1)}{N_u} 
\]

\[
RTR = \frac{\sum_{i=1}^{N_u} \mathbb{1}(\text{record}_{u_i} = 1)}{\sum_{i=1}^{N_u} \mathbb{1}((\text{click}_{u_i} = 1))} 
\]

\[
STR = \frac{\sum_{i=1}^{N_u} \mathbb{1}(\text{save}_{u_i} = 1)}{\sum_{i=1}^{N_u} \mathbb{1}(\text{record}_{u_i}) = 1} 
\]

where \(N_u\) is the total count of producers, \(\mathbb{1}(\cdot)\) is the indicator function, \(\text{click}_{u_i}, \text{record}_{u_i}, \text{and save}_{u_i}\) stand for if producer \(i\) clicks one template, records and saves pictures or videos with the template, respectively.

As mentioned above, CTR/RTR/STR tasks are closely related to each other, however, there are still some differences among these tasks. For example, the appearance of a template’s icon could affect CTR task, i.e., templates with fancy icons are more likely to be clicked than templates with unsightly icons, but the impact of icons on RTR or STR task may be little. On the other hand, special effects contained in templates may have a significant impact on a producer’s recording and saving behaviors. If a template’s special effect is simple and easy to use, it can greatly improve the quality of a video and the efficiency of recording, but this special effect cannot be seen before a producer clicks on the template to trigger the preview function, which means the click behavior cannot be affected by the special effects if the producer has never used that template.

Besides handling negative transfer problem, PFE can also extract generalizable features based on coarse clusters from the shared information for different tasks. Motivated by the strong correlation and the obvious differences among click/record/save tasks, we extend and deploy our PFE model to a real-world industrial template recommendation scenario and conduct massive experiments to show its high effective capability of shared feature generations.

### 5.2 Extended PFE Model for the Template Recommendation Scenario

The modern recommender system [12] includes two stages, namely matching and ranking. Matching refers to a process of returning thousands of candidate items out of hundreds of millions of items in the full set to the ranking stage. Ranking is to sort the thousands of candidate items according to the objectives of some specific tasks and returns top ranked items to users. Our goal is to predict the templates that producers are most likely to use based on the diverse producers’ previous behaviors and the candidate templates’ profile information. More specifically, we focus on the ranking phase of the template recommendation scenario. The specific tasks are CTR, RTR and STR predictions and the predicted results are denoted as \(P_{CTR}, P_{RTR}, P_{STR}\), respectively. Our online ranking module calculates the candidate templates ranking score according to the following Equation (16) and recommends the top-ranked templates in descending order to the producer:

\[
\text{score} = w_{CTR} \times P_{CTR}^{w_{CTR}} \times P_{RTR}^{w_{RTR}} \times P_{STR}^{w_{STR}} 
\]

where each \(w\) indicates the importance factor of each predicted score, \(w_{CTR}, w_{RTR}\) and \(w_{STR}\) are hyper-parameters optimized based on the online performance.

### Table 4: Positive and Negative Label for Each Task

<table>
<thead>
<tr>
<th>Behavior</th>
<th>CTR Task</th>
<th>RTR Task</th>
<th>STR Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show</td>
<td>Negative</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Click</td>
<td>Positive</td>
<td>Negative</td>
<td>-</td>
</tr>
<tr>
<td>Record</td>
<td>Positive</td>
<td>Positive</td>
<td>Negative</td>
</tr>
<tr>
<td>Save</td>
<td>Positive</td>
<td>Positive</td>
<td>Positive</td>
</tr>
</tbody>
</table>

All these aforementioned tasks are binary classification tasks, whose models are trained with cross-entropy loss, and models’ final outputs are the probability of the click/record/save behavior with respect to a given producer and a template. For the same training dataset extracted from the dumped behavior logs, the label and the validity of each training sample for these three tasks are different. For example, if a training sample indicates the behavior that a template was shown to the producer but not clicked, this training sample is a negative sample for the CTR prediction task and an invalid sample for the RTR task, because according to the Equation (14), the RTR task only considers the template that has been clicked. Specifically, we summarize the positive and negative labels and validity of various behavior samples for these 3 tasks in Table 4.

With the labels of each task at hand, the loss function of the CTR prediction task is defined as \(L_C\):

\[
L_C = \frac{1}{\sum_{i=1}^{N} \delta_i^C} \sum_{i=1}^{N} \delta_i^C (y_i^C \log \hat{y}_i^C + (1 - y_i^C) \log (1 - \hat{y}_i^C)) 
\]

where \(y_i^C \in \{0, 1\}\) is sample \(i\)’s labels of CTR prediction task, \(\hat{y}_i^C\) is the predicted sample \(i\)’s labels of the CTR prediction model, \(\delta_i^C\) indicates whether sample \(i\) is valid for the CTR prediction task, \(N\) is the count of the total samples. Similar with \(L_C\), we also define the loss functions of the RTR and STR prediction tasks as \(L_R\) and \(L_S\), respectively.

As mentioned in Section 5.1, the data in RTR and STR task is far more sparse than that in the CTR one. Therefore, we consider adding the loss weight for these two tasks to enhance the importance of the training samples in these two tasks. The final loss function of our extended PFE model is defined as:

\[
\mathcal{L} = L_C + \lambda_1 L_R + \lambda_2 L_S 
\]

where \(\lambda_1, \lambda_2\) are hyper-parameters and we set \(\lambda_1 = 2\) and \(\lambda_2 = 4\) according to the business bias, which is inversely related to the number of samples for three tasks.

Figure 3 shows the industrial template ranking system with our extended PFE model. We deploy the PFE model with a wide & deep [10] framework. Deep side features are fed into the downstream PFE and task-specific experts, while wide side features are concatenated with the outputs of gate-based networks as inputs of CTR/RTR/STR models to predict samples’ labels. This ranking system contains an offline training module and an online serving module. The offline training module processes dumped logs and converts them into features and labels. The model will periodically store checkpoints during the training process. The online serving module loads checkpoints to restore pre-trained ranking models and predicts the ranking score according to Equation (16). We summarize the training step of the template ranking system with the extended PFE model as Algorithm 2 in Appendix B.
5.3 Experimental Settings and Results

We collect an industrial dataset by sampling the dumped producer behavior logs during 9 days for offline experiments. The dataset contains 319.6 million samples and 6 million producers. We utilize the first 7 days for training and the last 2 days for testing.

5.3.1 Compared Algorithms. Following the settings in Section 4, we replace the PFE MTL module in our recommender system with Shared-bottom, MMOE and PLE as the compared algorithms.

5.3.2 Experimental Settings. All models are optimized using the mini-batch Adam Optimizer with batch size 1024 and learning rate 0.0001. The number of experts and the training steps for each model are hyperparameters tuned through grid search.

5.3.3 Evaluation Metrics. The evaluation metrics are slightly different from those in Section 4. AUC score measures the quality of order by ranking all the templates with predicted CTR/RTR/STR, including intra-user and inter-user orders. Recommender systems in industry focus on user experience, which is more relevant to the goodness of intra-user orders. We use the uAUC scores, which are the average AUC scores of users, as the ranking model’s performance indicators. The uAUC can be calculated by:

\[
uAUC = \frac{1}{N_u} \sum_{i=1}^{N_u} AUC_i
\]

where \(N_u\) is the total count of producers, \(AUC_i\) stands for the AUC score of the ranking model with respect to producer \(i\).

5.3.4 Experimental Results. Although there is a negative transfer problem among tasks as well, we can only deploy one model considering resources and maintainability. Therefore, we report the uAUC performances on CTR/RTR/STR task group of models with the highest average uAUC of each algorithm in Table 5. Our PFE outperforms others in this industrial scenario as well, showing the excellent capability and robustness of this model.

5.3.5 Online A/B Test and Deployments. We not only evaluate our PFE model on offline experiment but also deploy it on an online real-time data engine with 20% serving traffic for 10 days. The training module is implemented in Pytorch and requires 1 GPU (NVIDIA 2080Ti), 30 CPUs with 60G running memory. The serving module is compiled into a C++ inference engine. The online A/B test shows the performance indicator, i.e., number of works saved per producer, has increased by 1.24%, which is a huge improvement for a leading industrial platform.

6 CONCLUSIONS

In this paper, we propose a novel multi-task learning layer, prototype feature extraction (PFE), to obtain disentangled prototype features of each sample. The overall MTL network structure combines the shared information from extracted prototype features and task-specific information from task-specific experts. We utilize a two-step learning solution to update prototype features and prototype feature centers iteratively. PFE is tested on two popular public datasets, UCI and AE. Experimental results demonstrate that PFE outperforms all corresponding single-task models and state-of-the-art MTL models. These results show that PFE addresses the negative transfer problem well. We also extend and deploy PFE on a novel multi-task recommender application in one of the world’s top-tier short video sharing platforms. Both online and offline results show the effectiveness of our method and its capability to extract disentangled prototype features.

ACKNOWLEDGMENTS

This work is partially supported by the MOE AcRF Tier 1 funding (RG90/20) awarded to Dr. Jie Zhang. We would like to thank Kuaishou for valuable datasets and technical support. SHEN Xin would like to express the deepest appreciation to Jingyi Zhang for her continued assistance and encouragement.
A ALGORITHM OF PROTOTYPE FEATURE EXTRACTION

We summarize the two-step training operations for PFE in Algorithm 1.

Algorithm 1: Prototype Feature Extraction

Input: \( a_i \in \mathbb{R}^d \), the features obtained by the \( i \)-th shared expert
Output: \( z \in \mathbb{R}^{dM} \), the extracted prototype feature representation of this sample

1. Initialize \( c_j \) as \( L \times d \)-dimension network parameters randomly;
2. for routing iteration \( t = 1, 2, ..., T \) do
   3. for each shared expert \( i = 1, 2, ..., M_s \) do
      4. \( \hat{p}_{l,i,j}^{(t)} = \frac{a_i^T e_j^{(t-1)}}{\|a_i\|_2} \), where \( j = 1, 2, ..., L; \)
      5. \( \hat{p}_{l,1,1}^{(t)}, ..., \hat{p}_{l,L,1}^{(t)} \) = \( \text{softmax}(\{ \hat{p}_{l,1,1}^{(t)}, ..., \hat{p}_{l,L,1}^{(t)} \}) \);
   4. end
3. end
10. \( c_j = \sum_{i=1}^{M_s} \hat{p}_{l,i,j}^{(T)} a_i \);
11. \( z = \) the concatenation of \( c_1, c_2, ..., c_L \);

B ALGORITHM OF TEMPLATE RANKING SYSTEM TRAINING WITH PFE

We summarize the training step of the template ranking system with the extended PFE model as Algorithm 2. where \( H(\cdot) \) function indicates outputs of gate-based networks, \( f_c(\cdot), f_R(\cdot), f_s(\cdot) \) functions indicate the CTR, RTR, STR prediction scores, respectively.

Algorithm 2: Template Ranking System Training with PFE

Input: \( X_{\text{wide}} \in \mathbb{R}^{d_{\text{wide}}} \), the wide side features obtained from original inputs; \( X_{\text{deep}} \in \mathbb{R}^{d_{\text{deep}}} \), the deep side features obtained from original inputs; \( Y^C, Y^R, Y^S \in \{0, 1\} \), the training labels for CTR/RTR/STR tasks; \( I \in \mathbb{R} \), the iteration interval for saving checkpoints
Output: \( \text{CKPT} \), the saved checkpoint

1. for number of training iteration \( t = 1, 2, ..., T \) do
   2. if \( t \% I = = 0 \) then
      3. Save current model checkpoint \( \text{CKPT} \);
   3. else
      4. \( X_{\text{final}} \equiv \text{CONCAT}(X_{\text{wide}}, H(X_{\text{deep}})) \);
      5. Update model by minimizing loss:
        6. \( L = L_C(f_c(X_{\text{final}}), Y^C) + \lambda_1 L_R(f_R(X_{\text{final}}), Y^R) + \lambda_2 L_S(f_s(X_{\text{final}}), Y^S) \);
   7. end
9. end

C HYPER-PARAMETER SENSITIVITY

To figure out the influence of iterations, we explore the performance of PFE on UCI dataset at different iterations in this section. As shown in Figure 4, the AUC value increases as the number of iterations increases and arrives at the highest point at iteration 4 in both groups. It then decreases as the number of iterations increases further. Therefore, we set the number of iterations as 4 in our experiments. This could be explained as follows. When the number of iterations is small, the method of updating the value of the new parameter with the value of the previous parameter is effective. With the increase of the number of iterations, the prediction ability of the model reaches a peak. In the subsequent iterations, the continuously updated parameters may lead to overfitting, and therefore the AUC value of the model decreases.

In addition, the best AUC of Task 2 is usually not obtained at the same time as that of Task 1. For example, the best value of Task 2 of group 1 is obtained when the number of iterations is 3. But they have a similar trend (first increasing and then decreasing). Therefore a similar explanation applies to Task 2.