Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts

Jiaqi Ma[1*], Zhe Zhao[2], Xinyang Yi[2], Jilin Chen[2], Lichan Hong[2], Ed H. Chi[2]
[1]School of Information, University of Michigan, Ann Arbor [2]Google Inc.
[1]jiaqima@umich.edu [2]{zhezhao, xinyang, jilinc, lichan, edchi}@google.com

ABSTRACT

Neural-based multi-task learning has been successfully used in many real-world large-scale applications such as recommendation systems. For example, in movie recommendations, beyond providing users movies which they tend to purchase and watch, the system might also optimize for users liking the movies afterwards. With multi-task learning, we aim to build a single model that learns these multiple goals and tasks simultaneously. However, the prediction quality of commonly used multi-task models is often sensitive to the relationships between tasks. It is therefore important to study the modeling tradeoffs between task-specific objectives and inter-task relationships.

In this work, we propose a novel multi-task learning approach, Multi-gate Mixture-of-Experts (MMoE), which explicitly learns to model task relationships from data. We adapt the Mixture-of-Experts (MoE) structure to multi-task learning by sharing the expert submodels across all tasks, while also having a gating network trained to optimize each task. To validate our approach on data with different levels of task relatedness, we first apply it to a synthetic dataset where we control the task relatedness. We show that the proposed approach performs better than baseline methods when the tasks are less related. We also show that the MMoE structure results in an additional trainability benefit, depending on different levels of randomness in the training data and model initialization. Furthermore, we demonstrate the performance improvements by MMoE on real tasks including a binary classification benchmark, and a large-scale content recommendation system at Google.

CCS CONCEPTS

• **Computing methodologies** → **Multi-task learning**; **Neural networks**; • **Information systems** → *Recommender systems*;

KEYWORDS

multi-task learning; mixture of experts; neural network; recommendation system

∗ Work done while the first author was an intern at Google Inc.

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220007

ACM Reference Format:
Jiaqi Ma[1*], Zhe Zhao[2], Xinyang Yi[2], Jilin Chen[2], Lichan Hong[2], Ed H. Chi[2]. 2018. Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts. In *Proceedings of The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220007

1 INTRODUCTION

In recent years, deep neural network models have been successfully applied in many real world large-scale applications, such as recommendation systems [11]. Such recommendation systems often need to optimize multiple objectives at the same time. For example, when recommending movies for users to watch, we may want the users to not only purchase and watch the movies, but also to like the movies afterwards so that they'll come back for more movies. That is, we can create models to predict both users' purchases and their ratings simultaneously. Indeed, many large-scale recommendation systems have adopted multi-task learning using Deep Neural Network (DNN) models [3].

Researchers have reported multi-task learning models can improve model predictions on all tasks by utilizing regularization and transfer learning [8]. However, in practice, multi-task learning models do not always outperform the corresponding single-task models on all tasks [23, 26]. In fact, many DNN-based multi-task learning models are sensitive to factors such as the data distribution differences and relationships among tasks [15, 34]. The inherent conflicts from task differences can actually harm the predictions of at least some of the tasks, particularly when model parameters are extensively shared among all tasks.

Prior works [4, 6, 8] investigated task differences in multi-task learning by assuming particular data generation processes for each task, measuring task differences according to the assumption, and then making suggestions based on how different the tasks are. However, as real applications often have much more complicated data patterns, it is often difficult to measure task differences and to make use of the suggested approaches of these prior works.

Several recent works proposed novel modeling techniques to handle task differences in multi-task learning without relying on an explicit task difference measurement [15, 27, 34]. However, these techniques often involve adding many more model parameters per task to accommodate task differences. As large-scale recommendation systems can contain millions or billions of parameters, those additional parameters are often under-constrained, which may hurt model quality. The additional computational cost of these parameters are also often prohibitive in real production settings due to limited serving resource.

In this paper, we propose a multi-task learning approach based on a novel Multi-gate Mixture-of-Experts (MMoE) structure, which

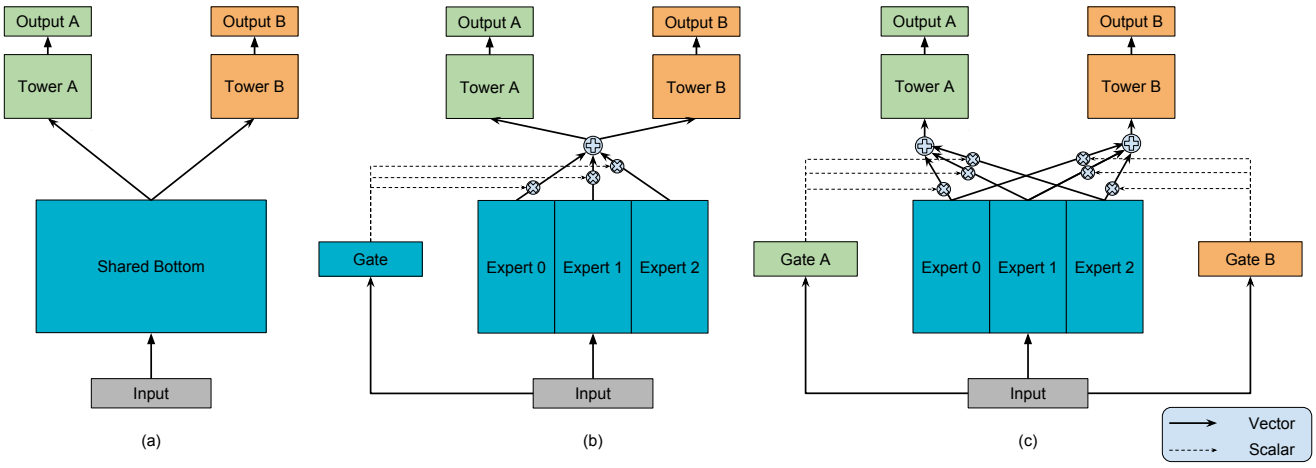


Figure 1: (a) Shared-Bottom model. (b) One-gate MoE model. (c) Multi-gate MoE model.

is inspired by the Mixture-of-Experts (MoE) model [21] and the recent MoE layer [16, 31]. MMoE explicitly models the task relationships and learns task-specific functionalities to leverage shared representations. It allows parameters to be automatically allocated to capture either shared task information or task-specific information, avoiding the need of adding many new parameters per task.

The backbone of MMoE is built upon the most commonly used Shared-Bottom multi-task DNN structure [8]. The Shared-Bottom model structure is shown in Figure 1 (a), where several bottom layers following the input layer are shared across all the tasks and then each task has an individual "tower" of network on top of the bottom representations. Instead of having one bottom network shared by all tasks, our model, shown in Figure 1 (c), has a group of bottom networks, each of which is called an expert. In our paper, each expert is a feed-forward network. We then introduce a gating network for each task. The gating networks take the input features and output softmax gates assembling the experts with different weights, allowing different tasks to utilize experts differently. The results of the assembled experts are then passed into the task-specific tower networks. In this way, the gating networks for different tasks can learn different mixture patterns of experts assembling, and thus capture the task relationships.

To understand how MMoE learns its experts and task gating networks for different levels of task relatedness, we conduct a synthetic experiment where we can measure and control task relatedness by their Pearson correlation. Similar to [24], we use two synthetic regression tasks and use sinusoidal functions as the data generation mechanism to introduce non-linearity. Our approach outperforms baseline methods under this setup, especially when task correlation is low. In this set of experiments, we also discover that MMoE is easier to train and converges to a better loss during multiple runs. This relates to recent discoveries that modulation and gating mechanisms can improve the trainability in training non-convex deep neural networks [10, 19].

We further evaluate the performance of MMoE on a benchmark dataset, UCI Census-income dataset, with a multi-task problem setup. We compare with several state-of-the-art multi-task models which model task relations with soft parameter sharing, and observe improvement in our method.

Finally, we test MMoE on a real large-scale content recommendation system, where two classification tasks are learned at the same time when recommending items to users. We train MMoE model with hundreds of billions of training examples and compare it with a shared-bottom production model. We observe significant improvements in offline metrics such as AUC. In addition, our MMoE model consistently improves online metrics in live experiments.

The contribution of this paper is threefold: First, we propose a novel Multi-gate Mixture-of-Experts model which explicitly models task relationships. Through modulation and gating networks, our model automatically adjusts parameterization between modeling shared information and modeling task-specific information. Second, we conduct control experiments on synthetic data. We report how task relatedness affects training dynamics in multi-task learning and how MMoE improves both model expressiveness and trainability. Finally, we conduct experiments on real benchmark data and a large-scale production recommendation system with hundreds of millions of users and items. Our experiments verify the efficiency and effectiveness of our proposed method in real-world settings.

2 RELATED WORK

2.1 Multi-task Learning in DNNs

Multi-task models can learn commonalities and differences across different tasks. Doing so can result in both improved efficiency and model quality for each task [4, 8, 30]. One of the widely used multi-task learning models is proposed by Caruana [8, 9], which has a shared-bottom model structure, where the bottom hidden layers are shared across tasks. This structure substantially reduces the risk of overfitting, but can suffer from optimization conflicts caused by task differences, because all tasks need to use the same set of parameters on shared-bottom layers.

To understand how task relatedness affects model quality, prior works used synthetic data generation and manipulated different types of task relatedness so as to evaluate the effectiveness of multi-task models [4–6, 8].

Instead of sharing hidden layers and same model parameters across tasks, some recent approaches add different types of constraints on task-specific parameters [15, 27, 34]. For example, for two tasks, Duong et al. [15] adds L-2 constraints between the two sets of parameters. The cross-stitch network [27] learns a unique combination of task-specific hidden-layer embeddings for each task. Yang et al. [34] uses a tensor factorization model to generate hidden-layer parameters for each task. Compared to shared-bottom models, these approaches have more task-specific parameters and can achieve better performance when task differences lead to conflicts in updating shared parameters. However, the larger number of task-specific parameters require more training data to fit and may not be efficient in large-scale models.

2.2 Ensemble of Subnets & Mixture of Experts

In this paper, we apply some recent findings in deep learning such as parameter modulation and ensemble method to model task relationships for multi-task learning. In DNNs, ensemble models and ensemble of subnetworks have been proven to be able to improve model performance [9, 20].

Eigen et al [16] and Shazeer et al [31] turn the mixture-of-experts model into basic building blocks (MoE layer) and stack them in a DNN. The MoE layer selects subnets (experts) based on the input of the layer at both training time and serving time. Therefore, this model is not only more powerful in modeling but also lowers computation cost by introducing sparsity into the gating networks. Similarly, PathNet [17], which is designed for artificial general intelligence to handle different tasks, is a huge neural network with multiple layers and multiple submodules within each layer. While training for one task, multiple pathways are randomly selected and trained by different workers in parallel. The parameters of the best pathway is fixed and new pathways are selected for training new tasks. We took inspiration from these works by using an ensemble of subnets (experts) to achieve transfer learning while saving computation.

2.3 Multi-task Learning Applications

Thanks to the development of distributed machine learning systems [13], many large-scale real-world applications have adopted DNN-based multi-task learning algorithms and observed substantial quality improvements. On multi-lingual machine translation tasks, with shared model parameters, translation tasks having limited training data can be improved by jointly learning with tasks having large amount of training data [22]. For building recommendation systems, multi-task learning is found helpful for providing context-aware recommendations [28, 35]. In [3], a text recommendation task is improved by sharing feature representations and lower level hidden layers. In [11], a shared-bottom model is used to learn a ranking algorithm for video recommendation. Similar to these prior works, we evaluate our modeling approach on a real-world large-scale recommendation system. We demonstrate that our approach is indeed scalable, and has favorable performance compared with other state-of-the-art modeling approaches.

3 PRELIMINARY

3.1 Shared-bottom Multi-task Model

We first introduce the shared-bottom multi-task model in Figure 1 (a), which is a framework proposed by Rich Caruana [8] and widely adopted in many multi-task learning applications [18, 29]. Therefore, we treat it as a representative baseline approach in multi-task modeling.

Given K tasks, the model consists of a shared-bottom network, represented as function f, and K tower networks h^k, where $k = 1, 2, ..., K$ for each task respectively. The shared-bottom network follows the input layer, and the tower networks are built upon the output of the shared-bottom. Then individual output y_k for each task follows the corresponding task-specific tower. For task k, the model can be formulated as,

$$y_k = h^k(f(x)). \tag{1}$$

3.2 Synthetic Data Generation

Prior works [15, 27] indicate that the performance of multi-task learning models highly depends on the inherent task relatedness in the data. It is however difficult to study directly how task relatedness affects multi-task models in real applications, since in real applications we cannot easily change the relatedness between tasks and observe the effect. Therefore to establish an empirical study for this relationship, we first use synthetic data where we can easily measure and control the task relatedness.

Inspired by Kang et al. [24], we generate two regression tasks and use the Pearson correlation of the labels of these two tasks as the quantitative indicator of task relationships. Since we focus on DNN models, instead of the linear functions used in [24], we set the regression model as a combination of sinusoidal functions as used in [33]. Specifically, we generate the synthetic data as follows.

(1) Given the input feature dimension d, we generate two orthogonal unit vectors $u_1, u_2 \in \mathbb{R}^d$, i.e.,
$$u_1^T u_2 = 0, \|u_1\|_2 = 1, \|u_2\|_2 = 1.$$
(2) Given a scale constant c and a correlation score $-1 \le p \le 1$, generate two weight vectors w_1, w_2 such that
$$w_1 = cu_1,\ w_2 = c\left(pu_1 + \sqrt{(1-p^2)}u_2\right). \tag{2}$$
(3) Randomly sample an input data point $x \in \mathbb{R}^d$ with each of its element from $\mathcal{N}(0, 1)$.
(4) Generate two labels y_1, y_2 for two regression tasks as follows,
$$y_1 = w_1^T x + \sum_{i=1}^{m} \sin\left(\alpha_i w_1^T x + \beta_i\right) + \epsilon_1 \tag{3}$$
$$y_2 = w_2^T x + \sum_{i=1}^{m} \sin\left(\alpha_i w_2^T x + \beta_i\right) + \epsilon_2 \tag{4}$$
where $\alpha_i, \beta_i, i = 1, 2, ..., m$ are given parameters that control the shape of the sinusoidal functions and $\epsilon_1, \epsilon_2 \overset{\text{i.i.d}}{\sim} \mathcal{N}(0, 0.01)$,
(5) Repeat (3) and (4) until enough data are generated.

Due to the non-linear data generation procedure, it's not straightforward to generate tasks with a given label Pearson correlation. Instead, we manipulate the cosine similarity of the weight vectors in Eq 2, which is $\cos(w_1, w_2) = p$, and measuring the resulting label Pearson correlation afterwards. Note that in the linear case where

$$y_1 = w_1^T x + \epsilon_1$$
$$y_2 = w_2^T x + \epsilon_2,$$

the label Pearson correlation of y_1, y_2 is exactly p.

In the nonlinear case, y_1 and y_2 in Eq 3 and Eq 4 are also positively correlated, as shown in Figure 2.

In the rest of this paper, for simplicity, we refer to cosine similarity of the weight vectors as "task correlation".

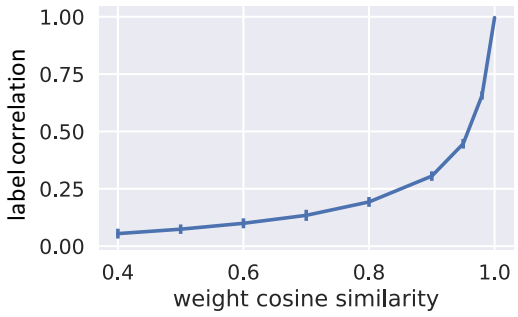


Figure 2: Label Pearson correlation v.s. weight cosine similarity (task correlation). X-axis shows the cosine similarities of weight vectors. Y-axis is the resulting Pearson correlation between the labels. For each weight cosine similarity, we generate 10k data points with two labels and calculate the Pearson correlation between these two labels. We repeat this process and plot the average with the error bar indicating 2 standard deviations among the 100 trials.

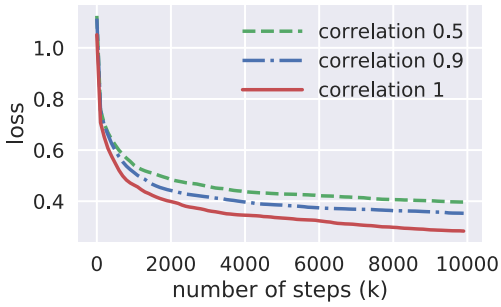


Figure 3: Performance of the Shared-Bottom model on synthetic data with different task correlation. Tasks with task correlation 1 means the two tasks have the same weight vectors but independent noises. X-axis is the number of training steps. Y-axis is the average loss of 200 independent runs.

3.3 Impact of Task Relatedness

To verify that low task relatedness hurts model quality in a baseline multi-task model setup, we conduct control experiments on the synthetic data as follows.

(1) Given a list of task correlation scores, generate a synthetic dataset for each score;
(2) Train one Shared-Bottom multi-task model on each of these datasets respectively while controlling all the model and training hyper-parameters to remain the same;
(3) Repeat step (1) and (2) hundreds of times with datasets generated independently but control the list of task correlation scores and the hyper-parameters the same;
(4) Calculate the average performance of the models for each task correlation score.

Figure 3 shows the loss curves for different task correlations. As expected, the performance of the model trends down as the task correlation decreases. This trend is general for many different hyper-parameter settings. Here we only show an example of the control experiment results in Figure 3. In this example, each tower network is a single-layer neural network with 8 hidden units, and the shared bottom network is a single-layer network with size=16. The model is implemented using TensorFlow [1] and trained using Adam optimizer [25] with the default setting. Note that the two regression tasks are symmetric so it's sufficient to report the results on one task. This phenomenon validates our hypothesis that the traditional multi-task model is sensitive to the task relationships.

4 MODELING APPROACHES

4.1 Mixture-of-Experts

The Original Mixture-of-Experts (MoE) Model [21] can be formulated as:

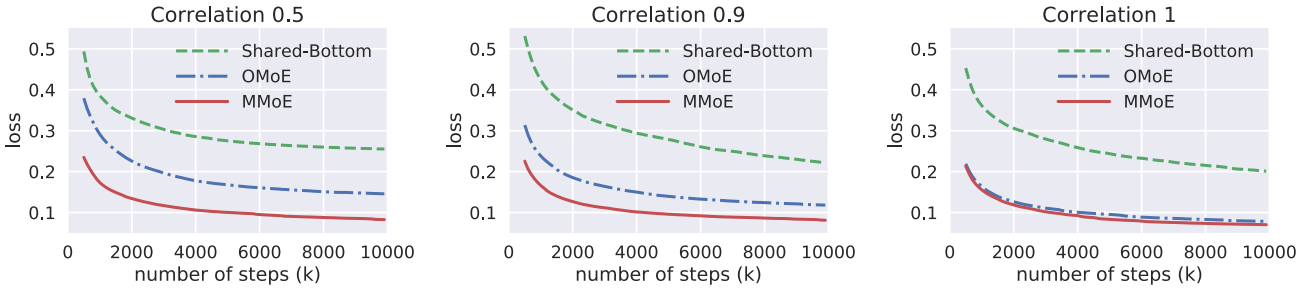
$$y = \sum_{i=1}^{n} g(x)_i f_i(x), \tag{5}$$

where $\sum_{i=1}^{n} g(x)_i = 1$ and $g(x)_i$, the ith logit of the output of $g(x)$, indicates the probability for expert f_i.

Here, $f_i, i = 1, ..., n$ are n expert networks and g represents a gating network that ensembles the results from all experts. More specifically, the gating network g produces a distribution over the n experts based on the input, and the final output is a weighted sum of the outputs of all experts.

MoE Layer : While MoE was first developed as an ensemble method of multiple individual models, Eigen et al [16] and Shazeer et al [31] turn it into basic building blocks (MoE layer) and stack them in a DNN. The MoE layer has the same structure as the MoE model but accepts the output of the previous layer as input and outputs to a successive layer. The whole model is then trained in an end-to-end way.

The main goal of the MoE layer structure proposed by Eigen et al [16] and Shazeer et al [31] is to achieve conditional computation [7, 12], where only parts of a network are active on a per-example basis. For each input example, the model is able to select only a subset of experts by the gating network conditioned on the input.



(a) Performance with correlation 0.5 (b) Performance with correlation 0.9 (c) Performance with two identical tasks

Figure 4: Average performance of MMoE, OMoE, and Shared-Bottom on synthetic data with different correlations.

4.2 Multi-gate Mixture-of-Experts

We propose a new MoE model that is designed to capture the task differences without requiring significantly more model parameters compared to the shared-bottom multi-task model. The new model is called Multi-gate Mixture-of-Experts (MMoE) model, where the key idea is to substitute the shared bottom network f in Eq 1 with the MoE layer in Eq 5. More importantly, we add a separate gating network g^k for each task k. More precisely, the output of task k is

$$y_k = h^k(f^k(x)), \tag{6}$$

$$\text{where } f^k(x) = \sum_{i=1}^{n} g^k(x)_i f_i(x). \tag{7}$$

See Figure 1 (c) for an illustration of the model structure.

Our implementation consists of identical multilayer perceptrons with ReLU activations. The gating networks are simply linear transformations of the input with a softmax layer:

$$g^k(x) = \text{softmax}(W_{gk}x), \tag{8}$$

where $W_{gk} \in \mathbb{R}^{n \times d}$ is a trainable matrix. n is the number of experts and d is the feature dimension.

Each gating network can learn to "select" a subset of experts to use conditioned on the input example. This is desirable for a flexible parameter sharing in the multi-task learning situation. As a special case, if only one expert with the highest gate score is selected, each gating network actually linearly separates the input space into n regions with each region corresponding to an expert. The MMoE is able to model the task relationships in a sophisticated way by deciding how the separations resulted by different gates overlap with each other. If the tasks are less related, then sharing experts will be penalized and the gating networks of these tasks will learn to utilize different experts instead. Compared to the Shared-Bottom model, the MMoE only has several additional gating networks, and the number of model parameters in the gating network is negligible. Therefore the whole model still enjoys the benefit of knowledge transfer in multi-task learning as much as possible.

To understand how introducing separate gating network for each task can help the model learn task-specific information, we compare with a model structure with all tasks sharing one gate. We call it One-gate Mixture-of-Experts (OMoE) model. This is a direct adaption of the MoE layer to the Shared-Bottom multi-task model. See Figure 1 (b) for an illustration of the model structure.

5 MMOE ON SYNTHETIC DATA

In this section, we want to understand if the MMoE model can indeed better handle the situation where tasks are less related. Similar to Section 3.3, we conduct control experiments on the synthetic data to investigate this problem. We vary the task correlation of the synthetic data and observe how the behavior changes for different models. We also conduct a trainability analysis and show that MoE based models can be more easily trained compared to Shared-Bottom models.

5.1 Performance on Data with Different Task Correlations

We repeat the experiments in section 3.3 for the proposed MMoE model and two baseline models: the Shared-Bottom model and the OMoE model.

Model Structures. The input dimension is 100. Both MoE based models have 8 experts with each expert implemented as a single-layer network. The size of the hidden layers in the expert network is 16. The tower networks are still single-layer networks with size=8. We note that the total number of model parameters in the shared experts and the towers is $100 \times 16 \times 8 + 16 \times 8 \times 2 = 13056$. For the baseline Shared-Bottom model, we still set the tower network as a single-layer network with size=8. We set the single-layer shared bottom network with size $13056/(100 + 8 \times 2) \approx 113$.

Results. All the models are trained with the Adam optimizer and the learning rate is grid searched from [0.0001, 0.001, 0.01]. For each model-correlation pair setting, we have 200 runs with independent random data generation and model initialization. The average results are shown in figure 4. The observations are outlined as follows:

(1) For all models, the performance on the data with higher correlation is better than that on the data with lower correlation.
(2) The gap between performances on data with different correlations of the MMoE model is much smaller than that of the OMoE model and the Shared-Bottom model. This trend is especially obvious when we compare the MMoE model with

the OMoE model: in the extreme case where the two tasks are identical, there is almost no difference in performance between the MMoE model and the OMoE model; when the correlation between tasks decreases, however, there is an obvious degeneration of performance for the OMoE model while there is little influence on the MMoE model. Therefore, it's critical to have task-specific gates to model the task differences in the low relatedness case.

(3) Both MoE models are better than the Shared-Bottom model in all scenarios in terms of average performance. This indicates that the MoE structure itself brings additional benefits. Following this observation, we show in the next subsection that the MoE models have better trainability than the Shared-Bottom model.

5.2 Trainability

For large neural network models, we care much about their trainability, i.e., how robust the model is within a range of hyper-parameter settings and model initializations.

Recently, Collins et al [10] find that some gated RNN models (like LSTM and GRU) we thought to perform better than the vanilla RNN are simply easier to train rather than having better model capacities. While we have demonstrated that MMoE can better handle the situation where tasks are less related, we also want to have a deeper understanding how it behaves in terms of trainability.

With our synthetic data, we can naturally investigate the robustness of our model against the randomness in the data and model initialization. We repeat the experiments under each setting multiple times. Each time the data are generated from the same distribution but different random seeds and the models are also initialized differently. We plot the histogram of the final loss values from repeated runs in Figure 5.

There are three interesting observations from the histogram. First, in all task correlation settings, the performance variances of Shared-Bottom model are much larger than those of the MoE based model. This means that Shared-Bottom models in general have much more poor quality local minima than the MoE based models do. Second, while the performance variance of OMoE models is similarly robust as that of MMoE models when task correlation is 1, the robustness of the OMoE has an obvious drop when the task correlation decreases to 0.5. Note that the only difference between MMoE and OMoE is whether there is a multi-gate structure. This validates the usefulness of the multi-gate structure in resolving bad local minima caused by the conflict from task difference. Finally, it's worth to observe that the lowest losses of all the three models are comparable. This is not surprising as neural networks are theoretically universal approximator. With enough model capacity, there should exist a "right" Shared-Bottom model that learns both tasks well. However, note that this is the distribution of 200 independent runs of experiments. And we suspect that for larger and more complicated model (e.g. when the shared bottom network is a recurrent neural network), the chance of getting the "right" model of the task relationship will be even lower. Therefore, explicitly modeling the task relationship is still desirable.

6 REAL DATA EXPERIMENTS

In this section, we conduct experiments on real datasets to validate the effectiveness of our approach.

6.1 Baseline Methods

Besides the Shared-Bottom multi-task model, we compare our approach with several state-of-the-art multi-task deep neural network models that attempt to learn the task relationship from the data.

L2-Constrained [15]: This method is designed for a cross-lingual problem with two tasks. In this method, parameters used for different tasks are shared softly by an L2 constraint.

Given y_k as the ground truth label for task k, $k \in 1, 2$, the prediction of task k is represented as

$$\hat{y}_k = f(x; \theta_k),$$

where θ_k are model parameters.

The objective function of this method is

$$\mathbb{E}L(y_1, f(x; \theta_1)) + \mathbb{E}L(y_2, f(x; \theta_2)) + \alpha \|\theta_1 - \theta_2\|_2^2$$

where y_1, y_2 are the ground truth label for task 1 and task 2, and α is a hyper-parameter. This method models the task relatedness with the magnitude of α.

Cross-Stitch [27]: This method shares knowledge between two tasks by introducing a "Cross-Stitch" unit. The Cross-Stitch unit takes the input of separated hidden layers x_1 and x_2 from task 1 and 2, and outputs $\tilde{x}_1^i$ and $\tilde{x}_2^i$ respectively by the following equation:

$$\begin{bmatrix} \tilde{x}_1^i \\ \tilde{x}_2^i \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} \begin{bmatrix} x_1^i \\ x_2^i \end{bmatrix},$$

where $\alpha_{jk}, j, k = 1, 2$ is a trainable parameter representing the cross transfer from task k to task j. The $\tilde{x}_1$ and $\tilde{x}_2$ are sent to the higher level layer in task 1 and task 2 respectively.

Tensor-Factorization [34]: In this method, weights from multiple tasks are modeled as tensors and tensor factorization methods are used for parameter sharing across tasks. For our comparison, we implement Tucker decomposition for learning multi-task models, which is reported to deliver the most reliable results [34]. For example, given input hidden-layer size m, output hidden-layer size n and task number k, the weights $\mathcal{W}$, which is a $m \times n \times k$ tensor, is derived from the following equation:

$$\mathcal{W} = \sum_{i_1}^{r_1} \sum_{i_2}^{r_2} \sum_{i_3}^{r_3} S(i_1, i_2, i_3) \cdot U_1(:, i_1) \circ U_2(:, i_2) \circ U_3(:, i_3),$$

where tensor S of size $r_1 \times r_2 \times r_3$, matrix U_1 of size $m \times r_1$, U_2 of size $n \times r_2$, and U_3 of size $k \times r_3$ are trainable parameters. All of them are trained together via standard backpropagation. r_1, r_2 and r_3 are hyper-parameters.

6.2 Hyper-Parameter Tuning

We adopt a hyper-parameter tuner, which is used in recent deep learning frameworks [10], to search the best hyperparameters for all the models in the experiments with real datasets. The tuning algorithm is a Gaussian Process model similar to Spearmint as introduced in [14, 32].

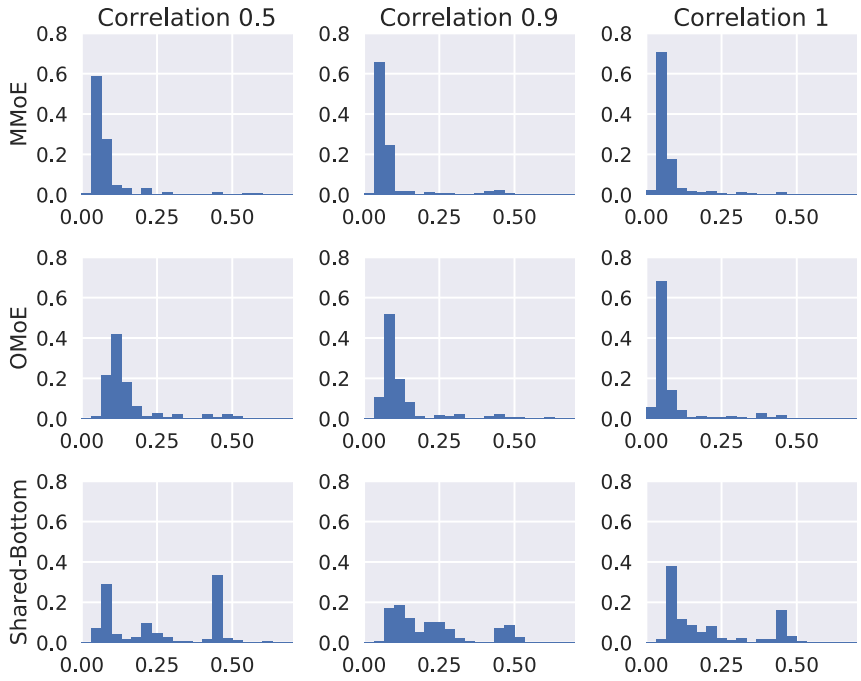


Figure 5: Histogram of performance of MMoE, OMoE, and Shared-Bottom multi-task model on synthetic data with different correlations.

To make the comparison fair, we constrain the maximum model size of all methods by setting a same upper bound for the number of hidden units per layer, which is 2048. For MMoE, it is the "number of experts" × "hidden units per expert". Our approach and all baseline methods are implemented using TensorFlow [1].

We tune the learning rates and the number of training steps for all methods. We also tune some method-specific hyper-parameters:

- **MMOE**: Number of experts, number of hidden units per expert.
- **L2-Constrained**: Hidden-layer size. Weight α of the L2 constraint.
- **Cross-Stitch**: Hidden-layer size, Cross-Stitch layer size.
- **Tensor-Factorization**: r_1, r_2, r_3 for Tuck Decomposition, hidden-layer size.

6.3 Census-income Data

In this subsection, we report and discuss experiment results on the census-income data.

6.3.1 Dataset Description. The UCI census-income dataset [2] is extracted from the 1994 census database. It contains 299,285 instances of demographic information of American adults. There are 40 features in total. We construct two multi-task learning problems from this dataset by setting some of the features as prediction targets and calculate the absolute value of Pearson correlation of the task labels over 10,000 random samples:

(1) Task 1: Predict whether the income exceeds \$50K;
Task 2: Predict whether this person's marital status is never married.
Absolute Pearson correlation: 0.1768.

(2) Task 1: Predict whether the education level is at least college;
Task 2: Predict whether this person's marital status is never married.
Absolute Pearson correlation: 0.2373.

In the dataset, there are 199,523 training examples and 99,762 test examples. We further randomly split test examples into a validation dataset and a test dataset by the fraction of 1:1.

Note that we remove education and marital status from input features as they are treated as labels in these setups. We compare MMoE with aforementioned baseline methods. Since both groups of tasks are binary classification problems, we use AUC scores as the evaluation metrics. In both groups, we treat the marital status task as the auxiliary task, and treat the income task in the first group and the education task in the second group as the main tasks. For hyper-parameter tuning, we use the AUC of the main task on the validation set as the objective. For each method, we use the hyper-parameter tuner conducting thousands of experiments to find the best hyper-parameter setup. After the hyper-parameter tuner finds the best hyper-parameter for each method, we train each method on training dataset 400 times with random parameter initialization and report the results on the test dataset.

6.3.2 Results. For both groups, we report the mean AUC over 400 runs, and the AUC of the run where best main task performance is obtained. Table 1 and Table 2 show the results of two groups

of tasks. We also tune and train single-task models by training a separate model for each task and report their results.

Table 1: Performance on the first group of UCI Census-income dataset.

Group 1	AUC/Income		AUC/Marital Stat	
	best	mean	w/ best income	mean
Single-Task	0.9398	0.9337	**0.9933**	0.9922
Shared-Bottom	0.9361	0.9295	0.9915	0.9921
L2-Constrained	0.9389	0.9359	0.9922	0.9918
Cross-Stitch	0.9406	**0.9361**	0.9917	0.9922
Tensor-Factorization	0.7460	0.6765	0.8175	0.8412
OMoE	0.9387	0.9319	0.9928	0.9923
MMoE	**0.9410**	0.9359	0.9926	**0.9927**

Table 2: Performance on the second group of UCI Census-income dataset.

Group 2	AUC/Education		AUC/Marital Stat	
	best	mean	w/ best education	mean
Single-Task	0.8843	0.8792	**0.9933**	0.9922
Shared-Bottom	0.8836	0.8813	0.9927	0.9917
L2-Constrained	0.8855	0.8823	0.9923	0.9918
Cross-Stitch	0.8855	0.8819	0.9919	0.9921
Tensor-Factorization	0.7367	0.7256	0.7453	0.7497
OMoE	0.8852	0.8813	0.9915	0.9912
MMoE	**0.8860**	**0.8826**	0.9932	**0.9924**

Given the task relatedness (roughly measured by the Pearson correlation) is not very strong in either group, the Shared-Bottom model is almost always the worst among multi-task models (except for Tensor-Factorization). Both L2-Constrained and Cross-Stitch have separate model parameters for each task and add constraints on how to learn these parameters, and therefore perform better than Shared-Bottom. However, having constraints on model parameter learning heavily relies on the task relationship assumptions, which is less flexible than the parameter modulation mechanism used by MMoE. So MMoE outperforms other multi-task models in all means in group 2, where the task relatedness is even smaller than group 1.

The Tensor-Factorization method is the worst in both groups. This is because it tends to generalize the hidden-layer weights for all of the tasks in lower rank tensor and matrices. This method can be very sensitive to task relatedness, since it tends to over-generalize when tasks are less related, and needs more data and longer time to train.

The multi-task models are not tuned for the auxiliary marital status task on validation set while the single-task model is. So it is reasonable that the single-task model gets the best performance on the auxiliary task.

6.4 Large-scale Content Recommendation

In this subsection, we conduct experiments on a large-scale content recommendation system in Google Inc., where the recommendations are generated from hundreds of millions of unique items for billions of users. Specifically, given a user's current behavior of consuming an item, this recommendation system targets at showing the user a list of relevant items to consume next.

Our recommendation system adopts similar framework as proposed in some existing content recommendation frameworks [11], which has a candidate generator followed by a deep ranking model. The deep ranking model in our setup is trained to optimize for two types of ranking objectives: (1) optimizing for engagement related objectives such as click through rate and engagement time; (2) optimizing for satisfaction related objectives, such as like rate. Our training data include hundreds of billions of user implicit feedbacks such as clicks and likes. If trained separately, the model for each task needs to learn billions of parameters. Therefore, compared to learning multiple objectives separately, a Shared-Bottom architecture comes with the benefit of smaller model size. In fact, such a Shared-Bottom model is already used in production.

6.4.1 Experiment Setup. We evaluate the multi-task models by creating two binary classification tasks for the deep ranking model: (1) predicting a user engagement related behavior; (2) predicting a user satisfaction related behavior. We name these two tasks as engagement subtask and satisfaction subtask.

Our recommendation system uses embeddings for sparse features and normalizes all dense features to [0, 1] scale. For the Shared-Bottom model, we implement the shared bottom network as a feed-forward neural network with several fully-connected layers with ReLU activation. A fully-connected layer built on top of the shared bottom network for each task serves as the tower network. For MMoE, we simply change the top layer of the shared bottom network to an MMoE layer and keep the output hidden units with the same dimensionality. Therefore, we don't add extra noticeable computation costs in model training and serving. We also implement baseline methods such as L2-Constrained and Cross-Stitch. Due to their model architectures, they have roughly double the number of parameters comparing to the Shared-Bottom model. We do not compare with Tensor-Factorization because the computation of the Tucker product cannot scale up to billion level without heavy efficiency engineering. All models are optimized using mini-batch Stochastic Gradient Descent (SGD) with batch size 1024.

6.4.2 Offline Evaluation Results. For offline evaluation, we train the models on a fixed set of 30 billion user implicit feedbacks and evaluate on a 1 million hold-out dataset. Given that the label of the satisfaction subtask is much sparser than the engagement subtask, the offline results have very high noise levels. We only show the AUC scores and R-Squared scores on the engagement subtask in Table 3.

We show the results after training 2 million steps (10 billion examples with batch size 1024), 4 million steps and 6 million steps. MMoE outperforms other models in terms of both metrics. L2-Constrained and Cross-Stitch are worse than the Shared-Bottom model. This is likely because these two models are built upon two

Table 3: Engagement performance on the real large-scale recommendation system.

Metric	AUC@2M	AUC@4M	AUC@6M	R2@2M	R2@4M	R2@6M
Shared-Bottom	0.6879	0.6888	0.6900	0.08812	0.09159	0.09287
L2-Constrained	0.6866	0.6881	0.6895	0.08668	0.09030	0.09213
Cross-Stitch	0.6880	0.6885	0.6899	0.08949	0.09112	0.09332
OMoE	0.6876	0.6891	0.6893	0.08749	0.09085	0.09230
MMoE	**0.6894**	**0.6897**	**0.6908**	**0.08978**	**0.09263**	**0.09362**

separate single-task models and have too many model parameters to be well constrained.

To better understand how the gates work, we show the distribution of the softmax gate of each task in Figure 6. We can see that MMoE learns the difference between these two tasks and automatically balances the shared and non-shared parameters. Since satisfaction subtask's labels are sparser than the engagement subtask's, the gate for satisfaction subtask is more focused on a single expert.

Figure 6: Softmax Gate Distribution for Engagement and Satisfaction Subtasks.

6.4.3 Live Experiment Results. At last, we conduct live experiments for our MMoE model on the content recommendation system. We do not conduct live experiments for L2-Constrained and Cross-Stitch methods because both models double the serving time by introducing more parameters.

We conduct two sets of experiments. The first experiment is to compare a Shared-Bottom model with a Single-Task model. The Shared-Bottom model is trained on both engagement subtask and satisfaction subtask. The Single-Task model is trained on the engagement subtask only. Note that though not trained on the satisfaction subtask, the Single-Task model serves as a ranking model at test time so we can also calculate satisfaction metrics on it. The second experiment is to compare our MMoE model with the Shared-Bottom model in the first experiment. Both experiments are done using the same amount of live traffic.

Table 4 shows the results of these live experiments. First, by using Shared-Bottom model, we see a huge improvement on the satisfaction live metric of 19.72%, and a slight decrease of -0.22% on the engagement live metric. Second, by using MMoE, we improve both metrics comparing with the Shared-Bottom model. In this recommendation system, engagement metric has a much larger raw value than the satisfaction metric, and it is desirable to have no engagement metric loss or even gains while improving satisfaction metric.

Table 4: Live experiment results

Live experiment	Engagement Metric	Satisfaction Metric
Shared-Bottom Improvement over Single-Task	-0.22% *	19.72% **
MMoE Improvement over Shared-Bottom	0.25% **	2.65% **

* indicates confidence interval level 90%
** indicates confidence interval level 95%

7 CONCLUSION

We propose a novel multi-task learning approach, Multi-gate MoE (MMoE), that explicitly learns to model task relationship from data. We show by control experiments on synthetic data that the proposed approach can better handle the scenario where tasks are less related. We also show that the MMoE is easier to train compared to baseline methods. With experiments on benchmark dataset and a real large-scale recommendation system, we demonstrate the success of the proposed method over several state-of-the-art baseline multi-task learning models.

Besides the benefits above, another major design consideration in real machine learning production systems is the computational efficiency. This is also one of the most important reasons that the Shared-Bottom multi-task model is widely used. The shared part of the model saves a lot of computation at serving time [18, 29]. All of the three state-of-the-art baseline models (see section 6.1) learn the task relationship at the loss of this computational benefit. The MMoE model, however, largely preserves the computational advantage since the gating networks are usually light-weight and the expert networks are shared across all the tasks. Moreover, this model has the potential to achieve even better computational efficiency by making the gating network as a sparse top-k gate [31]. We hope this work inspire other researchers to further investigate multi-task modeling using these approaches.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).
[2] Arthur Asuncion and David Newman. 2007. UCI machine learning repository. (2007).

[3] Trapit Bansal, David Belanger, and Andrew McCallum. 2016. Ask the gru: Multi-task learning for deep text recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 107–114.

[4] Jonathan Baxter et al. 2000. A model of inductive bias learning. *J. Artif. Intell. Res.(JAIR)* 12, 149-198 (2000), 3.

[5] Shai Ben-David, Johannes Gehrke, and Reba Schuller. 2002. A theoretical framework for learning from a pool of disparate data sources. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 443–449.

[6] Shai Ben-David, Reba Schuller, et al. 2003. Exploiting task relatedness for multiple task learning. *Lecture notes in computer science* (2003), 567–580.

[7] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[8] Rich Caruana. 1998. Multitask learning. In *Learning to learn*. Springer, 95–133.

[9] R Caruna. 1993. Multitask learning: A knowledge-based source of inductive bias. In *Machine Learning: Proceedings of the Tenth International Conference*. 41–48.

[10] Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. 2016. Capacity and Trainability in Recurrent Neural Networks. *arXiv preprint arXiv:1611.09913* (2016).

[11] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 191–198.

[12] Andrew Davis and Itamar Arel. 2013. Low-rank approximations for conditional feedforward computation in deep neural networks. *arXiv preprint arXiv:1312.4461* (2013).

[13] 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 *Advances in neural information processing systems*. 1223–1231.

[14] Thomas Desautels, Andreas Krause, and Joel W Burdick. 2014. Parallelizing exploration-exploitation tradeoffs in gaussian process bandit optimization. *The Journal of Machine Learning Research* 15, 1 (2014), 3873–3923.

[15] Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. 2015. Low Resource Dependency Parsing: Cross-lingual Parameter Sharing in a Neural Network Parser.. In *ACL (2)*. 845–850.

[16] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. 2013. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314* (2013).

[17] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. 2017. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734* (2017).

[18] Ross Girshick. 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*. 1440–1448.

[19] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 249–256.

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

[21] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation* 3, 1 (1991), 79–87.

[22] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. 2016. Google's multilingual neural machine translation system: enabling zero-shot translation. *arXiv preprint arXiv:1611.04558* (2016).

[23] Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. 2017. One Model To Learn Them All. *arXiv preprint arXiv:1706.05137* (2017).

[24] Zhuoliang Kang, Kristen Grauman, and Fei Sha. 2011. Learning with whom to share in multi-task feature learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 521–528.

[25] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[26] Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2015. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114* (2015).

[27] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. 2016. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3994–4003.

[28] Xia Ning and George Karypis. 2010. Multi-task learning for recommender system. In *Proceedings of 2nd Asian Conference on Machine Learning*. 269–284.

[29] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*. 91–99.

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

[31] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017).

[32] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.

[33] Shengyang Sun, Changyou Chen, and Lawrence Carin. 2017. Learning Structured Weight Uncertainty in Bayesian Neural Networks. In *Artificial Intelligence and Statistics*. 1283–1292.

[34] Yongxin Yang and Timothy Hospedales. 2016. Deep multi-task representation learning: A tensor factorisation approach. *arXiv preprint arXiv:1605.06391* (2016).

[35] Zhe Zhao, Zhiyuan Cheng, Lichan Hong, and Ed H Chi. 2015. Improving user topic interest profiles by behavior factorization. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1406–1416.