Adapting Deep RankNet for Personalized Search

[1]Yang Song, [2]Hongning Wang, [1]Xiaodong He

[1]Microsoft Research,
One Microsoft Way,
Redmond, WA 98052, USA
{yangsong, xiaohe}@microsoft.com

[2]Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana IL, 61801 USA
{wang296}@cs.uiuc.edu

ABSTRACT

RankNet is one of the widely adopted ranking models for web search tasks. However, adapting a generic RankNet for personalized search is little studied. In this paper, we first continue-trained a variety of RankNets with different number of hidden layers and network structures over a previously trained global RankNet model, and observed that a deep neural network with five hidden layers gives the best performance. To further improve the performance of adaptation, we propose a set of novel methods categorized into two groups. In the first group, three methods are proposed to properly assess the usefulness of each adaptation instance and only leverage the most informative instances to adapt a user-specific RankNet model. These assessments are based on KL-divergence, click entropy or a heuristic to ignore top clicks in adaptation queries. In the second group, two methods are proposed to regularize the training of the neural network in RankNet: one of these methods regularize the error back-propagation via a truncated gradient approach, while the other method limits the depth of the back propagation when adapting the neural network. We empirically evaluate our approaches using a large-scale real-world data set. Experimental results exhibit that our methods all give significant improvements over a strong baseline ranking system, and the truncated gradient approach gives the best performance, significantly better than all others.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Internet search*; H.3.0 [**Information Storage and Retrieval**]: General—*Web search*

General Terms

Measurement,Experimentation

Keywords

personalized search, deep learning, ranking adaptation

WSDM'14, February 24–28, 2014, New York, New York, USA.

http://dx.doi.org/10.1145/2556195.2556234.

1. INTRODUCTION

Search engine personalization, or otherwise known as personalized search, is getting more and more attention from the information retrieval community in recent years [24, 2, 5, 17, 18, 22, 23, 16]. In general, in order to better serve their users, search engines often record certain user activities such as queries and search-result clicks, and leverage such information to improve the relevance of the returned documents. To perform personalization, search engines try to match the user behaviors from the target users to others who have similar search behaviors and then aggregate the information from others to estimate the target user's search intent. Essentially, many of the existing works act fairly close to the collaborative filtering method [24, 22], where the user similarity is defined by considering their queries [22, 23], tasks [24], and etc.

In this paper, we address the personalized search problem from a model adaptation perspective. We adapt a generic RankNet model [3] that is trained on a user-independent data set to each individual user based on his/her search history and result preference. In literature, little has been studied on this direction until very recently, where the authors in [21] demonstrated competitive adaptation performance of RankNet when comparing with RankSVM and LambdaMART for personalized model adaptation. Interestingly, in their framework, the authors only used a RankNet without hidden layers which is mathematically equivalent to a logistic regression model. With the recent advance of leveraging deep learning models for information retrieval [9, 15, 11], we believe that, a well-trained deep RankNet with multiple layers of non-linear representations can offer an even better performance in training personalized models.

Our motivation of using deep learning for adaptation comes from two aspects. First and foremost, existing approaches on ranking model adaptation mainly focus on adapting from a source domain to a few number of target domains [8, 7] due to efficiency concern. It would be prohibitively expensive to adapt a generic model to each individual user using the current methods. Therefore, our proposal takes a model-based approach. i.e., we first train a user-independent ranker, and then for each individual user, we update the parameters of the global ranking model accordingly. Since a typical user's search history is short, adapting the global model becomes much more efficient than re-training the entire ranker.

Secondly, deep learning framework naturally constructs a semantic/abstract representation from the inputs. This property is desirable for model adaptation. Existing feature-based model adaptation methods, for example the CLRank

algorithm in [4], maps both the source and target domains into a lower-dimensional space to learn a common feature representation. However, finding a reduced dimension, e.g., using SVD, is prohibitively expensive. Instead, deep learning automatically constructs reduced feature spaces from increasingly more compact representations of the network and achieves semantic representations at higher levels.

However, there are many challenges in using deep learning for personalized search. One of the biggest concerns in personalized search is the data sparsity issue, where each user only has a handful of queries in their search histories and therefore limits the learning capability of the personalization algorithms. Despite the recent advances via using probabilistic models [17] and model adaptation frameworks [21], little has been explored on the relationship between data sparsity and model performance in personalized search. Also, issues like model complexity and overfitting in personalized ranking model adaptation remain to be discussed.

Specifically, our paper makes the following contributions:
1. We use a deep learning framework for personalized ranking adaptation. This framework works by first training a deep RankNet as a global ranking model from user-independent label data. We then adapt the global model to each user by training individual models based on the user's search history. We train a variety of RankNets with different numbers of hidden layers and network structures on a per-user basis, and observed that a deep neural network with five hidden layers gives the best performance.

2. To further improve the performance of adaptation, we introduce several novel ideas including re-weighting adaptation data and regularizing back-propagation. Among all of the five proposals, we observe that a truncated gradient approach performs the best.

3. We empirically analyze the issue of model complexity and how it affects model training and the overfitting issue, on a large-scale data set from a commercial search engine's logs. We discover that overfitting can indeed happen if the adaptation data is not sufficient, or early-stopping is not performed properly.

4. We investigate the relationship between data sparsity and model performance by sampling users with different search frequencies. We observe that the adaptation performance often increases with more user search history data. On the other hand, when adaptation data is limited, regularizing back-propagation becomes important.

The remainder of the paper is organized as follows: Section 2 describes related work in personalized search and deep learning for ranking; Section 3 describes the data set used in this paper; Section 4 presents our adaptation framework; Section 5 shows the empirical results; Section 6 concludes our work with discussion and future work.

2. RELATED WORK

We will be discussing two relevant areas of related work in this section: (1) personalized search and (2) deep learning for IR.

(**Personalized Search**) There have been many researches on personalized search and we will highlight a few that is considered interesting and representative. In [23], White et al. leveraged short-term user behavior signals such as browser history, query history, as well as desktop information to re-rank the search results and predict the user search interests in the future. In particular, they built a predictive model leveraging the contextual information by representing user interests as a list of Open Directory Project (ODP) categories so that the model is capable of scaling up to a quarter million users with billions of URLs.

In [17], Sontag et al. proposed to learn user profiles from user's long-term search history. The authors introduced a generative probabilistic model to infer the search relevance of URLs. Two models were introduced: the first probabilistic model built directly on the user history without considering any background models; the second model, on the other hand, considered the topic distribution of documents which is incorporated into the personalized models to further improve the predictive performance.

Shen et al. [16] introduced a context-sensitive method by leveraging implicit user feedback for re-ranking. Their model used both click-through information and contextual queries as implicit signals from users. Specifically, the authors proposed to enhance the KL-divergence based retrieval model by incorporating the context-sensitive information into language models. They empirically analyzed the performance by constructing a test set from TREC AP data set, the result of which indicated an improved retrieval performance than the baseline methods.

Recently, White et al. [24] introduced a novel model by modeling the search task behavior among search users. Comparing to previous work that often model user similarity based on queries, the authors were among the first to introduce a novel concept of task where a task is considered as an atomic user information need, often containing several queries and search-result clicks. The task information was used to group users together to cohorts for generating click features for a particular set of users in consideration. Results indicated that task-based user grouping outperforms query-based and session-based groupings significantly.

From the model adaptation perspective, Wang et al. [21] brought this concept to personalized search from domain adaptation. They introduced a general framework for adaptation by first training a global ranking model on a user-independent data set and then adapting models to each individual user. To reduce the parameter size, feature grouping based on linear transformations such as scaling and shifting is proposed. The authors then applied their general framework to three popular ranking models: RankSVM, RankNet and LambdaRank. Note that in the realization of the RankNet model, no hidden layers are considered concerning the feature grouping complexity, and therefore it could result a suboptimal performance of RankNet.

Dou et al. [5] conducted a large-scale evaluation on several personalized search algorithms, by using a 12-day query log from MSN. They divided personalization into person-level re-ranking and group-level re-ranking, and compared five different variants. They employed two metrics for evaluation, namely web page ranking score and average rank. Experiments indicated that all personalized methods significantly outperformed the default ranking algorithm. It also revealed that personalized search has different effectiveness on different types of queries. For example, they observed that for queries with small click entropy, the performance gain was not that obvious comparing to those with large entropy.

(**Deep Learning for IR**) In recent years, the deep learning technique has been introduced to domains such as speech and image recognition with great success. In information retrieval, Hinton et al. [9] first introduced the idea of us-

ing neural networks for dimensionality reduction. In their work, they leveraged the representation of deep belief network (DBN) to initialize the deep network structure, and then used a deep auto-encoder to reconstruct the input data. Their results indicated that the variables on the top layer were able to give a better representation of each document, so that semantically similar documents are mapped to a closer position to each other.

Mirowski et al. [15] presented a novel algorithm for topic modeling and information retrieval based on a deep auto-encoder architecture. The objective of auto-encoder in general is to minimize the reconstruction error of the input instance. In their work, the authors leveraged a deep structure which produces increasingly more compact representations of the document vector space to reduce the dimensionality so that semantic representation can be achieved at higher level of the network. Besides minimizing the reconstruction error, their framework also minimized a weighted cross-entropy loss between word histograms concurrently. Experimental results on the NIPS article data set demonstrated a lower perplexity score than Latent Dirichlet Allocation (LDA).

Recently, Huang et al. [11] leveraged this technique to improve web search. In particular, the authors tried to bridge the lexical gaps between queries and documents via semantic matching using deep learning framework. In their model, queries and documents were treated as bag-of-words vectors as the input of the neural network, the model then learnt several layers of deep non-linear projections and outputted a vector in the semantic space. During the learning phases, each query is given the title of a URL that is clicked by users along with several un-clicked URLs, where clicked URLs are assumed to be relevant. The learning objective is to minimize the relevance loss of query titles on the sematic space, where the relevance is computed using cosine similarity between the query semantic space and URL semantic space. Experimental results on a large-scale data set indicated significant improvement over the traditional tf-idf based model as well as deep auto-encoder models.

Similarly, Wang et al. [20] addressed the issue of data sparsity in social media question-answering application. Due to the small length of both questions and answers, traditional ways of modeling their relationship turned out to be suboptimal. Instead, the authors leveraged deep learning to address the semantic relationship between them, by using two deep belief networks to model the semantic relevance. Experiments on a Chinese forum corpora indicated better performance than traditional methods.

3. DATA COLLECTION

We collect two sources of data for (1) global ranking model training, and (2) personalized model adaptation, respectively. Both of these data sets are sampled from the logs of a large-scale commercial search engine. Since the focus of this paper is Web document ranking, we filtered the logs to remove non-Web vertical records. In this study, we limit the scope to be English queries within the US search market.

We first randomly sampled a set of 456,238 queries from the logs, between April 2011 and October 2011. For each of the queries, we collected 10 to 30 top URLs returned by the search engine and asked human judgers to assess their relevance. The total query-URL pairs to judge is 5,578,881. The judgers were given 5 choices to annotate each query-URL pair: Perfect, Excellent, Good, Fair and Bad. Each pair is judged by three different assessors. Majority voting is used to determine the final label of each pair.

	Users	Queries	URLs
Training Set	-	456,238	5,578,881
Adaptation Set	3,000	48,238	479,923

Table 1: Statistics of training and annotation sets.

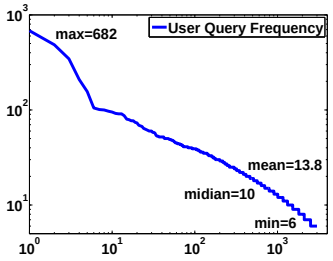


Figure 1: The distribution of user query frequency.

On the other hand, to generate user-specific data, we randomly selected 10,000 unique users based on their user id in the logs, between Jan 2013 and March 2013, a total of 60 days. User session information such as queries, clicked URLs, clicked positions are collected. For model adaptation purpose, we require a user to have at least 6 queries at minimum (2 for training, 2 for validation and 2 for testing). We further filtered out users with insufficient search history. Finally, we randomly sampled from the filtered set and gathered a total of 3,000 users. Table 1 summarizes the statistics of these two data sets. Figure 1 plots the user query frequency distribution, where the most frequent user had 682 queries and many users only had 6 queries.

4. METHODOLOGY

In this section, we introduce our deep learning framework for personalized model adaptation.

4.1 Preliminary

We first briefly review the RankNet model [3]. Unlike the traditional neural network framework, where each training instance first forward propagates the weights to the top layer, and then adjust the weights of individual neurons by back propagating the errors computed from the difference between the true label and the predicted label, RankNet works by considering query-URL pairs. Given a query q, each pair of URLs, say U_i and U_j, is considered as a training pair if they have different relevance labels. Given these training pairs, the model learns a scoring function $f(\cdot)$ for each of such pair, $s_i = f(x_i)$ and $s_j = f(x_j)$, where x represents the feature vector of the URL U. s_i and s_j are then mapped to probabilities using a logistic function

$$P_{ij} \equiv P(U_i \succ U_j) \equiv \frac{1}{1 + e^{-(s_i - s_j)}}, \quad (1)$$

where $U_i \succ U_j$ denotes a preference relationship of U_i over U_j. The model applies cross entropy as its cost function to penalize the deviation of the predicted probabilities from the true probabilities obtained from the labels. Denoting $\overline{P}_{ij}$ as

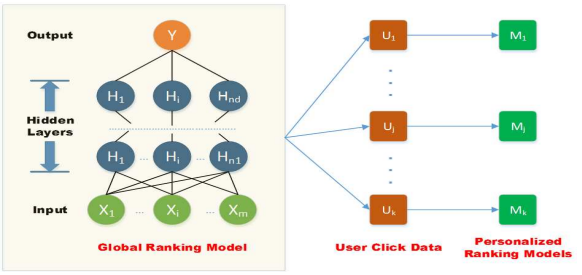


Figure 2: Deep learning framework for personalized ranking adaptation.

the true probability, the cost function can be written as

$$C_{ij} = -\overline{P}_{ij} \log P_{ij} - (1 - \overline{P}_{ij}) \log(1 - P_{ij}). \quad (2)$$

Taking the gradient of the above function, we have

$$\frac{\partial C}{\partial s_i} = -\frac{1}{1 + e^{s_i - s_j}} \quad (3)$$

for $U_i \succ U_j$. The gradient is used during the back propagation of the RankNet to update the weights of each neuron:

$$w_k \rightarrow w_k - \eta \frac{\partial C}{\partial w_k} = w_k - \eta \left(\frac{\partial C}{\partial s_i} \frac{\partial s_i}{\partial w_k} + \frac{\partial C}{\partial s_j} \frac{\partial s_j}{\partial w_k} \right), \quad (4)$$

with η denoting learning rate. The RankNet can be further speed up by accumulating the gradient for each URL to perform a mini-batch update [3], where the speed up if often close to quadratic.

4.2 Deep RankNet for Personalized Model Adaptation

In this section we detail our framework of using deep RankNet for personalized search. Our framework can be divided into two stages: (1) training a global ranking model, and (2) adapting the global model to each individual user.

Specifically, we leverage the human labeled data to first train a deep RankNet, where each input URL U_i is represented by its feature vector $X_i \in \mathbb{R}^m$ that has m dimensions. The network consists of d hidden layers $\{H_1, ...H_d\}$, where each layer can have different number of neurons. The output layer Y is a scalar value that calculates the predicted value of each URL, which is used to update the weights of neurons during the back propagation step as in eq. (4). The hyper-parameter space of the deep RankNet includes the number of hidden layers, the number of neurons in each layer, the learning rate during back propagation, the number of iterations in training, as well as the weights of each neuron. So far in the deep learning community, these parameters are often set empirically. For example, a held-out validation set is often used to determine whether to increase or decrease the learning rate, or when to perform early stop of the training. On the other hand, greedy layer-wise pre-train is often used to initialize the weights of neurons, which has shown some improvement over random initialization [9]. We will discuss the choices of these parameters in our empirical study.

To adapt the global model to each user, we first need to construct user preference data. Since each user's information need is hidden from the search logs, it is not realistic to rely on human assessors to label the data. Therefore, we leverage the user clicks to construct user preference data. Specifically, we denote a preference pair $U_i \succ U_j$ if U_i receives a click from the users and U_j does not, given the same query. Additionally, in order to address the issue of positional bias, we also employed two click feedback strategies from [12], namely *Click > Skip Above* and *Click > No Click Next*, where the first one constructs pairs from the clicked URL and all the skipped URLs that are ranked above that URL. The second method constructs one pair if the current URL gets a click and the immediate URL ranked below does not receive any click.

Figure 2 illustrates our framework. In general, we perform *continue-train* on the global model for each individual user. E.g., we update the global model using the adaptation data from each user until model converges. Comparing to retraining the whole model by adding user preference data to the training set, the benefit of continue-train is two-fold. First, this strategy avoids revisiting the general user-independent training data, and thus is more efficient. Second, as illustrated in Figure 1, users often have only a handful of queries. Thus adding this small amount of data into millions of user-independent training samples may only incur subtle changes to the entire ranking model. In contrast, continue-train will refresh the model solely on the user preference data, and therefore can adapt the global model more effectively.

Despite the potential benefits, there are two issues we need to consider when performing continue-training. First, unlike the user-independent training data which are annotated by professionals, the user-preference data only contains click information, which is very noisy. Second, the amount of user-preference data is usually very limited, and therefore over-fitting becomes a critical issue. In order to address the first issue, we propose a variety of methods to estimate how informative the adaptation samples are, and update the model according to the data that mostly reflects the user-specific preference. In order to address the second issue, we regularize the back-propagation based training of the neural network in various ways. In the first method, inspired by the truncated gradient learning [13], we update the weight only when the corresponding gradient is significant, e.g., the absolute value is above certain threshold. In another method, we follow the techniques in [10, 25], by constraining the back-propagation to only the top layer. In the following sections, we describes the proposed techniques in details.

4.3 Strategy 1: Controlling Adaptation Data

The first strategy we use in adaptation is to properly estimate the weight of user data according to their importance. Take the query *amazon* for example, if a user issued this query and clicked on *www.amazon.com*, then re-training on this data point is most likely to have minimum effect on learning user preference since the majority of other users would also do so. However, if the user instead clicked on *wikipedia.org/amazon/* or *geography.about.com/amazonriver*, then we know that the user probably has a different intent than others, and the global ranking model should be adjusted by taking these clicks into account.

The above example is from a user's perspective. On the other hand, from a query's perspective, if the click concentration of a query is very low, i.e., different users tend to click on different returned documents of that query, it might also be a good indicator that the query needs to be treated with higher weight due to its ambiguity.

Models	Training Pair Error	Validation Pair Error	Training NDCG@3	Validation NDCG@3
50 50	0.14390 ± 0.0222	0.17383 ± 0.0492	0.6931 ± 0.0529	0.5678 ± 0.0503
100 100 100	0.20890 ± 0.0326	0.22909 ± 0.0329	0.6738 ± 0.0292	0.5211 ± 0.0322
100 100 20 20	0.17445 ± 0.0305	0.19694 ± 0.0684	0.687 ± 0.0690	0.5299 ± 0.0440
100 100 50 50 20	0.14375 ± 0.0653	0.16828 ± 0.0685	0.7033 ± 0.0295	0.5682 ± 0.0281

Table 2: Training and validation results of several deep RankNets. Each model was repeated five times with random split data of 1:1 ratio between two sets. The best performed model is highlighted.

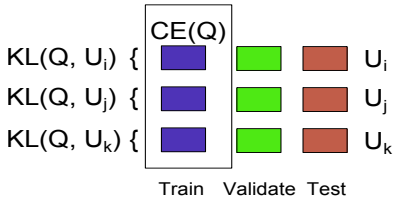


Figure 3: An illustration of how query weights are calculated based on KL and CE, where KL is estimated on a per-user basis and CE is estimated from the entire training set.

Therefore, we propose three heuristics here to assess the importance of adaptation queries.

Heuristic 1: local query weighting via Kullback-ŰLeibler (KL) divergence. Given a particular user u and all his/her queries in training $\{q_1^{(u)}, ..., q_n^{(u)}\}$, consider each query $q_i^{(u)}$ to have a click distribution $P_u(q_i^{(u)})$ for user u and $P_{\neg u}(q_i^{(u)})$ for the remaining users, normalized to be probability distributions. Its weight is determined by the KL divergence of these two probability distributions:

$$W_{KL}(P_u(q_i^{(u)})||P_{\neg u}(q_i^{(u)})) = \sum_j \log P_u(q_i^{(u)}, j) \left(\frac{P_u(q_i^{(u)}, j)}{P_{\neg u}(q_i^{(u)}, j)} \right), \tag{5}$$

so that the more similar user u's click pattern on $q_i^{(u)}$ to the remaining users' pattern, the lower that $q_i^{(u)}$ will be weighted. When the two click patterns are exactly the same, that query will receive zero weight and therefore not used for adaptation.

Heuristic 2: global query weighting via click entropy (CE) measurement. In this method, we do not consider individual user but to aggregate all clicks for a particular query q_i across all users. The weight of q_i is then calculated as:

$$W_{CE}(q_i) = -\sum_j P(q_i, j) \cdot \log P(q_i, j), \tag{6}$$

where $P(q_i, j)$ is the probability of document j being clicked given query q_i (regardless of users). This heuristic is inspired by previous research [19]. According to the authors, queries with higher click entropies are more useful for personalization rather than lower ones.

Figure 3 briefly shows the difference between the KL and CE measurements. Each user's adaptation data is first divided into three parts, for training, validation and testing purposes. The KL measurement calculates individual query's weight on a per-user basis using only the training data, while CE leverages the entire training part to estimate the global weight for each query.

Lastly, we propose a heuristic that ignores queries for which a user clicked on the top-1 returned results, on a per-user basis:

Heuristic 3: remove queries with top-one result clicks from the training set. The reason for this heuristic is quite straight-forward: search engines have already learnt to present the results in descending order of their relevance, having the user clicked on the top result brings few information to help learn a better personalized model.

4.4 Strategy 2: Regularizing Back Propagation

In this section, we address how to regularize the back propagation step to prevent model overfitting. Generally, from a regularization perspective, we hope to update the weight of a neuron only if the neuron is not certain about a particular adaptation example. From a semantic point of view, each neuron is trained to explore certain portion of the feature space for decision making, e.g., topic distribution in IR [15], edge detector in vision [14]. Thus, when a new training example exhibits different feature distribution than previous ones, the algorithm should take it into account and update the corresponding neurons accordingly. In machine learning, such technique is often referred to as L_1-regularized subgradient, or truncated gradient [13], or otherwise known as confidence-weighted learning [6]. In principle, our proposed technique mimics these methods but with a notable difference: our truncation is enforced on each neuron, while previous methods often truncate feature weights. In our study, we introduce a regularization method by leveraging the statistics from a held-out validation set.

To be specific, each neuron's output is computed using its activation function, often in the form of sigmoid, i.e., $a(z) = \frac{1}{1+e^{-z}}$. During adaptation, we update the weights of each neuron k using the truncated gradient as follows:

$$w_k \rightarrow w_k - \eta T_1 \left(\frac{\partial C}{\partial w_k}, a(k), \theta \right), \tag{7}$$

where $a(k)$ is the output of neuron k, C is the cost function defined in eq. (2) and T_1 is defined as follows:

$$T_1(v, a, \theta) = \begin{cases} \max(0, v - a) & \text{if } v \in [0, \theta] \\ \min(0, v + a) & \text{if } v \in [-\theta, 0] \\ v & otherwise \end{cases}$$

Figure 4 shows two examples of the truncated gradient with $a(k) = 3$, $\theta = 3$ and $a(k) = 1.5$, $\theta = 3$. For the first case, when v falls into the range of $[-\theta, \theta]$, T_1 is always pushed to zero. In the second case which is less aggressive, only when v is between $[-\alpha, \alpha]$, the gradient is rounded to zero. Otherwise, T_1 outputs the same value as v for $v > \theta$.

It has been show in [13] that this version of truncated gradient T_1 outperforms direct rounding algorithm (i.e.,

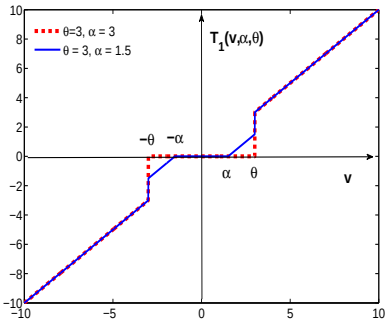


Figure 4: An illustration of the truncated gradient with different values of α and θ. X-axis is the value of v and Y-axis the value of gradient.

$T_0(v, \theta) = 0$ if $|v| \leq \theta$) in most scenarios with a proper choice of θ. Consequently, we propose a variance-truncation heuristic in our RankNet framework:

Heuristic 4: perform truncated gradient on adaptation using a validation set. Given a validation set $\{v_1, ..., v_n\}$, for each neuron k, calculate the output of each validation example as $\{a_k(v_1), ...a_k(v_n)\}$. We assume that the outputs follow a Normal distribution $\mathcal{N} \sim (\mu_k, \sigma_k^2)$. For each adaptation example, update the model weight using truncated gradient with parameter $\theta = \mu_k + \sigma_k$.

Specifically, after the model converges in training, we run a set of held-out data through the network, without performing back propagation, and store the output (activation) values for all held-out examples at each neuron. During adaptation, if the output α of a new example at a specific neuron falls into the range of the mean plus one standard deviation of the validation set, we will truncate the gradient using T_1, to propagate a smaller value of error term to the next layer.

In another way, from the model structure perspective of view, we can also consider training and adaptation to be two different learning tasks but share the same network structure, i.e., they share the same input layer, hidden layers but not the output layer due to a different objective function. Indeed, this scenario has been explored in speech adaptation for cross-lingual model transfer [10]. To be concrete, the authors propose to back propagate only to the top hidden layer and fix the rest hidden layers during adaptation. Semantically, the top layer contains the most abstract features which is more likely to be applicable to different domains than those concrete features at bottom layers. Therefore, when the adaptation data is sparse, it could be beneficial for the model to only adapting the top layer. And therefore we have the following heuristic:

Heuristic 5: back propagate to the highest layer only. During the adaptation stage, we fix the neuron weights of the remaining hidden layers but only perform back propagation to the highest layer of the network and update its neuron's weight accordingly.

5. EMPIRICAL ANALYSIS

In this section, we present a comprehensive empirical study on the real-world data set introduced in Section 3. Although our focus is on the adaptation performance, we will first discuss the results from training global ranking models.

5.1 Global Ranking Model Performance

For each training instance in the 5,578,881 query-URL pair collection, we first extract top-400 most discriminative features which are used by the commercial search engine. These features include query-level features such as query term number, URL-level features such as domain rank, and query-URL features such as BM25 and so on. The reason for only using top-400 features for global model training is due to efficiency concern. Since the main focus on this paper is on adaptation rather than tuning a best deep RankNet for generic ranking purpose, we choose to use a limited set of features to speed up training. Later, we show that even with this limited set of features, the adaptation performance of deep RankNet can still significantly outperform a generic ranker which is trained using thousands of ranking features. We then divided the data into training set and validation set at a 1:1 ratio based on queries, ending up with roughly 228,119 queries in each set. We experiment with different configuration of the RankNet, from very shallow ones to deep ones that have five hidden layers. For each of the configuration, we repeat the data splitting and training for five times and report on the average results. We set the initial learning rate for each configuration to be fairly large $\eta = 1e^{-2}$. We empirically reduce the learning rate (at a step length of 5, $\eta \leftarrow \eta/5$) after an iteration when the validation performance becomes worse — in our scenario, the trigger condition is either that the pair-wise error goes up by more than 2%, or the validation NDCG@3 reduces by more than 1%. The minimum learning rate is controlled to be no less than $1e^{-6}$. The maximum number of training iteration is set to be 2,000. However, we often terminate training earlier if we observe the model has already converged, i.e., when the change of validation NDCG@3 is less than 0.01%.

We tested a total of over 20 different configurations of RankNet. Due to space limitation, Table 2 only lists the results of several typical models that have different numbers of hidden layers. Note that here model "50 50" means a RankNet model with two hidden layers (in addition to the input and output layers), each of which has 50 neurons. The most complex model we have trained is a model with five hidden layers where each one has 100 neurons. Overall, the best performance was achieved by the model "100 100 50 50 20", followed by a very simple configuration with only two hidden layers, "50 50". In general, adding more hidden layers does seem to help improve the ranking performance on average, as can be seen from the table that complex models often outperform simple ones. However, we have also observed that larger models tend to have higher variance than smaller ones, which means that they are more likely to be trapped into local minimums during back-propagation.

5.2 Adaptation Performance

In this section, we focus on reporting the adaptation performance based on two of the most well-performed models in the previous section due to space concern, i.e., "50 50" and "100 100 50 50 20". From now on, we refer them as 2-layer and 5-layer models, respectively.

5.2.1 *Data preparation and evaluation metrics*

As described in Section 4.2, the user-dependent adaptation data is constructed based on user click feedback. However, since the click signal is binary, it becomes inappropriate to use NDCG for evaluation. Therefore, we mea-

sure the ranking quality by using mean average precision (MAP) and mean reciprocal rank (MRR), where MAP is defined as the average precision scores on all queries, and MRR is the average reciprocal ranks of all results, defined as $MRR = \frac{1}{N}\sum_i \frac{1}{rank_i}$, with $rank_i$ being the rank of the *first* relevant URL in the list. For details of the metrics, readers can refer to [1].

Unlike the user-independent data used for training global models which we can randomly split into training and validation sets according to queries, user-specific data comes in chronological order where queries have time dependencies. Therefore, a rigorous way for performance evaluation is to split the user queries by their timestamps, use the first part for training, second part for validation and the latest queries for testing purposes. We uniformly set the ratios to be $\frac{1}{3} : \frac{1}{3} : \frac{1}{3}$ in our experiments.

5.2.2 Baseline Ranking Performance

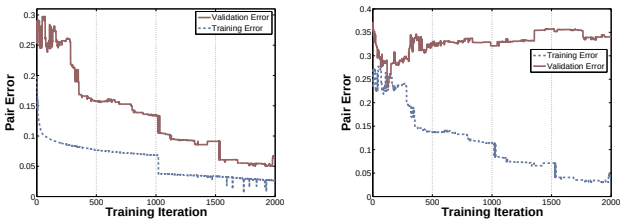
We compare with three baselines. The first baseline is the performance of the production ranking system which uses much more training data than our data set, with more features (over 3,000 features) than our model (400 features). The second and third baselines are the results without performing adaptation. i.e., we directly use the two global RankNet models trained using user-independent data and check their performance on the user-specific portion testing data. Finally, we adapt the two global models to the user data by performing continue-train on the 1/3 portion training data. We monitor the adaptation performance using the validation data. Parameter settings and the early stopping criteria for adaptation are the same as training stage described in Sec. 5.1.

Table 3 lists the results. We see that without adaptation, deep RankNet models perform worse than the baseline system, which is quite expected since the production system leverages many more training data and features. With adaptation conducted, the ranking performance is improved quite drastically. Both models were able to improve the MAP and MRR scores by more than 6%, while reducing the average user-click position by more than 0.2, when compared to the production system. Paired t-test shows that both improvements are significant with *p-value* less than 0.05, comparing to the baseline production ranking. Comparatively, the 5-layer adaptation model slightly beats the 2-layer one, but insignificantly with a *p-value* around 0.2.

We further analyze the adaptation performance of the 5-layer model by randomly selecting two users from the adaptation set. One of the users is considered as a heavy user of the search engine who has issued over 300 queries during the two months, while the other user uses the search engine much less frequently and only had 20 queries during that period. The purpose is to study how likely the data sparsity issue could cause model overfitting. Figure 5 plots their training and validation errors over 2,000 iterations. From the figure, it becomes quite clear that the model adapted for the heavy user is much less immune to overfitting, since the validation error keep decreasing. On the other hand, with only less than 10 queries for training, the model adapted for the light user greatly suffered from the overfitting issue, whose validation error continues to rise after approximately 500 iterations and never decreases again. This illustration demonstrates the importance of early stopping. Note that for illustration purpose, we deliberately allow overfitting to happen in this case. During the actual adaptation stage, the process is stopped earlier before the overfitting happens.

Model	MAP	MRR	Avg. Click Position
Production System	0.5782	0.5783	1.83928
Baseline (2-layer)	0.5017	0.5102	2.46839
Baseline (5-layer)	0.5092	0.5132	2.40555
2-layer adaptation	0.6311	0.6325	1.55006
5-layer adaptation	0.6421	0.6494	1.54805

Table 3: Per-user basis performance comparison between three baselines and two adaptation models.



(a) A heavy user (b) A light user

Figure 5: Performance of two users, (a) is considered a frequent user and (b) is an infrequent user.

5.2.3 Do Adaptation Strategies Help?

Next, we evaluate the performance of the five heuristics for different types of users. To be concrete, we sort users based on their query numbers, which follows a power-law distribution as shown in Figure 1. We separate the users evenly into three categories, each of which has roughly 1,000 users, namely heavy, medium and light users.

(**Strategy 1 Performance**) We first compare the three heuristics in strategy 1 with the baseline adaptation performance. Overall, all three heuristics achieve performance improvement over the baseline adaptation, with 0.6506, 0.6977 and 0.6524 MRR scores for KL divergence (KL), Click Entropy (CE) and Drop Top Click (DT) for the 2-layer framework. Comparatively, the 5-layer RankNet improved even more with the heuristics, which in general outperforms the 2-layer model in terms of MRR scores by increasing roughly 0.15. Among all heuristics, CE has the best improvement while KL improves the least. In Figure 6, we plot the MRR scores in each category of users from the output of the 5-layer RankNet, along with their standard deviations. Most improvement comes from heavy users, where CE improves the MRR for more than 8%, and KL for 2.5% on average. However, for low-frequency users, we observe that KL performs even worse than the baselines, while DT has marginal gains and CE still shows significant improvement. We attribute the reason of different performance improvement to the fact of their query coverage, i.e., how many queries are affected by each heuristic. Since CE measures the global weight of the query, its coverage is always 100%. Meanwhile, KL means user-specific query weights, which requires a query to be issued by both the current user and at least once by one of the remaining users according to eq. (5), and therefore has a much lower coverage. DT, on the oth-

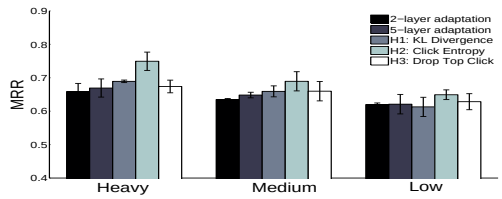


Figure 6: Comparison of three heuristics in strategy 1 with baseline adaptations in terms of MRR scores, divided by three user groups.

Coverage	Heavy	Medium	Low
H1: KL Divergence	43%	25%	6%
H2: Click Entropy	100%	100%	100%
H3: Drop Top Click	44%	31%	18%

Table 4: Percentage of queries re-weighted by three heuristics. Note in DT, re-weighting means setting the query weight to 0 and drop from training.

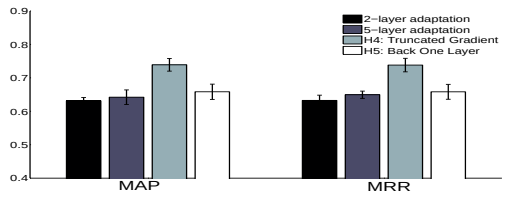


Figure 7: Comparison of two heuristics in strategy 2 with baseline adaptations in terms of MAP and MRR scores.

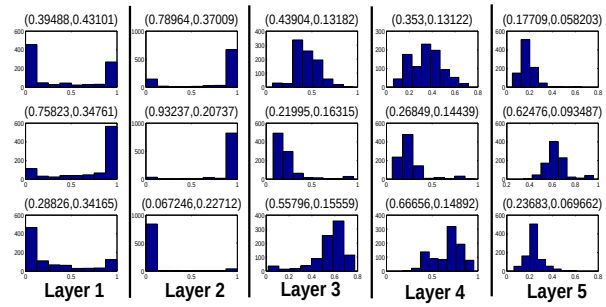


Figure 8: Example of neuron outputs for the validation set on different hidden layer of RankNet. Title indicates (mean, std) of each neuron. Higher-layer neuron generally has less output variances.

er hand, is also a user-specific weighting heuristic, which is determined by the percentage of top-1 clicks of the users.

The statistics from Table 4 confirms our hypothesis. Clearly, we can infer the reason why KL does not perform as well as CE. In general, KL has much lower query coverage than CE. Even for the heavy users, only 43% of the queries are found common among different users, while for low-frequency users, only 6% of queries can benefit from the re-weighting by KL divergence. On the other hand, DT continues to drop more queries when the user type becomes less frequent. In particular, for low-frequency users, DT drops as much as 82% of all queries. We analyzed the logs for a few of low-frequency users, and found out that indeed, those users have a much higher percentage of navigational queries like *amazon* and *yahoo email* than heavy users. Clicks of these queries are mostly on the first results, and therefore filtered from adaptation. In summary, the three re-weighting mechanisms review that heuristics with a high query coverage is critical to achieve good adaptation performance.

(**Strategy 2 Performance**) We now turn to analyzing heuristics of strategy 2, where the focus is more on the regularization of back propagation. Figure 7 plots the overall MRR and MAP scores of the two heuristics comparing with two baseline adaptation methods. The Truncated Gradient (TG) method expresses very compelling performance, which substantially improves the metrics from around 0.63 to 0.73, a 16% relative increase. However, the other heuristic, Back One Layer (BO), does not perform as well. BO only has a marginal 1% increase over the baselines, which are not statistically significant ($p\text{-}value \approx 0.12$).

For a deep understanding of layer-wise performance, we randomly sampled three neurons from each layer of the 5-layer model, and plotted the histograms of the output values given the validation set. Figure 8 shows the results. Note that here Layer 1 is the first/bottom layer of the network and Layer 5 the highest/top layer. The pattern looks quite consistent for neurons within each layer. In general, lower-layer neurons tend to have larger variances than neurons at high layers, which causes less updates to happen in the lower layers since more gradients are truncated during back propagation. From the statistics of over 2 million validation query-URL pairs, the gradients of neurons on layer 5 are truncated roughly 23% of all adaptation pairs, while the gradients of the first-layer neurons are truncated over 79% of the time.

Table 5 shows the comprehensive results of the heuristics performance along with the baseline. From the overall performance as well as user-category specific performance, we can observe that Truncated Gradient outperforms all other heuristics with significance. The results also suggest that applying heuristics on deep network (5 hidden layers) has better effect than shallow ones (2 hidden layers).

5.2.4 *Repeated/non-repeated Query Performance*

In this section, we analyze the ranking performance by using different criteria in terms of queries. Table 6 lists the fairly comprehensive results of the five heuristics. We only report the performance of the 5-layer adaptation due to space limit. First, we separate queries in the test set into either repeated queries and new queries for each user, where repeated queries were previously seen in that user's adaptation query set while new queries not. We see that for all methods, most gains come from the repeated queries, which agrees with previous research results in [21]. For new queries, however, unlike in [21] where sometimes the results got worse than baselines, all our five heuristics are still able to increase the MRR scores comparing to the baselines. Compared to the 5-layer baseline (0.5602), Truncated Gradient has the largest improvement (0.6082), followed by Click Entropy (0.5827), both of which are statistically significant, followed by marginal improvements of the remaining ones.

Next, we compare the individual query performance to the baseline production ranking. Our criteria is to look at

Model	MAP	MRR	Avg. Click Position	MRR For User Category		
				Heavy	Medium	Low
2-layer Adaptation	0.6311	0.6325	1.55006	0.6587	0.6348	0.6198
5-layer Adaptation	0.6421	0.6494	1.54805	0.6693	0.6483	0.6210
KL(2-layer)	0.6541	0.6506	1.51197	0.6893	0.6593	0.6129
Entropy(2-layer)	0.6982	0.6977	1.45615	0.7494	0.6894	0.6492
Drop Top(2-layer)	0.6582	0.6524	1.50758	0.6739	0.6598	0.6284
Truncated Gradient(2-layer)	0.7389	0.7382	1.41913	0.7789	0.7385	0.7194
Back One Layer(2-layer)	0.6583	0.6582	1.44059	0.6878	0.6589	0.6283
KL(5-layer)	0.6672	0.6649	1.43849	0.6928	0.6679	0.6230
Entropy(5-layer)	0.7046	0.7085	1.41837	0.7589	0.6978	0.6589
Drop Top(5-layer)	0.6691	0.6692	1.47823	0.6819	0.6683	0.6375
Truncated Gradient(5-layer)	0.7473	0.7497	1.38973	0.7859	0.7508	0.7239
Back One Layer(5-layer)	0.6696	0.6673	1.42974	0.6928	0.6700	0.6359

Table 5: Overall performance of all heuristics compared to baseline. The performance for different categories of users are also shown. Both RankNet frameworks with heuristics outperform adaptation without heuristics.

the combination of MRR, MAP and click position to see whether a particular query gets improved or worsen, and then calculate the percentage for each heuristic. Table 6 also shows these two comparisons in column 4 and 5. Note that the majority of the queries remain the same performance so these two columns do not add up to 100%. Similarly, Truncated Gradient and Click Entropy improve more queries than other heuristics.

Another similar criterion we used is to see how many test queries are pushed to top rank after adaptation, while how many top-ranked queries are dropped down from the No.1 ranking position. Column 6 and 7 in Table 6 demonstrate that Truncated Gradient and Drop Top push more queries to No.1 than others, while Back One Layer and Truncated Gradient drop less number of top-ranked queries to lower positions than the remaining heuristics.

Finally, we separate the test queries into navigational and informational queries based on user intent. We use a rough classification method here: if more than 75% of all clicks for a query is on the same hyperlink, we classify the query as navigational. Otherwise the query is informational. Our classifier is by no means the state-of-the-art, but is sufficient for our task here to observe performance by different query types. We notice that in general, navigational queries have a much higher MRR scores than informational queries, for both baseline methods and all five heuristics. In comparison, Truncated Gradient still exhibits the most impressive performance, with both categories of queries improved noticeably from the baselines.

5.3 Model Comparison

We conclude our experiments with the performance comparison of the five proposed heuristics. To show the relative strength, we simply average their MAP and MRR scores for each pair of the methods in comparison, i.e., $Score(H_i|H_j) = \frac{1}{2}\left(\frac{MAP(H_i)}{MAP(H_j)} + \frac{MRR(H_i)}{MRR(H_j)}\right)$, which indicates the relative performance of H_i compared to H_j. Figure 9 plots the heatmap of their scores, where each cell can be read, from y-axis to x-axis, as H_i vs. H_j.

Overall, we observe that TG performs best among all, while KL performs the worst. Other than the case that CE outperforms BO, in general, strategy 2 outperforms strategy 1, which suggests that regularizing back-propagation

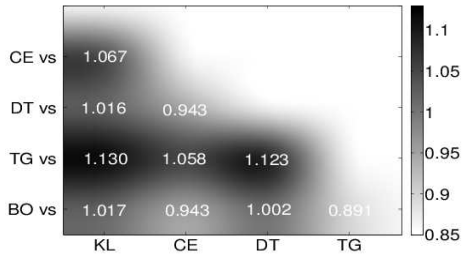


Figure 9: Comparison of the 5 heuristics used in this paper in terms of MAP and MRR scores. Higher values indicate better comparative performance.

can potentially lead better performance gain than adding constraints directly to the adaptation instances. The result also suggests that keeping as much data as possible for adaptation seems better than dropping some portion of the data which we assumed would not help adaptation. A global weighting mechanism for queries in general beats user-specific query weighting, as can be seen by the comparison between CE and KL. Finally, regularizing the gradient for each individual neuron (TG) seems to outperform a layer-wise treatment (BO) in back propagation. In summary, the relative strength of the 5 methods is: $TG > CE > BO > DT > KL$.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a large-scale study of personalized search adaptation, from a deep learning perspective. We proposed to use deep RankNet to first train a global ranking model, and then adapted to each individual user via continue-train. We proposed two strategies to improve the performance of adaptation. Strategy 1 focused on re-weighting the user-specific adaptation data so that more important adaptation instances would be assigned higher importance weights. Strategy 2 emphasized on regularizing the back propagation stage, where we introduced the usage of truncated gradient when updating the neuron weights. We then empirically evaluated our framework on a real-world data set. The results indicated that a deep RankNet with 5 hidden layers in general outperformed a shallow RankNet

Model	Repeated Q MRR	New Q MRR	% Query Improves	% Query Worsen	% Q Push to Rank 1	% Rank-1 Q Drop Down	Nav Q MRR	Info Q MRR
2-layer	0.6893	0.5538	10.30%	6.87%	2.30%	0.82%	0.7283	0.5489
5-layer	0.6913	0.5602	10.89%	6.41%	2.26%	0.78%	0.7385	0.5512
H1: KL	0.7093	0.5603	10.99%	6.35%	2.37%	0.65%	0.7483	0.5389
H2: CE	0.7243	0.5827	12.83%	6.68%	2.44%	0.44%	0.7531	0.5523
H3: DT	0.7033	0.5786	11.17%	6.13%	2.45%	0.67%	0.7392	0.5493
H4: TG	0.7483	0.6082	13.27%	4.19%	2.62%	0.49%	0.7592	0.5747
H5: BO	0.7034	0.5793	11.01%	6.17%	2.07%	0.43%	0.7804	0.5486

Table 6: (5-layer RankNet) Performance breakdown of the five heuristics by different query criteria.

with 2 hidden layers. Among the 5 heuristics we proposed to improve adaptation, the truncated gradient method performed the best, followed by the click entropy method that re-weighted adaptation instances.

There are several future work that we want to explore. We saw that click entropy has one of the best performance in all heuristics. Nevertheless, calculating click entropy for all queries is quite expensive in practice. The system needs to constantly update the weights of all queries whenever new user clicks become available, which is impractical for a production system. We therefore think of coming up with some approximation methods that can easily get the query weights without going back to the search logs constantly. Same for the truncated gradient heuristic, we want to approximate the truncation values more efficiently. On the other hand, since we observed that most performance gain came from heavy users rather than users with fewer queries, one way to improve the performance of light users is to group users together and use other similar users' search information to compensate for the short search history for light users.

7. ACKNOWLEDGEMENTS

We thank Jianfeng Gao for the insightful discussion on deep learning architecture. We thank Lihong Li and Ming-Wei Chang for their suggestions on tuning back propagation.

8. REFERENCES

[1] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[2] P. N. Bennett, F. Radlinski, R. W. White, and E. Yilmaz. Inferring and using location metadata to personalize web search. In *Proc. of SIGIR '11*, pages 135–144. ACM, 2011.

[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proc. of ICML '05*, pages 89–96, 2005.

[4] D. Chen, Y. Xiong, J. Yan, G.-R. Xue, G. Wang, and Z. Chen. Knowledge transfer for cross domain learning to rank. *Inf. Retr.*, 13(3):236–253, June 2010.

[5] Z. Dou, R. Song, and J.-R. Wen. A large-scale evaluation and analysis of personalized search strategies. In *Proc. of WWW '07*, pages 581–590. ACM, 2007.

[6] M. Dredze, K. Crammer, and F. Pereira. Confidence-weighted linear classification. In *Proc. of ICML '08*, pages 264–271. ACM, 2008.

[7] J. Gao, Q. Wu, C. Burges, K. M. Svore, Y. Su, N. Khan, S. Shah, and H. Zhou. Model adaptation via model interpolation and boosting for web search ranking. In *EMNLP*, pages 505–513. ACL, 2009.

[8] B. Geng, L. Yang, C. Xu, and X.-S. Hua. Ranking model adaptation for domain-specific search. In *Proc. of CIKM '09*, pages 197–206. ACM, 2009.

[9] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[10] J.-T. Huang, J. Li, D. Yu, L. Deng, and Y. Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. ICASSP 2013.

[11] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, , and L. Heck. Learning deep structured semantic models for web search using clickthrough dat. In *Proc. of CIKM '13*, 2013.

[12] T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proc. of SIGIR '05*. ACM, 2005.

[13] J. Langford, L. Li, and T. Zhang. Sparse online learning via truncated gradient. *J. Mach. Learn. Res.*, 10:777–801, 2009.

[14] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng. Building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning*, 2012.

[15] P. Mirowski, M. Ranzato, and Y. LeCun. Dynamic auto-encoders for semantic indexing. In *Proc. of the NIPS 2010 Workshop on Deep Learning*, 2010.

[16] X. Shen, B. Tan, and C. Zhai. Context-sensitive information retrieval using implicit feedback. In *Proc. of SIGIR '05*, pages 43–50. ACM, 2005.

[17] D. Sontag, K. Collins-Thompson, P. N. Bennett, R. W. White, S. Dumais, and B. Billerbeck. Probabilistic models for personalizing web search. In *Proc. of WSDM '12*, pages 433–442. ACM, 2012.

[18] J. Teevan, S. T. Dumais, and E. Horvitz. Personalizing search via automated analysis of interests and activities. In *Proc. of SIGIR '05*, pages 449–456, 2005.

[19] J. Teevan, S. T. Dumais, and D. J. Liebling. To personalize or not to personalize: modeling queries with variation in user intent. In *Proc. of SIGIR '08*, pages 163–170, 2008.

[20] B. Wang, B. Liu, X. Wang, C. Sun, and D. Zhang. Deep learning approaches to semantic relevance modeling for chinese question-answer pairs. *ACM Transactions on Asian Language Information Processing*, 10(4), Dec. 2011.

[21] H. Wang, X. He, M.-W. Chang, Y. Song, R. W. White, and W. Chu. Personalized ranking model adaptation for web search. In *Proc. of SIGIR '13*, pages 323–332. ACM, 2013.

[22] R. W. White, P. Bailey, and L. Chen. Predicting user interests from contextual information. In *Proc. of SIGIR '09*, pages 363–370. ACM, 2009.

[23] R. W. White, P. N. Bennett, and S. T. Dumais. Predicting short-term interests using activity-based search context. In *Proc. of CIKM '10*, pages 1009–1018. ACM, 2010.

[24] R. W. White, W. Chu, A. Hassan, X. He, Y. Song, and H. Wang. Enhancing personalized search by mining and modeling task behavior. In *Proceedings of WWW '13*, pages 1411–1420, 2013.

[25] D. Yu, K. Yao, H. Su, G. Li, and F. Seide. Kl-divergence regularized deep neural network adaptation for improved large vocabulary speech recognition. ICASSP 2013, 2013.