Personalized recommendation for extremely sparse and large scale data

Tongda Zhang
Department of Electrical Engineering
Stanford University
Email: tdzhang@stanford.edu

Zhifeng Sun
Google, Inc.
Kirkland, WA, USA
Email: zhsun@google.com

Xintian Yang
Google, Inc.
Kirkland, WA, USA
Email: xyang@google.com

1. Introduction

Personalized recommendation is a critcal component for most web systems, including online advertising, e-commerce websites, social network, etc. Let's take online advertising as an example. It's a multi-billion dollar business, accounts for majority of the revenue for companies like Google and Facebook. It's also the main income source for millions web content publishers and most e-commerce websites. Therefore, the impact of designing good personalized recommendations is huge.

In this paper, we present an innovative machine learning model for personalized recommendations that can handle extremely sparse and large scale data. Since our team works on the personalized keyword recommendation for advertisers in Google Adsense [4], for the rest of this paper, we will use our project as an example. But by no means our model is limited to this example. It can be easily applied to other personalized recommendation systems.

Google Adsense allows publishers in the network of content sites to serve automatic advertisements (for text, image, videos, or interactive media), which are targeted to site content and audience. It generates billions of dollars each year, supports hundreds of millions of publishers on the Internet eco-system, and can reach more than 80% of all Internet users worldwide in more than 30 languages and over 100 countries. All these make it one of advertiser' favorite online advertising systems. In a nutshell, how it works can be illustrated by Figure 1.

- *Publisher* posts content on the Internet, and insert a code snippet into its web pages.
- *User* visits the web page, which triggers the code snippet to pull relevant advertisements from Google Adsense servers, and show them on the same web page.
- If user click on the advertisement, the *advertiser* who created it will pay Google certain amount of money, and Google will share majority of that payment with the content publisher.

To make the best out of Adsense, it requires the advertisers to provide a good list of keywords, to match advertisements with web contents more precisely, hence increase the click through rate and better advertisement performance. However, the reality is most advertisers are lack of experience to provide good keywords for their advertisements. Therefore, intelligently making use of historical data to provide good personalized keyword recommendations to advertisers is crutial to the success of Adsense. There are mainly two types of approaches.

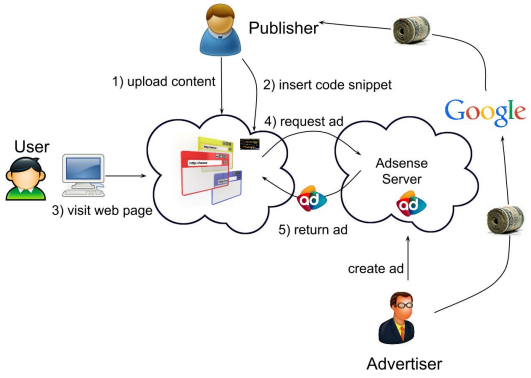

Figure 1. Google Adsense system work flow.

- A/B-testing [1]. In this approach, we split the web traffice into two parts: majority of the traffic uses the original keyword list to match advertisement, and the other small fraction of the trafic uses the new keyword list (original keyword list along with some semantically relevant keywords). Do this experiment for some period of time (usually in days or weeks) and compare the stats. If the new list performs significantly better, we then surface this recommendation to the advertiser. The main drawback of this approach is the long turnaround time. Lots of business opportunities may be lost after days or weeks of experiments.
- Model based solutions using collaborative filtering [2], [3]. This approach makes use of machine learning technology and predicts the performance of the new keyword list without running experiments. There are three type of collaborative filtering models that are commonly used, user neighborhood model (considering the similarity between users, in our case will be advertisers), item neighborhood model (con-

sidering the similarity between items, in oursecase will be keywords), and matrix factorization based model []. The matrix factorization based model has drawn a lot of attention recently due to its success in the Netflix Challenge []. It's proven to produce better estimates than user neighborhood or item neighborhood models, and can handle sparse data set like the data in Netflix Challenge. However, the data set we deal with is more than 10 times sparser than the Netflix data. We observe significant overfitting even with very strong regularization.

As a result, we went out and designed a novel new machine learning model that is resilient to extremely sparse data set and doesn't need to run days of experiments. First, the complexity of the model (a.k.a. the number of parameters) grows with respect to observed data, not to the number of advertisers and number of keywords like matrix factorization based models. This makes our model resilient to extremely sparse data. Secondly, our training algorithm is based on gradient descent, which is easy to parallelize, and thus can handle very large scale data set. Thirdly, our model combines the user neighborhood and item neighborhood ideas in collaborative filtering smartly, where similarities are "learned" from training algorithm other than defining a global similarity metric for neighborhood. This makes the estimation quality of our model very good.

In summary, our main contributions in this paper include:

- We successfully exploit the potential of improving personalized recommendation for extremely sparse and large scale data.
- Novel ideas in model design include: (a) control model complexity with respect to observed data; (b) intelligently combine ideas in collaborative filtering.
- We have applied our model successfully to personalized keyword recoomendation system in Google Adsense. And we have conducted extensive comparison experiments with matrix factorization based methods.
- We have implimented a distrubted training system that can handle large scale data set and is ready to apply to other personalized recommendation problems.

The remainder of this paper is organized as follows. We first formulate the problem in Section 2. Then we describe the details of our model, including the high-level workflow, intuition and assumptions, as well as different components of the model in Section 3. After that, we introduce the training algorithms in Section 4. We present experimental results in Section 6, and related works in Section 7. Finally, we conclude this paper in Section 8.

2. Problem Formulation

The input traffic estimation data can be viewed as a matrix, as shown in Figure 2. Each row represents an advertisement and each column represents a keyword. If a keyword i is already associated with advertisement u, then we have some observed traffic value β_{ui}. Otherwise, no value is observed (noted as X).

ADD THE MEASUREMENT METRIX HERE: precision? Given the input traffic estimation matrix M, the goal for our model is to predict the value of unobserved items X. We measure the overall precision of prediction using the formula (???) Our goal is to maximiz the prediction accuracy in Eq ()?

change the matrix figure as follows: make the X as red highlighted

Keyword / Advertiser	1	⋯	i	⋯	j	⋯
1	β_{11}	⋯	X	⋯	X	⋯
⋮	⋮	⋱	⋮		⋮	
u	X	⋯	β_{ui}	⋯	β_{uj}	⋯
⋮	⋮		⋮	⋱	⋮	
v	X	⋯	β_{vi}	⋯	β_{vj}	⋯
⋮	⋮		⋮		⋮	⋱

Figure 2. Matrix ??

We approach this problem by mining the homogeneous similartiy across advertisement and keyword dimentions, in another word, similarity ratio of different keywords between similar advertisement should be close to each other. Instead of using only item-based or user-based method, it is a synthetic multi-dimension model, which evaluates the similarity cross item and user dimensions. Also, different from item-based or user-based method, which uses a predefine static similarity function and calculate the global optimal similarity, we are using a local optimal function, which can dynamic involving similarity using learned parameters, hence can support large scale data in an efficient computation cost.

3. Similarity Amplifier Network

We first give an overview of our solution, and then describe its deployment in production.

3.1. Model Intuition

Our model is inspired by the similar idea of collaborative filtering that the preference of a user on an item is predicted based on the preferences of other users with similar interests. There are mainly two categories of collaborative filtering, either in user-centric or item-centric manner:

- item-based collaborative filtering: build an item-item matrix determining relationships between pairs of items, then infer the tastes of the current user by examining the matrix and matching that user's data, e.g. users who bought x also bought y

- user-based collaborative filtering: look for users who share the same rating patterns with the active user (the user whom the prediction is for). Then use the ratings from those like-minded users found in step 1 to calculate a prediction for the active user, e.g. people also looks at the following items

Both item-based and user-based models are trying to emphasize only one perspective of the problem, but did not consider the correlation cross them.

Is it possible to use the information from both dimensions to further improve the model's prediction performance?

The answer is positive. Why? because the two dimention model further amplify the effect of homogeneous cross both item and users, which we named as *Homogeneous Amplifying Effect* in this paper. NEED MORE EXPLAINATION IN HUMAN LANGUAGE

The main idea of our model is illustrated by Figure 3, where u, v, w are advertisements, and i, j, k are keywords, and the value in each cell is the average daily clicks for the corresponding advertisement from the keyword query. Let's say we want to predict the average daily click value if we want to add keyword k to advertisement v. For keyword pair k and j, we observed ratio $20/12$ in advertisement u and $5/18$ in advertisement w. If we can derive the performance ratio between k and j in advertisement v, then we can make use of j's observed daily average clicks in v to give one estimate of k's daily average clicks in v. To do so, we introduce a set of parameters called *similarity*, one for each pair of advertisements, to capture how similar the performance ratios will be between advertisements. Let S_{uv}, S_{vw} be the similarity between u, v and v, w respectively. The estimate given by keyword pair k, j can be expressed as

$$6 \times \left(\frac{20}{12} \times S_{uv} + \frac{5}{18} \times S_{vw} \right) \div (S_{uv} + S_{vw})$$

Similarly, we can have another estimate by keyword pair k, i, which is

$$9 \times \left(\frac{20}{5} \times S_{uv} + \frac{5}{22} \times S_{vw} \right) \div (S_{uv} + S_{vw})$$

Now how do we combine these estimates? We introduce another set of parameters called *confidence*, one for each pair of keywords. Therefore, the final estimate can be represented as the linear combination of the above two estimates, weighted by the confidences of keyword pairs k, j and k, i. All the parameters are learned from SPPAN trainer by gradient descent algorithm. They will be introduced later in Section 3.2.1 and 3.2.2.

3.2. Modeling Similarity Amplifier Network

Under the context of advertisement traffic estimation, it can be interpreted as:

- Similar advertisers are likely to have similar traffic ratios between the same pair of keywords, i.e. $\beta_{uj}/\beta_{ui} \approx \beta_{vj}/\beta_{vi}$ where u and v are "similar" adgroups.

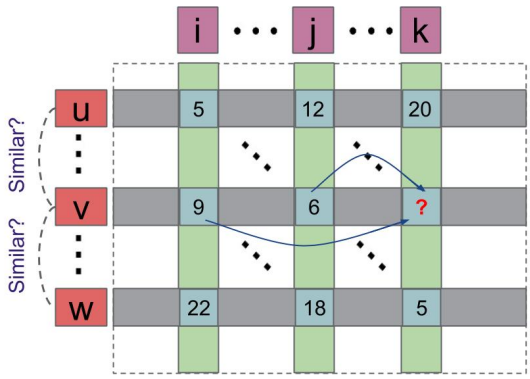

Figure 3. High level idea of SPPAN model.

- Similar keywords are likely to have similar entry value ratios between the same pair of advertisers, i.e. $\beta_{vi}/\beta_{ui} \approx \beta_{vj}/\beta_{uj}$ where i and j are "similar" criteria.

Table 1 shows the notations in this paper, specially u and v will be used for advertisers, while i and j will be used for keywords.

Notation	Description
N_a	total number of advertisements
N_k	total number of keywords
a_u	the uth advertisement, $u \in [1, N_a]$
k_i	the ith keywords, $i \in [1, N_k]$
Λ	missing values in data matrix M
$\Lambda_{(*,i)}$	missing values in the ith column of matrix M
$\Lambda_{(u,*)}$	missing values in the uth row of matrix M
Γ	observed (known) values in the matrix M
$\Gamma_{(*,i)}$	observed values in the ith column of matrix M
$\Gamma_{(u,*)}$	observed values in the uth row of matrix M

TABLE 1. NOTATIONS FOR SPAN MODEL

3.2.1. Similarity Graph. Similarity Graph is the graph which describes the similarity between different advertisers. To be more specific, it is a complete undirected graph $G_{\text{sim}} =< V, E >$, where V is the set of all the advertisers $V = \{a_1, a_2, ..., a_{N_a}\}$. For each edge in R, there is a value which represents how similar the corresponding two advertisers are with each other. Let θ_{uv} denote the similarity between advertiser a_u and a_v.

3.2.2. Pairwise Amplifier Graph. Pairwise Amplifier Graph contains all the ratio information between different pairs of keywords. It is defined as a directed graph $G_{\text{amp}} =< V, E >$, where V contains all the keywords $V = \{k_1, k_2, ..., k_{N_k}\}$, and E contains all pairs of keywords that have been targeted concurrently in at least one advertiser, $E = \{(k_i, k_j) \mid i \in [1, N_k], j \in [1, N_k], \exists u \in [1, N_a] \text{ s.t. } \beta_{ui}, \beta_{uj} \in \Gamma\}$. For each edge $(k_i, k_j) \in E$, there

are a corresponding confidence value $conf_{ij}$ and an amplifier set $amp_{ij} = \{(a_u, gain_{ij}^u) \mid u \in [1, N_a], \beta_{ui}, \beta_{uj} \in \Gamma\} \cup \{(a_0, c_{ij})\}$, where $gain_{ij}^u = \beta_{uj}/\beta_{ui}$, and (a_0, c_{ij}) are just the bias terms which we will use later in the model.

4. Training a Simplify Amplify Network

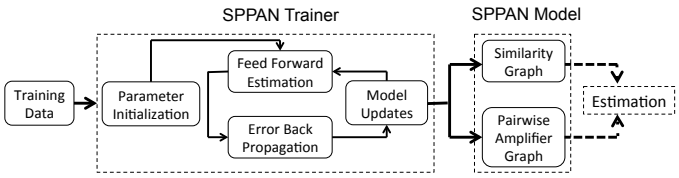

Figure 4. SPAN system overview.

The mainl workflow of SPAN model is summarized in Figure 4. The training dataset can be regarded as a sparse 2D matrix with known entries shown in Figure 2. Given the training dataset, the training process starts with a set of initial parameters, then there is an update loop between feeding forward estimation and backword propogation, in which *Feed Forward Estimation* uses the given data points to estimate the value of a target entry. *Error Back Propagation* updates current model parameters. The training process is in an incrementally update way, and generate the final model which is composed of similarity graph and pairwise amplifier graph. The learned SPAN model will be directly used to estimate missing values. In the following sections, we will discuss each component in details.

4.1. Feed Forward Estimation

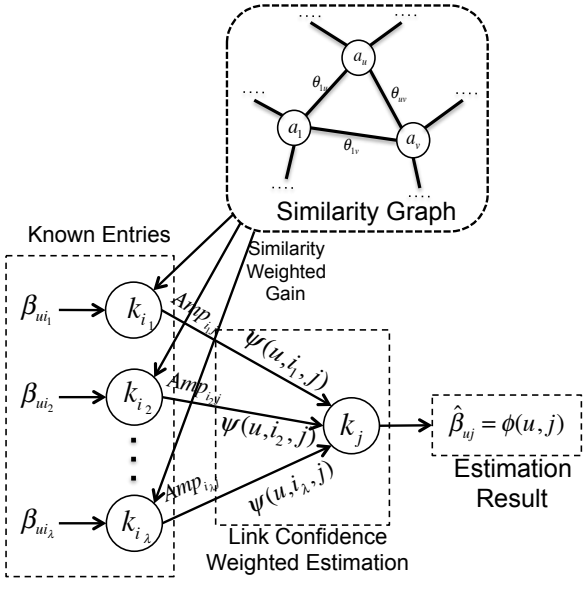

Figure 5. Feed Forward Estimation

ADD MORE DESCRIPTION HERE, need to understand what is two layers calculation units? ...

Given particular values for initial parameters and the inputs data, to predict parameter settings that minimize the error, our model uses feed-forward network: the inputs feed into a layer of hidden units, which can feed into layers of more hidden units, which eventually feed into the output layer. Each of the hidden units is a squashed linear function of its inputs.

Our SPAN model uses known entry values $\{\beta_{ui_1}, \beta_{ui_2}, \cdots, \beta_{ui_\lambda}\}$ (where λ is the number of known entry values of advertiser u) to estimate the value of target entry β_{uj}. It uses feed forward network to estimate the parameters in a layer-by-layer way: takes known entry values as inputs, it goes through a two layers of calculation units to get the final estimated result. Similarity weighted sum of entry estimation (in Equation 2.) and a link confidence weighted sum of entry estimation (in Equation 1) are used to estimate the value of an unknown entry $\beta_{uj} \in \Lambda$.

As shown in Figure 5, the first layer make estimation of target β_{uj} using $\{\beta_{ui_1}, \beta_{ui_2}, \cdots, \beta_{ui_\lambda}\}$ separately with equation 1, which generates λ output estimated values. The second layer takes all the output estimations from the first layer $\{\psi(u, i_1, j), \psi(u, i_2, j), \cdots \psi(u, i_\lambda, j)\}$ as inputs to calculate the final prediction using equation 2.

$$\begin{cases} \psi(u,i,j) = \frac{\sum_{v \in V} \theta_{uv} \cdot gain_{ij}^v}{\sum_{v \in V} \theta_{uv}} \cdot \beta_{ui} \\ V = \{v | (a_v, gain_{ij}^v) \in amp_{ij}\} \end{cases} \tag{1}$$

$$\begin{cases} \hat{\beta}_{uj} = \phi(u,j) = \frac{\sum_{i \in I} conf_{ij} \cdot \psi(u,i,j)}{\sum_{i \in I} conf_{ij}} \\ I = \{i | \beta_{ui} \in \Gamma_{(u,*)}\} \end{cases} \tag{2}$$

4.2. Error Back Propagation

In Feed Forward Estimation process , we assume the similarity parameters of advertiser pairs (in Similarity Graph) and the link confidence parameters in the Pairwise Amplifier Graph are given. Indeed the model needs to learn these parameters from the training dataset using error back proprgation. The idea of error back propagation is very similar to artificial neural networks: for an entry in the training dataset, we hide this entry from the rest of the training dataset and use the feed forward estimation algorithm in Section 4.1 to estimate its value with the rest of the training dataset. Then we compare the true value of that entry with current estimation, and back propagate the estimation error (the deviation from estimated value to the true value) to update the parameters. The idea is shown in Figure 6. Gradient descent method [**?**] is used to optimize the model parameters, which calculates the gradient of a loss function with respects to all the weights in the network. The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the loss function.

Before developing the parameter update function, we need to calculate the derivative of the squared error function with

respect to all the parameters in SPAN model, including both $\{\theta_{uv}\}_{u,v\in[1,N_a]}$ and $\{conf_{ij}\}_{(i,j):(k_i,k_j)\in E(G_{amp})}$.

The objective function of the parameter optimization problem is shown in Equation 3, which is the sum of squared estimation error for all the entries in the training dataset.

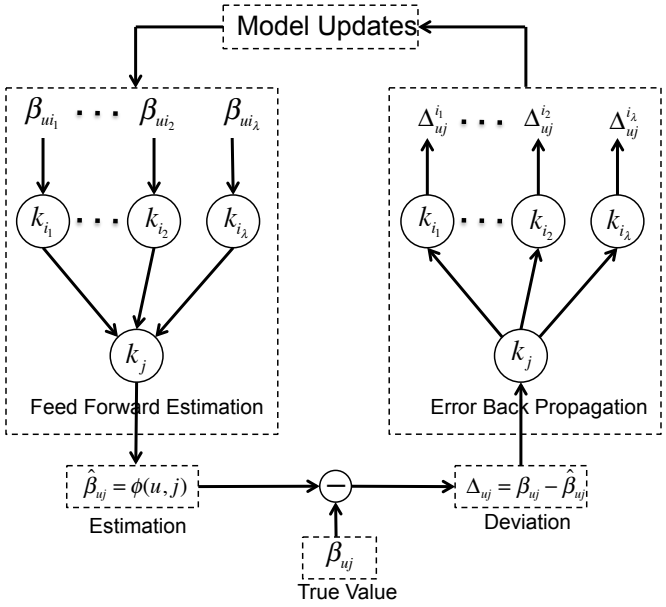

Figure 6. The Loop of the Parameter Update

$$\begin{aligned} J &= \frac{1}{2}\sum_{u,i:\beta_{ui}\in\Gamma}\left(\beta_{ui}-\hat{\beta}_{ui}\right)^2 \\ &= \frac{1}{2}\sum_{u,i:\beta_{ui}\in\Gamma}\left(\beta_{ui}-\phi(u,i)\right)^2 \\ &= \frac{1}{2}\sum_{u,i:\beta_{ui}\in\Gamma}\Delta_{ui}^2 \end{aligned} \tag{3}$$

Where β_{ui} is the true value, and $\hat{\beta}_{ui} = \phi(u,i)$ is the estimated value given by Feed Forward Estimation algorithm we described in Section 4.1. The factor of $\frac{1}{2}$ at the beginning of the objective function is a constant factor which would simplify the expression after the differentiation operation. Noted that an arbitrary learning rate factor will be multiplied with this expression in our later learning process, so that it doesn't make any differences if a constant coefficient is introduced here.

With Equation 1, 2 and 3, the derivatives of our objective function with respect to parameters $\{\theta_{uv}\}_{u,v\in[1,N_a]}$ and $\{conf_{ij}\}_{i,j\in[1,N_k]}$ are

$$\frac{\partial J}{\partial \theta_{uv}} = \sum_{j:\beta_{uj},\beta_{vj}\in\Gamma}\frac{\partial\left[\frac{1}{2}\left(\beta_{uj}-\hat{\beta}_{uj}\right)^2\right]}{\partial\theta_{uv}}$$

and

$$\frac{\partial J}{\partial conf_{ij}} = \sum_{u:(a_u,gain_{ij}^u)\in amp_{ij}}\frac{\partial\left[\frac{1}{2}\left(\beta_{uj}-\hat{\beta}_{uj}\right)^2\right]}{\partial conf_{ij}}$$

With further formula deductions, we can have Equation 4 as the derivative result of the objective function with respect to $\{\theta_{uv}\}_{u,v\in[1,N_a]}$, and Equation 5 as the derivative result of the objective function with respect to $\{conf_{ij}\}_{i,j\in[1,N_k]}$.

$$\begin{cases} \frac{\partial J}{\partial\theta_{uv}} = \sum_{j\in J}\sum_{i\in I}\frac{-\Delta_{uj}^i}{\sum_{v'\in V}\theta_{uv'}}\left(gain_{ij}^v\cdot\beta_{ui}-\psi(u,i,j)\right) \\ J=\{j|\beta_{uj},\beta_{vj}\in\Gamma\} \\ I=\{i|\beta_{ui}\in\Gamma_{(u,*)}\} \\ V=\{v|(a_v,gain_{ij}^v)\in amp_{ij}\} \end{cases} \tag{4}$$

$$\begin{cases} \frac{\partial J}{\partial conf_{ij}} = \sum_{u\in U}\left[-\frac{\Delta_{uj}}{\sum_{i'\in I}conf_{i'j}}\left(\psi(u,i,j)-\phi(u,j)\right)\right] \\ U=\{u|(a_u,gain_{ij}^u)\in amp_{ij}\} \\ I=\{i:\beta_{ui}\in\Gamma_{(u,*)}\} \end{cases} \tag{5}$$

where Δ_{uj} is the estimation error and Δ_{uj}^i is back propagated error, their expressions are given by Equation 6.

$$\begin{cases} \Delta_{uj} = \beta_{uj}-\hat{\beta}_{uj} \\ \Delta_{uj}^i = \frac{conf_{ij}}{\sum_{i':\beta_{ui'}\in\Gamma_{(u,*)}}conf_{i'j}}\cdot\Delta_{uj} \end{cases} \tag{6}$$

With the derivatives and back propagated errors given by Equation 4, 5 and 6, we can update the parameter of SPPAN model in the Similarity Graph and the Pairwise Amplifier Graph using the gradient descent approach. Given a specific learning rate η, the changes of model parameters are equal to the product of the learning rate and the corresponding gradient value, multiplied by -1. Therefore, the parameter update function for training target β_{uj} can be expressed in Equation 7.

$$\begin{cases} conf_{ij} := conf_{ij}-\eta\cdot\frac{\partial J}{\partial conf_{ij}} \\ \theta_{uv} := \theta_{uv}-\eta\cdot\frac{\partial J}{\partial\theta_{uv}} \end{cases} \tag{7}$$

where $\frac{\partial J}{\partial conf_{ij}}$ and $\frac{\partial J}{\partial\theta_{uv}}$ are given by Equation 4, 5 and 6. Figure 6 shows the parameter update loop of the Feed Forward Estimation and the Error Back Propagation for a training target entry β_{uj}.

4.3. Training Algorithm for SPAN model

Combining the Feed Forward Estimation and Error Back Propagation, The training algorithm for SPPAN model can be summarized in Algorithm 1.

The stop criterion for the training algorithm is a training error threshold: the training will stop if the training error drops to a specific value. Other criterion may also be used,

such as maximum iteration number [**?**], error reduction rate threshold [**?**], the convergence of the parameters [**?**] and so on.

The training algorithm generates final SPAN model, which includes the Similarity Graph and Pairwise Amplifier Graph. To estimate unknown entries, we can just follow the Feed Forward Estimation described in Section 4.1, using the well-trained Similarity Graph and Pairwise Amplifier Graph.

Data: The set of given/known entries Γ in matrix M
Learning rate η
Result: $\{\theta_{uv}\}_{u,v\in[1,N_a]}$ in adgroups similarity graph
$\{conf_{ij}\}_{(i,j):(k_i,k_j)\in E(G_{amp})}$ in criteria pairwise amplifier graph
Initialization:

- initialize the pairwise amplifier set $Amp_{ij} = \{(a_u, gain^u_{ij}) \mid u \in [1, N_a], \beta_{ui}, \beta_{uj} \in \Gamma\} \cup \{(a_0, c_{ij})\}$ using $gain^u_{ij} = \beta_{uj}/\beta_{ui}$
- initialize parameters $\{\theta_{uv}\}$ in adgroup similarity graph using small random numbers when encountered.
- initialize parameters $\{conf_{ij}\}$ in criteria pairwise amplifier graph using small random numbers when encountered.

begin
  while *stop criterion not meet* **do**
    for *each* β_{uj} *in* Γ **do**
      /* SPPAN Model feed forward estimation of training example β_{uj} */
      /* with Equation 1 and 2 */
      $\hat{\beta}_{uj} = FeedForwardEstimation(\text{SPPAN Model}, u, j)$
      /* error back propagation using Equation 6 */
      $(\Delta_{uj}, \{\Delta^i_{uj}\}) = ErrorBackPropagation(\text{SPPAN Model}, u, j, \hat{\beta}_{uj})$
      /* update SPPAN model using gradient descent Equation 4, 5 and 6 */
      $update(\text{SPPAN Model}, \Delta_{uj}, \{\Delta^i_{uj}\})$
    end
  end
  return (SPPAN Model)
end

Algorithm 1: Training Algorithm for SPPAN Model

4.4. Handling Large Scale Data

Algorithm 1 works well in most cases. However, as the data set become much bigger and less sparse, the iterative process of error back propagation based parameter update for SPPAN model often take a great deal of time to completely go through the whole training set. Another advantage of our model is that it can handle large scale sparse data in an effective way. As we can see from Algorithm 1, the SPPAN model updates can use different independent training samplesr. Thus, map-reduce techniques can be used to greatly decrease the amount of time that the training algorithm takes to converge.

For each training iteration, the mapping part can takes each training example separately and executes the feed forward and error backward propagation in parallel to generate model updates. Then, the updates for all the parameters of SPPAN model are summed up in the reducing units. At the end of each iteration, the SPPAN model can be updated using the outputs from the reducing unites. This process continues until the stop criterion is met.

4.5. Handling Extreme Sparse Data

add a section about how to handle extreme sparse data, explain why your model can handle extrem sparse data

5. System Architecture

This section explains how we implement our SPAN model and deploy into Google's Adsense system. add one architecture photo, add some description about how does the systme work

6. Experiments

Based on Section 3 and 4, we have implemented the proposed SPPAN model in both Python and C++ that can run on a single machine. We also developed an parallel C++ version of SPPAN model based on Map-reduce.

In this section, we provide evaluation experiments for the performance of the SPPAN model. More specifically, we use the Python version SPPAN model, and compared its performance with a baseline method and two Nonnegative Matrix Factorization based methods provided by nimfa Python library [5]:

- *Baseline*: The baseline method is giving prediction simply based on the average value in the training set.
- *LSNMF*: It is based on Alternating nonnegative least squares matrix factorization using projected gradient method for subproblems [6].
- *NMF*: It is based on Standard nonnegative matrix factorization with Euclidean / Kullback-Leibler update equations and Frobenius / divergence / connectivity cost functions [7], [8].

To provide more interpretable experiment results and for data privacy considerations, we show the performance result of NMF, LSNMF and SPPAN as their relative performance to the baseline method in all the following subsections. The results of the evaluation experiments show salient advantages of SPPAN model compare to those two NMF based methods:

- The matrix factorization based methods LSNMF and NMF are not handling the present extreme sparse

CTR data set very well. Their prediction accuracy is even worse than the baseline method.

- In terms of the training and testing error (Root-mean-square deviation), SPPAN model performs more than 65% and 40% better than the matrix factorization based methods, and around 60% and 30% better than the baseline method.
- SPPAN model has a normal error distribution centered at 0, which means a balanced estimation; while LSNMF and NMF have biased error distributions that are constantly under estimating.

6.1. Data Description

Our evaluation experiments are performed on a click-through rate (CTR) dataset generated from Google Adsense, which was collected with appropriate end-user license agreement and was fully anonymized without any retrievable personally identifiable information.

The CTR data set can be considered as a 2D matrix shown in Figure 2, where each row represents an advertiser and each column represent a target keyword. The value of each matrix entry in Figure 2 is the weekly average click-through rate (CTR) that we observed for the corresponding advertiser on that specific keyword. The whole dataset is extremely sparse because there are hundreds of thousands of different keywords but advertisers usually only target at several of them. To be more specific, the dataset contains more than 400k different advertisers (number of rows) and 500k keywords (number of columns), but more than 99.98% entries in the dataset matrix are missing.

Because the two Nonnegative Matrix Factorization based methods provided by nimfa cannot handle the entire CTR dataset ($400k \times 500k$ sparse matrix) on a single machine, we also generated a sub-sampled version of data set in addition to the original one. To generate the sub-sampled dataset, we go through each entry in the original dataset and only keep that entry with specified probability. If all the entry for a row (advertiser) or column (keyword) are dropped, we simply remove that row (advertiser) or column (keyword) all together. In Table 2, we summarized the basic information of both the 10% sub-sampled version and original data set, including the number of different advertisers, keywords and known CTR entries. It can be seen that both datasets are extremely sparse with more than 99.98% of the entries missing.

Dataset	whole	10% sub-sampled
Advertisers	$\sim 400K$	$\sim 76K$
Keywords	$\sim 511K$	$\sim 73K$
Known CTR entries	$\sim 21M$	$\sim 2M$

TABLE 2. BASIC INFORMATION

6.2. Evaluation

To compare the estimation performance between LSNMF, NMF and SPPAN methods, we separate the sub-sampled CTR dataset described in Section 6.1 into a 95% training set and a 5% testing set through random selecting. The experiment is very straightforward. For each method, we use the 95% training set for model training, and test the model on estimating the entry values in the 5% testing set.

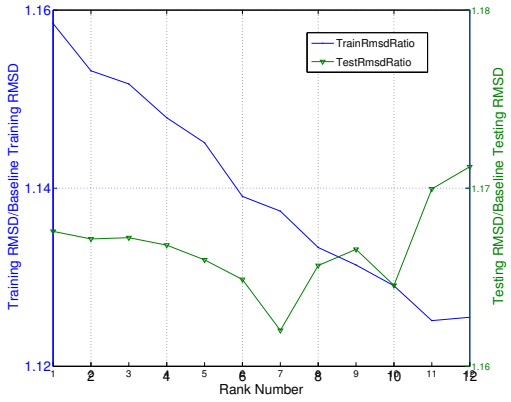

Figure 7. LSNMF: Relative Training/Testing Error Versus Rank Number

For the Nonnegative Matrix Factorization based method LSNMF and NMF, the Factorization Matrix Rank need to be specified before training the model. Using a higher rank number often means a more complicated matrix model, which is more powerful to represent a dataset but with higher risk of over-fitting. On the other hand, using a lower rank number means a simpler matrix model, which is less possible to be over-fitted but might be too simple to represent the data set. We have tried rank number from 1 to 12 for both LSNMF and NMF methods, their results are very similar. The result training error and testing error versus rank parameter of LSNMF method is shown in Figure 7.

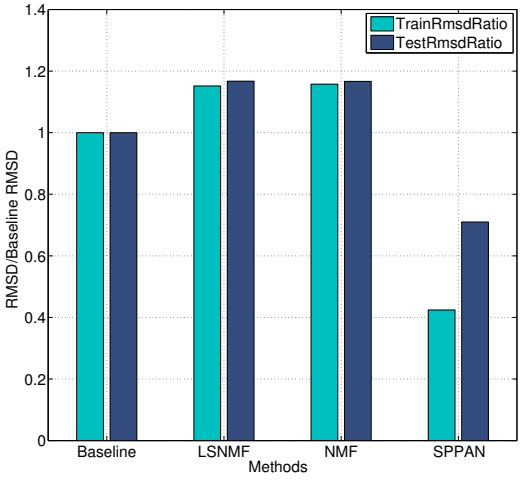

Figure 8. Compare Relative RMSD Between Different Methods

For a fair comparison with proposed SPPAN model, we selected the rank numbers with the best test errors for both

LSNMF and NMF. The best rank numbers for LSNMF and NMF are 3 and 7 respectively.

Method	TrainRmsdRatio	TestRmsdRatio
Baseline	1	1
LSNMF	1.15	1.17
NMF	1.16	1.17
SPPAN	0.42	0.71

TABLE 3. RELATIVE RMSD IN THE BEST TEST ERROR ROUND

Here we list the result relative Root-mean-square deviation(RMSD) of both training set and testing set from Baseline, LSNMF, NMF and SPPAN in Table 3 and Figure 8 to compare the estimation performance among these three methods. The root-mean-square deviation (RMSD) is a measure of the differences between the estimated value using a model and the true observed value. It can be expressed using Equation 8, where n is the number of entries that need be to estimated, and $\hat{\beta}_i$ and β_i are the estimated value and observed value of entry i respectively. The RMSD represents the sample standard deviation of the differences between estimated values and observed values [9].

$$RMSD = \sqrt{\frac{\sum_{i=1}^{n}(\hat{\beta}_i - \beta_i)^2}{n}} \tag{8}$$

In order to make those results more meaningful, we use the relative criteria TrainRmsdRatio and TestRmsdRatio, that are the Train/Test RMSD of current methods divided by the corresponding RMSD of the baseline method:

$$TrainRmsdRatio = \frac{\textit{Train RMSD}}{BaselineTrainRMSD}$$

$$TestRmsdRatio = \frac{\textit{Test RMSD}}{BaselineTestRMSD}$$

As can be seen from Figure 8, even though we chose the best rank numbers for the Nonnegative Matrix Factorization based methods, they perform even worse than the baseline method in this extreme sparse case. SPPAN model, on the other hand, outperforms the Baseline, LSNMF and NMF on both training and testing RMSD. To be more specific, the training and testing RMSD of SPPAN are only around 40% and 70% of the training and testing RMSD of Baseline methods, respectively; they are aroung 35% and 60% of the training and testing RMSD of the Matrix Factorization based methods LSNMF and NMF; which shows a great improvement.

We also draw the histogram of estimated error generated using SPPAN and LSNMF on the testing set as in Figure 9. The estimated error is defined in Equation 9. For data privacy considerations, we multiply an constant k in Equation 9, which doesn't not affect the interpretation of the following analysis at all. Note that the testing error distribution of LSNMF and NMF are very similar. Thus we regard the LSNMF result in Figure 9 as the representative result of Nonnegative Matrix Factorization based method.

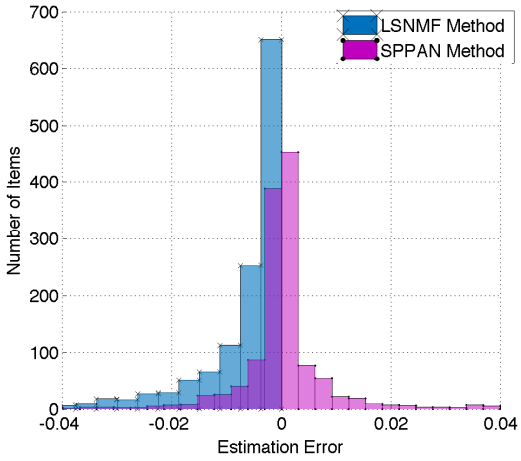

Figure 9. Histogram of Estimated Error

$$Estimated\ Error = k \cdot (Estimated\ CTR - True\ CTR) \tag{9}$$

It can be noticed that the distribution of estimation error of SPPAN model is centered on 0 with a normal distribution shape, while LSNMF's estimation errors are distributed on the left side of 0. It means the estimations given by LSNMF always have a negative bias which leads to a constant under-estimation. Therefore, the estimations from the SPPAN model is more accurate and reasonable.

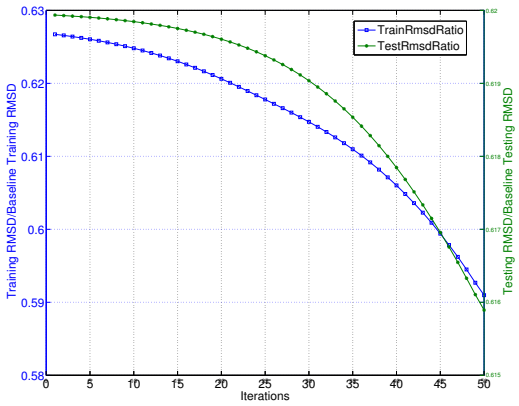

Figure 10. SPPAN Learning Curve on Whole Dataset

Besides the experiments on 10% sub sampled data set, it is worth mentioning that the proposed SPPAN model can also handle the whole CTR data set on a single computer. Similar to the model evaluation experiment on the sub-sampled data set, we separate the whole CTR data set described in Section 6.1 into a 95% training set and a 5% testing set through random selection. Then, we use the 95% training set to train the SPPAN model, and test the model on the 5% testing set. The output training RMSD ratio and testing RMSD

ratio are 0.59 and 0.62 respectively, which shows a great improvement from the baseline method.

We also show the learning curves of SPPAN model in Figure 10. It can be seen that both the training error and testing error were decreasing are decreasing during each training iterations until the training stop criterion was met, which is a strong indicator that the SPPAN model is not over-fitted on the whole CTR data set.

7. Related Work

Among the prevalence in online advertisements system, e-commerce platform and social networks, there has been an increasing interest in developing recommendation systems [2], [10] that can automatically predict the "preference", "rating", "performance" or "interest" of a item for a end user or a client based the history data, such as the task we introduces in Section 1. One popular technique that has been used in many recommendation systems is called collaborative filtering, which is based on the intuition that if two users(or clients) have a similar opinion on an item(or issue), they are more likely to have similar opinions on other items(or issues) [2], [3], [10], [11]. Generally, collaborative filtering is an approach that make use of the preference information from many users to give prediction of the interest of one user. There are many forms of collaborative filtering systems. For examples, Linden et al. uses the item-to-item collaborative filtering to offer personalized recommendations for each customer in the online store [12]; Cai et al. captures the interaction between users within a social network and formulates a collaborative filtering approach to allow high quality people to people recommendations in social networks [13]; Hu et al. implements a large scale TV recommender system with a collaborative filtering based on prior implicit feedback that can recommend new TV programs to their users with high accuracy [14].

As a special type of collaborative filtering, various forms of matrix factorization based recommendation system were used by researchers for different recommendation tasks [5], [6], [7], [8], [10], [15]. For instance, to help Flickr users more easily engage in group activities, Zhang et al. proposes a tensor decomposition-based Flickr group recommendation model, which is based on CANDECOMP/PARAFAC tensor decomposition method to capture the underlying patterns in the user-tag-group relations [16]. Gu et al. introcudes a graph regularized nonnegative matrix factorization model for general collaborative filtering tasks, which outperform many state of the art collaborative filtering methods on benchmark data sets [17]. Other work by Baltrunas et al. presents an context-aware matrix factorization (CAMF) method which extends the classical Matrix Factorization approach by taking contextual information into consideration [18]. By applying their method on MovieAT data set and Yahoo Webscope movie data to do movie rating prediction, Baltrunas et al . have shown that the CAMF method can substantially improve the rating prediction accuracy comparing to the the classical Matrix Factorization approaches in certain circumstances in which the relevant context information is available.

Despite there being a lot of researches of matrix factorization based collaborative filtering methods, many algorithms still have salient weaknesses under sparsity conditions [19]. Even though there are approaches like multi-Domain collaborative filtering by Zhang et al. and adapting neighborhood and matrix factorization model by Liu et al. trying to improve the model's performance by integrating other available context information [20], [21], an extreme sparse data set would easily make most of the matrix factorization based models over-fitting. The SPPAN model in this present paper, however, adjust the complexity of the model automatically according to the sparsity of the data set, which makes it more accustomed to extreme sparse data set.

8. Conclusions

In summary, the Similarity Powered Pairwise Amplifier Network (SPPAN) model we proposed in this article shows promising results in predicting click-through rate (CTR) in a very sparse CTR data set. It utilizes the pairwise ratio similarity information that embedded in the data set, and its model complexity is automatically adjusted according to the sparsity of the data set. Comparing with several matrix factorization based recommendation methods, our evaluation experiment results show that this model can substantially improving the performance of the recommendation system in extreme sparse situations. It has also been shown that the training process of the SPPAN model can be easily implemented in a paralleled version through map-reduce, which makes the model can handle even bigger data set efficiently.

Since all evaluation experiments in this paper were performed on a click-through rate data set, we must observe that more experimental evaluations of the proposed SPPAN model should be performed in the future in order to make a more comprehensive conclusion of the performance of the model. In fact, out next plan is to apply the SPPAN model on other sparse data sets from different domains so that we could confirm if the *Homogeneous Amplifying Effect* described in Section 3.1 is valid only in online advertisement traffic data or can be generalized in other areas. Moreover, the experiment result in the present paper only shows the advantages of the SPPAN model in a situation with a certain sparsity level. A sensitivity analysis about the effect of the sparsity on the model's performance would help us to decide when to use SPPAN model instead of other approaches.

Acknowledgments

The authors would like to thank...

References

[1] "A/B Testing," http://en.wikipedia.org/wiki/A/B_testing.

[2] P. Resnick and H. R. Varian, "Recommender systems," *Communications of the ACM*, vol. 40, no. 3, pp. 56–58, 1997.

[3] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 285–295.

[4] "Google Adsense," http://en.wikipedia.org/wiki/AdSense.

[5] M. Zitnik and B. Zupan, "Nimfa: A python library for nonnegative matrix factorization," *Journal of Machine Learning Research*, vol. 13, pp. 849–853, 2012.

[6] C.-J. Lin, "Projected gradient methods for nonnegative matrix factorization," *Neural computation*, vol. 19, no. 10, pp. 2756–2779, 2007.

[7] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in neural information processing systems*, 2001, pp. 556–562.

[8] J.-P. Brunet, P. Tamayo, T. R. Golub, and J. P. Mesirov, "Metagenes and molecular pattern discovery using matrix factorization," *Proceedings of the national academy of sciences*, vol. 101, no. 12, pp. 4164–4169, 2004.

[9] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International journal of forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[10] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, *Recommender systems handbook*. Springer, 2011, vol. 1.

[11] Y. Koren and R. Bell, "Advances in collaborative filtering," in *Recommender systems handbook*. Springer, 2011, pp. 145–186.

[12] G. Linden, B. Smith, and J. York, "Amazon. com recommendations: Item-to-item collaborative filtering," *Internet Computing, IEEE*, vol. 7, no. 1, pp. 76–80, 2003.

[13] X. Cai, M. Bain, A. Krzywicki, W. Wobcke, Y. S. Kim, P. Compton, and A. Mahidadia, "Collaborative filtering for people to people recommendation in social networks," in *AI 2010: Advances in Artificial Intelligence*. Springer, 2011, pp. 476–485.

[14] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *Proceedings of ICDM 2008*.

[15] S. A. P. Parambath, "Matrix factorization methods for recommender systems," 2013.

[16] N. Zheng, Q. Li, S. Liao, and L. Zhang, "Flickr group recommendation based on tensor decomposition," in *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2010, pp. 737–738.

[17] Q. Gu, J. Zhou, and C. H. Ding, "Collaborative filtering: Weighted nonnegative matrix factorization incorporating user and item graphs." in *SDM*. SIAM, 2010, pp. 199–210.

[18] L. Baltrunas, B. Ludwig, and F. Ricci, "Matrix factorization techniques for context aware recommendation," in *Proceedings of the fifth ACM conference on Recommender systems*. ACM, 2011, pp. 301–304.

[19] F. Cacheda, V. Carneiro, D. Fernández, and V. Formoso, "Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems," *ACM Transactions on the Web (TWEB)*, vol. 5, no. 1, p. 2, 2011.

[20] Y. Zhang, B. Cao, and D.-Y. Yeung, "Multi-domain collaborative filtering," *arXiv preprint arXiv:1203.3535*, 2012.

[21] N. N. Liu, B. Cao, M. Zhao, and Q. Yang, "Adapting neighborhood and matrix factorization models for context aware recommendation," in *Proceedings of the Workshop on Context-Aware Movie Recommendation*. ACM, 2010, pp. 7–13.