A Hybrid Latent Variable Neural Network Model for Item Recommendation

Michael R. Smith
Computer Science Department
Brigham Young University
Provo, Utah 84601, USA
Email: msmith@axon.cs.byu.edu

Michael S. Gashler
Department of Computer Science
and Computer Engineering
University of Arkansas
Fayetteville, AR 72701, USA
Email: mgashler@uark.edu

Tony Martinez
Computer Science Department
Brigham Young University
Provo, Utah 84601, USA
Email: martinez@cs.byu.edu

***Abstract*—Collaborative filtering is used to recommend items to a user without requiring a knowledge of the item itself and tends to outperform other techniques. However, pure collaborative filtering technique suffer from the *cold-start* problem, which occurs when an item has not yet been rated or a user has not rated any items. Incorporating additional information, such as item or user descriptions, into collaborative filtering can address the cold-start problem. In this paper, we present a neural network model with latent input variables (*latent neural network* or LNN) as a hybrid collaborative filtering technique that addresses the cold-start problem. LNN outperforms a broad selection of content-based filters (which make recommendations based on item descriptions) and other hybrid approaches while maintaining the accuracy of state-of-the-art collaborative filtering techniques.**

I. Introduction

Modern technology enables users to access an abundance of information. This deluge of data makes it difficult to sift through it all to find what is desired. This problem is of particular concern to companies who are trying sell products (e.g. Amazon or Walmart) or recommend movies (e.g. Netflix). To lessen the severity of information overload, recommender systems help a user find what he or she is looking for. Two commonly used classes of recommender systems are content-based filters and collaborative filters.

Content-based filters (CBF) make recommendations based on item/user descriptions and users' ratings of the items. Creating item/user descriptions that are predictive of how a user will rate an item, however, is not a trivial process. On the other hand, collaborative filtering (CF) techniques use correlations between users' ratings to infer the rating of unrated items for a user and make recommendations without having to understand the item or user itself. CF does not depend on item descriptions and tends to produce higher accuracies than CBF.

However, CF suffers from the *cold-start problem* which occurs when an item cannot be recommended unless it is has been rated before (*first-rater problem*) or when a user has not rated any items (*new-user problem*). This is particularly important in domains where new items are frequently added to a set of items and users are more interested in the new items. For example, many users are more interested, and likely to purchase, new styles of shoes rather than out-dated styles or many users are more interested in watching newly released movies rather than older movies. Recommending old items has the potential to drive away customers. In addition, making inappropriate recommendations for new users who have not built a profile can also drive away users.

One approach for addressing the cold-start problem is using a hybrid recommender system that can leverage the advantages of multiple recommendation systems. Developing hybrid models is a significant research direction [1], [2], [3], [4], [5]. Many hybrid approaches combine a content-based filter with a collaborative filter through methods such as averaging the predicted ratings or combining the top recommendations from both techniques [6]. In this paper, we present a neural network model that infers latent item/user trait vectors (*latent neural network* or LNN) as a hybrid recommendation algorithm that addresses the cold-start problem. LNN uses a matrix of user/item ratings and user/item descriptions to simultaneously train the weights in a neural network and induce a set of latent user/item trait vectors for matrix factorization.

By incorporating user/item descriptions, LNN is able to address the cold-start problem. We also propose a method for recommending items *without any ratings*. Most previous methods require at least some ratings in order to recommend an item. Using a neural network allows for flexible architecture configurations to model higher-order and non-linear dependencies in the data. We find that LNN outperforms other content-based filters and hybrid filters on the cold-start problem. Additionally, LNN outperforms its predecessor (UBP) and maintains an accuracy similar to matrix factorization (which does not implicitly handle the cold-start problem) on non-cold-start recommendations.

In Section II we review related work to the LNN and in dealing with the cold-start problem. We then formally describe LNN in Section III. The results from our experiments are presented in Section IV followed by our conclusions and directions for future work.

II. Related Work

Matrix factorization (MF) has become a popular technique, in part due to its effectiveness with the data used in the NetFlix competition [7] and is widely considered a state-of-the-art recommendation technique. Given a matrix $\mathbf{X}$ (typically ratings for items given by users), MF finds two smaller

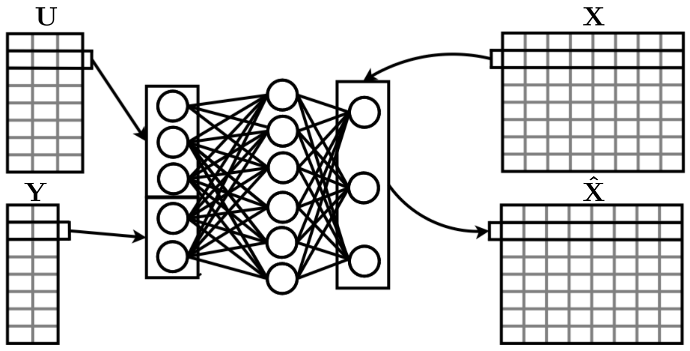

Fig. 1. High-level overview of LNN. Induced latent features are used as input to a neural network along with provided user/item features. The network weights and the latern featuers are updated by calculating the gradient with respect to the error signal.

matrices $\mathbf{U}$ and $\mathbf{V}$ such that their product approximates $\mathbf{X}$ as close as possible:

$$\mathbf{X} \approx \mathbf{U} \times \mathbf{V}^T = \hat{\mathbf{X}}.$$

These smaller matrices can then be combined to predict all of the missing elements in the original matrix. The value of an element from row r and column c in $\mathbf{X}$ is calculated as:

$$\hat{x}_{rc} = \mathbf{u}_r \times \mathbf{v}_c^T + b_u + b_i$$

where b_u and b_i are biases for $\mathbf{u}_r$ and $\mathbf{v}_c$ respectively (typically user and item biases). Since its initial success using stochastic gradient descent, several other variations of MF have also been proposed [8], [9], [10]. Salakhutdinov and Mnih extended MF using probabilistic approaches [11], [12]. Lee et al. extended MF assuming that the matrix is locally low-rank [13].

While MF is a general technique, it is often used to recommend items to users. Pure collaborative filtering (CF) techniques are not able to handle the cold-start problem for items or users. As a result, several hybrid methods have been developed that incorporate item and/or user descriptions into collaborative filtering approaches. The most common, as surveyed by Burke [6], involves using separate content based filter (CBF) and CF techniques and then combining their outputs (i.e. weighted average, combining the output from both techniques, or switching depending on the context) or using the output from one technique as input to another. Content-boosted collaborative filtering [14] uses CBF to fill in the missing values in the ratings matrix and then the dense ratings matrix is passed to a collaborative filtering method (in their implementation, a neighbor based CF).

Other work addresses the cold-start problem by building user/item descriptions for later use in a recommendation system [15]. The Matchbox system [16] uses user and item meta-data $\mathbf{y}$ and $\mathbf{z}$ when calculating ratings as:

$$\hat{x}_{rc} = \mathbf{s}_r \times \mathbf{t}_c^T + b_u + b_i$$

where $\mathbf{s}_r = \mathbf{U}_r\mathbf{y}_r$, $\mathbf{t}_c = \mathbf{V}_c\mathbf{z}_c$ and $\mathbf{U}$ and $\mathbf{V}$ represent the $\mathbf{U}$ and $\mathbf{V}$ matrices from matrix factorization. Other hybrid techniques take a similar approach of using a linear combination of the item features and the induced latent vectors [12], [17]. Each of these approaches require some known ratings in order to address the cold-start problem.

In contrast to other hybrid approaches, LNN does not require a linear combination of latent features. LNN is based on the idea of generative backpropagation (GenBP) [18] and expands upon unsupervised backpropagation (UBP) [19]. Both GenBP and UBP are neural network methods that induce a set of latent trait vectors. The latent trait vectors form an internal representation of observed values. GenBP adjusts its latent vectors while holding the network weights constant. It has been used to generate labels for images [20], and for natural language [21]. UBP differs from GenBP in that it trains network weights simultaneously with the latent vectors, instead of training the weights as a pre-processing step. LNN is a further development of UBP that incorporates input features (i.e. item descriptions) with the latent trait vectors.

In the case where LNN is used with linear activation functions and no hidden layers, LNN can be described as:

$$\hat{x}_{rc} = (\mathbf{y}_r, \mathbf{u}_r) \times \mathbf{v}_c + b_u + b_i$$

where $(\mathbf{y}_r, \mathbf{u}_c)$ represents concatenating the given item descriptor vector and the inferred user meta-data vector. LNN may also be used with nonlinear activation functions and an arbitrary number of hidden layers and nodes which give it power to fit nonlinear manifolds. LNN can also make predictions without any prior ratings from a user/for an item. When $\mathbf{Y}$ is empty, and only a single-layer neural network is used with a linear activation function, LNN reduces to MF where the network weights correspond to $\mathbf{V}$ and the latent vectors correspond to $\mathbf{U}$ [22].

III. Latent Neural Network

In this section, we formally describe *latent neural networks* (LNN). At a high-level, a LNN is a neural network that induces a set of latent vectors using generative backpropagation while also updating the weights in the network. Generative backpropagation calculates the gradient of the latent vectors with respect to the error and updates them in a manner similar to how the weights are updated in the backpropagation algorithm. Figure 1 provides a high-level diagram of LNN. The underlying idea of LNN is to seamlessly incorporate additional user/item information into the recommender system. By using non-linear activation functions, unsupervised backpropagation (UBP) may be viewed as a non-linear generalization of MF. UBP utilizes three phases for training to initialize the latent variables, the weights of the model and then to update the weights and latent variables simultaneously. LNN further builds on UBP by integrating item or user descriptions with the latent input variables.

A. Preliminaries

In order to formally describe LNNs, we define the following terms.

- Let $\mathbf{X}$ be a given $m \times n$ sparse user/item rating matrix, where m is the number of items and n is the number of users.
- Let $\mathbf{Y}$ be an $m \times y$ matrix, representing the given portion of the item profiles (the item features such as the movie genres that a movie is in or actors in the movie). Again, m is the number of items and y is the number of item features.
- Let $\mathbf{U}$ be an $m \times t$ matrix, representing the latent portion of the item profiles. These values are inferred by LNN. m is the number of items and t represents the number of latent variables (provided by the user).
- If x_{rc} is the rating for item r by user c in $\mathbf{X}$, then $\hat{x}_{rc}$ is the predicted rating when $\mathbf{y}_r \in \mathbf{Y}$ and $\mathbf{u}_r \in \mathbf{U}$ are concatenated into a single vector $\mathbf{q}_r$ and then fed forward into the LNN.
- Let w_{ij} be the weight that feeds from unit i to unit j in the LNN.
- For each network unit i on hidden layer m, let β_{mi} be the net input into the unit, α_{mi} be the output or activation value of the unit, and δ_{mi} be an error term associated with the unit.
- Let l be the number of hidden layers in the LNN.
- Let $\mathbf{g}$ be a vector representing the gradient with respect to the weights of the LNN, such that g_{ij} is the component of the gradient that is used to refine w_{ij}.
- Let $\mathbf{h}$ be a vector representing the gradient with respect to the latent inputs of the LNN, such that h_i is the component of the gradient that is used to refine $u_{ri} \in \mathbf{u}_r$.

We use item descriptions, but user descriptions could easily be used by transposing the $\mathbf{X}$ and using user descriptions instead of item descriptions.

As using generative backpropagation to compute the gradient with respect to the latent inputs, $\mathbf{h}$, is less commonly used, we provide a derivation of it here. We compute each $h_i \in \mathbf{h}$ from the presentation of a single element $x_{rc} \in \mathbf{X}$ since we assume that $\mathbf{X}$ is typically high-dimensional and sparse. It is significantly more efficient to train with the presentation of each known element individually. We begin by defining an error signal for an individual element, $E_{rc} = (x_{rc} - \hat{x}_{rc})^2$, and then express the gradient as the partial derivative of this error signal with respect to each latent input in $\mathbf{U}$ (the non-latent inputs $\mathbf{Y}$ do not change):

$$h_i = \frac{\partial E_{rc}}{\partial u_{ri}}. \quad (1)$$

The latent input u_{ri} affects the value of E_{rc} through the net value of a unit (β_{ji}) and further through the output of a unit (α_{ji}). Using the chain rule, Equation 1 becomes:

$$h_i = \frac{\partial E_{rc}}{\partial \alpha_{0c}} \frac{\partial \alpha_{0c}}{\partial \beta_{0c}} \frac{\partial \beta_{0c}}{\partial u_{ri}} \quad (2)$$

where α_{0c} and β_{0c} represent, respectively, the output values and the net input values of the output nodes (the 0^{th} layer). The backpropagation algorithm calculates $\frac{\partial E_{rc}}{\partial \alpha_{0c}} \frac{\partial \alpha_{0c}}{\partial \beta_{0c}}$ (which is $\frac{\partial E_{rc}}{\partial \beta_{j,i}}$ for a network unit) as the error term δ_{ji} associated with a network unit. Thus, to calculate h_i, the only additional calculation to the backpropagation algorithm that needs to be made is $\frac{\partial \beta_{jc}}{\partial u_{ri}}$. For a single layer perceptron:

$$\frac{\partial \beta_{0c}}{\partial u_{ri}} = \frac{\partial}{\partial u_{ri}} \sum_t w_{tc}\, u_{rt}$$

which is non-zero only when t equals i and is equal to w_{ic} since the error is being calculated with respect to a single element in $\mathbf{X}$. When there are no hidden layers ($l = 0$) and using the error from a single element x_{rc}:

$$h_i = -w_{ic}\delta_c. \quad (3)$$

If there is at least one hidden layer ($l > 0$), then,

$$\frac{\partial \beta_{0c}}{\partial u_{ri}} = \frac{\partial \beta_{0c}}{\partial \boldsymbol{\alpha}_1} \frac{\partial \boldsymbol{\alpha}_1}{\partial \boldsymbol{\beta}_1} \cdots \frac{\partial \boldsymbol{\alpha}_l}{\partial \boldsymbol{\beta}_l} \frac{\partial \boldsymbol{\beta}_l}{\partial u_{ri}},$$

where $\boldsymbol{\alpha}_k$ and $\boldsymbol{\beta}_k$ are vectors that represent the output values and the net values for the units in the k^{th} hidden layer. As part of the error term for the units in the l^{th} layer, backpropagation calculates $\frac{\partial \beta_{0,c}}{\partial \alpha_1} \frac{\partial \alpha_1}{\partial \beta_1} \cdots \frac{\partial \alpha_l}{\partial \beta_l}$ as the error term associated with each network unit. Thus, the only additional calculation for h_i is:

$$\frac{\partial \boldsymbol{\beta}_l}{\partial u_{ri}} = \frac{\partial}{\partial u_{ri}} \sum_j \sum_t w_{jt}\, u_{rt}.$$

As before, $\frac{\partial \boldsymbol{\beta}_l}{\partial u_{ri}}$ is non-zero only when t equals i. For networks with at least one hidden layer:

$$h_i = -\sum_j w_{ij}\delta_j. \quad (4)$$

Equation 4 is a strict generalization of Equation 3. Equation 3 only considers the one output unit, c, for which a known target value is being presented, whereas Equation 4 sums over each unit, j, into which the latent value u_{ri} feeds.

B. Three-Phase Training

To integrate generative backpropagation into the training process, LNN uses three phases to train $\mathbf{U}$ and $\mathbf{W}$: 1) the first phase computes an initial estimate for the latent vectors, $\mathbf{U}$, 2) the second phase computes an initial estimate for the network weights, $\mathbf{W}$, and 3) the third phase refines them both together. All three phases train using stochastic gradient descent. In phase 1, the latent vectors in $\mathbf{U}$ are induced while there are no hidden layers to avoid the complexities of nonlinear separations. Likewise, phase 2 allows the weights to converge without having to train against moving inputs (i.e. $\mathbf{U}$ is held constant). These two preprocessing phases initialize the system (consisting of both latent vectors and weights) to a good initial starting point, such that gradient descent is more likely to find a local optimum of higher quality. Empirical results comparing three-phase and single-phase training show that three-phase training produces more accurate results than single-phase training, which only refines $\mathbf{U}$ and $\mathbf{W}$ together (see Gashler et al. [19]).

Algorithm 1 LNN($\mathbf{U}, \mathbf{Y}, \mathbf{X}, \eta', \eta'', \gamma, \lambda$)

```
 1: Initialize each element in $\mathbf{U}$ with small random values $\sim \mathcal{N}(0, 0.01)$
    // First phase of training
    // Create a temporary single-layer perceptron for initial
    // training of $\mathbf{U}$
 2: Let $\mathbf{T}$ be the weights of a single-layer perceptron
 3: Initialize each element in $\mathbf{T}$ with small random values $\sim \mathcal{N}(0, 0.01)$
 4: $\eta \leftarrow \eta'$; $s' \leftarrow \infty$
 5: while $\eta > \eta''$ do
 6:   $s \leftarrow$ train_epoch($\mathbf{U}$,$\mathbf{Y}$,$\mathbf{X}$, $\mathbf{T}$, $\lambda$, true, $0$)
 7:   if $1 - s/s' < \gamma$ then $\eta \leftarrow \eta/2$
 8:   $s' \leftarrow s$
 9: end while
    // Second phase of training
    // Hold $\mathbf{U}$ constant for training of $\mathbf{W}$
10: Let $\mathbf{W}$ be the weights of a multi-layer perceptron with $l$ hidden layers, $l \geq 0$
11: Initialize each element in $\mathbf{W}$ with small random values
12: $\eta \leftarrow \eta'$; $s' \leftarrow \infty$
13: while $\eta > \eta''$ do
14:   $s \leftarrow$ train_epoch($\mathbf{U}$,$\mathbf{Y}$,$\mathbf{X}$, $\mathbf{W}$, $\lambda$, false, $l$)
15:   if $1 - s/s' < \gamma$ then $\eta \leftarrow \eta/2$
16:   $s' \leftarrow s$
17: end while
    // Third phase of training
    // Update both $\mathbf{U}$ and $\mathbf{W}$
18: $\eta \leftarrow \eta'$; $s' \leftarrow \infty$
19: while $\eta > \eta''$ do
20:   $s \leftarrow$ train_epoch($\mathbf{U}$,$\mathbf{Y}$,$\mathbf{X}$, $\mathbf{W}$, $0$, true, $l$)
21:   if $1 - s/s' < \gamma$ then $\eta \leftarrow \eta/2$
22:   $s' \leftarrow s$
23: end while
24: return $\{\mathbf{U}, \mathbf{W}\}$
```

Pseudo-code for the LNN algorithm, which trains $\mathbf{U}$ and $\mathbf{W}$ in three phases, is given in Algorithm 1. LNN calls the train_epoch function (shown in Algorithm 2) which performs a single epoch of training. Matrices containing the known data values, $\mathbf{X}$, and the item descriptions, $\mathbf{A}$, are passed into LNN along with the parameters $\eta', \eta'', \gamma, \lambda$. η is the learning rate and s' is used to store the previous error score. We note that many techniques could be used to detect convergence. Our implementation decays the learning rate whenever predictions fail to improve by a sufficient amount. Convergence is detected when the learning rate η falls below η''. γ specifies the amount of improvement that is expected after each epoch, or else the learning rate is decayed. λ is the regularization term used in train_epoch. LNN returns $\mathbf{U}$ and $\mathbf{W}$. $\mathbf{W}$ is a ragged matrix containing weight values for an MLP that maps from each $\mathbf{v}_i$ to an approximation of $\mathbf{x}_i \in \mathbf{X}$.

C. Stochastic gradient descent

For completeness, train_epoch is given in Algorithm 2, which performs a single epoch of training by stochastic gradient descent. This algorithm is very similar to an epoch of traditional backpropagation, except that it presents each element individually, instead of presenting each vector, and it conditionally refines the latent variables, $\mathbf{U}$, as well as the weights, $\mathbf{W}$. The variable p represents which pass of training LNN is in (whether or not to update the latent variables $\mathbf{U}$. Generative backpropagation is implemented in lines 16-19 making use of the error term(s) δ used in backpropagation to update the weights.

Algorithm 2 train_epoch($\mathbf{U}$,$\mathbf{Y}$,$\mathbf{X}$, $\mathbf{W}$, λ, p, l)

```
 1: for each known $x_{rc} \in \mathbf{X}$ in random order do
 2:   $\mathbf{q}_r \leftarrow (\mathbf{u}_r, \mathbf{a}_r)$
 3:   Compute $\alpha_c$ by forward-propagating $\mathbf{q}_r$ into a multi-layer perceptron with weights $\mathbf{W}$
      // Compute an error term for an output unit $c$ and
      // backpropagate the error
 4:   $\delta_c \leftarrow (x_{rc} - \alpha_c)f'(\beta_c)$
 5:   for each hidden unit $i$ feeding into output unit $c$ do
 6:     $\delta_i \leftarrow w_{ic}\delta_c f'(\beta_i)$
 7:   end for
 8:   for each hidden unit $j$ in an earlier hidden layer (in backward order) do
 9:     $\delta_j \leftarrow \sum_k w_{jk}\delta_k f'(\beta_j)$
10:   end for
      // Refine $\mathbf{W}$ by gradient descent
11:   for each $w_{ij} \in \mathbf{W}$ do
12:     $g_{ij} \leftarrow -\delta_j \alpha_i$
13:   end for
14:   $\mathbf{W} \leftarrow \mathbf{W} - \eta(\mathbf{g} + \lambda\mathbf{W})$
      // Refine $\mathbf{U}$ by gradient descent
      // Only update $\mathbf{U}$ during phases 1 and 3
15:   if $p =$ true then
16:     for $i$ from $0$ to $t - 1$ do
17:       if $l = 0$ then $h_i \leftarrow -w_{ic}\delta_c$
          else $h_i \leftarrow -\sum_j w_{ij}\delta_j$
18:     end for
19:     $\mathbf{u}_r \leftarrow \mathbf{u}_r - \eta(\mathbf{h} + \lambda\mathbf{u}_r)$
20:   end if
21: end for
22: $s \leftarrow$ measure RMSE with $\mathbf{X}$
23: return $s$
```

D. Recommending New Items

For many existing hybrid approaches, new items cannot be recommended unless they have recieved as least a certain amount of ratings. Thus, the cold start problem refers to recommending items that have very few ratings. Here, we propose a method for recommending items that have not received any ratings. Predicting a new item poses a challenge since the latent variables for a new item have not been induced. Recall that a content-based filtering creates a model for each user based on item descriptions and corresponding user ratings. LNN, on the other hand, produces a single model which is beneficial when using all of the ratings because the mutual information between users and items can be shared. The shared information is contained in the latent variables. The quality of the latent variables depends on the number of ratings that a user has given and/or an item has received.

To compensate for the lack of latent variables for the new items, we utilize the new_item_prediction function that takes a vector $\mathbf{y}_{newItem}$ representing the description of the new item and is outlined in Algorithm 3. At a high level,

Algorithm 3 new_item_prediction($\mathbf{y}_{newItem}$)

```
1:  Let $count$ be a map containing the count of how many
    times each rating was predicted
2:  Initialize each element in $count$ to 0
3:  $num \leftarrow 100$; $distThresh \leftarrow 0$
4:  $neighbors \leftarrow$ getNeighbors($\mathbf{y}_{newItem}$, $num$)
5:  for $i$ from 0 to $num - 1$ do
6:      $numR \leftarrow$ number of ratings for $neighbors[i]$
7:      if $numR > 50$ && $dist(neighbors[i]) > dist$ then
8:          $\mathbf{q}_{new} \leftarrow (\mathbf{v}_{neighbors[i]}, \mathbf{y}_{newItem})$
9:          $prediction \leftarrow$ rounded prediction of $\mathbf{q}_{new}$
10:         $counts[prediction] += numRatings$
11:     end if
12: end for
13: return maxIndex($counts$)
```

new_item_prediction uses $\mathbf{y}_{newItem}$ to find the most similar items. The induced latent input variables for each similar item are concatenated with $\mathbf{y}_{newItem}$ and fed into a trained LNN to predict a rating for the new item. The weighted mode of the predicted ratings of the new item is then returned. The rating from each neighbor is weighted according to how many times it has been rated. By weighting, we mean when selecting the mode from a set of numbers, the predicted rating is added r times to the set where r is the number times that the neighbor item has been rated. We chose to use the mode rather than the mean because the mode is more robust to outliers and achieves better empirical results on the validation sets in our experimentation. As we used binary item descriptions of movie genres, we only considered using the latent variables from items that have the same genre(s) (has a distance of 0). We also only consider items that have been rated at least 50 times. The value of 50 was chosen based on the evaluation of a content-based predictor [23]. The number of times that an item has been rated helps to determine the quality of the induced latent variables for an item and provides a confidence level for latent variables.

IV. Experimental Results

In this section we present the results from our experiments. We examine LNN using the MovieLens[1] data sets. Although the well-known NetFlix dataset is larger, the MovieLens data sets provide richer meta descriptions (in the form of movie genres), which we use to evaluate the effectiveness of LNN in addressing the cold start problem. The runtime complexity of LNN is approximately the same as matrix factorization in the case where only a single layer is used, and it scales linearly with the number of weights in the neural network. Therefore, LNN can easily handle datasets as large as, or larger than, the NetFlix data. Other data sets provide unstructured data such as twitter information or a set of friends on last.fm from which input variables could be created. As this paper focuses on the performance of LNN rather than feature creation from unstructured data, we chose to use the MovieLens data set.

The MovieLens data set has multiple versions of different sizes. We examine the versions "ml-100k", "ml-1m", and "ml-10M100k". The ml-100k data set contains 100,000 ratings from 943 users on 1682 movies where each user has rated at least 20 movies. It has a density level of 6.3%. The ml-1m data set contains 1,000,209 ratings from 6,040 users on 3,952 movies and has a density level of 4.2%. The ml-10M100K data set contains 10,000,054 ratings from 71,567 users on 10,681 movies and has a density level of 1.3%.

[1]http://www.grouplens.org

TABLE I. The MAE for each recommendation technique.

	ml-100k		ml-1m		ml-10M100k	
Alg	Val	Test	Val	Test	Val	Test
CBCF	0.72	0.92	0.91	0.91	-	-
CBF	1.04	1.04	1.01	1.01	0.99	0.98
LNN	**0.71**	0.74	0.68	0.70	0.64	0.65
LNN$_{\text{3PT}}$	**0.71**	**0.72**	0.68	0.69	0.64	0.64
MF	0.72	0.73	**0.67**	**0.67**	**0.61**	**0.61**
NLPCA	0.73	0.74	0.69	0.69	0.63	0.64
UBP	0.73	0.74	0.69	0.70	0.65	0.65

We compare LNN with several other recommendation systems: 1) content-boosted collaborative filtering (CBCF), 2) content-based filtering (CBF), 3) nonlinear principle component analysis (NLPCA), 4) unsupervised backpropagation (UBP), and 5) matrix factorization (MF). We use LNN with and without three phase training. This is equivalent to a hybrid UBP and hybrid NLPCA technique. LNN with three phase training is denoted as LNN$_{\text{3PT}}$.

For each recommendation system, we test several parameter settings. CBF uses a single learning algorithm to learn the rating preferences of a user. We experiment using naïve Bayes (as is commonly used Melville et al. [14]), linear regression, a decision tree, and a neural network trained with backpropagation. The same learning algorithms are also used for CBCF. The number of neighbors for the nearest-neighbor portion of CBCF ranges from 1 to 64. For MF, the number of latent variables ranges from 2 to 32 and the regularization term from 0.001 to 0.1. In addition to the values used for MF for the number of latent variables and the regularization term, the number of nodes in the hidden layer ranges from 0 to 32 for UBP, NLPCA, LNN, and LNN$_{\text{3PT}}$. For each experiment, we randomly select 20% of the ratings as a test set. We then use 10% of the training set as a validation set for parameter selection. Using the selected parameters, we test on the test set and using 10-fold cross-validation.

A. Results

The results comparing LNN with the other recommendation approaches are shown in Table I. We report the mean absolute error (MAE) for each approach. The bold values represent the lowest value for each approach. The algorithms that use latent variables are significantly lower than those that do not (CBCF and CBF), thus demonstrating the predictive power of using latent variables for item recommendation. Latent inputs also allows one to bypass feature engineering – often a difficult process.

On ml-100k, the addition of the item descriptions to NLPCA and UBP (LNN and LNN$_{\text{3PT}}$) improves the performance compared to only using the latent variables. The runtime performance of LNN and LNN$_{\text{3PT}}$ is similar to matrix factorization, which is widely considered state-of-the-art in

TABLE II. THE MAE FOR THE TOP 10 MOST RATED MOVIES (INDIVIDUALLY AND COMBINED) WHEN HELD OUT OF THE TRAINING SET.

	alg	106	1182	1356	1407	1654	233	282	614	641	717	Ave
ml-100k	CBCF	**1.045**	1.137	1.188	1.326	2.718	0.835	0.831	0.757	1.478	**1.079**	1.239
	CBF	1.130	1.166	1.179	1.337	2.571	0.857	0.883	0.707	1.617	1.137	1.258
	LNN	1.218	**0.846**	**0.333**	**0.975**	**2.000**	0.839	**0.750**	0.627	**0.591**	1.268	**0.945**
	LNN_{3PT}	1.148	0.885	0.500	1.075	2.500	**0.694**	0.774	**0.588**	0.636	1.152	0.995
	alg	106	2070	2470	2509	2676	2678	3430	3462	614	717	Ave
ml-1m	CBCF	1.248	0.841	0.733	0.939	0.875	1.029	0.880	1.124	**1.030**	0.681	0.938
	CBF	**1.221**	0.868	**0.862**	1.003	1.028	1.045	0.949	1.062	1.042	0.460	0.954
	LNN	1.247	**0.715**	0.864	0.964	**1.004**	**0.972**	**0.758**	**0.935**	1.174	0.125	**0.876**
	LNN_{3PT}	1.250	0.729	**0.867**	**0.833**	1.197	1.060	0.791	0.952	1.222	**0.000**	0.890
	alg	106	2070	2676	26796	3059	3462	59265	614	6966	8909	Ave
10M100K	CBCF	-	-	-	-	-	-	-	-	-	-	-
	CBF	1.242	0.851	0.970	0.591	1.227	**0.907**	1.162	1.002	1.117	0.629	0.970
	LNN	1.013	0.718	**0.912**	**0.569**	**0.833**	1.093	1.013	**0.772**	1.013	1.013	0.895
	LNN_{3PT}	**1.006**	**0.684**	0.917	0.655	0.917	1.024	**0.566**	0.833	**0.777**	**0.250**	**0.763**

recommendation systems when comparing MAE.[2] For the other data sets, the MAE scores are similar to, but do not exceed those of matrix factorization. In fact, they are similar to those of NLPCA and UBP. This implies 1) that having more data improves the values of the latent variables and 2) that the sparser the ratings matrices are the less predictive the item descriptions become.

B. Cold Start Problem

The power of LNN and LNN_{3PT} is demonstrated with the cold-start problem. As was discussed previously, MF and other pure collaborative filtering techniques are not able to address the cold-start problem despite being able to perform very well on items that have been rated previously a certain number of times. (They also suffer from the gray sheep problem which occurs when an item has only been rated a small number of times.) LNN and LNN_{3PT} are capable of addressing the cold-start problem *while* still obtaining similar performance to matrix factorization. To examine the cold-start problem, we remove the ratings for ten randomly chosen movies from each data set. The recommendation systems were trained using the remaining ratings using the parameter setting found in the previous experiments.

The results for recommending new items using new_item_prediction are provided in Table II. The values at the top of each section of the table correspond to the movie id in the MovieLens data set. The bold values represent the lowest MAE value obtained. No single recommendation system produces the lowest MAE for all of the movies, suggesting that some recommendation systems are better than others for a given user and/or item as has been suggested previously [24]. In most individual cases and on average, LNN and LNN_{3PT} produce the lowest MAE on the new items. This shows the importance of using latent variables. CBCF uses CBF to create a dense matrix (except for the ratings corresponding to the active user) and then uses a collaborative filtering technique on the dense matrix to recommend items to the user. Thus, more emphasis is given to the CBF which generally produces poorer item recommendations than a collaborative filtering approach. LNN, on the other hand, utilizes the latent variables and their predictive power. CBCF is also very memory and computationally expensive. CBCF did not finish on the 10M100K data set.

[2]The values for other versions of matrix factorization that are not based on stochastic gradient descent perform similarly. The results for other matrix factorization approaches can be found at http://prea.gatech.edu/features.html#benchmark on the ml-1m data set. We choose to use stochastic gradient decent since it is most similar to LNN.

V. Conclusions and Future Work

In this paper, we presented a neural network with latent input variables capable of recommending unrated items to users or items to new users which we call a *latent neural network* (LNN). The combination of the latent variables and the input variables allow information and correlations among the rated items to be represented while also incorporating the item descriptions in the recommendation. Thus, LNN is a hybrid recommendation algorithm that leverages the advantages of collaborative filtering and content based filtering.

Empirically, a LNN is able to achieve similar results to state-of-the-art collaborative filtering techniques such as matrix factorization while *also* addressing the cold-start problem. Compared with other hybrid filters and content-based filtering, LNN achieves much lower error when recommending previously unrated items. As LNN achieves similar error rates to the state-of-the-art filtering techniques and can make recommendations for previously unrated items, LNN does not *have* to be retrained once new items are rated in order to recommend them.

As LNN is built on a neural network, it is capable of modeling higher-order dependencies and non-linearities in the data. However, the data in the MovieLens data set and many similar data sets is well suited to using linear models such as matrix factorization. This may be due in part to the fact many of the data sets are inherently sparse and nonlinear models could overfit them and reduce their generalization. As a direction of future work, we are examining how to better incorporate the non-linear component of LNN. We are also looking at integrating *both* user and item descriptions with latent input variables to address the new user problem *and* the new item problem in a single model.

References

[1] P. Cremonesi, R. Turrin, and F. Airoldi, "Hybrid algorithms for recommending new items," in *Proceedings of the 2Nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, ser. HetRec '11. New York, NY, USA: ACM, 2011, pp. 33–40.

[2] P. Forbes and M. Zhu, "Content-boosted matrix factorization for recommender systems: experiments with recipe recommendation," in *RecSys*, B. Mobasher, R. D. Burke, D. Jannach, and G. Adomavicius, Eds. ACM, 2011, pp. 261–264.

[3] J. Lin, K. Sugiyama, M.-Y. Kan, and T.-S. Chua, "Addressing cold-start in app recommendation: latent user models constructed from twitter followers," in *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '13. New York, NY, USA: ACM, 2013, pp. 283–292.

[4] N. Koenigstein and U. Paquet, "Xbox movies recommendations: variational bayes matrix factorization with embedded feature selection." in *RecSys*, Q. Y. 0001, I. King, Q. Li, P. Pu, and G. Karypis, Eds. ACM, 2013, pp. 129–136.

[5] Y. Bao, H. Fang, and J. Zhang, "Topicmf: Simultaneously exploiting ratings and reviews for recommendation." in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 2014, pp. 2–8.

[6] R. D. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331–370, 2002.

[7] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[8] D. D. Lee and H. S. Seung, "Learning the parts of objects by nonnegative matrix factorization," *Nature*, vol. 401, pp. 788–791, 1999.

[9] J. D. M. Rennie and N. Srebro, "Fast maximum margin matrix factorization for collaborative prediction," in *ICML '05: Proceedings of the 22nd international conference on Machine learning*. New York, NY, USA: ACM, 2005, pp. 713–719.

[10] N. D. Lawrence and R. Urtasun, "Non-linear matrix factorization with gaussian processes." in *ICML*, ser. ACM International Conference Proceeding Series, A. P. Danyluk, L. Bottou, and M. L. Littman, Eds., vol. 382. ACM, 2009, p. 76.

[11] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization," in *Advances in Neural Information Processing Systems 20*, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, Eds. Curran Associates, Inc., 2007.

[12] R. Salakhutdinov and A. Mnih, "Bayesian probabilistic matrix factorization using Markov chain Monte Carlo," in *Proceedings of the 25th International Conference on Machine Learning*, 2008.

[13] J. Lee, S. Kim, G. Lebanon, and Y. Singer, "Local low-rank matrix approximation." in *ICML (2)*, ser. JMLR Proceedings, vol. 28. JMLR.org, 2013, pp. 82–90.

[14] P. Melville, N. Shah, L. Mihalkova, and R. J. Mooney, "Experiments on ensembles with missing and noisy data." in *Multiple Classifier Systems*, ser. Lecture Notes in Computer Science, vol. 3077, 2004, pp. 293–302.

[15] K. Zhou, S.-H. Yang, and H. Zha, "Functional matrix factorizations for cold-start recommendation," in *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '11*, 2011, pp. 315–324.

[16] D. H. Stern, R. Herbrich, and T. Graepel, "Matchbox: large scale online bayesian recommendations," in *Proceedings of the 18th international conference on World wide web*, ser. WWW '09. New York, NY, USA: ACM, 2009, pp. 111–120.

[17] I. Porteous, A. U. Asuncion, and M. Welling, "Bayesian matrix factorization with side information and dirichlet process mixtures." in *AAAI*, M. Fox and D. Poole, Eds. AAAI Press, 2010.

[18] G. E. Hinton, "Generative back-propagation," in *Abstracts 1st INNS*, 1988.

[19] M. S. Gashler, M. R. Smith, R. Morris, and T. Martinez, "Missing value imputation with unsupervised backpropagation," *Computational Intelligence*, p. To Appear, 2014.

[20] D. Coheh and J. Shawe-Taylor, "Daugman's gabor transform as a simple generative back propagation network," *Electronics Letters*, vol. 26, no. 16, pp. 1241–1243, 1990.

[21] Y. Bengio, H. Schwenk, J. Senécal, F. Morin, and J. Gauvain, "Neural probabilistic language models," in *Innovations in Machine Learning*. Springer, 2006, pp. 137–186.

[22] G. Takács, I. Pilászy, B. Németh, and D. Tikk, "Scalable collaborative filtering approaches for large recommender systems," *The Journal of Machine Learning Research*, vol. 10, pp. 623–656, 2009.

[23] T. M. Mitchell, *Machine Learning*. McGraw-Hill New York, 1997, vol. 1.

[24] J. Lee, M. Sun, G. Lebanon, and S. jean Kim, "Automatic feature induction for stagewise collaborative filtering," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 314–322.