Recurrent Recommender Networks

Chao-Yuan Wu
University of Texas at Austin
Austin, TX
cywu@cs.utexas.edu

Amr Ahmed*
Google Research
Mountain View, CA
amra@google.com

Alex Beutel*†
Google Research
Mountain View, CA
alexbeutel@google.com

Alexander J. Smola*
Carnegie Mellon University
Pittsburgh, PA
alex@smola.org

How Jing
LinkedIn
Mountain View, CA
kublai.jing@gmail.com

ABSTRACT

Recommender systems traditionally assume that user profiles and movie attributes are static. Temporal dynamics are purely reactive, that is, they are inferred after they are observed, e.g. after a user's taste has changed or based on hand-engineered temporal bias corrections for movies. We propose Recurrent Recommender Networks (RRN) that are able to *predict* future behavioral trajectories. This is achieved by endowing both users and movies with a Long Short-Term Memory (LSTM) [14] autoregressive model that captures dynamics, in addition to a more traditional low-rank factorization. On multiple real-world datasets, our model offers excellent prediction accuracy and it is very compact, since we need not learn latent state but rather just the state transition function.

1. INTRODUCTION

The design of practical recommender systems is a well-established and well-studied subject. A common approach is to study problems of the form introduced in the Netflix contest [4]. That is, given a set of tuples consisting of users, movies, timestamps and ratings, the goal is to find ratings for alternative combinations of the first three attributes (user, movie, time). Performance is then measured by the deviation of the prediction from the actual rating.

This formulation is easy to understand and it has led to numerous highly successful approaches, such as Probabilistic Matrix Factorization [19], nearest neighbor based approaches [20], and clustering [5]. Moreover, it is easy to define appropriate performance measures (deviation between rating estimates and true ratings over the matrix), simply by selecting a random subset of the tuples for training and the rest for testing purposes.

*These authors contributed equally.
†A majority of this work was done while the author was at Carnegie Mellon University.

WSDM '17, February 6–10, 2017, Cambridge, United Kingdom.

ACM ISBN 978-1-4503-4675-7/17/02.
DOI: http://dx.doi.org/10.1145/3018661.3018689

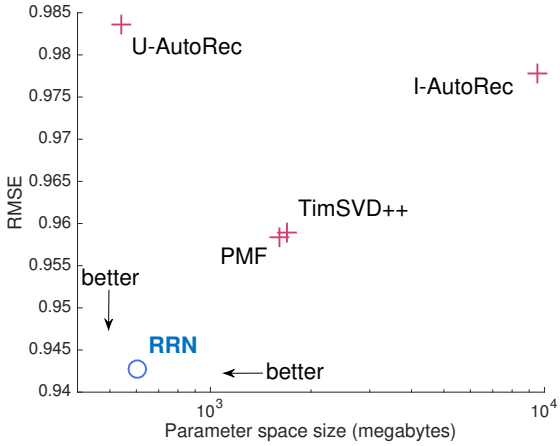

Figure 1: Accuracy and model size of RRN for the 6-month Netflix dataset, compared to [19], [16], and [21]. Our model achieves excellent *prediction* accuracy by being more flexible and dynamic than prior methods, while being comparable in model size.

Unfortunately, these approaches are lacking when it comes to temporal and causal aspects inherent in the data. The following examples illustrate this in some more detail:

Change in Movie Perception *Plan 9 from Outer Space* has achieved cult movie status by being arguably one of the world's worst movies. As a result of the *social* notion of being a movie that is so bad that it is great to watch, the *perception* changed over time from a truly awful movie to a popular one. To capture this appropriately, the movie attribute parameters would have to change over time to *track* such a trend. While maybe not quite as pronounced, similar effects hold for movie awards such as the Oscars. After all, it is much more challenging to hold a contrarian view about a critically acclaimed movie than about, say, *Star Wars 1, The Phantom Menace.*

Seasonal Changes While not quite so extreme, the relative appreciation of romantic comedies, Christmas movies and summer blockbusters is seasonal. Beyond the *appreciation*, users are unlikely to *watch* movies about overweight bearded old men wearing red robes in summer.

User Interest User's preferences change over time. This is well established in online communities [8] and it arguably also applies to online consumption. A user might take a liking to a particular actor, might discover the intricacies of a specific genre, or her interest in a particular show might wane, due to maturity or a change in lifestyle. Any such aspects render existing profiles moot, yet it is difficult to model all such changes *explicitly*.

Beyond the mere need of modeling temporal evolution, evaluating ratings with the benefit of hindsight also violates basic requirements of causality. For instance, *knowing* that a user will have developed a liking for *Pedro Almodóvar* in one month in the future makes it much easier to estimate what his opinion about *La Mala Educación* might be. In other words, we violate causality in our statistical analysis when we use *future* ratings for the benefit of estimating current reviews. It also makes it impossible to translate reported accuracies on benchmarks into meaningful assessments as to whether such a system would work well in practice. While the Netflix prize generated a flurry of research, evaluating different models' success on future predictions is hindered by the mixed distribution of training and testing data, as seen in Figure 2. Rather, by having an explicit model of profile *dynamics*, we can predict future behavior based on current trends.

A model capable of capturing the actual data distribution inherent in recommender systems needs to be able to model both the temporal dynamics within each user and movie, in addition to capturing the rating interaction between both sets. This suggests the use of latent variable models to infer the unobserved *state* governing their behavior. See Figure 3 for an example of such a model. The key difference between this and temporal models such as the ones proposed by [16] is that [16] uses a specific parametrization to *interpolate* temporal behavior. Instead, we use a nonparametric model that is able to *extrapolate* behavior into the future by learning the inherent user and movie dynamics. This makes it more adaptive to the true dynamics and less demanding on the statistical modeling skills of the experimenter.

Given the overall structure of the model, we are at liberty to posit a specific type of state for the latent variable. Popular choices are to assume a discrete latent state, as used e.g. when modeling web browsing behavior [9]. Likewise, we could resort to spectral methods [22, 1] or nonparametric estimators, such as Recurrent Neural Networks (RNNs). To address the vanishing gradient problem we resort to Long Short-Term Memory [14] in our model. Our contributions are as follows:

Nonlinear nonparametric recommender systems have proven to be somewhat elusive. In particular, nonlinear substitutes of the inner product formulation showed only limited promise in our experiments. To the best of our knowledge this is the first paper addressing movie recommendation in a fully causal and integrated fashion. That is, we believe that this is the first model which attempts to capture the dynamics of both users and movies. Moreover, our model is nonparametric. This allows us to model the data rather than having to assume a specific form of a state space.

Recurrent Recommender Networks are very concise since we only learn the dynamics rather than the state. This is one of the key differences to typical latent variable models where considerable effort is spent on estimating the latent state. In this aspect it closely resembles the neural autoencoder for recommendation of [21]. In fact, one could view the latter as a special case of our model.

Experiments show that our model outperforms all others in terms of forward prediction, i.e. in the realistic scenario where we attempt to estimate future ratings given data that occurred strictly prior to the to-be-predicted ratings. We show that our model is able to capture exogenous dynamics (e.g. an Oscar award) and endogenous dynamics (e.g. Christmas movies) quite accurately. Moreover, we demonstrate that the model is able to predict changes in future user preferences accurately.

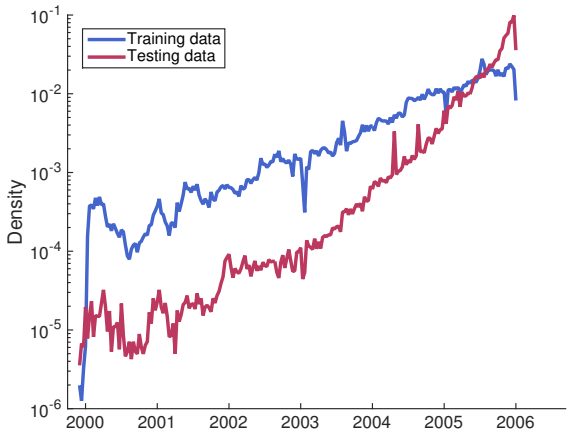

Figure 2: Data density in the Netflix contest. Note that while the test data is substantially overweighted to recent ratings, the training set is much more uniformly spaced out. This illustrates that the problem posed in the contest is much more one of interpolating ratings *with the benefit of hindsight* rather than predicting future user behavior.

2. RELATED WORK

2.1 Recommender Systems

Basic recommender systems ignore temporal information. This is a reasonable approximation, in particular for the Netflix contest, since opinions about movies and users do not change too rapidly and too dramatically in most cases.

Probably one of the most popular variants is *Probabilistic Matrix Factorization* (PMF) [19]. Albeit simple, PMF achieves robust and strong results in rating prediction. It is the prototypical factorization model where preferences are attributed to users and movies alike. Furthermore, implementations such as LIBPMF [24] are readily available, which make it easy to replicate experiments. Our proposed model uses the same factorization as PMF to model stationary effects. In this sense, it is a strict generalization.

Temporal aspects in recommendation were discussed in great length in Koren's prize-winning paper [16]. It proposed *TimeSVD++*, a temporal extension of the SVD++ matrix

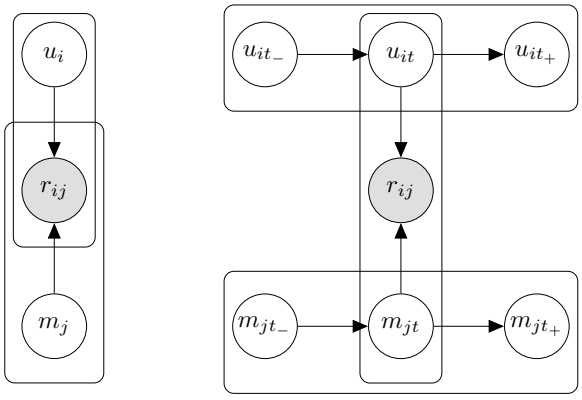

Figure 3: Left: time-independent recommendation model. User and movie parameters are stationary, ratings are drawn from $p(r_{ij}|u_i, m_j)$. Right: time-dependent recommendation model. Here both user and movie models follow a Markov chain and the ratings are drawn from $p(r_{ij}|u_{it}, m_{jt})$. Note that the plate notation is not entirely suitable for capturing the *additional* information of *which* (user,movie) pairs are rated and when.

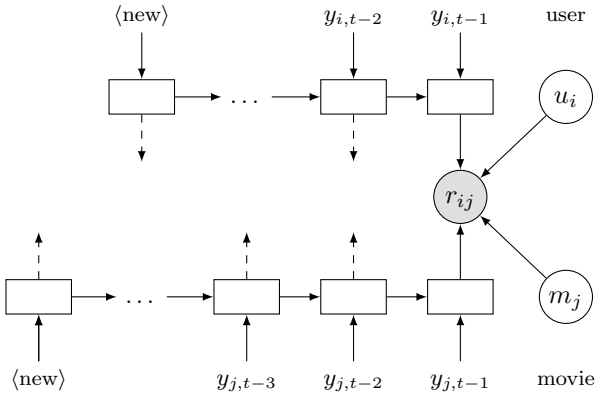

Figure 4: Recurrent Recommender Networks. We assume individual recurrent networks to address the temporal evolution of user and movie state respectively. The state evolution for a user depends on which movies (and how) a user rated previously. Likewise, a movie's parameters are dependent on the users that rated it in the previous time interval and its popularity among them. To capture stationary attributes we use an additional (conventional) set of auxiliary parameters u_i and m_j for users and movies respectively.

factorization algorithm. The key innovation here is to use a separate model to capture temporal dynamics, i.e. to allow for explicit temporal bias in the data. This allowed [16] to capture temporal inhomogeneity due to a change in rating labels and popularity changes in an integrated fashion.

Note that the features of TimeSVD++ are *hand engineered*, relying on intimate knowledge of the dataset. This makes the model difficult to port to new problems, e.g. from movies to music or books. Secondly, the model explicitly does not attempt to estimate *future behavior*, particularly because the Netflix competition did not require it, as seen in Figure 2. Instead, it only interpolates between past observations. In contrast, our model does not depend on feature engineering and is able to predict future ratings without the (unrealistic) assumption of having seen future user behavior.

Probably most closely related to our approach is *AutoRec* [21], one of the few Neural Networks models for recommendation. It uses the fact that matrix factorization can be viewed as an encoding problem: find a low-dimensional representation of a user's activity such that we can reconstitute all of a user's ratings from this vector. Given a set of movie attributes, this is achieved in the linear case by projecting a user's activity onto a lower-dimensional space and then projecting back from it (the proof is basic linear algebra).

AutoRec extends this idea from a linear autoencoder to a nonlinear one. In it, ratings are first aggregated by a lower-dimensional mapping and then processed by a number of intermediate compression and decompression layers as is common in autoencoders [3]. It is among the best neural network models so far in terms of rating prediction, and achieves state-of-the-art results on several datasets.

2.2 Recurrent Deep Networks

One of the key challenges in a graphical model described in Figure 3 is that it requires us to infer future states given observations, e.g. via message passing or particle filtering. This is costly and fraught with difficulty, since we need to match the emission model (of ratings) to the latent state. In other words, the only way to feed information about ratings back into the latent state is via the likelihood $p(r_{ij}|u_{it}, m_{jt})$. Common strategies for achieving this are via message passing or via particle filtering, i.e. via Sequential Monte Carlo sampling [2]. They are approximate and can be nontrivial to implement accurately at scale.

Alternatively we can simply learn the mapping as part of a nonparametric state update, i.e. via a Recurrent Neural Network (RNN). See [12] for an extensive overview and tutorial. The key idea is to use a latent variable autoregressive model as follows:

$$\hat{z}_{t+1} = f(h_t, z_t) \text{ and } h_{t+1} = g(h_t, z_{t+1}).$$

Here z_t denotes the observation at time t, $\hat{z}_t$ is the associated estimate, and h_t is the *latent state*. While the general functional form is well established [18] (e.g. the venerable Kalman Filter satisfies this condition), it is the use of nonlinearities and of tools to address stability and vanishing gradients that make the expression particularly effective.

A popular choice is the Long Short-Term Memory (LSTM) [14]. It captures temporal dynamics and we use it as a building block for a collaborative filtering system. The overall model is shown in Figure 4. The state updates satisfy the following operations:

$$[f_t, i_t, o_t] = \sigma\left[W\left[h_{t-1}, z_t\right] + b\right] \quad (1)$$

$$l_t = \tanh\left[V\left[h_{t-1}, z_t\right] + d\right] \quad (2)$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot l_t \quad (3)$$

$$h_t = o_t \cdot \tanh(c_t) \quad (4)$$

where f_t, i_t and o_t denote forget gate, input gate, and output gate respectively. They control how information flows through the sequence. For simplicity, in the following sec-

tions we use $h_t = \text{LSTM}(h_{t-1}, z_t)$ to denote these operations.

Note that LSTMs are not the only option. For instance, [7] propose the use of a Gated Recurrent Unit (GRU). It is very similar in spirit, albeit computationally cheaper and often similar in its results. For the purpose of the current paper we limit ourselves to LSTMs since they are slightly more general. [13] propose to use RNN for session-based recommendation, but it does not consider personalization, nor does it attempt to model user or item state transitions.

3. MODEL

In our approach we use LSTM Recurrent Neural Networks to capture temporal dependencies for both users and movies. This way we are able to incorporate past observations and to predict future trajectories in an integrated manner. Denote by u_i and m_j latent attributes for user i and movie j respectively. To deal with temporal dynamics we endow both with a time index, i.e. u_{it} and m_{jt}. Now all that is required is to define update functions. We use

$$\hat{r}_{ij|t} = f(u_{it}, m_{jt}) \text{ and } u_{i,t+1} = g(u_{it}, \{r_{ij|t}\})$$
$$m_{j,t+1} = h(m_{jt}, \{r_{ij|t}\}),$$

where $\hat{r}_{ij|t}$ denotes the predicted rating for user i on movie j, and $r_{ij|t}$ is the actual rating, at time step t. The functions f, g and h are learned such that we can infer a new user's state directly without need for further optimization. Rather than solving the optimization problem to find the user parameters, we solve an optimization problem to *find the functions that find the user parameters*. This aspect is similar to the deep autoencoder of [21] who learn an encoding function for past ratings. The key difference in our context is that we learn a function that sequentially updates scores and that is able to make forward predictions one set of ratings at a time.

3.1 User and Movie State

For clarity we illustrate the idea assuming a user-state RNN, as the movie-state RNN is defined in the same manner. Given a dataset of M movies, we denote by $x_t \in \mathbb{R}^M$ the rating vector for a given user at time t, where $x_{tj} = k$ if the user rated movie j with score k at time step t and $x_{tj} = 0$ otherwise. Moreover, we denote by τ_t the wallclock at time step t and we use $1_{\text{newbie}} = 1$ to indicate that the user is new.

$$y_t := W_{\text{embed}} \left[x_t, 1_{\text{newbie}}, \tau_t, \tau_{t-1}\right], \tag{5}$$

where W_{embed} is the transformation to be learned to project source information into embedding space. This yields y_t which serves as the input to an LSTM at step t. That is, we have the following model

$$u_t := \text{LSTM}(u_{t-1}, y_t) \tag{6}$$

Note that the steps where a user has not rated any movies are not included in the RNNs to save computations. Nonetheless, the inclusion of the wallclock gives the model the information needed to account for the no-rating steps and additionally capture effects such as rating scale change or movie age. See Section 4 for more details regarding a user model. To distinguish different users (and movies alike) we use the additional index i to denote user i when dealing with u_{it}. Likewise, for movies we use m_{jt} for movie j at time t.

3.2 Rating Emissions

Even though the user and movie states can be time-varying, we conjecture that there is still some *stationary* components that encode fixed properties such as the profile, long-term preference of a user, or the genre of a movie. To accomplish this, we supplement the time-varying profile vectors u_{it} and m_{jt} with stationary ones u_i and m_j respectively. We thus propose the rating being a function of both dynamic and stationary states, i.e.

$$\hat{r}_{ij|t} = f(u_{it}, m_{jt}, u_i, m_j) := \langle \tilde{u}_{it}, \tilde{m}_{jt} \rangle + \langle u_i, m_j \rangle \tag{7}$$

where $\tilde{u}_{it}$ and $\tilde{m}_{jt}$ are affine functions of u_{it} and m_{jt} respectively. That is, we have

$$\tilde{u}_{it} = W_{\text{user}} u_{it} + b_{\text{user}} \text{ and } \tilde{m}_{jt} = W_{\text{movie}} m_{jt} + b_{\text{movie}}$$

In short, the standard factorization accounts for stationary effects, while we use LSTMs for longer-range dynamic updates. This makes our model a strict superset of the more commonly used matrix factorization recommender systems.

3.3 Rating Prediction

Different from traditional recommender systems, in prediction time we use the *extrapolated* states instead of the estimated states for rating prediction. That is, we take the latest observations as input, update the states and make predictions based on the newly updated states. In this way we naturally take the causal effects brought by the previous ratings into account. For example, we become able to address hedonic adaptation [11], which in the context of movie recommendation, refers to the reduction in the level of satisfaction with a movie after watching another satisfying movie, or similarly for disappointing movies.

3.4 Inference

The optimization objective is to find parameters that yield predictions that are close to the actual ratings, i.e.

$$\underset{\theta}{\text{minimize}} \sum_{(i,j,t) \in \mathcal{I}_{\text{train}}} \left(r_{ij|t} - \hat{r}_{ij|t}(\theta)\right)^2 + R(\theta) \tag{8}$$

Here θ denotes all parameters to be learned, I_{train} is the set of observed (user, movie, timestamp) tuples in training set, and R denotes some regularization function. While the objective function and building blocks in our model are quite standard, a naive application of backpropagation cannot solve this problem easily. The key challenge is that each individual rating depends on both user-state RNN and movie-state RNN. Backpropagation through 2 sequences for every rating is computationally prohibitive. We can alleviate the problem a bit by back-propagating gradients from all the ratings of a user (movie) at once, but still, each rating would depend on its movie (user) state, and in turn, the movie's (user's) full sequence.

Instead, we propose an alternating subspace descent strategy that does not suffer from this issue. That is, we still back-propagate the gradients of all the ratings of a user at once to update user-sequence parameters, but now we assume movie states are fixed, so there is no need to propagate gradients into those movie sequences. Then we alternate between updating user sequences and updating movie-sequences. This way, we can perform only one time of standard feed-forward and back-propagation for each user (movie). This strategy is well known as subspace descent.

Dataset	IMDb	Netflix 6 months	Netflix 1 year	Netflix full
Users	440.8k	311.3k	345.9k	477.4k
Items	114.3k	17.7k	16.9k	17.8k
Train size	1.4M	13.7M	41.5M	98.1M
Test size	29.6k	2.1M	3.9M	2.3M
Train data	7/98-12-12	6/05-11/05	6/04-5/05	12/99-11/05
Test data	1/13-9/13	12/05	6/05	12/05
Sparsity	0.0028%	0.25%	0.7%	1.2%

Table 1: IMDb and different splits on the Netflix dataset used in experiments. To evaluate the ability to model state transition dynamics, users and items not shown in the training set are removed from the testing set.

4. EXPERIMENTS

We present both quantitative and qualitative analysis of our model. We demonstrate RRN's ability to automatically model a variety of temporal effects and make accurate predictions to future ratings. In particular, we show that the ratings are the best currently available (for models satisfying realistic causality conditions). Moreover, they accurately reproduce the temporal effects that are usually hand-engineered in temporal recommender systems.

In order to study the effectiveness modeling temporal dynamics, we evaluate our model on the well-understood dataset from Netflix contest and the IMDb dataset, first used in [10]. The IMDb dataset comprises of 1.4M ratings collected between July 1998 and September 2013, and the Netflix dataset consists of 100M ratings collected between November 1999 and December 2005. Each data point is a (user id, item id, time stamp, rating) tuple with a time stamp granularity of 1 day. To better understand different aspects of our model, we test our model on several different time windows with different training and testing period. A detailed summary is shown in Table 1. Note that we split our data *based on time* to simulate the actual situations where we need to predict future ratings instead of interpolate previous ratings. Ratings from the testing period are evenly split into validation and testing sets.

4.1 Setup

We found that our model is able to achieve good accuracy even with a very small amount of parameters. In the following experiments, we use a single-layer LSTM with 40 hidden neurons, 40-dimensional input embeddings, and 20-dimensional dynamic states. We use 20-dimensional and 160-dimensional stationary latent factors for Netflix and IMDb dataset respectively. Our model is implemented on MXNet [6], an open-source deep learning framework.

We use ADAM [15] to optimize neural network parameters and stochastic gradient descent (SGD) to update stationary latent factors. Architecture and learning algorithm parameters are selected by cross-validation. We found that we can generally obtain better results if we first train stationary states only, and then jointly train the full model. In the following experiments, stationary latent states are initialized by a small pre-trained PMF and a U-AutoRec model for Netflix and IMDb experiments respectively. Wallclock time τ_ts are rescaled to have zero mean and unit variance.

We compare RRN against the following state-of-the-art models.

PMF [19] Albeit simple, PMF achieves robust and strong results in rating prediction. As our model adopts the same factorization as PMF to model stationary effects, comparing to PMF directly shows the benefits of modeling temporal dynamics with our model. We use LIBPMF [24] in experiments. Grid search of regularization parameter over $\lambda \in \{10^0, \ldots, 10^{-5}\}$ and factor size over $k \in \{20, 40, 80, 160\}$ is performed to select the best parameters.

TimeSVD++ [16] TimeSVD++ is one of the most successful models to capture temporal dynamics and has shown strong results at Netflix contest. The implementation in GraphChi [17] is used in experiments. Similarly, regularization parameter and factor size are selected by grid-search over $\lambda \in \{10^0, \ldots, 10^{-5}\}$ and $k \in \{20, 40, 80, 160\}$.

AutoRec [21] learns an autoencoder that encodes each movie (or user) into lower-dimensional space and then decodes to make predictions. It is among the best neural network models so far in terms of rating prediction, and achieves state-of-the-art results on several datasets. We use the software provided by the authors to evaluate AutoRec. Parameters that yield the best performance in original [21] are used in our experiments (latent state dimension $k = 500$).

4.2 Rating Prediction

	PMF	I-AR	U-AR	T-SVD	**RRN**
IMDb	2.3913	2.0521	2.0290	2.0037	**1.9703**
Netflix 6 months	0.9584	0.9778	0.9836	0.9589	**0.9427**
Netflix full	0.9252	0.9364	0.9647	0.9275	**0.9224**

Table 2: RRN outperforms competing models in terms of RMSE. On Netflix datasets, a RRN with only 20-dimensional stationary factors and 40-dimensional embedding is enough to outperform PMF and TimeSVD++ of dimensionality 160 and AutoRec with 500-dimensional embeddings. Dimensionality and regularization parameter of PMF and TimeSVD++ are selected by cross-validation. (I-AR: I-AutoRec, U-AR: U-AutoRec, T-SVD: TimeSVD++.)

We evaluate the performance of rating prediction based on standard root-mean-square error (RMSE). A summary of results on different datasets is shown in Table 2. Here we use a time step granularity of 2 months for full Netflix and IMDb and 1 day/7 days (users/movies) for the 6-month Netflix dataset. See Section 4.6 for discussions on the choice of time step granularities. We run our model for 10 epochs. After every epoch, RMSE is computed. We report the RMSE on the testing set for the model that gives the best results on validation set.

Accuracy and Size.

Our model achieves the best accuracy on all datasets among all compared methods including the *best neural network model* and the *best temporal model* available. Compared to PMF, RRN offers an improvement of 1.7% on IMDb and 1.6% on

6-month Netflix dataset. Note that the RMSEs presented in Table 2 are higher than those achieved in Netflix contest because we test on *purely future ratings*.

In addition, our model is very concise, as we can store the *transition function* instead of the actual states to model temporal dynamics. On the Netflix datasets, while outperforming all baseline models, RRN is 2.7 times smaller than PMF and TimeSVD++, and 15.8 times smaller than I-AutoRec. Specifically, a RRN with 40-dimensional embeddings and 20-dimensional stationary states is enough to outperform PMF and TimeSVD++ of dimensionality 160 and AutoRec with 500-dimensional embeddings. Figure 1 shows the model size and RMSE on the 6-month dataset. For IMDb, RRN is of comparable size to PMF, TimeSVD++ and U-AutoRec, and is much smaller than I-AutoRec; this is because RRN uses the same dimension stationary states as the factorization models and includes a relatively small model to capture the dynamics. We see clear advantage of RRN in terms of flexibility and thus accuracy, without sacrificing on size[1].

Robustness.

While other methods swap rankings across different datasets, RRN shows consistent improvements. In particular, we see that for PMF and Time-SVD++ the relative improvement decreases on IMDb, as it is 90 times sparser than the 6-month Netflix dataset as shown in Table 1. On the other hand, RRN does not suffer from sparsity. We conjecture that this is due to the fact that our model learns the functions that find parameters instead of learning the parameters directly, so the statistical strength are shared across all data points. In dense settings, RRN still achieves a smaller size and comparable or better accuracy. In addition, PMF performs much worse than Time-SVD++ on the 14-year-spanning IMDb dataset. This again stresses the need for modeling temporal dynamics.

4.3 Temporal Dynamics

Exogenous Dynamics on movies.

The goal is to understand how our model reacts to exogenous effects, such as awards. Specifically, we run RRN on the 1-year dataset with a time step granularity of a month, and see how the average predicted ratings evolve along the sequence. The average ratings are calculated over 1000 randomly sampled users, who might or might not have rated the movie at that time step. This represents the average predicted rating for a movie, and avoids bias by who chose to rate the movie at that time. Figure 5 shows average predicted ratings on award winners. We see when a movie receives awards or nominations, there is a clear rise in predicted ratings, which matches the data trend. This confirms our expectation that RRN should adapt automatically to exogenous changes and adjust predictions.

Endogenous Dynamics on movies.

In addition to exogenous events that cause perception changes, a movie can experience variations over time for endogenous reasons. To understand how RRN models these effects, we test on the full 6-year dataset with a time step granularity of 2 months. Experiments are conducted in the same way as the previous experiment, but considering the fact that most users are only active on the system for some time, here for each time step we sample a different set of 5,000 users that are active at that time. A user is considered active if she rated any movie in the time step. We look at the movies with big change in ratings and observe how RRN models their dynamics[2]. First of all, we observe the "age effect", as pointed out in [16]. That is, a movie's rating tends to decrease slightly in the first year after its release, and then increase with age. From Figure 6 we see our model clearly captures this "age effect" and adapts effectively. In addition, we also notice how RRN models movies with different reception differently. Figure 7 shows the predicted ratings for Oscar best picture nominees and Golden Raspberry worst picture nominees. We see movies within each group show consistent patterns that match intuitive sense. Raspberry nominees initially experience a dip and then the average ratings increase, while Oscar nominees rise at the beginning and then stay relatively stable. Note that these patterns are more than a simple shift that can be captured by a bias term.

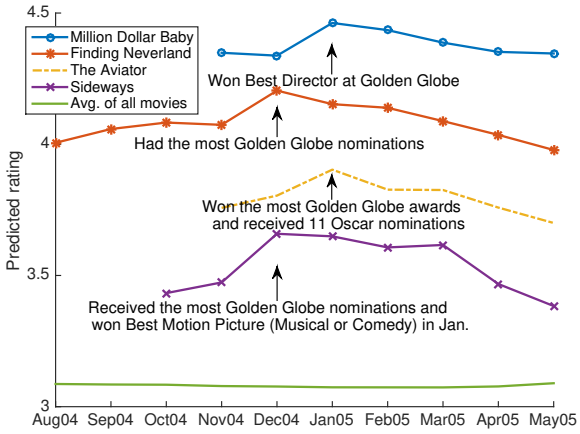

Figure 5: Ratings for a number of prize winning movies relative to a general baseline. We sampled 1,000 and tracked how *the same users* would have rated *the same movie* as a function of time. Awards lead to a short-time positive effect until a user's default preferences prevail again (Netflix does not display awards in its user interface). For reference, Golden Globe nominees were announced on 12/13/04, the award ceremony was held on 1/16/05. Oscar nominees were announced on 1/25/05 and the award ceremony followed on 2/27/05.

User Interface Changes.

As pointed out in [16], there is a change of rating scale in early 2004. Clearly systems that fail to take this into account will have inferior estimation accuracy. We test our model on

[1] Here we consider each factor to be a 32-bit floating point number in calculating model size.

[2] Movies that joined the dataset before 2004, have at least 1000 ratings every year, and have a fluctuation greater than 0.5 are selected. The fluctuation is defined as the difference between the average rating of the highest-rated year and that of the lowest-rated year. This procedure results in a set of 256 movies.

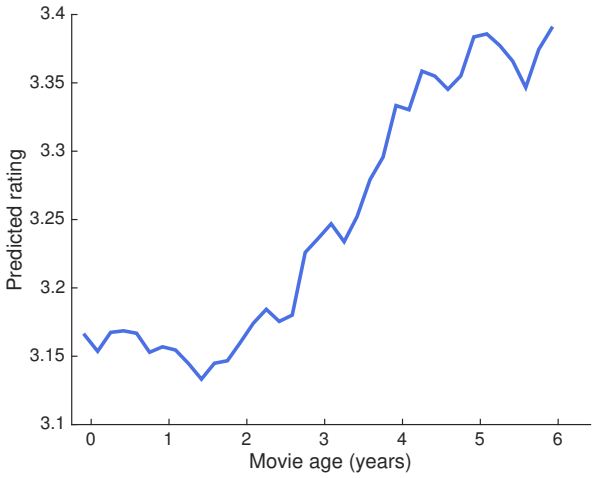

Figure 6: Average predicted ratings as a function of movie age. Our estimates are consistent with the fact a movie's impression tends to degrade initially after its release. Subsequently it may achieve cult status (relatively speaking) and its public perception increases over time.

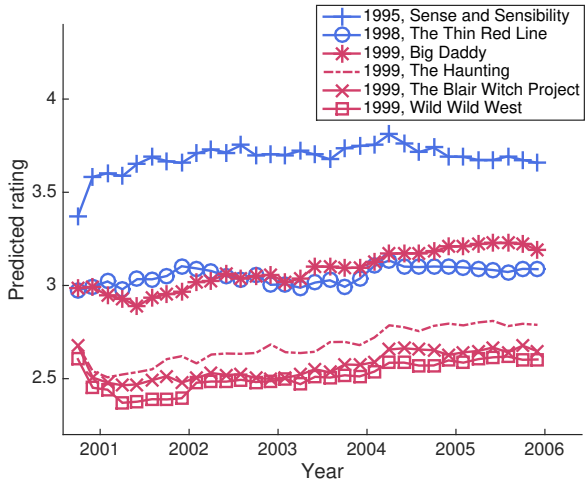

Figure 7: Average predicted ratings for Oscar best picture nominees (in blue) and Golden Raspberry worst picture nominees (in red). Movies within each group show consistent patterns that match intuitive sense. Note that these patterns are more than a simple shift that can be captured by a bias term.

full Netflix dataset with a time step granularity of 2 months, and look at the average rating that each time step generates for randomly selected users. Note that since we are calculating the average prediction over random users over time, we observe how our model's dynamic embeddings change over time. Figure 8 shows the average predicted ratings grouped by movie release year. We see for all the curves, there is a clear rise in early 2004 that matches the scale change. This is in contrast to PMF, for which embeddings are static and thus average predictions over time are constant.

4.4 Cold-Start

To understand the strength and weakness of our model, we perform a careful comparison to the stationary PMF model on users and movies for which we have few ratings in our training dataset. As can be seen in Figure 9, RRN improves over PMF for users with few ratings in the training data and provides the greatest improvements for users for which we observe *very* few ratings. In Figure 10, we find that RRN still consistently provides improvements for movies with few ratings, but the relationship between improvement and the number of observations is far more noisy. This is likely due to the relatively few ratings in the test set for these movies.

4.5 Incorporate New Data without Re-Training

One advantage of estimating the transition function instead of the state itself is that even without re-training, we are able to incorporate the information from the newly observed ratings to update states (by simply feeding the new ratings into the network). Here we evaluate this strategy on the 6-month Netflix dataset. Specifically, we extrapolate the states using the ratings observed from the first testing time step, and use the extrapolated states to predict ratings in the second time step. We test on movies with different levels of rating fluctuations. Fluctuation is defined as the difference between the average rating of the highest-rated time-step

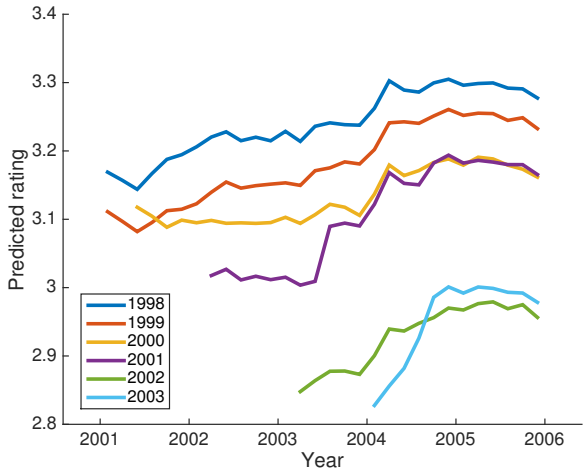

Figure 8: Average predicted ratings, grouped by release year. Note the spike between 2003 and 2004, i.e. the time when the labels of star ratings on the Likert scale were changed.

and that of the lowest-rated time-step[3]. RMSE improvements from this strategy are summarized in Figure 11. We observe that while very small fluctuations are not captured by our pre-trained model, large fluctuations are successfully described, leading to significant improvements in prediction accuracy. That is, using RRN not only relieves the need of frequent expensive re-training, but also opens a new avenue to provide real-time recommendations [23].

[3] To eliminate the noise when estimating fluctuations, we remove movies with less then 100 ratings within the testing month. This results in 3,552 movies and 1,808,654 ratings tested in this experiment.

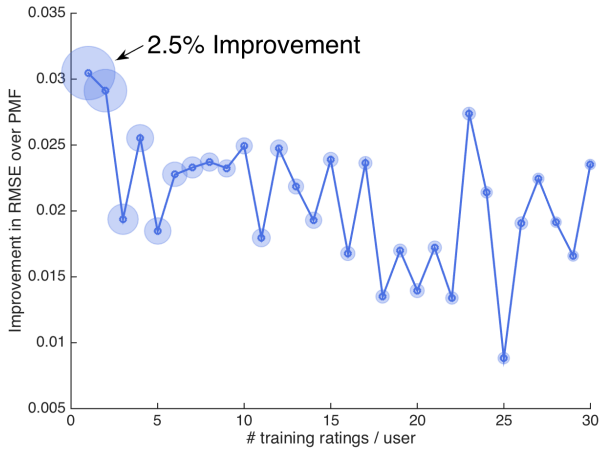

Figure 9: Gain of RRN over PMF in RMSE. RNN achieves a 2.5% improvement for users with only 1 training rating. Users with few training data benefit the most. In fact, there are 37k users with only 1 training rating and only 5.2k users with 30 training ratings. Marker size denotes user counts.

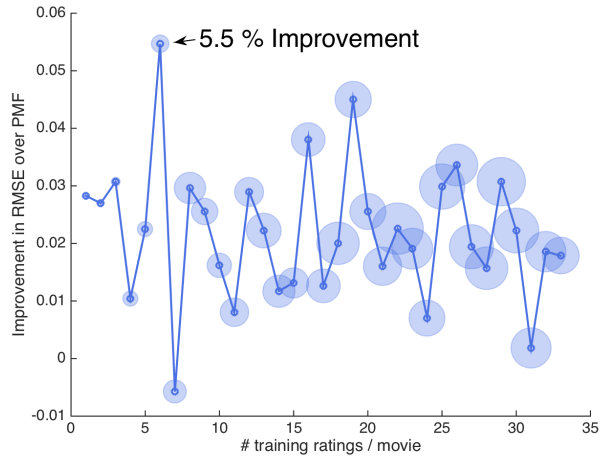

Figure 10: Gain of RRN over PMF in RMSE. Movies with more than 10 training ratings show consistent improvements. Marker size denotes movie counts.

4.6 Time Step Granularity and Sensitivity

A smaller time-step granularity allows the model to update states frequently and capture short-term effects. However, it also results in longer LSTM sequences, that are computationally more expensive to train. Table 3 summarizes the training time and RMSE over different granularities on the 6-month dataset. We see a trade-off between RMSE and training-time here. One can obtain better accuracy at the cost of longer training time. Note that the performance is not sensitive to granularity, and even the worst of these RMSEs outperforms all the baseline models. This could be due to the fact that RRN is completely general, in the sense that it assumes no specific form or distribution of data.

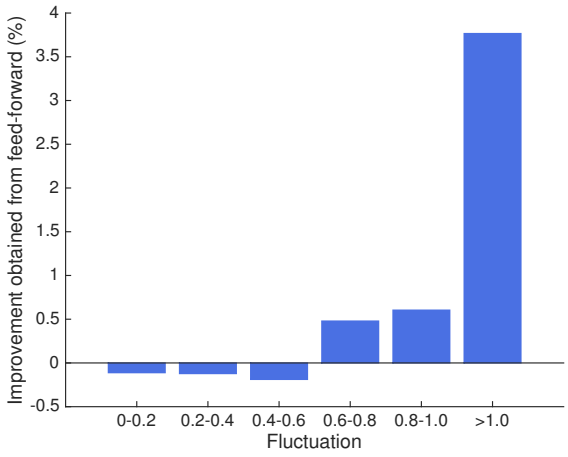

Figure 11: RMSE improvement gained from feed-forwarding ratings *without* re-training. We see clear improvements for movies with big fluctuations.

Granularity (user/movie)	RMSE	Time (user/movie)
1 day / 1 day	0.9440	9,235 / 7,225
1 day / 7 days	**0.9427**	8,587 / 4,478
7 days / 1 day	0.9459	6,845 / 7,219
7 days / 7 days	0.9440	**6,800** / **4,298**

Table 3: RMSE and training-time trade-off for different time step granularities. Training-time (in seconds) is measured for one epoch of user/item-sequence update.

5. CONCLUSION AND DISCUSSION

In summary, we have provided RRN, a novel recommender system based on recurrent neural networks that can accurately model user and movie dynamics. In this paper, we have provided the following contributions:

1. **Nonlinear, Nonparametric, Dynamic Recommender:** We offer the first, to the best of our knowledge, recommender system that jointly models the evolution of both user and item states, and focuses on extrapolating predictions into the future without hand-crafted features. We accomplish this by adapting recurrent neural networks architectures, particularly LSTMs, to recommendation data in order to learn dynamic embeddings of users and movies.

2. **Efficient Training:** We provide an efficient training procedure for our model, based on alternating optimization of user and movie embeddings. This enables us to scale to over 100 million ratings from over 6 years.

3. **Empirical Evidence:** We demonstrate that our model, RRN, improves prediction accuracy over previous methods, and implicitly captures a variety of known temporal patterns in ratings data without explicit inclusion in the model.

Acknowledgement

This research was supported by funds from Google, Bloomberg, Adobe, a National Science Foundation Graduate Research Fellowship (Grant No. DGE-1252522), and the National Science Foundation under Grant No. IIS-1408924. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties.

6. REFERENCES

[1] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky. Tensor decompositions for learning latent variable models. *arXiv preprint arXiv:1210.7559*, 2012.

[2] C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan. An introduction to mcmc for machine learning. *Machine Learning*, 50:5–43, 2003.

[3] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al. Greedy layer-wise training of deep networks. *NIPS*, 19:153, 2007.

[4] J. Bennett and S. Lanning. The netflix prize. In *Proceedings of KDD Cup and Workshop*, volume 2007, page 35, 2007.

[5] A. Beutel, A. Ahmed, and A. J. Smola. ACCAMS: Additive co-clustering to approximate matrices succinctly. In *WWW*, pages 119–129. ACM, 2015.

[6] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. W. T. Xiao, B. Xu, C. Zhang, and Z. Zhang. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[7] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Gated feedback recurrent neural networks. *arXiv preprint arXiv:1502.02367*, 2015.

[8] C. Danescu-Niculescu-Mizil, R. West, D. Jurafsky, J. Leskovec, and C. Potts. No country for old members: User lifecycle and linguistic change in online communities. In *WWW*, pages 307–318, 2013.

[9] M. Deshpande and G. Karypis. Selective markov models for predicting web page accesses. *ACM Transactions on Internet Technology (TOIT)*, 4(2):163–184, 2004.

[10] Q. Diao, M. Qiu, C.-Y. Wu, A. J. Smola, J. Jiang, and C. Wang. Jointly modeling aspects, ratings and sentiments for movie recommendation (jmars). In *SIGKDD*, pages 193–202. ACM, 2014.

[11] S. Frederick and G. Loewenstein. Hedonic adaptation. *D. Kahneman, E. Diener, and N. Schwarz (Eds.), Well being. The foundations of hedonic psychology*, 1999.

[12] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[13] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[14] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[15] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Y. Koren. Collaborative filtering with temporal dynamics. In *Knowledge discovery and data mining KDD*, pages 447–456, 2009.

[17] A. Kyrola, G. Blelloch, and C. Guestrin. GraphChi: Large-scale graph computation on just a pc. In *OSDI*, 2012.

[18] S. Roweis and Z. Ghahramani. A unifying review of linear Gaussian models. *Neural Computation*, 11(2), 1999.

[19] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *NIPS*, volume 20, 2008.

[20] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295. ACM, 2001.

[21] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie. Autorec: Autoencoders meet collaborative filtering. In *WWW Companion*, pages 111–112, 2015.

[22] L. Song, B. Boots, S. Siddiqi, G. Gordon, and A. J. Smola. Hilbert space embeddings of hidden markov models. In *ICML*, 2010.

[23] C.-Y. Wu, C. V. Alvino, A. J. Smola, and J. Basilico. Using navigation to improve recommendations in real-time. In *RecSys*, pages 341–348. ACM, 2016.

[24] H.-F. Yu, C.-J. Hsieh, S. Si, and I. S. Dhillon. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *ICDM*, 2012.