Long and Short-Term Recommendations with Recurrent Neural Networks

Robin Devooght
IRIDIA, Université Libre de Bruxelles
1050, Brussels, Belgium
robin.devooght@ulb.ac.be

Hugues Bersini
IRIDIA, Université Libre de Bruxelles
1050, Brussels, Belgium
bersini@ulb.ac.be

ABSTRACT

Recurrent neural networks have recently been successfully applied to the session-based recommendation problem, and is part of a growing interest for collaborative filtering based on sequence prediction. This new approach to recommendations reveals an aspect that was previously overlooked: the difference between short-term and long-term recommendations. In this work we characterize the full short-term/long-term profile of many collaborative filtering methods, and we show how recurrent neural networks can be steered towards better short or long-term predictions. We also show that RNNs are not only adapted to session-based collaborative filtering, but are perfectly suited for collaborative filtering on dense datasets where it outperforms traditional item recommendation algorithms.

KEYWORDS

Collaborative Filtering; Recommender Systems; Recurrent Neural Network; Sequence Prediction

1 INTRODUCTION

Collaborative filtering methods most often consider the user as a static entity whose interests are fixed in time. Matrix factorization for example uses all the ratings (or implicit feedback) of a user to build a representation of its general tastes, oblivious to the possible evolution of taste or fading interests of the user. Some, like time-SVD++, incorporate the timestamp of ratings in order to improve ratings prediction [12], but the use of the sequential nature of interactions between users and items has been poorly studied.

Recently however some methods have started to frame the item recommendation problem of collaborative filtering as a sequence prediction problem [8, 9, 20]: given that the user consumed this item, then this one, etc. which item will he consume next ? The motivation for those works comes from the nature of the data: they either tackle datasets with a very large number of users but extremely sparse information about those users [8] or session-based dataset where users are only identified for the time of a session, and recommendations must therefore be based only on the last few clicks of the user[9, 20]. In both cases it is not realistic to build a model for every users, and they instead take the approach of directly recommending items based on the few past interactions of the user. The use of sequence information allowed to compensate for the scarcity of recorded interactions between users and items.

In the first part of our paper we show that methods based on sequence prediction (and especially recurrent neural networks) are similarly powerful on dense datasets. The sequence of actions of the user holds indeed a lot of information: it can reveal the evolution of a user's taste, it might help to identify which items became irrelevant with regards to the current user's interests, or which items make part of a vanishing interest.

This paper then highlights an aspect of recommender systems that was previously overlooked but is revealed by the sequence prediction approach: the difference between short-term and long-term predictions. Long term predictions aim to identify which items the user will consume *eventually*, without regards for when exactly he will consume them, while short-term predictions should accurately predict the immediate behavior of the user: what he will consume *soon*, and in the extreme case, what he will consume *next*. In the static setting, this distinction does not make sense because the order of items is ignored (both during training and testing), but in the sequence-based approach this distinction is evident because sequence prediction algorithms are specifically design to predict *the next* item, which make them especially good at short-term predictions, sometimes at the cost of worse long-term predictions.

Some applications are more oriented towards short-term predictions while others aim to make good long-term predictions (e.g. recommending the next song in a playlist versus recommending a book based on everything that the user has read). We study in details this short-term/long-term aspect of recommendation systems and make the following contributions:

- We introduce a practical visualization of the short-term/long-term profile of any recommender system and use it to compare several algorithms.
- We show how to modify the RNN to find a good trade-off between long and short-term predictions.

UMAP'17, July 9–12, 2017, Bratislava, Slovakia
 978-1-4503-4635-1/17/07...$15.00
DOI: http://dx.doi.org/10.1145/3079628.3079670

- We explore the relationship between short-term predictions and diversity.

Our code is available at https://github.com/rdevooght/sequence-based-recommendations, and our data, with the precise training/validation/test splits used in the experiments, is available at http://iridia.ulb.ac.be/~rdevooght/rnn_cf_data.zip.

2 RELATED WORK

Some early works have framed collaborative filtering as a sequence prediction problem and used simple Markov chain methods to solve it. In the early 2000s, Zimdars et al. tested a Markov model for web-page recommendation[21]. Mobasher et al. adopted a similar approach, using sequential pattern mining[15]. Both showed the superiority of methods based on sequence over nearest-neighbors approaches. In [2, 18], Brafman et al. defended the view of recommendation systems as a Markov decision process, and although the predictive model was not their main focus, they did present in [18] a Markov chain approach, improved by some heuristics such as skipping and clustering.

The community then has gradually lost interest in the sequence prediction approach, with the only major work in many years being FPMC[17], which combines the matrix factorization of the user-item matrix with Markov chains, and is still considered to be the state-of-the-art in sequence-based recommendations.

Recently however, several papers have explored sequence-based recommendation, mostly motivated by the need to produce recommendations without building a user model[8, 9, 20]. Hidasi et al. has applied gated RNN to session based collaborative filtering using two objective functions, BPR (from [16]) and the newly formulated TOP1[9]. Tan et al. then improved on those results through data manipulation and variation in the training methods[20].

It is also worth noting that Spotify might have been using it as far back as 2014 [1] to build playlists. They however do not seem to be using gated RNN, and they are using a hierachical softmax output in order to deal with the very large number of items (they have much more songs than netflix or movielens have movies).

He and McAuley took another approach with Fossil[8], an algorithm similar in spirit to FPMC, but instead of factorizing the user-item matrix, they take inspiration from the item-similarity based approaches[11] and factorize the item-item matrix.

Guàrdia-Sebaoun et al. used word2vec (a well-known unsupervised word embedding algorithm [14]) to build a representation of the item based on sequences[7]. They then model the users as a transition vector between item representations and used it to predict the next items in the sequence[1].

[1]We tried it on our datasets but the results were very poor. Our implementation of the method is available with the rest of our code.

Table 1: Dataset descriptions. For Movielens and Netflix each sequence represents a user, while in RSC15 they represent sessions.

Dataset	#sequences	#items	#interactions
Movielens	6040	3706	1,000,209
Netflix	480,189	17,770	100,480,507
RSC15	7,981,581	37,486	31,708,505

3 COLLABORATIVE FILTERING WITH RNN

The idea of using recurrent neural networks for collaborative filtering is a recent one, and they have not yet been thoroughly tested on dense datasets. In the following we compare the RNN with multiple state-of-the-art collaborative filtering algorithm and show that they perform well on dense and sparse datasets.

We use n and m to represent the number of users and items in the dataset. $\mathcal{S}_u$ represents the sequence of items with which user u interacted.

3.1 Datasets

We use three datasets whose characteristics are shown in Table 1. **Movielens** and **Netflix** are well-known datasets recording the ratings of users on a catalog of movies. Because the timestamp of each rating is available, it is possible to construct for each user the sequence of movies he rated. It is worth precising that we do not use the values of the ratings in any way, we are only trying to predict which movie a user will rate, based on what he rated before.

The **RecSys'15 Challenge** (RSC15) dataset is a collection of navigation sessions on an e-commerce website and it differs from the two movie datasets in many ways. First of all, its sequential nature is much more evident and concerns much shorter time scales. The sequences do not represent users, but only short navigation sessions, which means that long sequences are much less frequent (the median length of a sequence is 3 in RSC15 against 96 in Movielens). The much larger number of sequences and their relative shortness makes it less practical and less pertinent to use methods based matrix factorization, that would need to build a representation for each session.

Each set is divided into training, validation and test subsets: N randomly chosen users and all their ratings to constitute the test set, N others to constitute the validation set and all the remaining users for the training set. We used N = 500 for Movielens 1M, N = 1000 for Netflix and N = 10k for RSC15.

In Movielens and Netflix, any item can appear only once in the rating history of any user, and we helped all the methods on those datasets by forcing them to recommend items that the user had not yet seen.

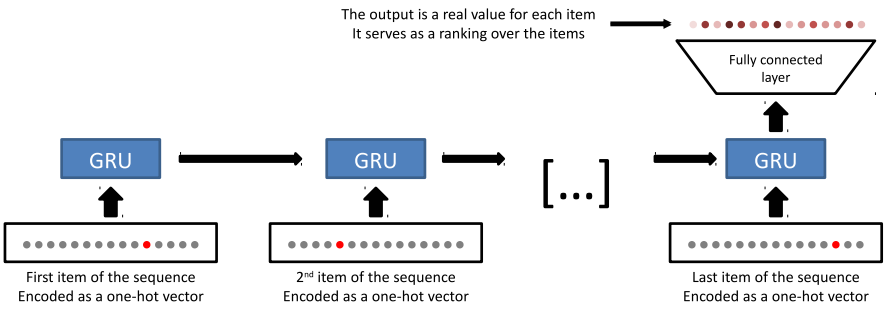


Figure 1: Schematic representation of the RNN.

3.2 Recurrent neural networks

Recurrent neural networks (RNN) are commonly used for language modeling, where they are trained to learn sequences of words [13]. We take a similar approach by considering each item as a word, the catalog of items as the full vocabulary, and the history of each user as a sample sequence. The RNN runs through the sequence of items consumed by a user, item by item, as depicted in Figure 1. At each time step, the input is the one-hot encoding of the current item, and the final output is a fully-connect layer with a neuron for each item in the catalog. The k items whose neurons are activated the most are used as the k recommendations.

The state-of-the-art in recurrent neural networks is what are called "gated" RNNs, where the internal state of the RNN is controlled by one or more small neural networks called gates, with the most common architectures being the LSTM[10] and the GRU[3]. In the following we used the GRU for easier comparison with other works[9, 20], but the LSTM works just as well.

A major aspect of the RNN is the choice of an appropriate objective function. In the following experiments we compare two different objective functions[2].

Categorical cross-entropy (CCE), the most common loss in language modeling. It is defined as $\text{CCE}(\mathbf{o}, i) = \log(\text{softmax}(\mathbf{o})_i)$ where $\mathbf{o}$ is the output vector of the RNN and i is the correct item. A major problem of CCE is that its complexity is linear in the number of items (because of the computation of the softmax) which can be very expensive in application with large catalog.

Hinge is an objective function that is based on the objective function of SVMs. Unlike CCE, it does not depend on the comparison of outputs between items, but on the independent comparison of the output of each item against a fixed threshold. We define it as follows: $\text{Hinge}(\mathbf{o}, i) = \sum_{j \in \mathcal{C}} \max(0, 1 - o_j) - \gamma \sum_{j \in \mathcal{F}} \max(0, o_j)$, where $\mathcal{C}$ is the set of good recommendations, $\mathcal{F}$ is the set of bad recommendations and γ is a factor used to balance the influence of the error on the correct item and the error on the much more numerous incorrect items. In the following, $\mathcal{C}$ contains only the next item in the sequence, and $\mathcal{F}$ contains all the other items. Because γ is difficult to choose intuitively, we use instead $\beta = \gamma|\mathcal{F}|/|\mathcal{C}|$. β is easier to interpret: $\beta = 1$ means that correct and incorrect items have the same weight on the objective function, $\beta = 2$ means that the incorrect items weight twice as much, etc. The Hinge objective function is as expensive to compute as the CCE, but its interest relies in the fact that it can be used to define several items as being correct (see Section 5.3).

3.2.1 Training Procedure. We used the standard mini-batch stochastic gradient descent method (see e.g. [6, Section 8.1.3]). A training instance is produced by randomly cutting the sequence of a user's interactions into two parts; the first part is fed to the RNN, and the first item of the second part is used as the correct item in the computation of the objective function. The training consists of looping over the users, and for each user generating b training instances (or less if the length of the sequence is less than b) that are used as a mini-batch. We used $b = 16$ in our experiments.

3.2.2 Parameters. In the following experiments we arbitrarily set the size of the GRU cell to $k = 50$. We used the Adagrad learning scheduler[4]. The learning rate (l), the β parameter of Hinge are tuned on the validation set of Movielens. RNNs are usually trained on sequences up to a certain length, and the maximum length (M_l) is a difficult parameter to choose: longer length means potentially more informed prediction, but it also significantly increases the training time. We arbitrarily set $M_l = 30$ after observing its influence on training time and recommendation quality on the Movielens validation set. Our implementation is based on the Lasagne framework for Theano[3], and parameters that were not mentioned are kept to Lasagne defaults.

3.3 Competing Methods

We compare the RNN with two static methods of top-N recommendation: one based on nearest neighbors and one on matrix factorization. We also compare the RNN with a simple Markov chain that can be seen as a baseline for the sequence prediction approach, and with FPMC [17] and Fossil [8], the state-of-the-art for recommendations based on

[2]Objective functions are often paired with a specific choice of non-linearity on the last layer, like softmax and CCE. We chose to consider the last layer without non-linearity and to incorporate the non-linearity in the definition of the objective function.

[3]See http://lasagne.readthedocs.io and http://www.deeplearning.net/software/theano/

sequences. Finally, POP is the baseline that always rank items according to their frequency in the training set.

3.3.1 User-Based Nearest Neighbors. User-based nearest neighbors or user KNN is one of the oldest method of collaborative filtering, yet still a strong baseline for top-N recommendation. A score s_{ij} is computed between a user i and an item j:

$$s_{ij} = \sum_{u \in \mathcal{N}_k(i)} c_{iu} \mathbb{1}(j \in \mathcal{S}_u) \quad (1)$$

Where c_{iu} is the similarity between users i and u, $\mathcal{N}_k(i)$ is the set k users closest to i according to the similarity measure c, and $\mathbb{1}(j \in \mathcal{S}_u)$ is the indicator function that evaluates to 1 if item j belongs to the sequence of items of user u, or else evaluates to 0. We used the cosine similarity measure, which is usually preferred for item recommendation:

$$c_{iu} = |\mathcal{S}_i \cap \mathcal{S}_u| / \sqrt{|\mathcal{S}_i||\mathcal{S}_u|} \quad (2)$$

The size of the neighborhood (k) was tuned on the Movielens validation set.

3.3.2 BPR-MF. BPR-MF is a state-of-the-art matrix factorization method for top-N recommendation devised by [16] and based on the BPR ranking loss. We used the original implementation of BPR-MF, available in the MyMediaLite framework [5]. BPR-MF has many parameters; we tuned the number of features (k) ourselves and kept the default values of MyMediaLite for the others.

3.3.3 Markov Chain. In this simple method, the users' behavior is modeled by a Markov chain whose states are the different items. The transition probabilities between items are inferred from the transition frequencies observed in the training set. At any time the state of a user corresponds to the last item that he consumed, and the recommendations for that user will be the k items with the highest transition probabilities from that state. In other words, if the last item consumed by a user is the item j, the k first recommendations of the Markov model will be the k items that followed most often the item j in the sequences coming from other users. This is equivalent to a bigram model in language modeling in which the words become the items.

3.3.4 FPMC. FPMC combines traditional factorization method and factorized Markov chains[17]. Factorized Markov chains are a way to estimate transition probabilities from sparse data by factorizing the transition matrix (i.e. the matrix whose element i, j is the transition probability between items i and j). FPMC factorizes two matrices simultaneously: the user-item matrix and the transition matrix, using a BPR-like loss. For a given user and a given item, the predictions are simply made by summing the similarity based on the user-item matrix factorization and based on the transition matrix factorization.

The main parameters of FPMC are the rank k of the factorizations (in principle one could use a different rank for the user-item and transition matrix factorization, but we use the same), the initial learning rate l, the learning rate decay factor α (which multiplies the learning rate after each epoch) and the regularization λ.

Table 2: Parameters values used in the experiments. We tuned each method on the validation set of Movielens, then used the same parameters in all the experiments (except if specified otherwise). See methods description for the meaning of the parameters.

Method	parameters
RNN-CCE	$k = 50$, $l = 0.1$, $M_l = 30$
RNN-Hinge	$k = 50$, $l = 0.1$, $M_l = 30$, $\beta = 6$
UserKNN	$k = 80$
BPRMF	$k = 200$ for Movielens, $k = 100$ for Netflix
FPMC	$k = 100$, $l = 0.016$, $\alpha = 0.99$, $\lambda = 0.003$
Fossil	$k = 100$, $l = 0.02$, $\alpha = 0.97$, $\lambda = 0.0025$, $\beta = 0.5$

3.3.5 Fossil. Fossil[8] is inspired by FPMC and by FISM[11]: instead of factorizing the user-item matrix, it factorizes the item-item matrix. The predicted score of an item i, for a user u is:

$$s_{ui} = b_i + \frac{1}{|\mathcal{S}_u|^\beta} \sum_{j \in \mathcal{S}_u} < \mathbf{v}_j, \mathbf{h}_i > + (\eta + \eta_u) < \mathbf{v}_{\text{last}(\mathcal{S}_u)}, \mathbf{h}_i > \quad (3)$$

Where the vector of bias $\mathbf{b}$, the $m \times k$ matrices $\mathbf{V}$ and $\mathbf{H}$, the trade-off vector η and the trade-off bias η are the parameters learned by the model, β is an hyper-parameters, and last($\mathcal{S}_u$) is the last item with which the user interacted.

The trading procedure is similar to the one of FPMC, and the parameters are the same (k, l, α, and λ), with the addition of the β parameter.

3.4 Testing Procedure

For each user of the test set, the method can base its recommendations on the first half of the user's ratings and on the model previously built on the training set (or on the training set directly, in the case of the nearest neighbors methods). Those recommendations are evaluated against the second half of the user's rating, using the metrics described in Section 3.5. The metrics are then averaged over all test users. We use $\mathcal{S}_u^T$ to represent the second half of the ratings of user u.

BPR-MF, Fossil and FPMC can only produce recommendations to user that were in the training set (as they need to built a model for each user). For that reason, those methods are trained on a larger training set than the other methods, that includes the first half of the ratings of each of the test users. This is an unfair but unfortunately unavoidable advantage for those methods.

Since BPR-MF, FPMC, Fossil and RNNs produce stochastic models, table 3 gives the average and the standard deviation of the results over ten models.

3.5 Metrics

We compare the different methods through a range of metrics designed to capture various qualities of the recommendation

systems. In the following metrics an item i is a "correct recommendation" for user u if and only if $i \in \mathcal{S}_u^t$.

- **Recall**. The usual metrics for top-N recommendation, defined as the number of correct recommendations divided by the number of unique items in $\mathcal{S}_u^t$.
- **sps**. The Short-term Prediction Success captures the ability of the method to predict the next item in the sequence. It is 1 if the next item (i.e. the first item of $\mathcal{S}_u^t$) is present in the recommendations, 0 else.
- **User coverage**. The fraction of users who received at least one correct recommendation. The average recall (and precision) hides the distribution of success among users. A high recall could still mean that many users do not receive any good recommendation. This metrics captures the generality of the method.
- **Item coverage**. The number of distinct items that were correctly recommended. It captures the capacity of the method to make diverse, successful, recommendations.
- **Blockbusters share**. The percentage of correct recommendations that are about items among the 1% most popular items in the dataset. This metric should be read knowing that the 1% most popular items constitute 8.8% of the interactions in Movielens, 22.4% in Netflix and 28.6% in RSC15.

All those metrics are computed "at 10", i.e. in a setting where the recommendation systems produces ten recommendations for each user.

3.6 Analysis

The results are shown in Table 3. The methods using the sequence information are impressively much better than the others in terms of sps. It is worth underlying the quality of those results: given ten trials (i.e. ten recommendations), the RNN-CCE is able to predict the next movie seen by 33% of the users of Movielens and 41% of the users of Netflix, while methods not based on the sequence are below 15%.

Interestingly, the methods with a high sps generally have a larger item coverage and recommend less often the most frequent items. This link between diversity of recommendation and short-term predictions will be explored in Section 6. In particular, the item coverage of the RNN-CCE is more than twice the one of the User KNN.

As a global observation, the RNN proves to be a very promising method for all considered metrics, not only on session-based datasets where it was tested by [9, 20], but also on dense datasets. It dominates the other methods in every aspects on the Movielens dataset and is only beaten in terms of recall by BPR-MF and User KNN on Netflix, where both methods rely heavily on the skewed distribution of the items frequency.

4 SHORT-TERM / LONG-TERM PROFILE

The experiments show that some methods can be good on short-term prediction (according to the sps) but bad on long-term predictions (according to the recall) and vice-versa. We believe that this is an important characteristic to consider when adopting an algorithm for a real application. For example, online shops or music radio should probably favors short-term predictions as they need to capture the temporary interest of the user, while information websites and books recommender systems might be more interested in building deep understanding of their users' long term preferences. A finer representation of the short-term/long-term profile of recommendation methods is therefore useful.

Each recommendation method is producing a ranking over all the items in the catalog. Our approach is to measure the average rank of the N next items in the user sequence. As N increases, the average rank changes smoothly from a measure of short-term predictions to a measure of long-term predictions. For example, consider a user who interacted with $2q$ items in the following order:$x_1, x_2, \ldots, x_{2q}$. We use the algorithm of interest to produce a ranking $\mathbf{r}$ over all the items, based on the first half of the sequence (from x_1 to x_q), where r_i is the rank of item i (smaller rank being better). We then compute the average rank at N as: av-r@$N = \sum_{i=1}^{N} r_{x_{q+i}}/N$ for all $N \in 1, \ldots, q$. When av-r@N is average over all users in the test set, it offers a useful representation of the quality of an algorithm on the complete short-term/long-term spectrum[4].

Figure 2 shows the short-term/long-term profile of the different CF algorithms. The only method that performs evenly on short and long term is POP, which does not build a user model at all. As expected, the three methods based on sequence prediction (MC, RNN and FPMC) are much better on short-term prediction than on long-term ones, with this tendency being marked the most for RNN. Interestingly, User KNN is also better on short-term predictions than long-term ones. This reveals that the taste and interests of users do change over time, and that it is hard to predict the interest of a user in the far future.

User KNN is still better than RNN and FPMC for long-term prediction, and Figure 2 suggests that, on Movielens at least, it becomes an interesting option when it is important to make good predictions about actions more than 20 steps in the future. Although not shown here, the choice of the objective function of the RNN with Hinge loss as a similar short-term / long-term profile than RNN-CCE.

5 IMPROVING RNNS ON LONG-TERM PREDICTIONS

As we saw, RNNs are trained specifically to perform short-term predictions and can be outperformed by other methods in terms of long-term prediction. In this section we explore three ways to increase the performances of RNNs on long-term predictions, two based on manipulating the training set and one on changing the objective function. These approaches offer a practical tuning mechanism to trade some short-term

[4]It is important to notice the difference between this use of the "@N" notation from its more common usage in section 3: here N refers to the number of items that form the ground truth, while it generally refers to the number of recommendations that the algorithm is allowed to make.

Table 3: Comparison of top-N recommendation methods on Movielens 1M, Netflix and RSC15

	Method	Metrics (@10) sps (%)	Item coverage	User coverage (%)	rec (%)	Blockbusters share (%)
Movielens	POP	5.0	50	65.6	3.62	97.73
	BPR-MF	12.44 ± 1.26	388.50 ± 14.32	82.94 ± 1.27	5.58 ± 0.12	44.08 ± 1.43
	User KNN	14.40	277	80.8	6.31	50.36
	MC	29.20	518	77.0	4.90	16.47
	FPMC	28.10 ± 0.71	614.00 ± 10.32	82.94 ± 1.45	5.70 ± 0.13	14.26 ± 0.71
	Fossil	22.46 ± 1.67	556.40 ± 39.81	83.04 ± 0.61	6.32 ± 0.14	11.22 ± 1.98
	RNN-CCE	33.45 ± 1.17	669.75 ± 15.58	87.73 ± 0.98	7.52 ± 0.14	13.91 ± 0.28
	RNN-Hinge	30.91 ± 2.15	611.56 ± 45.52	88.40 ± 0.82	7.72 ± 0.17	16.40 ± 1.92
Netflix	POP	5.92	54	68.2	4.65	100
	BPR-MF	9.93 ± 0.70	591.60 ± 16.36	79.19 ± 0.71	6.88 ± 0.09	73.84 ± 1.32
	User KNN	13.04	383	80.84	8.49	79.86
	MC	32.50	594	64.50	3.17	53.02
	FPMC	26.38 ± 0.80	542.56 ± 6.87	70.98 ± 0.46	3.75 ± 0.08	64.53 ± 0.51
	Fossil	24.26 ± 0.64	852.78 ± 28.32	76.93 ± 0.77	5.17 ± 0.12	40.07 ± 1.78
	RNN-CCE	41.00 ± 0.99	845.88 ± 26.00	82.43 ± 0.48	6.37 ± 0.16	53.15 ± 1.39
	RNN-Hinge	32.28 ± 0.87	935.20 ± 38.23	78.92 ± 0.65	5.76 ± 0.24	39.29 ± 0.43
RSC15	POP	1.24	11	2.24	1.2	100
	MC	39.50	2945	46.55	32.76	33.33
	RNN-CCE	52.78 ± 0.08	3536.67 ± 4.99	59.34 ± 0.10	44.52 ± 0.06	33.14 ± 0.17
	RNN-Hinge	35.12 ± 0.27	2374.33 ± 21.93	43.18 ± 0.16	30.24 ± 0.23	33.76 ± 0.30

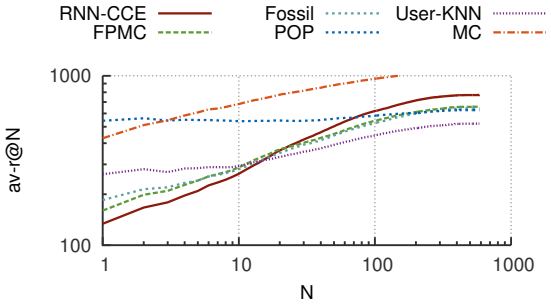


Figure 2: Short-term/long-term profile of RNN-CCE, User KNN, FPMC, Fossil, MC and POP on Movielens.

accuracy for more long-term accuracy, making the RNN a more versatile solution for collaborative filtering.

The three approaches explored in the following are to randomly alter the training sequences with either a dropout or a shuffling mechanism, or to train the RNN to predict the n next items (rather the *the* next one).

5.1 Dropout

The dropout mechanism is well known in deep learning. It consists in randomly deactivating some neurons during training in order to improve the robustness of the model [19]. Our concept of dropout is slightly different, and consists in randomly removing some items from a training sequence, before using it to train the RNN. More precisely, each time a user is drawn from the dataset, each of the items in its sequence of interactions is deleted with a probability p, and that altered sequence is used to train the RNN. This means that the RNN is often trained to predict not the next item in the sequence, but actually the second or third next item.

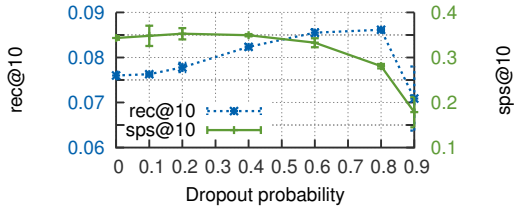
Figure 3a shows the influence of the dropout probability on the sps and recall. As expected, when the dropout probability increases the long-term predictions improve while the short-term ones decline. It is nevertheless impressive that the recall still increases with very high (80%) dropout.

Interestingly, the models seem to be overall improved with small dropout probabilities, with both the sps and recall increasing, which suggest that, as observed in many other deep learning methods, a little bit of dropout is always helpful.

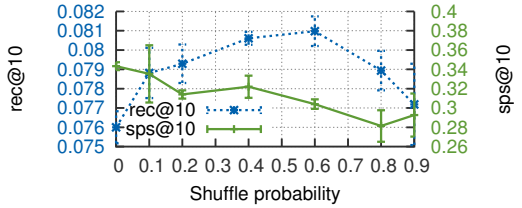
5.2 Shuffle Sequences

Our second approach is similar to dropout in spirit: the introduction of noise into the training sequences can affect short patterns more than long-term ones, and the resulting model will therefore be more attuned to those long-term patterns. Instead of dropout, the noise we use here is a slight shuffling of the training sequences. The difficulty is to shuffle in a way that will destroy very short patterns without completely loosing the sequential aspect of the data. Our solution is that each item in the sequence can be swapped with another item with a probability p, and it is swapped with the item in position $i + r$ in the sequence, where i is the position of the current item and r is a normally distributed offset: $r \sim round(N(0, \sigma))$.

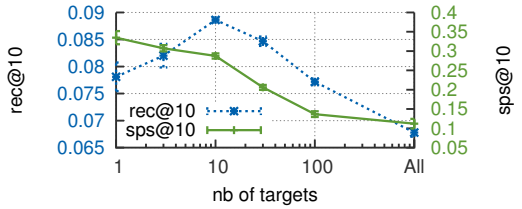
Figure 3b shows the influence of the shuffle probability on the sps and recall, with $\sigma = 3$. The impact is smaller than with dropout, but nonetheless clear. With a shuffle



(a) Dropout



(b) Shuffling



(c) Multi-targets, hinge loss

Figure 3: Influence of dropout, shuffle and multiple targets on short and long-term predictions on Movielens.

probability between 0.2 and 0.6 the recall is improved without decreasing much the sps.

5.3 Multiple Targets

The objective functions of RNNs are designed to build models good at short-term predictions. Indeed, CCE is comparing the output value of the true next item to other items, and pushing the model to give to the true next item a higher value than to the others. The hinge loss on the other hand does not work by comparing items between themselves, but by comparing the output value of each item against a fixed threshold. With the hinge loss, the recommendation problem is treated as m binary classification problem (one for each item) where the question is whether or not it should be recommended to the user. This allows to select multiple items as positive targets, as they are all treated independently, and it can be used to produce a loss function that favors long-term rather than short term predictions.

Many ways can be imagined to select which items should be positive or negative targets. In the following we used the t next items in the sequence as the positive items. If $t = 1$ it is equivalent to the experiments in Section 3, if $t = 10$ the RNN is trained to produce an output value higher than 1 for the 10 next items in the sequence, and lower than 0 for the other items.

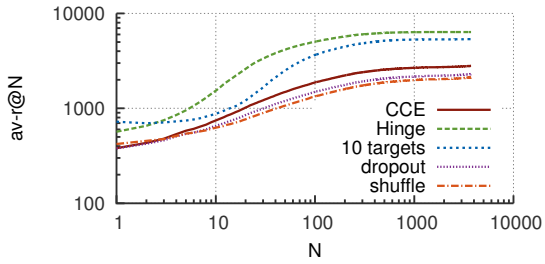


Figure 4: Impact of the different strategies to improve long-term recommendations on the short-term/long-term profile of RNNs on Netflix

Figure 3c shows the influence of t on the sps and recall on Movielens. When t increases, the sps decreases as expected, and the recall increases at first and then decreases. The fact that the recall decreases when the number of targets becomes too large might be due to the inability of the RNN to master an increasingly difficult problem. Indeed, the set of possible solutions dramatically increases when you go from predicting one item to predicting a hundred, and the training set might be too small to learn a good model.

5.4 Comparison of the Three Approaches

We compare the three strategies for improving long-term recommendations on Netflix. We fixed the dropout probability and the shuffle probability to 0.6 based on Figure 3a and 3b. We also tested the combination of dropout and shuffling. Table 4 shows the impact of the different strategies in terms of the metrics described in Section 3.5, and Figure 4 shows how those strategies affect the short-term/long-term profile of the methods.

Each strategy significantly improves the recall, with the combination of dropout and shuffling working the best. Although the User KNN still has a higher recall, the quality of its recommendations is questionable, as they are much less diverse than those of the RNNs. Interestingly, RNN-Hinge with 10 targets is much more diverse than the other methods (in terms of item coverage and blockbusters share), and although it is worse than RNN-CCE in terms of sps and recall, its high diversity might make it the preferred method.

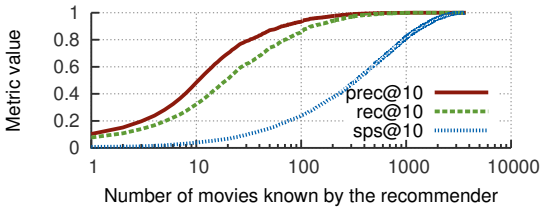
6 SEQUENCE PREDICTION FAVORS DIVERSITY IN RECOMMENDATIONS

As observed in Section 3, methods with a high sps tend to also have more diverse recommendations (in terms of item coverage). We argue that this is because it is possible to produce high recall using only very frequent items, but reaching a high sps requires to use less frequent items. In other words, optimizing short-term predictions puts more pressure on the capacity of the model to produce diverse recommendations.

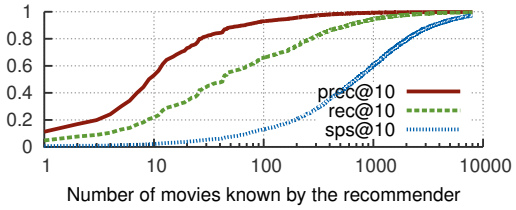
The reasoning is the following: the correct predictions in terms of sps are a subset of the correct predictions in terms

Table 4: Performances on Netflix of the RNN trained with dropout, shuffling or multiple targets. The results of the User KNN, RNN-CCE and RNN-Hinge where copied from Table 3 for easier comparison.

	Metrics (@10)				
Method	sps (%)	Item coverage	User coverage (%)	rec (%)	Blockbusters share (%)
User KNN	13.04	383	80.84	8.49	79.86
RNN-Hinge	32.28 ± 0.87	935.20 ± 38.23	78.92 ± 0.65	5.76 ± 0.24	39.29 ± 0.43
RNN-Hinge 10 targets	11.51 ± 0.48	1013.00 ± 7.07	83.05 ± 0.45	6.63 ± 0.05	24.57 ± 0.25
RNN-CCE	41.00 ± 0.99	845.88 ± 26.00	82.43 ± 0.48	6.37 ± 0.16	53.15 ± 1.39
RNN-CCE + dropout	38.01 ± 0.77	868.75 ± 6.94	83.38 ± 0.76	6.82 ± 0.09	52.29 ± 0.36
RNN-CCE + shuffle	33.77 ± 0.58	851.00 ± 13.06	85.19 ± 0.13	6.96 ± 0.12	54.58 ± 0.54
RNN-CCE + dropout & shuffle	32.76 ± 0.61	866.67 ± 8.01	85.02 ± 0.41	7.25 ± 0.05	53.32 ± 0.56



(a) Movielens 1M



(b) Netflix

Figure 5: Evolution of the rec@10, prec@10 and sps@10 for a recommendation that can only recommend the top t most popular items. The maximum value of the rec@10 is actually much smaller than 1 but we normalized the recall here by dividing it by its maximum value to make the graph more readable.

of recall; because of that, any given item will be a correct prediction for more users in terms of long term predictions than in terms of short term predictions; the result is that it takes fewer items to make correct long term predictions for a given percentage of users than it takes to make correct short term predictions.

We illustrate that with a simple experiment. Consider an oracle (i.e. a perfect recommendation system) that can only recommend items within the t most popular ones; Figure 5 shows how the rec@10, prec@10 and sps@10 increase as t increases on two real datasets[5]. The rec@10 and prec@10 converge very fast, they reach 80% of their maximum value with a small fraction of the items, and each new item brings only a marginal improvement. The sps@10 on the other hand has a much slower convergence and requires therefore a higher diversity of recommendation to reach 80% of its maximum value; we therefore expect that training a recommendation system to optimize the sps rather than the recall or precision will force it to produce more diverse recommendations.

[5]prec@10 stands for "precision at 10" which is defined as the fraction of correct recommendations in the first 10 recommendations made by the algorithm.

7 CONCLUSION

Our results show that recurrent neural networks are a powerful tool for collaborative filtering, even outside of the sparse session-based settings where it was first introduced. We achieved the best results using the categorical cross-entropy objective function.

RNNs performed especially well on short-term recommendations, but we also observed that adding noise to the training sequence (such as dropout and shuffling) improves its success on long-term recommendations. Moreover, the recommendations produced by the RNN are more diverse and depends less on the most frequent items than those of factorization-based and neighborhood-based approaches. Interestingly, producing diverse recommendations might be linked to the performances in short-term predictions, making the methods based on sequence predictions inherently more diverse.

Acknowledgments. R. Devooght is supported by the Belgian Fonds pour la Recherche dans l'Industrie et l'Agriculture (FRIA, 1.E041.14). Some of the results were obtained using a Tesla K40 donated by the NVIDIA Corporation.

REFERENCES

[1] E. Bernhardsson. Recurrent neural networks for collaborative filtering, 2014. [Online; accessed 20-Mai-2016].
[2] R. I. Brafman, D. Heckerman, and G. Shani. Recommendation as a stochastic sequential decision problem. In *ICAPS*, pages 164–173, 2003.
[3] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.
[4] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
[5] Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. MyMediaLite: A free recommender system library. In *Proceedings of the 5th ACM Conference on Recommender Systems (RecSys 2011)*, 2011.
[6] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016. http://www.deeplearningbook.org.

[7] E. Guàrdia-Sebaoun, V. Guigue, and P. Gallinari. Latent trajectory modeling: A light and efficient way to introduce time in recommender systems. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 281–284. ACM, 2015.

[8] R. He and J. McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *Proceedings of ICDM'16*, 2016.

[9] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. In *Proceedings of ICLR'16*, 2016.

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

[11] S. Kabbur, X. Ning, and G. Karypis. Fism: factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 659–667. ACM, 2013.

[12] Y. Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010.

[13] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, volume 2, page 3, 2010.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3111–3119, 2013.

[15] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa. Using sequential and non-sequential patterns in predictive web usage mining tasks. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 669–672. IEEE, 2002.

[16] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

[17] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 811–820. ACM, 2010.

[18] G. Shani, R. I. Brafman, and D. Heckerman. An mdp-based recommender system. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 453–460. Morgan Kaufmann Publishers Inc., 2002.

[19] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[20] Y. K. Tan, X. Xu, and Y. Liu. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 17–22. ACM, 2016.

[21] A. Zimdars, D. M. Chickering, and C. Meek. Using temporal data for making recommendations. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 580–588. Morgan Kaufmann Publishers Inc., 2001.