Memory Augmented Neural Model for Incremental Session-based Recommendation

Fei Mi and **Boi Faltings**

Artificial Intelligence Laboratory, École Polytechnique Fédérale de Lausanne (EPFL)
fei.mi@epfl.ch, boi.faltings@epfl.ch

Abstract

Increasing concerns with privacy have stimulated interests in *Session-based Recommendation* (SR) using no personal data other than what is observed in the current browser session. Existing methods are evaluated in static settings which rarely occur in real-world applications. To better address the dynamic nature of *SR* tasks, we study an *incremental SR* scenario, where new items and preferences appear continuously. We show that existing neural recommenders can be used in *incremental SR* scenarios with *small* incremental updates to alleviate computation overhead and catastrophic forgetting. More importantly, we propose a general framework called **M**emory **A**ugmented **N**eural model (MAN). MAN augments a base neural recommender with a continuously queried and updated *nonparametric* memory, and the predictions from the neural and the memory components are combined through another lightweight gating network. We empirically show that MAN is well-suited for the incremental *SR* task, and it consistently outperforms state-of-the-art neural and nonparametric methods. We analyze the results and demonstrate that it is particularly good at incrementally learning preferences on new and infrequent items.

1 Introduction

Due to new privacy regulations that prohibit building user preference models from historical user data, it is getting important to utilize short-term dynamic preferences within a browser session. *Session-based Recommendation* (SR) is therefore increasingly used in interactive online computing systems. The goal of *SR* is to make recommendations based on user behavior obtained in short web browser sessions, and the task is to predict the users' next actions, such as clicks, based on previous actions in the same session.

To better address the *dynamic* nature of *SR* tasks, we study it from an incremental learning perspective, referred to as *Incremental Session-based Recommendation*. In this setting, new items and preferences appear incrementally, and models need to incorporate the new preferences incrementally while preserving old ones that are still useful. The setup requires a recommender to be incrementally updated with data observed during the last time period and evaluated on the events in the following time period. We summarize three main challenges of incremental *SR* scenarios: (i). *catastrophic forgetting* [McCloskey and Cohen, 1989]: incorporating new patterns requires additional training and often reduces performance on old patterns. (ii). *computation efficiency*: models need to be efficient as they need to be frequently updated with new data. (iii). *sample efficiency*: the number of observations on new items and patterns is often small such that models need to capture them quickly with limited observations.

Recently proposed neural approaches [Hidasi *et al.*, 2016; Li *et al.*, 2017; Hidasi and Karatzoglou, 2018; Liu *et al.*, 2018; Sun *et al.*, 2019] have shown great success in a static setting where the set of items and the distribution of preferences during testing are assumed to be the same as in the training phase. However, this assumption rarely holds in real-world recommendation applications. As our first contribution, we show that neural recommenders can also be efficiently and effectively used for incremental *SR* scenarios by applying *small* incremental updates. To the best of our knowledge, this is the first study of neural models for *SR* from an incremental learning perspective. We found that incrementally updating neural models using a single pass with a small learning rate through new data helps to capture new patterns incrementally. However, using large learning rates (or equivalently multiple passes) degrades performance badly due to overfitting new patterns and forgetting old ones.

Our main contribution is to propose a method called **M**emory **A**ugmented **N**eural model (MAN) inspired by a framework proposed for language model [Merity *et al.*, 2017; Grave *et al.*, 2017a], neural machine translation [Tu *et al.*, 2018], and image recognition [Orhan, 2018] tasks. MAN augments a neural recommender with a *nonparametric* memory to capture new items and preferences incrementally. The predictions of neural and memory components are combined by a lightweight gating network. MAN is *agnostic* to the neural model as long as it learns meaningful sequence representations, therefore, it can be easily and broadly applied to various neural recommenders. The nonparametric memory component of MAN helps to deal with all three challenges of incremental *SR* mentioned above. **First**, it achieves a long-term memory to remember long histories of observations to mitigate catastrophic forgetting. **Second**, it is very efficient to up-

date because it is not trained jointly with the base neural recommender. **Third**, the nonparametric nature by itself helps to better capture new patterns with a smaller number of observations to address the third challenge of sample efficiency. Through extensive experiments, we show that MAN boosts the performance of different neural methods and achieves state-of-the-art. We also demonstrate that it is particularly good at capturing preferences on new and infrequent items.

2 Related Work

Incremental learning for recommendation is an important topic for practical recommendation systems where new users, items, and interactions are continuously streaming in. For the standard recommendation task of predicting ratings or clicks, different learners have been studied based on matrix factorization (MF) with online updating, including [He *et al.*, 2016]. For sequential recommendation tasks, the interaction matrix to be decomposed is constructed from sequential user feedback. [Rendle *et al.*, 2010] proposes FPMC to factorize the transitions in Markov chains with low-rank representation. Later, [He and McAuley, 2016] proposes to FOSSIL with factorized Markov chains to incorporate sequential information. Recently, SMF is proposed by [Ludewig and Jannach, 2018] for *SR* using session latent vectors. However, these MF-based methods are expensive to train. For example, [Li *et al.*, 2017; Wu *et al.*, 2019] reported that even 120GB memory is not enough to train FPMC. Therefore, they are in principle not suitable for incremental settings.

Session-based recommendation can be formulated as a sequence learning problem to predict the user's sequential behavior. It can be solved by recurrent neural networks (RNNs). The first work (GRU4Rec, [Hidasi *et al.*, 2016]) uses a gated recurrent unit (GRU) to learn session representations from previous clicks and predict the next click. Based on GRU4Rec, [Hidasi and Karatzoglou, 2018] proposes new ranking losses on relevant sessions, and [Tan *et al.*, 2016] proposes to augment training data. NARM [Li *et al.*, 2017] augments GRU4Rec with a bilinear decoder and an additional attention operation to pay attention to specific parts of the sequence. Base on NARM, [Liu *et al.*, 2018] proposes STAMP to model users' general and short-term interests using two separate attention operations, and [Ren *et al.*, 2019b] proposes RepeatNet to use an additional repeat decoder based on an attention mechanism to predict repetitive actions in a session. Recently, [Wu *et al.*, 2019] uses graph attention to capture complex transitions of items. Motivated by the recent success of *Tansformer* [Vaswani *et al.*, 2017] and *BERT* [Devlin *et al.*, 2018] for language model tasks, [Kang and McAuley, 2018] proposed SASRec using the the *Transformer* operation and [Sun *et al.*, 2019] proposed BERT4Rec using the training scheme of *BERT* to model bi-directional information through Cloze tasks. Despite the broad exploration and success, these methods are all studied in a static *SR* scenario without considering new items and patterns that appear continuously.

Nonparametric methods [Härdle and Linton, 1994] are ideal for incremental learning tasks thanks to their computation efficiency. The amount of information they capture increases as the number of observations grows. Simple item-

based collaborative filtering methods using nearest neighbors have been proven to be effective and are widely employed in industry. Markov models [Shani *et al.*, 2005] also support updating the transition probabilities incrementally. However, the Markov assumption is often too strong, and it limits the recommendation performance. Recently, [Jannach and Ludewig, 2017] proposed SKNN to compare the entire current session with historical sessions in the training data. They show that SKNN is very efficient and achieves strong results. Lately, variations [Ludewig and Jannach, 2018; Garg *et al.*, 2019] of SKNN have been proposed to consider the position of items in a session or the timestamp of a past session. [Mi and Faltings, 2017; 2018] built a nonparametric recommender based on a structure called *context tree* to model suffixes of a sequence. These simple nonparametric methods are independent from the neural approaches. In this paper, we combine the edges of both neural and nonparametric models for incremental *SR* scenarios.

Memory-augmented neural models [Graves *et al.*, 2014] have been well-known for the purpose of maintaining long-term memory, which is also recently explored for sequential recommendation tasks [Chen *et al.*, 2018; Ren *et al.*, 2019a]. However, the goal of maintaining long-term memory is not the central concern in incremental *SR* scenarios. This category of memory-augmented neural models differ from ours in the sense that their memory components are *parametric* as they need to be trained jointly with the rest of the model. Therefore, they are not suitable be updated frequently without catastrophic forgetting in incremental *SR* scenarios.

Our work is largely inspired by a recently proposed *nonparametric* memory module that is not trained jointly with neural models. [Grave *et al.*, 2017b; Merity *et al.*, 2017] introduce a cache to augment RNNs for language modeling task. They later improve this cache to unbounded size [Grave *et al.*, 2017a] and achieve significant performance improvement. A similar memory module is also proposed for neural machine translation tasks [Tu *et al.*, 2018] and image recognition tasks [Orhan, 2018].

3 Model

Background. session-based recommendation, an *event* is a *click* on an item and the task is to predict the next event based on a sequence of events in the current web-browser session. Suppose $\mathbf{Y}$ is the set of all candidate items, and the size is n. Existing neural session-based recommenders typically contain two modules: an **encoder** $g_\theta(\mathbf{x}_{1:t})$ to compute a compact *sequence representation* $\mathbf{c}_t$ of the sequence of events $\mathbf{x}_{1:t}$ until time t, and a **decoder** $f_\omega(\mathbf{c}_t)$ to compute an output distribution $P^N \in \mathbb{R}^n$ to predict the next event. Recurrent neural networks (RNNs) and fully-connected layers are common choices for encoder and decoder respectively. In our later experiments, we included models that use different types of encoders and decoders. Our MAN framework is agnostic to the neural recommender, therefore, readers can use many other neural architectures with an encoder-decoder structure.

Next, we present the **M**emory **A**ugmented **N**eural recommender (MAN) to augment a neural session-based recommender with a cache-like nonparametric memory to incre-

mentally incorporate new items and preferences. We first introduce the architecture of the cache-like memory. Then we describe how to use it to generate nonparametric memory predictions, and how to merge memory and neural predictions through a lightweight gating network.

3.1 Nonparametric Memory Structure

Our memory $\mathbf{M}$ is an array of slots in the form of (key, value) pairs [Miller *et al.*, 2016] in Eq.(1). $\mathbf{M}$ is queried by keys and returns corresponding values. We define the keys to be input sequence representations $\mathbf{c}_m$ computed by the neural encoder, and the values to be the corresponding label y_m of the next event. To scale to a large set of observations and long histories in practical recommendation scenarios, we do not restrict the size of $\mathbf{M}$ but store all pairs $(\mathbf{c}_m, y_m)$ from previous events.

$$\mathbf{M} = \{(\mathbf{c}_1, y_1), \ldots, (\mathbf{c}_m, y_m), \ldots\} \quad (1)$$

3.2 Memory Prediction

To support efficient incremental learning, the memory module is *not* trained jointly with the neural recommender. Instead, it directly predicts the next event by computing a probability distribution using entries stored in $\mathbf{M}$. For an input sequence $\mathbf{x}_{1:t}$, we first match the current sequence representation $\mathbf{c}_t = g_\theta(\mathbf{x}_{1:t})$ against $\mathbf{M}$ to retrieve K nearest neighbors $\mathcal{N}(\mathbf{c}_t) = \{(\mathbf{c}_k, y_k, d_k)\}_{k=1}^{K}$ of $\mathbf{c}_t$, where d_k is the Euclidean distance between the sequence representation $\mathbf{c}_k$ of the k-th neighbor and $\mathbf{c}_t$. Then, we use the K nearest neighbors to compute a non-parametric memory prediction $P^M \in \mathbb{R}^n$ using a variable kernel density estimation by:

$$P^M(y_i) \propto \sum_{k=1}^{K} \delta(y_i = y_k) \mathbf{K}\left(\frac{d_k}{d^\star}\right) \quad (2)$$

where $P^M(y_i)$ is the probability on an item $y_i \in \mathbf{Y}$, $\delta(y_t = y_k)$ is Kronecker delta which equals one when the equality holds and zero otherwise, $\mathbf{K}$ is a Gaussian kernel ($\mathbf{K}(x) = exp(-x^2/2)$), and $d^\star$ is the Euclidean distance between $\mathbf{c}_t$ and its closest neighbor in $\mathcal{N}(\mathbf{c}_t)$. P^M only assigns non-zero probability to at most K (number of neighbors) items because the probabilities assigned to items that do not appear in the nearest neighbors are zero. As a result, P^M is a very *sparse* distribution as a mixture of the labels in $\mathcal{N}(\mathbf{c}_t)$ weighted by their similarities to $\mathbf{c}_t$. To capture new preference patterns incrementally, $\mathbf{M}$ is queried and updated incrementally during the testing phase such that P^M is up-to-date.

3.3 Gating Network to Combine Memory and Neural Predictions

The neural prediction P^N mainly captures static old preference patterns while the nonparametric memory prediction P^M can model infrequent and new preference patterns incrementally. To flexibly work with both scenarios, these two predictions are combined. A simple way proposed by [Grave *et al.*, 2017a; Orhan, 2018] is linearly interpolating them with a fixed weight, and we later call this version "`MAN-Shallow`" in our experiments in Section 4.5.

To better merge these two predictions at different sequential contexts, we propose to use a lightweight **gating network**

[Bakhtin *et al.*, 2018] $w(\mathbf{c}_t)$ to learn the mixing coefficient as a function of the sequence representation $\mathbf{c}_t$. We use a lightweight fully connected neural network defined in Eq.(3) with a single hidden layer of 100 hidden units, *tanh* as hidden layer activation function, and a Sigmoid function at the output layer.

$$w(\mathbf{c}_t) = \sigma(\mathbf{W}_o * tanh(\mathbf{W}_h \mathbf{c}_t + \mathbf{b}_h) + b_o) \quad (3)$$

The output of $w(\mathbf{c}_t)$ is a scalar between 0 and 1 that measures the relative importance of P^N, while the other $1 - w(\mathbf{c}_t)$ fraction is multiplied to P^M. The final prediction distribution P^{MAN} is an learning interpolation of P^M and P^N weighted by the output of $w(\mathbf{c}_t)$ computed in Eq.(4) .

$$P^{MAN}(y_t) = w(\mathbf{c}_t)P^N(y_t) + (1 - w(\mathbf{c}_t))P^M(y_t) \quad (4)$$

The gating network is trained with cross-entropy loss using P^{MAN} as predictive distribution *after* the normal training of the neural model with both P^N and P^M are fixed. The gradients are not computed for the large number parameters of the neural model to avoid computation overhead and interference with the trained neural model. Inspired by [Bakhtin *et al.*, 2018], it is trained using only validation data. The idea is to train it using data not seen during the pre-training phase to better predict new preferences that might appear in incremental *SR* scenarios. We randomly select 90% validation data for training and the remaining 10% for early stopping. We compared this setup with training using the whole training set, and we found that our setup achieves better performance while being much more efficient.

3.4 Efficient Large-scale Nearest Neighbor Computation

As the number of events in practical *SR* scenarios is huge and we do not restrict the size of $\mathbf{M}$, computing nearest neighbors frequently to generate P^M can be expensive. We apply a scalable retrieval method used by [Grave *et al.*, 2017a]. To avoid exhaustive search, an *inverted file table* $\mathbf{T}$ is maintained. Keys in $\mathbf{M}$ are first clustered to a set of clusters using k-means, then all keys in $\mathbf{M}$ can be associated with one centroid. When we query $\mathbf{c}_t$ in $\mathbf{M}$, it is searched by firstly matching to a set of centroids to get the closest cluster and then the set of keys in this cluster.

The clustered memory supports efficient querying, yet it is memory consuming because each key in $\mathbf{M}$ needs to be stored. This can be greatly reduced by Product Quantization (PQ [Jegou *et al.*, 2010]) that quantizes a vector by parts (sub-quantizers), and it does not directly store the vector but its residual, i.e., the difference between the vector and its associated centroids. We use 2^8 centroids, 8 sub-quantizers per sequence representation, and 8 bits allocated per sub-quantizer, then we only need the size of 16 (quantization code + centroid id = 8 + 8) bytes per vector. Therefore, a million sequence representations can be stored with only 16 Mb memory. With an inverted table and PQ, we have a fast approximate nearest neighbor retrieval method with a low memory footprint. We use the FAISS [1] open-source library that also supports GPU acceleration for implementation.
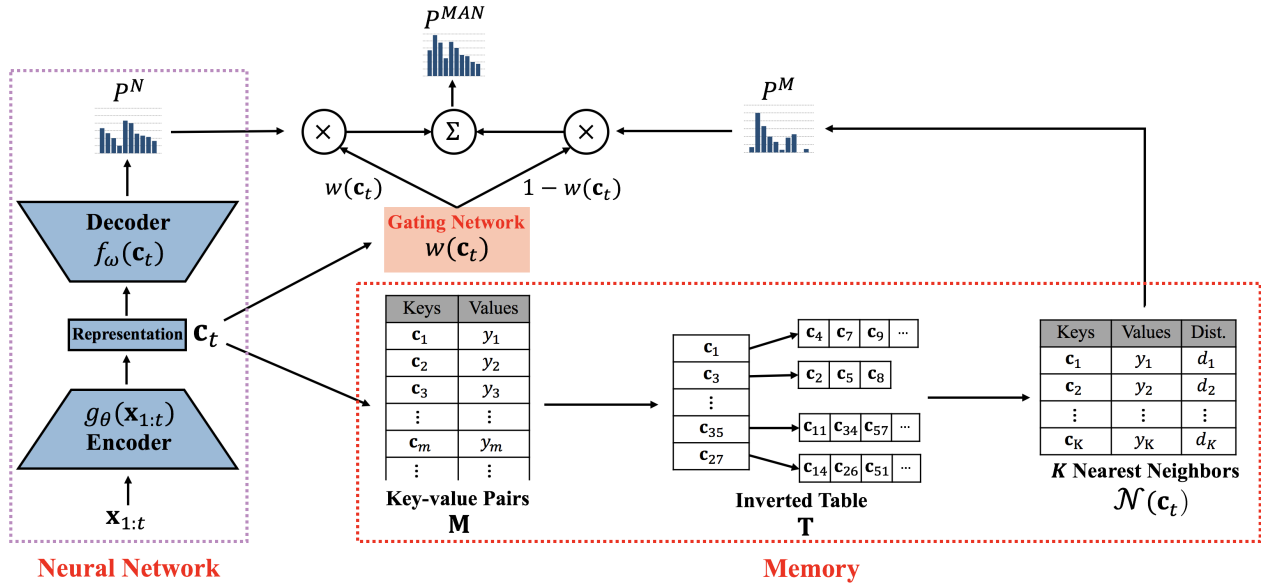
---

[1] https://github.com/facebookresearch/faiss

Figure 1: Computation pipeline of **M**emory **A**ugmented **N**eural recommender (MAN). Predictions from a neural network are augmented by predictions from a memory module through a gating network.

Algorithm 1 **M**emory **A**ugmented **N**eural Recommender

1: **procedure** TRAIN($D_{train}, D_{valid}, \mathbf{M}$)
2:     Train g_θ, f_ω w.r.t. D_{train}
3:     **for** $(\mathbf{x}_{1:t}, y_t) \in D_{train}$ **do**
4:         Compute $\mathbf{c}_t = g_\theta(\mathbf{x}_{1:t})$; store $(\mathbf{c}_t, y_t)$ to $\mathbf{M}$
5:     **end for**
6:     Build the inverted table $\mathbf{T}$ for $\mathbf{M}$
7:     Fix g_θ, f_ω, and train $w(\mathbf{c}_t)$ w.r.t. D_{valid}
8: **end procedure**
9: **procedure** TEST($D_{test}, \mathbf{M}$)
10:     **for** $(\mathbf{x}_{1:t}, y_t) \in D_{test}$ **do**
11:         Compute $\mathbf{c}_t = g_\theta(\mathbf{x}_{1:t})$, and $P^N = f_\omega(\mathbf{c}_t)$
12:         Query $\mathbf{M}$ and $\mathbf{T}$ to retrieve $\mathcal{N}(\mathbf{c}_t)$
13:         Compute P^M by Eq.(2), P^{MAN} by Eq.(4)
14:         Update $\mathbf{M}$ and $\mathbf{T}$ with $(\mathbf{c}_t, y_t)$
15:     **end for**
16: **end procedure**

3.5 Overall MAN Algorithm

The computation pipeline and algorithm of MAN presented in Figure 1 and Algorithm 1. Next, we describe the training and testing procedures of MAN in detail, and also analyzed its computation efficiency.

Training procedure. MAN first trains the neural encoder and decoder on the training set D_{train}. Then, it computes sequence representations for all training data and stores them with corresponding labels to $\mathbf{M}$. Afterwards, the clustered memory of the inverted table $\mathbf{T}$ is built with entries in $\mathbf{M}$. Lastly, The gating network is trained on validation set D_{valid}.

Testing procedure. The sequence representation $\mathbf{c}_t$ and the neural prediction P^N is first computed. Then, $\mathbf{M}$ and $\mathbf{T}$ are queried with $\mathbf{c}_t$ to retrieve K nearest neighbors $\mathcal{N}(\mathbf{c}_t)$ to compute P^M. To generate the final recommendation P^{MAN},

P^M is merged with P^N weighted by the output of the gating network. Lastly, $\mathbf{M}$ and $\mathbf{T}$ are incrementally updated with the new testing pair $(\mathbf{c}_t, y_t)$. During testing, the clustered memory in $\mathbf{T}$ is not updated for the purpose of computation efficiency. Running k-means to update the clustered memory on huge data sets with large dimensions is computationally intensive. Therefore, we need to decide when and how to update the clustering algorithm, and there will be a trade-off between the performance benefits and the computation overhead. Studies on this part are left for interesting future work.

Computation efficiency analysis. During training, the additional training procedures of MAN on top of the regular neural recommender training are efficient. (i) Sequence representations of training data can be obtained directly from the last regular training epoch of the neural recommender, therefore, no computation overhead is injected at this step (line 3-5). 2). (ii) Building the clustered memory and the inverted table for entries in $\mathbf{M}$ (line 6) is also fast with the FAISS library. (iii) Training the lightweight gating network using only validation split (line 7) is much more efficient than the regular training of the base neural recommender. During testing, querying the memory to retrieve nearest neighbors can be done very efficiently supported by FAISS; a forward computation through the lightweight gating network is also efficient.

4 Experiments and Analyses

4.1 Datasets

YOOCHOOSE: This is a public dataset for RecSys Challenge 2015.[2] It contains click-streams on an e-commerce site over 6 months. Events in the last week are tested. Following [Tan *et al.*, 2016; Li *et al.*, 2017], we use the latest 1/4 training sequences because YOOCHOOSE is very large.

---

[2]http://2015.recsyschallenge.com/challenge.html

	Training Data			Validation Data		Testing Data		
	Events	Sessions	Items	Events	Sessions	Events	Sessions	New Events
YOOCHOOSE	6,245,412	1,535,693	22,594	693,935	170,633	748,269	178,920	8.6%
DIGINETICA	636,506	130,994	42,294	70,723	14,555	286,254	59,240	3.3%

Table 1: Statistics of two datasets. The last column indicates the percentage of testing events that involve new items not in the training set.

DIGINETICA: This dataset contains click-streams data on another e-commerce site over a span of 5 months for CIKM Cup 2016.[3] Events in the last four weeks are tested.

Items that appear less than five times, and sessions of length shorter than two or longer than 20 are filtered out. Statistics of the two datasets after pruning are summarized in Table 1, and the last 10% of the training data based on time is used as the validation set. Different from previous static settings that remove items not in the training phase from test sets, our test sets for the incremental *SR* task include events on new items that appear only during the testing phase. The last column indicates the percentage of events in the test data that involve items not part of the training data.

4.2 Evaluation Metrics

HR@k: Average hit rate when the desired item is amongst the top-k recommended items. It can be interpreted as precision [Liu *et al.*, 2018; Wu *et al.*, 2019] or recall [Hidasi *et al.*, 2016; Li *et al.*, 2017; Jannach and Ludewig, 2017] because we predict the immediate next event.

MRR@k: HR@k does not consider the order of the items recommended. MRR@k measures the mean reciprocal ranks of the desired items in top-k recommended items.

4.3 Models and Training Details

A wide range of neural and non-paramtric session-based recommenders are compared:

Item-KNN: This simple baseline recommends top items similar to the single last item in the current session based on co-occurrence statistics of two items in other sessions.

(S)-SKNN: Instead of considering only the last item, SKNN [Jannach and Ludewig, 2017] compares all items in the current session with items in other sessions. S-SKNN [Ludewig and Jannach, 2018] is an improved version that assigns more weights to items that appear later in a session.

CT [Mi and Faltings, 2018]: It builds a nonparametric recommender based on a structure called Context Tree to model suffixes of a sequence.

GRU4Rec [Hidasi and Karatzoglou, 2018]: It uses an RNN with GRU as encoder. It also uses specialized ranking-based losses computed w.r.t. the most relevant samples, which perform better than their initial version [Hidasi *et al.*, 2016].

NARM [Li *et al.*, 2017]: It improves the encoder of GRU4Rec with an item-level attention and replaces the decoder by a bilinear decoding scheme.

MAN (proposed): The method proposed in this paper. Two versions (MAN-GRU4Rec and MAN-NARM) are tested using

η	1e-2	5e-3	1e-3	5e-4	1e-4	5e-5	0
YOOCHOOSE							
NARM	0.389	0.422	0.460	**0.463**	0.447	0.440	0.420
DIGINETICA							
NARM	0.235	0.255	0.315	0.338	**0.355**	0.350	0.324

Table 2: HR@5 with different learning rate η to update NARM incrementally. NARM is fixed during testing when $\eta = 0$.

GRU4Rec and NARM as base neural models respectively. Unless mentioned specifically, MAN is based on NARM. [4]

During training, hidden layer size of GRU4Rec and NARM is set to 100, and the item embedding size of NARM is set to 50; the batch size is set to 512 and 30 epochs are trained for both models. Hyper-parameters of different models are tuned on validation splits to maximize HR@5. The number of nearest neighbors of MAN is set to 50 for YOOCHOOSE and 100 for DIGINETICA. During testing, we update different neural models and MAN incrementally using *a single gradient descent step* as every batch of 100 events are tested. Learning rates for neural models are 5e-4 and 1e-4 for YOOCHOOSE and DIGINETICA, and the learning rate to update the gating network is 1e-3. Other pure nonparametric methods (Item-KNN, (S)-SKNN, CT) are also incrementally updated as every batch of 100 events is tested.

4.4 Experiment Results

In this section, we first study the effect of using different learning rates to update neural models incrementally. Then, we analyze the overall performance of different methods and plot the incremental performance of two representative methods as more events are tested.

Effect of incremental learning rate. Before we proceed to our main results, we first highlight that the degree of updates is important when neural models are updated incrementally. As an example, we show results of using different learning rates to update NARM in Table 2. We can see that using large learning rates [5] to incrementally update NARM *degrades* performance severely, while using relatively small learning rates outperforms the version when NARM is fixed during testing ($\eta = 0$). We believe that large learning rates cause the model to overfit new patterns while catastrophically forgetting old

---

[3]http://cikm2016.cs.iupui.edu/cikm-cup

[4]In this paper, we chose the two most representative base neural models (GRU4Rec and NARM) to evaluate MAN. As MAN is agnostic to the neural model, the exploration of using MAN on top of other neural recommenders is left for future work.

[5]Using multiple incremental training epochs has a similar effect as using large earning rates in one epoch.

	YOOCHOOSE				DIGINETICA			
	HR@5	MRR@5	HR@20	MRR@20	HR@5	MRR@5	HR@20	MRR@20
Item-KNN	0.205	0.114	0.403	0.127	0.112	0.042	0.186	0.056
GRU4Rec	0.359	0.216	0.582	0.228	0.191	0.114	0.382	0.135
SKNN	0.411	0.245	0.625	0.268	0.262	0.156	0.489	0.177
S-SKNN	0.416	0.247	0.628	0.272	0.279	0.170	0.497	0.185
CT	0.427	0.263	0.618	0.286	0.290	0.189	0.515	0.206
MAN-GRU4Rec	0.447	0.269	0.657	0.293	0.331	0.203	0.545	0.226
NARM	0.463	0.280	0.682	0.303	0.358	0.221	0.566	0.242
MAN-NARM	**0.476**	**0.292**	**0.689**	**0.314**	**0.381**	**0.234**	**0.599**	**0.258**

Table 3: Overall results of different models for the incremental *SR* task on two datasets. Models are ranked by HR@5, and the best method in each column is in bold.
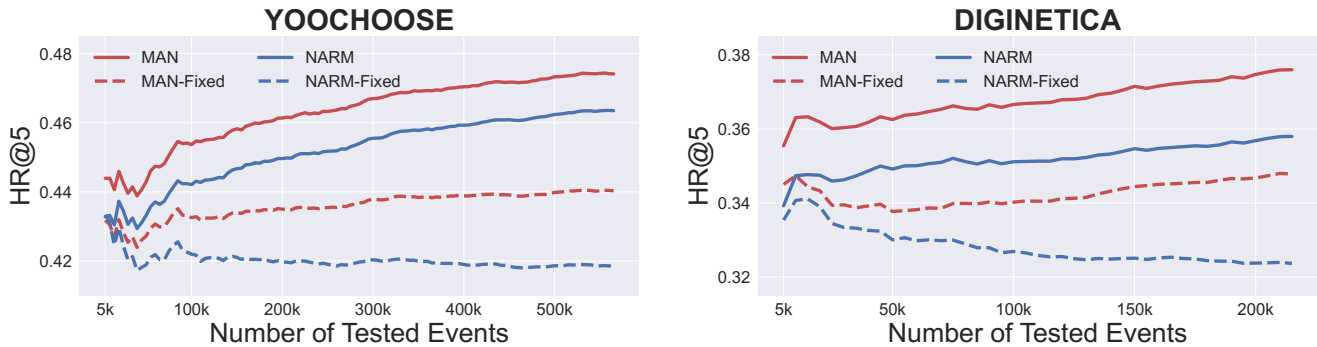

Figure 2: Incremental performance (HR@5) as the number of tested events increases.

patterns. Therefore, we contend that *incremental updates for neural models needs to be small*.

Overall performance. The results of different methods on two datasets are summarized in Table 3. Several interesting empirical results can be noted: **First**, MAN-NARM achieves the top performance. Both MAN-NARM and MAN-GRU4Rec consistently outperform their individual neural modules (NARM and GRU4Rec) with notable margins. This result shows that our MAN architecture effectively helps standard neural recommenders for incremental *SR* scenarios. **Second**, MAN is not sensitive to neural recommenders. We observed that even though NARM significantly outperforms GRU4Rec, MAN-GRU4Rec and MAN-NARM show comparable performances. We contend that the memory predictions effectively compensate the failed predictions of GRU4Rec. **Third**, (S)-SKNN and CT are much stronger nonparametric methods than Item-KNN, with CT being slightly better. They both outperform GRU4Rec on two datasets, and their performance gaps compared to MAN and NARM are smaller on the YOUCHOOSE dataset that contains more new events.

Incremental performance. In Figure 2, we present incremental performance of MAN and NARM as more events are evaluated during testing. We also included versions that are *fixed* during the testing phase. In NARM-Fixed, the neural model is fixed. In MAN-Fixed, both the neural model and the gating networks are fixed, and only memory entries are incrementally expanded. Several results can be noted from Figure 2. **First**, MAN helps to boost performance both when

	YOOCHOOSE		DIGINETICA	
	HR@5	MRR@5	HR@5	MRR@5
MAN	**0.476**	**0.292**	**0.381**	**0.234**
MAN-Shallow	0.469	0.286	0.374	0.228
MAN-50k	0.469	0.287	0.376	0.231
MAN-10k	0.466	0.284	0.369	0.226

Table 4: Ablation study for MAN. The large memory size and the gating network of MAN gain performance benefits.

the neural model is fixed and incrementally updated. MAN outperforms NARM by consistent margins, and MAN-Fix outperforms NARM-Fix by increasing margins. **Second**, incrementally updated models consistently outperform their fixed versions. Both MAN and NARM outperform their corresponding fixed versions with significant and increasing margins.

4.5 In-depth Analysis

In the following experiments, we conduct in-depth analysis to further understand the effectiveness and efficiency of different components of MAN and for what types of events MAN most improves predictions.

Ablation study. We further studied in Table 4 the effect of two setups in MAN, i.e., the large memory size and the combining scheme with a gating network. Two simpler versions are compared. **(i)** MAN-Shallow linearly combines neural and memory predictions with a fixed scalar rather than
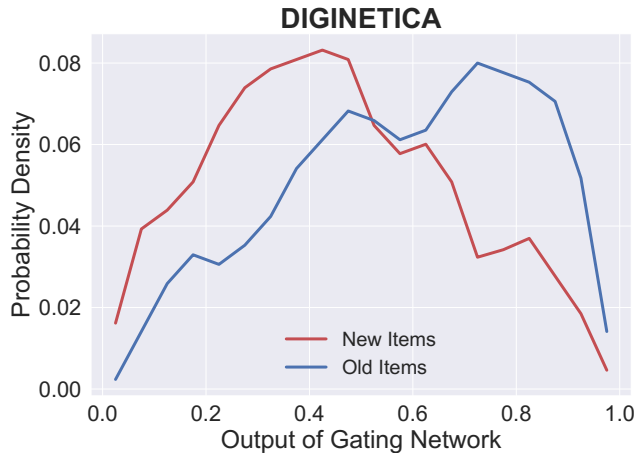
Figure 3: Visualizing the empirical distribution of the Gating Network outputs assigned to the parametric part on DIGINETICA. Outputs on old (in the training phase) and new (not in the training phase) items are plotted separately.

	YOOCHOOSE		DIGINETICA	
	Train	Avg. Response	Train	Avg. Response
NARM	375.5	25.5	42.5	23.5
MAN	392.2	29.5	45.8	27.8

Table 5: Computation time of MAN and NARM. Total training time (in minutes) and average response time (in milliseconds) per recommendation are reported for both datasets. The computation overhead of MAN on top of NARM is trivial.

the weight output by the gating network. The scalar weight is tuned on validation splits to be 0.7 for YOOCHOOSE and 0.8 for DIGINETICA. This simple method serves as a strong baseline in learning new vocabularies in language model tasks [Merity *et al.*, 2017; Grave *et al.*, 2017a]. **(ii)** MAN-50k/10k use fixed-size memories that only store a limited number of recent events (50k/10k). MAN-Shallow is consistently inferior to MAN. It means the gating network makes a better decision to combine neural and memory predictions. Furthermore, the performance drops as the size of the memory decrease from unbounded (MAN) to 50k, and to 10k. It means that keeping a big memory that handles long histories achieves better recommendation performance. Despite the slight performance drop, the three simplified versions are more efficient, therefore, they are still suitable candidates for industry.

Visualization of gating network outputs. In this experiment, we further visualize the empirical distribution of the gating network outputs assigned to the parametric part. We demonstrate in Figure 3 that it indeed learns desired weight distributions to the parametric and non-parametric parts. For new items denoted by the red curve, outputs of the gating network tend to be small such that larger weights are assigned to the non-parametric memory prediction. Similarly, for old items denoted by the blue curve, outputs of the gating net-

Bucket #	1	2	3	4	5
YOOCHOOSE					
MAN	0.318	0.263	0.319	0.378	0.517
NARM	0.287	0.247	0.302	0.362	0.510
DIGINETICA					
MAN	0.145	0.187	0.251	0.329	0.515
NARM	0.117	0.169	0.230	0.303	0.494

Table 6: Disentangled performance (HR@5) at different item frequency buckets. MAN is consistently better at all buckets, and the improvement margin on infrequent items is the most significant.

work tend to be large such that smaller weights are given to the non-parametric memory prediction.

Computation time. In Table 5, we report the total training time and average response time per recommendation of MAN compared to NARM. Both models are trained using a NVIDIA TITAN X GPU with 12GB memory. We can see that the extra training time and response time of MAN is **trivial** on top of the NARM. This empirical results, together with our analysis in Section 3.5, demonstrate that MAN is computation efficient.

Disentangled performance. To further understand for what types of events that MAN most improves predictions, we studied the disentangled performance when items are bucketed into five groups by their occurrence frequency in training data. Bucket 1 contains the least frequent items, and bucket 5 contains the most frequent items. Bucket splitting intervals are chosen to ensure the bucket size is the same. The disentangled performances of MAN and NARM across five buckets are reported in Table 6, and the results reveal that:

- Infrequent items are more challenging to predict. The performances of both methods have an increasing trend as item frequency increases.

- MAN is consistently better than NARM on all levels of item frequency.

- The improvement margin of MAN over NARM is very significant on infrequent items (small bucket number). Therefore, we contend that MAN it is especially good at learning new patterns and items with a small number of observations.

5 Conclusion

In this paper, we study neural methods in a realistic incremental session-based recommendation scenario. We show that existing neural models can be used in this scenario with small incremental updates, and we propose a general method called **M**emory **A**ugmented **N**eural recommender (MAN) that is widely applicable to augment different existing neural models. MAN uses a efficient nonparametric memory to compute a memory prediction, which is combined with the neural prediction through a lightweight gating network. We show that MAN consistently outperforms state-of-the-art neural and nonparametric methods and it is particularly good at learning new items with insufficient number of observations.

References

[Bakhtin *et al.*, 2018] Anton Bakhtin, Arthur Szlam, Marc'Aurelio Ranzato, and Edouard Grave. Lightweight adaptive mixture of neural and n-gram language models. *arXiv preprint arXiv:1804.07705*, 2018.

[Chen *et al.*, 2018] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. Sequential recommendation with user memory networks. In *WSDM*, pages 108–116. ACM, 2018.

[Devlin *et al.*, 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[Garg *et al.*, 2019] Diksha Garg, Priyanka Gupta, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. Sequence and time aware neighborhood for session-based recommendations: Stan. In *SIGIR*, pages 1069–1072, 2019.

[Grave *et al.*, 2017a] Edouard Grave, Moustapha M Cisse, and Armand Joulin. Unbounded cache model for online language modeling with open vocabulary. In *NIPS*, pages 6042–6052, 2017.

[Grave *et al.*, 2017b] Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. In *ICLR*, 2017.

[Graves *et al.*, 2014] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[Härdle and Linton, 1994] Wolfgang Härdle and Oliver Linton. Applied nonparametric methods. *Handbook of Econometrics*, 4:2295–2339, 1994.

[He and McAuley, 2016] Ruining He and Julian McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *ICDM*, pages 191–200. IEEE, 2016.

[He *et al.*, 2016] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*, pages 549–558. ACM, 2016.

[Hidasi and Karatzoglou, 2018] Balázs Hidasi and Alexandros Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. In *CIKM*, pages 843–852. ACM, 2018.

[Hidasi *et al.*, 2016] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *ICLR*, 2016.

[Jannach and Ludewig, 2017] Dietmar Jannach and Malte Ludewig. When recurrent neural networks meet the neighborhood for session-based recommendation. In *RecSys*, pages 306–310. ACM, 2017.

[Jegou *et al.*, 2010] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *PAMI*, 33(1):117–128, 2010.

[Kang and McAuley, 2018] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *ICDM*, pages 197–206. IEEE, 2018.

[Li *et al.*, 2017] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In *CIKM*, pages 1419–1428. ACM, 2017.

[Liu *et al.*, 2018] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. Stamp: Short-term attention/memory priority model for session-based recommendation. In *SIGKDD*, pages 1831–1839. ACM, 2018.

[Ludewig and Jannach, 2018] Malte Ludewig and Dietmar Jannach. Evaluation of session-based recommendation algorithms. *User Modeling and User-Adapted Interaction*, 28(4-5):331–390, 2018.

[McCloskey and Cohen, 1989] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pages 109–165. Elsevier, 1989.

[Merity *et al.*, 2017] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *ICLR*, 2017.

[Mi and Faltings, 2017] Fei Mi and Boi Faltings. Adaptive sequential recommendation for discussion forums on moocs using context trees. In *Proceedings of the 10th international conference on educational data mining*, number CONF, 2017.

[Mi and Faltings, 2018] Fei Mi and Boi Faltings. Context tree for adaptive session-based recommendation. *arXiv preprint arXiv:1806.03733*, 2018.

[Miller *et al.*, 2016] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. In *EMNLP*, pages 1400–1409, 2016.

[Orhan, 2018] Emin Orhan. A simple cache model for image recognition. In *NIPS*, pages 10107–10116, 2018.

[Ren *et al.*, 2019a] Kan Ren, Jiarui Qin, Yuchen Fang, Weinan Zhang, Lei Zheng, Weijie Bian, Guorui Zhou, Jian Xu, Yong Yu, Xiaoqiang Zhu, et al. Lifelong sequential modeling with personalized memorization for user response prediction. In *SIGIR*, pages 565–574, 2019.

[Ren *et al.*, 2019b] Pengjie Ren, Zhumin Chen, Jing Li, Zhaochun Ren, Jun Ma, and Maarten de Rijke. Repeatnet: A repeat aware neural recommendation machine for session-based recommendation. In *AAAI*, 2019.

[Rendle *et al.*, 2010] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *WWW*, pages 811–820. ACM, 2010.

[Shani *et al.*, 2005] Guy Shani, David Heckerman, and Ronen I Brafman. An mdp-based recommender system. *JMLR*, 6(Sep):1265–1295, 2005.

[Sun *et al.*, 2019] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *CIKM*, pages 1441–1450, 2019.

[Tan *et al.*, 2016] Yong Kiam Tan, Xinxing Xu, and Yong Liu. Improved recurrent neural networks for session-based recommendations. In *1st Workshop on Deep Learning for Recommender Systems*, pages 17–22. ACM, 2016.

[Tu *et al.*, 2018] Zhaopeng Tu, Yang Liu, Shuming Shi, and Tong Zhang. Learning to remember translation history with a continuous cache. *TACL*, 6:407–420, 2018.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

[Wu *et al.*, 2019] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based recommendation with graph neural networks. In *AAAI*, 2019.