Towards Neural Mixture Recommender for Long Range Dependent User Sequences

Jiaxi Tang*†
School of Computing Science, Simon Fraser University, Canada
jiaxit@sfu.ca

Francois Belletti†
Google
belletti@google.com

Sagar Jain
Google
sagarj@google.com

Minmin Chen
Google
minminc@google.com

Alex Beutel
Google
alexbeutel@google.com

Can Xu
Google
canxu@google.com

Ed H. Chi
Google
edchi@google.com

ABSTRACT

Understanding temporal dynamics has proved to be highly valuable for accurate recommendation. Sequential recommenders have been successful in modeling the dynamics of users and items over time. However, while different model architectures excel at capturing various temporal ranges or dynamics, distinct application contexts require adapting to diverse behaviors.

In this paper we examine how to build a model that can make use of different temporal ranges and dynamics depending on the request context. We begin with the analysis of an anonymized Youtube dataset comprising millions of user sequences. We quantify the degree of long-range dependence in these sequences and demonstrate that both short-term and long-term dependent behavioral patterns co-exist. We then propose a neural Multi-temporal-range Mixture Model (*M3*) as a tailored solution to deal with both short-term and long-term dependencies. Our approach employs a mixture of models, each with a different temporal range. These models are combined by a learned gating mechanism capable of exerting different model combinations given different contextual information. In empirical evaluations on a public dataset and our own anonymized YouTube dataset, *M3* consistently outperforms state-of-the-art sequential recommendation methods.

CCS CONCEPTS

• **Information systems** → **Personalization**; **Recommender systems**; • **Computing methodologies** → **Neural networks**.

*Work done while interning at Google
†The two authors contributed equally to this work

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313650

KEYWORDS

Recommender System; User Modeling; Sequential Prediction

ACM Reference Format:
Jiaxi Tang, Francois Belletti, Sagar Jain, Minmin Chen, Alex Beutel, Can Xu, and Ed H. Chi. 2019. Towards Neural Mixture Recommender for Long Range Dependent User Sequences. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3308558.3313650

1 INTRODUCTION

Across the web and mobile applications, recommender systems are relied upon to surface the right items to users at the right time. Some of their success can be attributed to advances in modeling as well as the ingenuity of applied researchers in adopting and inventing new techniques to solve this important problem [19, 30, 44, 46]. Fundamentally, recommenders match users in a particular context with the best personalized items that they will engage with [17, 34]. In order to do this effectively, recommenders need to understand the users, typically based on their previous actions, and to understand items, most often based on the users who previously interacted with them. This presents a fundamental challenge: users' preferences and items' perception are continuously changing over time, and the recommender system needs to understand these dynamics.

A significant amount of research has recognized forms of this problem. Sequence information has been generally shown to improve recommender performance [18, 45]. Koren [29] identified multiple user and item dynamics in the Netflix Prize competition, and incorporated these dynamics as biases in a collaborative filtering model. [7, 56] demonstrated that Recurrent Neural Networks (RNNs) could learn many of these patterns, and likewise [20] demonstrated that RNNs can learn patterns in individual sessions. Despite these successes, RNNs are known to have difficulties learning long-range dependent temporal patterns [4].

We observe and study an open challenge for such sequential recommender systems: *while different applications and contexts require different temporal ranges and patterns, model architectures are typically designed to capture a particular temporal dynamic.* For

example, when a user comes to the Amazon home page they may be looking for something new to buy or watch, but on an item specific page they may be looking for other items that are closely related to recently browsed items. *How can we design a model that works, simultaneously, across all of these contexts and temporal ranges?*

Contributions: We address the issue of providing a single model adapted to the diversity of contexts and scales of temporal dependencies in sequential recommendations through data analysis and the design of a Multi-temporal-range Mixture Model, or *M3* for short. We make the following contributions to this problem:

- **Data-driven design:** We demonstrate that in real world recommendation tasks there are significant long-range temporal dependencies in user sequence data, and that previous approaches are limited in their ability to capture those dynamics. M3's design is informed by this quantitative analysis.
- **Multi-range Model:** We offer a single model, M3, which is a mixture model consisting of three sub-models (each with a distinct manually designed architecture) that specialize in capturing different ranges of temporal dependencies. M3 can learn how to dynamically choose to focus on different temporal dynamics and ranges depending on the application context.
- **Empirical Benefits and Interpretability:** We show on both public academic and private data that our approach provides significantly better recommendations. Further, using its interpretable design, we analyze how M3 dynamically switches between patterns present at different temporal ranges for different contexts, thus showing the value in enabling context-specific multi-range modeling. Our private dataset consists in anonymized user sequences from YouTube. To the best of our knowledge this paper is the first to focus on sequential patterns in such a setting.

2 RELATED WORK

Before we describe our sequential recommendation problem and provide the quantitative insights orienting the design of a novel sequential neural model based on a mixture of models, we briefly introduce the reader to some key pre-existing related work.

Matrix factorization [30] is among the most popular techniques used in classic recommender research, in which a similarity score for each user-item pair is learned by building latent user and item representations to recover historical user-item interactions. The predicted similarity score is then used to indicate the *relatedness* and find the most relevant items to recommend to a user. Followup work on introducing auxiliary sources of information beyond user-item interactions have been proven successful [11], especially for cold-start problems. Pazzani and Billsus [39] use item content (*e.g.,* product image, video's visual/audio content, etc) to provide a better item representation.

Neural Recommender Systems. Deep neural networks have gained tremendous success in the fields of Computer Vision [28, 31] and Natural Language Processing [2, 36]. In recommender research, we have witnessed growing interest of using deep neural networks to model complex contextual interactions between user and items, which surpass classic factorization-based methods [30, 43]. Auto-encoders[33, 47, 57] constitute an early example of success for a framework based on neural networks to better infer un-observed user/item affinities in a recommendation problem. He et al. [19] also proved that traditional Collaborative Filtering methods can be effectively generalized by a deep neural network. Besides,

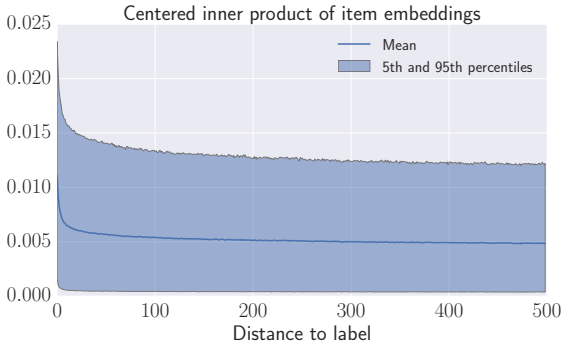


Figure 1: Trace of covariance (i.e. centered inner product similarity) of item embeddings between the last item in user sequence and the item located L steps before (100K samples).

For the specific problem of sequential recommendation using neural networks, RNNs [20, 56] have become a common choice. Other methods based on Convolutional Neural Networks (CNNs) [51, 59], Attention Models [61] have also been explored. While most of existing methods developed for sequential recommendations perform well [18, 20, 45, 49, 51], they still have some limitations when dealing with long user sequences found in production recommender systems. As we shall discuss in Section 3, such approaches do not scale well to very long sequences.

Mixture of Models. Despite being simpler and more elegant, monolithic models are in general less effective than mixtures of models to take advantage of different model capacities and architectural biases. Gehring et al. [15] used an RNN in combination with an attention model for neural machine translation which provided a substantial performance gain. Pinheiro and Collobert [40] proposed to combine a CNN with an RNN for scene labeling. In the field of sequential recommendation, an earlier work on mixing of a Latent Factor Model (LFM) and a Factorized Markov Chain (FMC) has been shown to offer superior performance than each individual one [45]. A similar trend was observed in [18, 60]. While sharing similar spirit to these aforementioned methods, we designed our mixture of models with the goal to model varying ranges of dependence in long user sequences found in real production systems. Unlike model ensembles [13, 62, 63] that learn individual models separately prior to ensembling them, a mixture of models learns individual models as well as combination logic simultaneously.

3 QUANTIFYING LONG RANGE TEMPORAL DEPENDENCIES IN USER SEQUENCES

We first present some findings on our anonymized proprietary dataset which uncover properties of behavioral patterns as observed in extremely-long user-item interaction sequences. We then pinpoint some limitations of existing methods which motivate us to design a better adapted solution.

Sequential Recommendation Problem: We consider a sequential recommendation problem [18, 20, 45, 49, 51] defined as follows: assume we have a set of users $u \in \mathcal{U}$, a set of items $v \in \mathcal{V}$, and for each user we have access to a sequence of user historical events $\mathcal{E}^u = (e_1^u, e_2^u, \cdots)$ ordered by time. Each e_τ^u records the item consumed at time τ as well as context information of the interaction. Given the historical interactions, our goal is to recommend to each user a subset of items in order to maximize a performance metric such as user satisfaction.

3.1 Hidden Values in Long User Sequences

We now describe how we developed a better understanding of long user sequences in our proprietary dataset through quantitative data exploration. To quantify how past events can influence a user's current behavior in our internal dataset, *i.e.* measure the range of temporal dependency within a sequence of events, one can examine the covariance matrix of two events L-step apart [5, 41], where step denotes the relative order of events within sequence. In particular, we look at the trace of the covariance matrix as a measurement of dependency:

$$\mathrm{Dep}_L = \mathrm{tr}\left(\mathrm{Cov}\left(Q_{e_N}, Q_{e_{N-L}}\right)\right)$$

where e_N is the item in last event in a logged user/item interaction sequence and e_{N-L} is the item corresponding to the interaction that occurred L time steps before the last event. We focus on the trace of the covariance matrix as it equals the sum of the eigenvalues of the covariance matrix and its rate of decay is therefore informative of the rate of decay of these eigenvalues as a whole.

We utilize the embeddings Q that have been learned by a pre-existing model—in our case an RNN-based sequential recommender which we describe later as one of M3's sub-models. Dep_L here measures the similarity between the current event and the event L steps back from it. To estimate Dep_L for a particular value of L we employ a classic empirical averaging across user sequences in our dataset. From Figure 1, we can extract multiple findings:

- The dependency between two events decreases as the time separating their consumption grows. This suggests that recent events bear most of the influence of past user behavior on a user's future behavior.
- The dependency slowly approaches zero even as the temporal distance becomes very large (*i.e.* $L > 100$). The clear hyperbolic-decay of the level of temporal dependencies indicates the presence of long-range-dependent patterns existing in user sequences [41]. In other words, a user's past interactions, though far from the current time step, still cumulatively influence their current behavior significantly.

These findings suggest that users do have long-term preferences and better capturing such long-range-dependent pattern could help predicting their future interests. In further work, we plan to use off-policy correction methods such as [8, 16] to remove presentation bias when estimating correlations.

3.2 Limitations of Existing Sequential Models

The previous section has demonstrated the informational value of long-range temporal patterns in user sequences. Unfortunately, it is still generally challenging for existing sequential predictive models to fully utilize information located far into the past.

Most prior models have difficulties when learning to account for sequential patterns involving long-range dependence. Existing sequential recommenders with factorized Markov chain methods [18] or CNNs [51] arguably provide reliable sequential recommendation strategies. Unfortunately they are all limited by a short window of significant temporal dependence when leveraging sequential data to make a recommendation prediction. RNNs [7, 20, 24] and their variants [12, 42] are widely used in sequential recommendation. RNN-based models, though effective for short user sequences (*e.g.* short event sequences within a session), are challenged by long-range dependent patterns in long user sequences. Because of the way they iterate over sequential items [36] and their use of saturating non-linear functions such as tanh to propagate information through time, RNNs tend to have difficulties leveraging the information contained in states located far into the past due to gradient propagation issues [4, 38]. Even recent architectures designed to facilitate gradient propagation such as Gated Recurrent Unit [10] and Long-short Term Memory [21, 50] have also been shown to suffer from the same problem of not being able to provably account for long-range dependent patterns in sequences [4].

A second challenge in sequential recommendations is learning user latent factors P_u explicitly from data, which has been observed to create many difficulties [9, 18, 45, 51]. In the corresponding works, users' long-term preferences have been modeled through learning a set of latent factors P_u for each user. However, learning P_u explicitly is difficult in large-scale production systems. As the number of users is usually several magnitudes higher than the number of items, building such a large user vocabulary and storing the latent factors in a persistent manner is challenging. Also, the long-tail users (*a.k.a* cold users) and visitor users (*i.e.* users who are not logged in) could have much worse recommendations than engaged users [6].

3.3 Limitations of Single Monolithic Models

Figure 1 clearly indicates that although the influence of past user events on future interactions follows a significant decaying trend, significant predictive power can still be carried by events located arbitrarily far in the past. Very recent events (*i.e.* $1 \leq L \leq 10$) have large magnitude similarities with the current user behavior and this similarity depends strongly on the sequential order of related events. As the distance L grows larger, the informative power of previously consumed items on future user behavior is affected by more uncertainty (*e.g.* variance) and is less sensitive to relative sequential position. That is, the events from 100 steps ago and from 110 steps ago may have a generally similar influence on future user decisions regardless of their relative temporal location. Therefore, *for the kind of sequential signals we intend to leverage, in which different scales of temporal dependencies co-exist, it may be better to no longer consider a single model.* While simple monolithic models such as Deep Neural Network (DNN) with pooling and dropout [11, 55] are provably robust to noise, they are unfortunately not sensitive to sequential order (without substantial modifications). On the other hand, RNNs [7, 20] provide cutting-edge sequential modeling capabilities but they are heavily sensitive to noise in sequential patterns. Therefore, it is natural to choose a mixture of diverse models which would then complement each other to provide better overall predictive power.

4 MULTI-TEMPORAL-RANGE MIXTURE MODEL FOR LONG USER SEQUENCES

Motivated by our earlier analyses, we now introduce a novel method aimed at addressing the shortcoming of pre-existing approaches for long user/item interaction sequences: Multi-temporal-range Mixture Model (M3) and its two variants (M3R/M3C). For simplicity, we omit the superscripts related to users (*i.e.* e_τ^u will now be denoted e_τ) and use a single user sequence to describe the neural architecture we introduce.

4.1 Overview

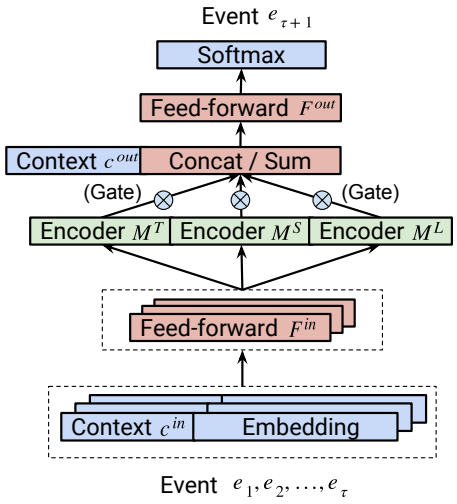


Figure 2: An overview of the proposed M3 model.

Figure 2 gives a general schematic depiction of M3. We will now introduce each part of the model separately in a bottom-up manner, starting from the inputs and progressively abstracting their representation which finally determines the model's output. When predicting the behavior of a user in the next event $e_{\tau+1}$ of their logged sequence, we employ item embeddings and context features (optional) from past events as inputs:

$$x_\tau = [Q_\tau \oplus c_\tau^{\text{in}}], \tag{1}$$

where $\oplus$ denotes the concatenation operator. To map the raw context features and item embeddings to the same high-dimensional space for future use, a feed-forward layer F^{in} is used:

$$Z_\tau^{\text{in}} = \{z_i^{\text{in}}\}_{i=1\cdots\tau} \text{ where } z_\tau^{\text{in}} = F^{\text{in}}(x_\tau) \tag{2}$$

here $z_\tau^{\text{in}} \in \mathbb{R}^{1\times d_{\text{in}}}$ represents the input processed at step τ and $Z_\tau^{\text{in}} \in \mathbb{R}^{\tau\times d_{\text{in}}}$ stands for the collection of all processed inputs before step τ (included). Either the identity function or a ReLU [37] can be used to instantiate the feed-forward layer F^{in}.

In the previous section, we assessed the limitations of using a single model on long user sequence. To circumvent the issues we highlighted, we employ in M3 three different sequence models (encoders) in conjunction, namely M^T, M^S and M^L, on top of the processed input Z_τ^{in}. We will later explain their individual architectures in details. The general insight is that we want each of these sub-models to focus on different ranges of temporal dependencies in user sequences to provide a better representation (*i.e.,* embedding) of the sequence. We want the sub-models to be architecturally diverse and address each other's shortcomings. Hence

$$SE_\tau^T = M^T(Z_\tau^{\text{in}}),\ SE_\tau^S = M^S(Z_\tau^{\text{in}}),\ SE_\tau^L = M^L(Z_\tau^{\text{in}}), \tag{3}$$

which yields three different representations, one produced by each of the three sequence encoders. The three different sub-model encoders are expected to produce outputs—denoted by d_{enc}—of identical dimension. By construction, each sequential encoder produces its own abstract representation of a given user's logged sequence, providing diverse latent semantics for the same input data.

Our approach builds upon the success of Mixture-of-Experts (MOE) model [23]. One key difference is that our 'experts' are constructed to work with different ranges of temporal dependencies, instead of letting the cohort of 'experts' specialize by learning from data. As shown in [48], heavy regularization is needed to learn different experts sharing the same architecture in order to induce specialization and prevent starvation when learning (only one expert performs well because it is the only one to learn which creates a self-reinforcing loop when learning with back-propagation).

Informed by the insights underlying the architecture of MOE models, we aggregate all sequence encoders' results by weighted-concatenate or weighted-sum, with weights G_τ computed by a small gating network. In fact, we concatenate the outputs with

$$SE_\tau = (G_\tau^T \times SE_\tau^T) \oplus (G_\tau^S \times SE_\tau^S) \oplus (G_\tau^L \times SE_\tau^L), \tag{4}$$

where $G_\tau \in \mathbb{R}^3$ corresponds to the outputs of our gating network. We can also aggregate outputs with a weighted-sum:

$$SE_\tau = (G_\tau^T \times SE_\tau^T) + (G_\tau^S \times SE_\tau^S) + (G_\tau^L \times SE_\tau^L). \tag{5}$$

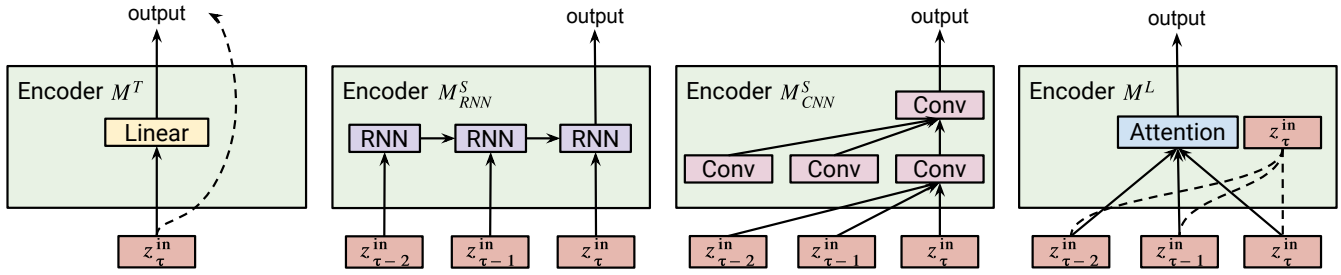
Note that there is no theoretical guarantee whether concatenation is better than summation or not. The choice of aggregation, as well as the choice of activation functions, is determined by observing a given model's performance from a validation set extracted from different datasets. Such a procedure is usual in machine learning and will help practitioners determine which variant of the model we propose is best suited to their particular application.

Because of its MOE-like structure, our model can adapt to different recommendation scenarios and provide insightful interpretability (as we shall see in Section 5). In many recommendation applications, some features annotate each event and represent the context in which the recommendation query is produced. Such features are for instance indicative of the page or device on which a user is being served a recommendation. After obtaining a sequence encoding at step τ (*i.e.* SE_τ), we fuse it with the annotation's context features (optional) and project them to the same latent space with another hidden feed-forward layer F^{out}:

$$z_\tau^{\text{out}} = F^{\text{out}}([SE_\tau \oplus c_\tau^{\text{out}}]) \tag{6}$$

where c^{out} is a vector encoding contextual information to use after the sequence has been encoded. Here the $z_\tau^{\text{out}} \in \mathbb{R}^{1\times d_{\text{out}}}$ is what we name *user representation*, it is computed based on the user's history as it has been gathered in logs. Finally, a user similarity score r_v is predicted for each item via an inner-product (which can be changed to another similarity scoring function):

$$r_v = z_\tau^{\text{out}} \cdot Q_v' \tag{7}$$



(a) Tiny-range encoder **(b) Short-range encoder (RNN)** **(c) Short-range encoder (CNN)** **(d) Long-range encoder**

Figure 3: The sequence encoders of *M3*. The solid lines are used to denote the data flow. The dotted line in (a) means an identity copy whereas in (d) it means the interaction of attention queries and keys.

where Q'_v is a vector representing the item. For a given user, item similarity scores are then normalized by a softmax layer which yields a recommendation distribution over the item vocabulary. After training M3, the recommendations for a user at step τ are served by sorting the similarity scores r_v obtained for all $v \in \mathcal{V}$ and retrieving the items associated with the highest scores.

4.2 Three Different Range Encoders

Item Co-occurrence as a Tiny-range Encoder The Tiny-range encoder M^T only focuses on the user's last event e_τ, ignoring all previous events. In other words, given the processed inputs from past events Z^{in}_τ, this encoder will only consider z^{in}_τ. As in factorizing Markov chain (FMC) models [45], M^T makes predictions based on item range-1 co-occurrence within observed sequences. For example, if most of users buy iPhone cases after purchasing an iPhone, then M^T should learn this item-to-item co-occurrence pattern. As shown in Figure 3a, we compute M^T's output as:

$$M^T(Z^{\text{in}}_\tau) = \phi(z^{\text{in}}_\tau), \text{ where } \phi(x) = \begin{cases} xW^{(T)} + b^{(T)}, & \text{if } d_{\text{in}} \neq d_{\text{enc}}, \\ x, & \text{otherwise.} \end{cases} \tag{8}$$

That is, when the dimensionality of processed input and encoder output are the same, the tiny-range encoder performs a role of residual for the other encoders in mixture. If $d_{\text{in}} \neq d_{\text{enc}}$, it is possible to down-sample (if $d_{\text{in}} > d_{\text{enc}}$) or up-sample (if $d_{\text{in}} < d_{\text{enc}}$) from z^{in}_τ by learned parameters $W^{(T)} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{enc}}}$ and $b^{(T)} \in \mathbb{R}^{d_{\text{enc}}}$.

In summary, the tiny-range encoder M^T can only focus on the last event by construction, meaning *it has a temporal range of 1 by design*. If we only use the output of M^T to make predictions, we obtain recommendations results based on item co-occurrence.

RNN/CNN as Short-range Encoder As discussed in Section 3, the recent behavior of a user has substantial predictive power on current and future interactions. Therefore, to leverage the corresponding signals entailed in observations, we consider instantiating a short-range sequence encoder that puts more emphasis on recent past events. Given the processed input from past events Z^{in}_τ, this encoder, represented as M^S, focuses by design on a recent subset of logged events. Based on our quantitative data exploration, we believe it is suitable for M^S to be highly sensitive to sequence order. For instance, we expect this encoder to capture the purchasing pattern iPhone → iPhone case → iPhone charger if it appears frequently in user sequences. As a result, we believe the Recurrent Neural Network (RNN [36]) and the Temporal Convolutional Network ([3, 53, 59]) are fitting potential architectural choices. Such neural architectures have shown superb performances when modeling high-order causalities. Beyond accuracy, these two encoders are also order sensitive, unlike early sequence modeling method (*i.e.* Bag-of-Word [27]). As a result we develop two interchangeable variants of M3: *M3R* and *M3C* using an RNN and a CNN respectively.

To further describe each of these options, let us introduce our RNN encoder M^S_{RNN}. As shown in Figure 3b we obtain the output of M^S_{RNN} by first computing the *hidden state* of RNN at step τ:

$$h_\tau = \text{RNN}(z^{\text{in}}_\tau, h_{\tau-1}), \tag{9}$$

where RNN$(\cdot)$ is a *recurrent cell* that updates the hidden state at each step based on the previous hidden state $h_{\tau-1} \in \mathbb{R}^{1 \times d_{\text{in}}}$ and the current RNN input z^{in}_τ. Several choices such as Gated Recurrent Unit (GRU) [10] or Long Short Term Memory (LSTM) [21] can be used. The output is then computed as follows:

$$M^S_{\text{RNN}}(Z^{\text{in}}_\tau) = h_\tau W^{(R)}, \text{ where } W^{(R)} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{enc}}} \tag{10}$$

where $W^{(R)}$ maps the hidden state to the encoder output space. We design our CNN encoder M^S_{CNN} as a Temporal Convolutional Networks which has provided state-of-art sequential modeling performance [3, 15, 53]. As shown in Figure 3c, this encoder consists of several stacked layers. Each layer computes

$$h^{(1)}_\tau = \text{Conv}(Z^{\text{in}}_\tau), \ldots, h^{(k)}_\tau = \text{Conv}(h^{(k-1)}_\tau), \tag{11}$$

where k indicates the layer number. The Conv$(\cdot)$ is a 1-D convolutional operator (combined with non-linear activations, see [3] for more details), which contains d_{enc} convolutional filters and operates on the convolutional inputs. With K layers in our CNN encoder, the final output will be:

$$M^S_{\text{CNN}}(Z^{\text{in}}_\tau) = h^{(K)}_\tau. \tag{12}$$

As highly valuable signals exist in the short-range part of user sequence, we propose two types of encoders to capture them. Our model can be instantiated in its first variant, *M3R*, if we use RNN encoder or *M3C* if a CNN is employed. Here M3C and M3R are totally interchangeable with each other and they show comparable results in our experiments (see Section 5.2.1). We believe such

flexibility will help practitioners adapt their model to the hardware they intend to use, *i.e.* typically using GPU for faster CNN training or CPU for which RNNs are better suited. In terms of temporal range, the CNN only considers a limited finite window of inputs when producing any output. The RNN, although it does not have a finite receptive field, is hampered by difficulties when learning to leverage events located further back into the past (to leverage an event located L observations ago the RNN needs $L-1$ steps). Regardless of the choice of a CNN or an RNN, our short-range encoder M^S *has a temporal range greater than 1, although it is challenging for this sub-model to capture signals too far away from current step.* This second encoder is specifically designed to capture sequence patterns that concern recent events.

Attention Model as Long-range Encoder The choice of an attention model is also influenced by our preliminary quantitative analysis. As discussed in Section 3, as the temporal distance grows larger, the uncertainties affecting the influence of item consumption on future events get larger as well. Moreover, as opposed to the recent part of a given user's interaction sequence, relative position does not matter as much when it comes to capturing the influence of temporally distant events. As we take these properties into account, we choose to employ *Attention Model* [2, 54] as our long-range sequence encoder. Usually, an attention model consists of three parts: attention *queries*, attention *keys* and attention *values*. One can simply regard an attention model as weighted-sum over attention values with weights resulting from the interaction between attention queries and attention keys. In our setting, we use (1) the last event's processed input z_τ^{in} as attention queries, (2) all past events' processed inputs Z_τ^{in} as keys and values and (3) scaled dot-product [54] as the similarity metric in the attention softmax. For instance, if a user last purchased a pair of shoes, the attention mechanism will focus on footwear related previous purchases.

So that all encoders have the same output dimensionality, we need to transform[1] our processed input first as follows:

$$\tilde{Z}_\tau^{\text{in}} = Z_\tau^{\text{in}} W^{(A)}, \tag{13}$$

where $W^{(A)} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{enc}}}$ is a learned matrix of parameters. Then for each position $i \in [1, \tau]$, we obtain its raw attention weights, with respect to the processed input $\tilde{z}_i^{\text{in}}$, as follows:

$$\omega_{\tau,i} = \frac{\tilde{z}_\tau^{\text{in}} \cdot \tilde{z}_i^{\text{in}}}{\sqrt{d_{\text{enc}}}}, \tag{14}$$

where $\omega_{\tau,i}$ is the raw weight at position i. Similarly, we compute the raw attention weights $\boldsymbol{\omega}_\tau \in \mathbb{R}^{1\times\tau}$ for all positions $\boldsymbol{\omega}_\tau = \{\omega_i\}_{i=1..\tau}$ and normalize them with a softmax$(\cdot)$ function. Finally, we acquire the output of our long-range encoder as follows:

$$M^L(Z_\tau^{\text{in}}) = \text{softmax}(\boldsymbol{\omega}_\tau) Z_\tau^{\text{in}}. \tag{15}$$

Our long-range encoder borrows several advantages from the attention model. First, it is not limited by a certain finite temporal range. That is, *it has an unlimited temporal range and can 'attend' to anywhere in user's sequence with O(1) steps.* Second,because it computes its outputs as a weighted sum of inputs, the attention-based encoder is not as sensitive to sequential order as an RNN or a CNN as each event from the past has an equal chance of influencing the prediction. Third, the attention model is robust to noisy inputs due to its normalized attention weights and weighted-sum aggregation.

Gating Network Borrowing the idea from from Mixture-of-Experts model [23, 35], we build a gating network to aggregate our encoders' results. The gate is also helpful to better understand our model (see Section 5). To produce a simpler gating network, we use a feed-forward layer F^g on the gating network's inputs:

$$G_\tau = [G_\tau^T, G_\tau^S, G_\tau^L] = \text{sigmoid}(F^g(G_\tau^{\text{in}})), \tag{16}$$

where G_τ^{in} is the input we feed into our gating network. We will discuss how the model performs overall with different choices of gate inputs in Section 5.4. The resulting $G_\tau \in \mathbb{R}^3$ contains the gate value modulating each encoder. More importantly, an element-wise sigmoid function is applied to the gate values which allows encoders to 'corporate' with each other [4]. Note that a few previous works [26, 35, 48] also normalize the gate values, but we found this choice led to the degeneracy of our mixture model as it would learn to only use M^S which in turn hampers model performance.

Summary M3 is able to address limitations of pre-existing models as shown in Table 1: (1) M3 has a mixture of three encoders with different temporal ranges which can capture sequential patterns located anywhere in user sequences. (2) Instead of learning a set of latent factor P_u for each user, M3 represents the long-term user preferences by using a long-range sequence encoder that provides a representation of the entire history of a user. Furthermore, M3 is efficient in both model size and computational cost. In particular M3 does not introduce any extra parameters under certain settings (*i.e.* $d_{\text{in}} = d_{\text{enc}}$), and the computation of M^T and M^L are very efficient when using specialized hardware such as a GPU. With its simple gate design, M3 also provides good interpretability and adaptability.

- **Effectiveness.** Given our analysis on user sequences, we assume M3 to be effective. As compared to past works, M3 is capable to capture signals from the whole sequence, it also satisfies the properties we found in different parts of sequence. Moreover, our three encoders constitute a diverse set of sequential encoder and, if well-trained, can model user sequence in a multi-scale manner, which is a key to success in past literature [53, 58].
- **Efficiency.** In terms of model size, M3 is efficient. As compared to existing works which use short-range encoder only, though uses two other encoders, our M3 model doesn't introduce any extra parameters (if $d_{\text{in}} = d_{\text{enc}}$). In terms of computational efficiency, our M3 is good as well, as both M^T and M^L are nothing other than matrix multiplication, which is cheap when computed with optimized hardwares like Graphics Processing Unit (GPU).
- **Interpretability.** Model's interpretability is critical for diagnosing purpose. As we shall see later, with the gate network, we are able to visualize our network transparently by observing the gate values.
- **Adaptability.** One issue in production recommender system is modeling users for different recommendation scenarios, as people may behave very differently. Two typical scenarios are *HomePage* recommendation and product *DetailPage* recommendation. However, as we shall introduce in later

[1] It is unnecessary if d_{in} is same as d_{enc}.

Table 1: A summary of relationships and differences between sequence encoders in *M3*

	Base model	Temporal range	Model size	Sensitive to order	Robustness
M^T	Item Co-occurrence	1	small (or 0)	very high	no
M^S_{RNN}	Recurrent Neural Nets	unknown	large	high	no
M^S_{CNN}	Temporal Convolution Nets	limited	large	high	no
M^L	Attention Model	unlimited	small (or 0)	no	high

section, M3 is able to adapt to these scenarios if we use the scenario information as our gate input.

5 EXPERIMENTAL STUDIES

In this section, we study the two variants of M3 against several baseline state-of-the-art methods on both a publicly available dataset and our large-scale Youtube dataset.

Datasets We use MovieLens 20M[2], which is a publicly available dataset, along with a large-scale anonymized dataset from YouTube to which we have access because we are employees of Google working on improving YouTube as a platform. The dataset is private, anonymized and accessible only internally by few employees whose work is directly related to Youtube.

5.1 Experiments on MovieLens Dataset

As in previous works [18, 51], we process the MovieLens data by first converting numeric ratings to 1 values, turning them into implicit logged item consumption feedback. We remove the items with less than 20 ratings. Such items, because of how little user feedback is available for them, represent another research challenge — cold start — which is outside the scope of the present paper.

To focus on long user sequences, we filtered out users who had a sequence length of less than $\delta_{\text{min}} = 20$ item consumed, while we didn't filter items specifically. The maximum sequence length in the dataset being 7450, we follow the method proposed in [18, 51] and employ a sliding window of length $\delta_{\text{win}} = 300$ to generate similarly long sequences of user/item interactions in which we aim to capture long range dependent patterns. Some statistics can be found in the first row of Table 3.

We do not use contextual annotations for the MovieLens data.

Evaluation protocol We split the dataset into training and test set by randomly choosing 80% of users for training and the remaining 20% for validation (10%) and testing (10%). As with the training data, a sliding window is used on the validation and test sets to generate sequences. We measure the mean average precision (mAP) as an indicator for models' performances [6, 52]. We only focus on the top positions of our predictions, so we choose to use mAP@n with $n \in \{5, 10, 20\}$. There is only one target per instance here and therefore the mAP@n is expected to increase with n which is consistent with [4] but differs from [51].

Details on model architectures We keep architectural parameters consistent across all experiments on MovieLens. In particular, we use identical representation dimensions: $d_{\text{in}} = d_{\text{enc}} = d_{\text{out}} = 32$. Such a choice decreases the number of free parameters as the sub-models M^T and M^L will not have learned parameters. A GRU cell is employed for the RNN while 2 stacked temporal convolution layers [3] of width 5 are used in the CNN. A ReLU activation function is employed in the feed-forward layers F^{in} and F^{out}. Item embeddings of dimension 64 are learned with different weights on the input side (*i.e.,* Q in Eq. 1) and output side (*i.e.,* Q' in Eq. 7). Although previous work [18] has constrained such embeddings to be identical on the input and output side of the model, we found that increasing the number of degrees of freedom led to better results.

Baselines We compare our two variants, *i.e.,* M3R and M3C, with the following baselines:

- **FMC**: The Factorizing model for the first-order Markov chain (FMC) [45] is a simple but strong baseline in sequential recommendation task [7, 49, 51]. As discussed in Section 1, we do not want to use explicit user representations. Therefore, we do not compare the personalized version of this model (FPMC).
- **DeepBoW**: The Deep Bag-of-word model represent user by averaging item embeddings from all past events. The model then makes predictions through a feed-forward layer. In our experiments, we use a single hidden layer with size of 32 and ReLU as activation function.
- **GRU4Rec**: Originally presented in [20], this method uses a GRU RNN over user sequences and is a state-of-the-art model for sequential recommendation with anonymized data.
- **Caser**: The Convolutional Sequence Embeddings model [51] applying horizontal and vertical convolutional filters over the embedding matrix and achieves state-of-the-art sequential recommendation performance. We try $\{2, 4, 8\}$ vertical filters and $\{16, 32, 64\}$ horizontal filters of size $(3, 5, 7)$. In order to focus on the sequential encoding task, we discard the user embedding and only use the sequence embedding of this model to make predictions.

In the models above, due to the large number of items in input and output dictionaries, the learned embeddings comprise most of the free parameters. Therefore, having set the embedding dimension to 64 in all the baselines as well as in M3R and M3C, we consider models with similar numbers of learned parameters. The other hyperparameters mentioned above are tuned by looking at the mAP@20 on validation set. The training time of M3R/M3C is comparable with others and can be further improved with techniques like model compression [52], quantization [22], etc.

5.1.1 Overall Performances. We report each model's performance in Table 2. Each metric is averaged across all user sequences in test set. The best performer is highlighted in bold face. The results show that both M3C and M3R outperform other baselines by a large margin. Among the baselines, GRU4Rec achieves the best performance

[2]https://grouplens.org/datasets/movielens/20m/

and DeepBoW worst one, suggesting the sequence order plays a very important predictive role. FMC performs surprisingly well, suggesting we could get considerable results with a simple model only taking the last event into account. The poor results of Caser may be caused by its design which relies on vertical filters of fixed size. Caser performs better in the next subsection which considers sequences whose lengths vary less within the training data.

Table 2: Performance comparison on MovieLens 20M. M3C and M3R outperform the baselines significantly.

	mAP@5	mAP@10	mAP@20
FMC	0.0256	0.0291	0.0317
DeepBoW	0.0065	0.0079	0.0093
GRU4Rec	0.0256	0.0304	0.0343
Caser	0.0225	0.0269	0.0304
M3C	0.0295	0.0342	0.0379
M3R	**0.0315**	**0.0367**	**0.0421**
Improv.	+23.4%	+20.7%	+22.7%

5.1.2 Investigating the influence of sequence length through variants of MovieLens. The previous results have shown strong performance gains achieved by the models we intruced: M3C and M3R. We now investigate the origin of such improvements. The design of these models was inspired by an attempt to capture sequential patterns with different characteristic temporal extents. To check whether the models we introduced achieve this aim we construct multiple variants of MovieLens with different sequence lengths.

We vary the sequence length by having a maximum cutoff threshold $\delta_{\max}$ which complements the minimal sequence length threshold $\delta_{\min}$. A sequence with more than $\delta_{\max}$ only has its latest $\delta_{\max}$ observations remained. We vary the values of $\delta_{\min}$, $\delta_{\max}$ and the sequence generation window size. Table 3 summarizes the properties of the four variants of the MovieLens dataset we construct. It is noteworthy that such settings make Caser perform better as the sequence length is more consistent within each dataset variant.

GRU4Rec and Caser outperform the other baselines in the present setting and therefore we only report their performance. Figure 4 shows the improvements of M3C and M3R over the best baselines on four MovieLens variants. The improvement of each model is computed by its mAP@20 against the best baseline. In most cases, M3C and M3R can outperform the highest performing baseline. Specifically, on ML20M-S and ML20M-M, Caser performs similarly to GRU4Rec while both M3C and M3R have good performance. This is probably due to the contribution of the tiny-range encoder.

5.2 Anonymized YouTube dataset

For the YouTube dataset, we filtered out users whose logged sequence length was less than 150 ($\delta_{\min} = 150$) and keep each user's last 300 events ($\delta_{\max} = 300$) in their item consumption sequence. In the following experiments, we exploit contextual annotations such as user device (*e.g.,* from web browser or mobile App), time-based features (*e.g.,* dwelling time), etc. User sequences are all anonymized and precautions have been taken to guarantee that users cannot be re-identified. In particular, only public videos with enough views have been retained.

Neural recommender systems attempt at foreseeing the interest of users under extreme constraints of latency and scale. We define the task as predicting the next item the user will consume given a recorded history of items already consumed. Such a problem setting is indeed common in collaborative filtering [34, 46] recommendations. We present here results obtained on a dataset where only about 2 million items are present that correspond to most popular items. While the user history can span over months, only watches from the last 7 days are used for labels in training and watches in the last 2 days are used for testing. The train/test split is 90/10%. The test set does not overlap with the train set and corresponds to the last temporal slice of the dataset. In all, we have more than 200 million training sequences and more than 1 million test sequences, and with overall average sequence length approximately being 200.

The neural network predicts, for a sample of negatives, the probability that they are chosen and classically a negative sampling loss is employed in order to leverage observations belonging to a very large vocabulary [25]. The loss being minimized is

$$\sum_{l \in \text{Labels}} w_l \times \text{CrossEntropy}(\text{SampledSoftmax}(\xi(t+1)))$$

where the SampledSoftmax [25] uses 20000 randomly sampled negatives and w_l is the weight of each label.

Evaluation metrics To test the models' performances, we measure the mean average precision (mAP) as in [7, 51]. We only focus on the top positions of our predictions, so we choose to use mAP@n with $n \in \{5, 10, 20\}$. The mAP is computed with the entire dictionary of candidate items as opposed to the training loss which samples negatives. There is only one target per instance here and therefore the mAP@n is expected to increase with n which is consistent with [4] but differs from [51].

Baselines In order to make fair comparisons with all previous baselines, we used their contextual counterparts if they are proposed or compared in literature.

- **Context-FMC**: The Context-FMC condition the last event's embedding on last event's context features by concatenating them and having a feed-forward layer over them.
- **DeepYouTube**: Proposed by [11], the DeepYoutube model is a state-of-the-art neural model for recommendation. It concatenates: (1) item embedding from users' last event, (2) item embeddings averaged by all past events and (3) context features. The model then makes predictions through a feedforward layer composed of several ReLU layers.
- **Context-GRU**: We used the contextual version of GRU proposed in [49]. Among the three conditioning paradigms on context, we used the concatenation as it gives us better performances.

All models are implemented by TensorFlow [1] and by Adagrad [14] over a parameter server [32] with many workers.

Model details In the following experiments, we keep the dimensions of processed input d_{in} and encoder outputs d_{enc} identical for all experiments conducted on the same dataset. Once more, we also want to share some of our architectural parameters so that they are consistent across the two datasets. Again, by doing this, we make the parametrization of our models more parsimonious,

Table 3: Statistics of the variants of the MovieLens dataset.

	Min. Length	Max. Length	Window Size	Avg. Length	#Sequences	#Items
ML20M	20	∞	300	144.1	138.4K	13.1K
ML20M-S	20	50	20	42.8	138.4K	13.1K
ML20M-M	50	150	50	113.6	85.2K	13.1K
ML20M-L	150	300	150	250.7	35.8K	12.9K
ML20M-XL	300	∞	300	605.5	16.3K	12.5K

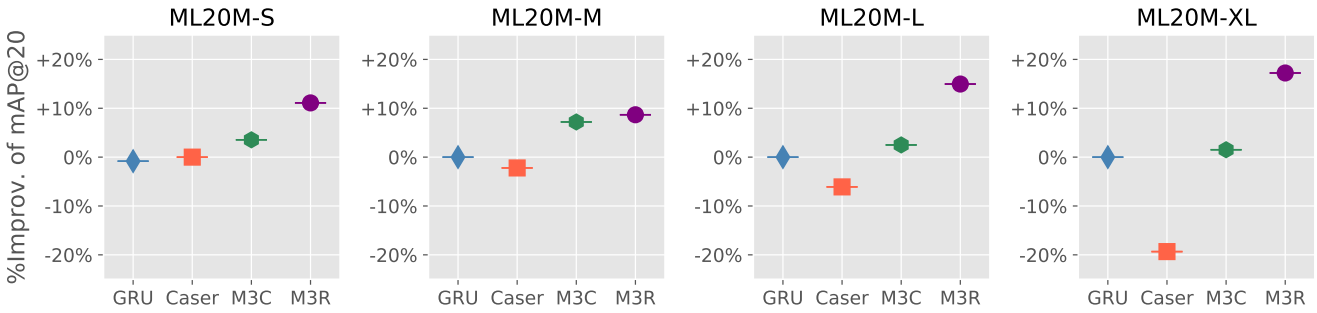


Figure 4: Uplifts with respect to the best baselines on four MovieLens variants. The improvement percentage of each model is computed by its relative mAP@20 gain against the best baseline. For all variants, M3R significantly outperforms the two baselines we consider according to a one-tail paired t-test at level 0.01, while M3C outperforms the other two significantly only on ML20M-M. Note that the standard error of all uplifts gets higher as we use a MovieLens variant with longer sequences. The standard error reaches 2.3% on ML20M-XL.

Table 4: Performance comparison on the anonymized YouTube dataset. M3C and M3R outperform the baselines significantly.

	mAP@5	mAP@10	mAP@20
Context-FMC	0.1103	0.119	0.1240
DeepYouTube	0.1295	0.1399	0.1455
Context-GRU	0.1319	0.1438	0.1503
M3C	0.1469	0.1591	0.1654
M3R	**0.1541**	**01670**	**0.1743**
Improv.	+16.8%	+16.1%	+16.0%

because the sub-models M^T and M^L will be parameter-free. For the RNN cell, we use a GRU on both datasets for its effectiveness as well as efficiency. For the CNN version, we stacked 3 layers of temporal convolution [3], with no dilation and width of 5. For the feed-forward layers F^{in} and F^{out}, we used ReLU as their activation functions, whereas they contains different number of sub-layers. For item embeddings on the input side (*i.e.,* Q in Eq. 1) and on the output side (*i.e.,* Q' in Eq. 7), we learn them separately which improves all results.

5.2.1 Overall Results. We report each model's performance on the private dataset in Table 4. The best performer is highlighted in bold face. As can be seen from this table, on our anonymized YouTube dataset, the Context-FMC performs worse followed by DeepYoutube while Context-GRU performs best among all baselines. The DeepYouTube and Context-GRU perform better than Context-FMC possibly because they have longer temporal range, which again shows that the temporal range matters significantly in long user sequences. One can therefore improve the performance of a sequential recommender if the model is able to leverage distant (long-range dependent) informantion in user sequences.

On both datasets, we observed our proposed two model variants M3R and M3C significantly outperform all other baselines. Within these two variants, the M3R preforms marginally better than the M3C, and it improves upon the best baselines by a large margin (more than 20% on MovieLens data and 16.0% on YouTube data).

5.3 Ablation Study of Mixture of Models

To demonstrate how each encoder contributes to the overall performance, we now present an ablation test on our M3R model (results from M3C are similar) on our proprietary data. We use T, S, L to denote M^T, M^S and M^L respectively. The results are described in Table 5. When we only enable single encoder for M3R, the best performer is M3R-T on MovieLens data and M3R-S on the YouTube data. This result is consistent with the results in Section 5.2.1. With more encoders involved in M3R the model performs better. In particular, when all encoders are incorporated, our M3R-TSL performs best on both datasets, indicating all three encoders matter for performance.

Table 5: mAP@20 vs. different components of M3R on both datasets, where T,S,L stands for M^T, M^S and M^L respectively.

	MovieLens 20M	YouTube Dataset
M3R-T	0.0269	0.1406
M3R-S	0.0363	0.1673
M3R-L	0.0266	0.1359
M3R-TS	0.0412	0.1700
M3R-TL	0.0293	0.1485
M3R-SL	0.0403	0.1702
M3R-TSL	0.0421	0.1743

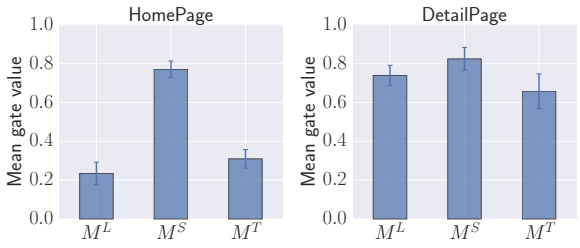


Figure 5: Average gate values of M3R in different scenarios. The model learns to use different combination of encoders in different recommendation scenarios.

5.4 Role of Gating Network

We now begin to study our gating network in order to answer the following questions: (1) Is the gating network beneficial to the overall model performance? (2) How do different gating network inputs influence the model performance? and (3) How can the gating network make our model more adaptable and interpretable?

Fixed gates versus learned gates: First of all, we examine the impact of our gating network by comparing it with a set of fixed gate values. More precisely, we fixed the gate values to be all equal to 1.0 during the model training: $G_\tau = \mathbf{1}$, here $\mathbf{1} \in \mathbb{R}^3$ is a vector. The first row of Table 6 shows the result of this fixed-gate model. We found that the fixed models are weaker than the best performing version of M3R (*i.e.,* mAP@20 of 0.1743) and M3C (*i.e.,* mAP@20 of 0.1654). This reveals that the gating network consistently improves M3-based models' performances.

Influence of different gate inputs: In this paragraph we investigate the potential choices of inputs for the gating network, and how they result in different performance scores. In the existing Mixture-of-Experts (MOE) literature, the input for the gating network G_τ^{in} can be categorized into Contextual-switch and Bottom-switch. The Contextual-switch, used in [4], uses context information as gate input:

$$G_\tau^{\text{in}} = [c_\tau^{\text{in}} \oplus c_\tau^{\text{out}}], \tag{17}$$

where c_τ^{in} and c_τ^{out} are context features from input and output side. Intuitively, this suggests how context may influence the choices of different encoders. If no context information is available, we can still use the output of a shared layer operating before the MOE layer [35, 48] as gate input, *i.e.,* Bottom-switch:

$$G_\tau^{\text{in}} = z_\tau^{\text{in}}. \tag{18}$$

The shared layer contains high-level semantic knowledge from the last event, which can also enable gate switching.

Table 6: mAP@20 vs. different types of gating network on the two datasets for M3R. 'Fixed' indicates we fix gate values to 1.0, 'Contextual-switch' means that we use context features c^{in} and c^{out} as gate input and 'Bottom-switch' corresponds to the use of z_τ^{in} as gate input.

	MovieLens	YouTube
Fixed	0.0413	0.1715
Bottom-switch	0.0421	0.1734
Contextual-switch	/	0.1743

On the MovieLens data, we used Bottom-switched gate for all the results above because of the absence of contextual annotations. On the YouTube dataset, the last two rows from Table 6 provide the comparison results between Contextual-switched gate and Bottom-switched gate. We observe that context information is more useful to the gates than a shared layer. In other words, the decision of whether to focus more on recent part (*i.e.* large gate values for M^T and M^S) or on the distant part (*i.e.* large values for M^L) from user sequence is easier to make based on contextual annotations.

5.5 Discussion on model architecture

The model architecture we design is based on quantitative findings and has two primary goals: capturing co-existing short-range and long-range behavioral patterns as well as serving recommendations given in different contexts with a single model.

We know for recommender systems in most applications (*e.g.* e-commerce like Amazon, streaming services like Netflix) that recommendations commonly occur in at least two different contexts: either a *HomePage* or a *DetailPage*. The Homepage is the page shown when users open the website or open the mobile App, while DetailPage is the page shown when users click on a certain item. User behaviors are different depending on which of these two pages they are browsing. Users are more likely to be satisfied by a recommendation related to recent events, especially the last event, when they are on a DetailPage. A straightforward solution to deal with these changing dynamics is to train two different models.

We now demonstrate that with the multi-temporal-range encoders architecture and gating mechanism in M3, we can have a single adaptive end-to-end model that provides good performance in a multi-faceted recommendation problem. To that end we analyze the behavior of our gating network and show the adaptability of the model as we gain a better understanding of its behavior.

What we observe in Figure 5 is that when contextual information is available to infer the recommendation scenario, the gating network can effectively automatically decide how to combine the results from different encoders in a dynamic manner to further improve performance. Figure 5 shows how gate values of M3R change *w.r.t.* across different recommendation scenarios. It is clear that M3R puts more emphasis on M^S when users are on the HomePage,

while it encourages all three encoders involved when users are on DetailPage. This result shows that the gating network uses different combinations of encoders for different recommendation scenarios.

As a result, we can argue that our architectural design choices do meet the expectations we set in our preliminary analysis. It is noteworthy that the gating mechanism we added on top of the three sub-models is helpful to improve predictive performance and ease model diagnosis. We have indeed been able to analyze recommendation patterns seamlessly.

6 CONCLUSION

M3 is an effective solution to provide better recommendations based on long user sequences. M3 is a neural model that avoids most of the limitations faced by pre-existing approaches and is well adapted to cases in which short term and long term temporal dependencies coexist. Other than effectiveness, this approach also provides several advantages such as the absence of a need extra parameters and interpretability. Our experiments on large public dataset as well as a large-scale production dataset suggest that M3 outperforms the state-of-the-art methods by a large margin for sequential recommendation with long user sequences. One shortcoming of the architecture we propose is that all sub-models are computed at serving time. As a next step, we plan to train a sparse context dependent gating network to address this shortcoming.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. [n. d.]. Tensorflow: a system for large-scale machine learning.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[3] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271* (2018).

[4] Francois Belletti, Alex Beutel, Sagar Jain, and Ed Chi. 2018. Factorized Recurrent Neural Architectures for Longer Range Dependence. In *International Conference on Artificial Intelligence and Statistics*. 1522–1530.

[5] Francois Belletti, Evan Sparks, Alexandre Bayen, and Joseph Gonzalez. 2017. Random projection design for scalable implicit smoothing of randomly observed stochastic processes. In *Artificial Intelligence and Statistics*. 700–708.

[6] Alex Beutel, Ed H Chi, Zhiyuan Cheng, Hubert Pham, and John Anderson. 2017. Beyond globally optimal: Focused learning for improved recommendations. In *International Conference on World Wide Web*. 203–212.

[7] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. 2018. Latent Cross: Making Use of Context in Recurrent Recommender Systems. In *International Conference on Web Search and Data Mining*. ACM, 46–54.

[8] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. 2019. Top-K Off-Policy Correction for a REINFORCE Recommender System. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 456–464.

[9] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential recommendation with user memory networks. In *International Conference on Web Search and Data Mining*. ACM, 108–116.

[10] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[11] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *ACM Conference on Recommender Systems*.

[12] Robin Devooght and Hugues Bersini. 2017. Long and short-term recommendations with recurrent neural networks. In *Conference on User Modeling, Adaptation and Personalization*. ACM, 13–21.

[13] Thomas G Dietterich. 2000. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*. Springer, 1–15.

[14] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.

[15] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122* (2017).

[16] Alexandre Gilotte, Clément Calauzènes, Thomas Nedelec, Alexandre Abraham, and Simon Dollé. 2018. Offline a/b testing for recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 198–206.

[17] Carlos A Gomez-Uribe and Neil Hunt. 2016. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)* 6, 4 (2016), 13.

[18] Ruining He and Julian McAuley. 2016. Fusing Similarity Models with Markov Chains for Sparse Sequential Recommendation. In *International Conference on Data Mining*. IEEE.

[19] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *International Conference on World Wide Web*. ACM, 173–182.

[20] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[21] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[22] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18, 1 (2017), 6869–6898.

[23] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation* 3, 1 (1991), 79–87.

[24] Dietmar Jannach and Malte Ludewig. 2017. When Recurrent Neural Networks meet the Neighborhood for Session-Based Recommendation. In *ACM Conference on Recommender systems*. ACM, 306–310.

[25] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2014. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007* (2014).

[26] Michael I Jordan and Robert A Jacobs. 1994. Hierarchical mixtures of experts and the EM algorithm. *Neural computation* 6, 2 (1994), 181–214.

[27] Dan Jurafsky and James H Martin. 2014. *Speech and language processing*. Vol. 3. Pearson London.

[28] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. 2014. Large-scale video classification with convolutional neural networks. In *IEEE conference on Computer Vision and Pattern Recognition*.

[29] Yehuda Koren. 2009. Collaborative filtering with temporal dynamics. In *ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM.

[30] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1097–1105.

[32] Mu Li, David G Andersen, and Jun Woo Park. [n. d.]. Scaling Distributed Machine Learning with the Parameter Server.. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation*.

[33] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. 2018. Variational Autoencoders for Collaborative Filtering. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 689–698.

[34] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* (2003).

[35] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H. Chi. 2018. Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1930–1939.

[36] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model.. In *Interspeech*.

[37] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*. 807–814.

[38] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*. 1310–1318.

[39] Michael J Pazzani and Daniel Billsus. 2007. Content-based recommendation systems. In *The adaptive web*. Springer, 325–341.

[40] Pedro HO Pinheiro and Ronan Collobert. 2014. Recurrent convolutional neural networks for scene labeling. In *International Conference on Machine Learning*.

[41] Vladas Pipiras and Murad S Taqqu. 2017. *Long-range dependence and self-similarity*. Vol. 45. Cambridge university press.

[42] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. 2017. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *ACM Conference on Recommender Systems*. ACM, 130–137.

[43] Steffen Rendle. 2010. Factorization machines. In *International Conference on Data Mining*. IEEE, 995–1000.

[44] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 452–461.

[45] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *International Conference on World Wide Web*. ACM, 811–820.

[46] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *International Conference on World Wide Web*. ACM, 285–295.

[47] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders meet collaborative filtering. In *International Conference on World Wide Web*. ACM, 111–112.

[48] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017).

[49] Elena Smirnova and Flavian Vasile. [n. d.]. Contextual Sequence Modeling for Recommendation with Recurrent Neural Networks. *arXiv preprint arXiv:1706.07684* ([n. d.]).

[50] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*.

[51] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 565–573.

[52] Jiaxi Tang and Ke Wang. 2018. Ranking Distillation: Learning Compact Ranking Models With High Performance for Recommender System. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2289–2298.

[53] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. 2016. WaveNet: A generative model for raw audio.. In *SSW*. 125.

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 5998–6008.

[55] Chen Wu and Ming Yan. 2017. Session-aware information embedding for e-commerce product recommendation. In *ACM on Conference on Information and Knowledge Management*. ACM, 2379–2382.

[56] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. 2017. Recurrent Recommender Networks. In *International Conference on Web Search and Data Mining*. ACM, 495–503.

[57] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *International Conference on Web Search and Data Mining*. ACM, 153–162.

[58] Fisher Yu and Vladlen Koltun. 2015. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122* (2015).

[59] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. 2019. A Simple Convolutional Generative Network for Next Item Recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 582–590.

[60] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2018. Deep Interest Evolution Network for Click-Through Rate Prediction. *arXiv preprint arXiv:1809.03672* (2018).

[61] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1059–1068.

[62] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. 2002. Ensembling neural networks: many could be better than all. *Artificial intelligence* 137, 1-2 (2002), 239–263.

[63] Yu Zhu, Junxiong Zhu, Jie Hou, Yongliang Li, Beidou Wang, Ziyu Guan, and Deng Cai. 2018. A Brand-level Ranking System with the Customized Attention-GRU Model. *arXiv preprint arXiv:1805.08958* (2018).