DynamicRec: A Dynamic Convolutional Network for Next Item Recommendation

Md Mehrab Tanjim
mtanjim@eng.ucsd.edu
UC San Diego

Hammad A. Ayyubi
hayyubi@eng.ucsd.edu
UC San Diego

Garrison W. Cottrell
gary@eng.ucsd.edu
UC San Diego

ABSTRACT

Recently convolutional networks have shown significant promise for modeling sequential user interactions for recommendations. Critically, such networks rely on fixed convolutional kernels to capture sequential behavior. In this paper, we argue that all the dynamics of the item-to-item transition in session-based settings may not be observable at training time. Hence we propose **DynamicRec**, which uses dynamic convolutions to compute the convolutional kernels on the fly based on the current input. We show through experiments that this approach significantly outperforms existing convolutional models on real datasets in session-based settings.

KEYWORDS

Session-Based Recommendation, Dynamic Convolutions

ACM Reference Format:
Md Mehrab Tanjim, Hammad A. Ayyubi, and Garrison W. Cottrell. 2020. DynamicRec: A Dynamic Convolutional Network for Next Item Recommendation. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3340531.3412118

1 INTRODUCTION

Everyday users interact with a variety of items on different e-commerce platforms, performing a dynamic sequence of actions. These platforms usually use session keys to record these sequential interactions within a short time period. In this setting, the goal of a recommender system is to capture the sequential dynamics from the short sequences and predict the item that a user will most likely interact with next. With the advent of recurrent neural networks (RNNs) and convolutional networks (CNNs), there has been an increasing interest in incorporating them into this session-based recommendation. The advantage of these models is that they can capture the sequential dynamics by summarizing previous actions via a hidden state (RNNs), or through convolutions over a fixed number of past actions (CNNs). For example, GRU4Rec uses Gated Recurrent Units (GRU) to model click sequences for session-based recommendation [6]. In each step, RNNs take the state from the last step and current action as its input. These dependencies make RNNs more complex and less efficient than CNNs, though techniques like "session parallelism" have been proposed to improve efficiency [6].

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10.
https://doi.org/10.1145/3340531.3412118

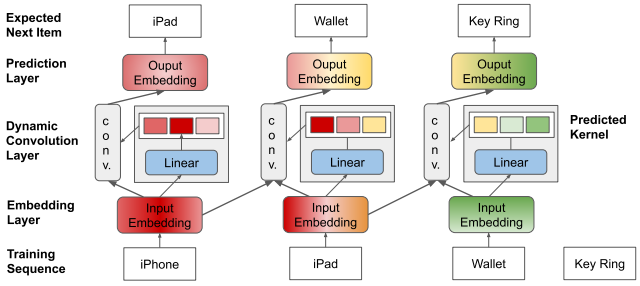

Figure 1: A simplified diagram showing the architecture of DynamicRec. At each time step, the model dynamically changes the kernel based on the current input and then applies the computed convolution on a fixed number of items (in this case, two) to predict the next item.

Recently, there has been a strong effort in the community to apply CNNs to this problem. The advantages of convolutional operations are straightforward: they require a smaller number of parameters while allowing parallel computation and are thus able to offer several orders of magnitude of speedup. This proves to be useful when deploying the model at industry-level scale. To this end, various convolutional models have been developed for recommender systems. For example, Convolutional Sequence Embedding (Caser), a CNN-based method, views the embedding matrix of L previous items as an "image" and applies convolutional operations to extract transitions [10]. However, treating the matrix as a static image may hinder capturing the dynamics of the item-to-item transitions. Recently, NextItNet [13] treats this problem as a sequence, and applies 1-D dilated convolutions to get the context of item-to-item transitions in the session-based setting.

The current state-of-the-art CNN models make the assumption that user preferences within a session are static enough to be captured by fixed kernels. However, the user preference in a particular session may change drastically. To illustrate this, take the example of a user who wants to buy different gifts for her family members (Figure 1). At first, she purchases two electronic products, namely an iPhone and an iPad. Then, in the same session, she changes her intention towards a different category of items, and purchases two non-electronic items such as a wallet and a key ring. This kind of randomness in the transitions makes it difficult for a system expecting strong sequential patterns to adapt quickly. Furthermore, in the session-based setting, we typically do not have a user model. This makes it difficult to generalize from current sequence only during test time.

To tackle such challenges in session-based settings, in this work, we seek to design a model that can adapt quickly to the dynamics of the interaction. In the Computer Vision community, Dynamic

Filter Networks [2] have been proposed to deal with dynamic data such as video and stereo. Recently, for various sequential NLP tasks, similar dynamic convolutions has been proposed as well [12]. The basic idea is to use a filter-generating network to compute the filter or convolution conditioned on the current input. This allows it to quickly adapt to changing dynamics.

Inspired by its success in the vision and NLP domains, we propose to use dynamic convolutions for the next item recommendation task in session-based settings. Figure 1 shows the conceptual diagram of our model, which we call **DynamicRec**. At each step, the model predicts a convolutional kernel based on the current input, which is then applied to a fixed number of previous items. This allows the model to correct or adapt its kernel solely based on current input, without requiring any prior metadata or user information. Because the kernel is dynamic, DynamicRec is adaptive to the current sequence of operations, which is useful in session-based settings as new users come to the system and interact with different sets of items. To demonstrate the effectiveness of dynamic convolutions in next item recommendation, we experiment with four real datasets: two from e-commerce platforms and two from music streaming media. Our experimental results show that DynamicRec performs better than the current state-of-the-art convolutional model, and is overall state of the art on three out the four datasets.

2 DYNAMICREC

Problem Formulation: In session-based settings, we have a sequence of user actions (e.g., click, purchase) $\mathbf{S} = (a_1, a_2, \ldots, a_n)$, where each action a_i contains an item id. In each step, the goal of a sequential recommendation is to predict the next action, a_n, based on $a_1, \ldots, a_{n-1}$ actions. We design our sequential model with three main layers: input embedding layer, dynamic convolutional layer, and prediction layer. Figure 2 shows the different components of our model. We describe these components of our model below.

Embedding Layer: In this layer, we embed the one-hot item features into a d-dimensional space, using a learned embedding matrix $W_e \in \mathbb{R}^{d \times V}$, where V is the number of items. This transforms the training sequence $\mathbf{S} = (a_1, a_2, \ldots, a_n)$ into $\mathbf{E} = (e_1, e_2, \ldots, e_n)$, where $e_i \in \mathbb{R}^d$ and n represents the maximum length that our model can handle. If the sequence length is greater than n, we consider the most recent n actions. If shorter, we pad with 0's.

Convolutional Layer: From the embeddings $(e_1, e_2, \ldots, e_{n-1})$, our goal is to predict e_n. For sequences, we can perform convolutions in a 1D space (i.e., a sequence) using a fixed kernel size which slides over the input sequence, $e_{1:n-1}$, and determines the importance of context elements via a set of weights. If the kernel length for the convolution is l then the standard convolution operation would require dl parameters for each output channel. If our output's embedding size (or number of channels) is also d, then the standard convolution operation would require $d^2 l$ parameters. To reduce the number of parameters, we use depthwise convolutions, where a fixed kernel of size l is applied independently for each of the d channels, requiring dl parameters. By using the same kernel for all channels this can be further reduced to just l parameters. Such a vast reduction in the number of parameters makes implementing dynamic convolutions feasible without incurring a lot of computational resources [12]. Let this weight be $\mathbf{w}$. Given the kernel or weight $\mathbf{w}$, the output for each element i in the input sequence $\mathbf{E}$ is

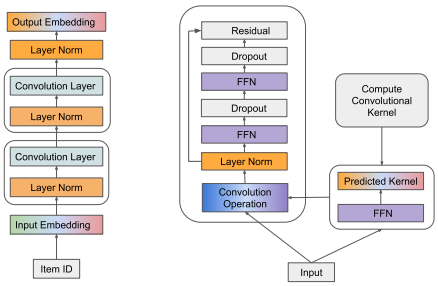

Figure 2: Left: Architecture of the overall model (here we only show the flow for a single input). Right: Components of the convolutional layer in the model.

calculated as follows:

$$\text{DepthwiseConv}(\mathbf{E}, \mathbf{w}, i, c) = \sum_{j=1}^{l} \mathbf{w}_j * \mathbf{E}_{(i+j-\lfloor l+1 \rfloor/2), c} \quad (1)$$

Here c is the current channel. As mentioned before, each dimension of the latent space can be considered as a channel, so usually $max(c) = d$. This has a time complexity of $O(ndl)$, where n=sequence length, d=embedding dimensions, and l=kernel size.

Dynamic convolutions make fixed kernels of depthwise convolutions a function of the input at each time step, making it dynamic and adaptive. Specifically, the fixed kernel weights $\mathbf{w}$ are now computed using a linear layer:

$$\text{DynamicConv}(\mathbf{E}, i, c) = \text{DepthwiseConv}(\mathbf{E}, f(\mathbf{E}_i), i, c) \quad (2)$$

Here, f is modeled using a learnable weight matrix $\mathbf{W}_f \in \mathbb{R}^{d \times l}$. Specifically, $f(\mathbf{E}_i) = \mathbf{W}_f^T \mathbf{E}_i$ and has a complexity of $O(ndl)$.

This proves to be useful in the session-based setting as we need to adapt the kernels dynamically based on the current input item. To introduce non-linearity into the model, the next step is to feed the output from the DynamicConv model to a one-hidden layer feed-forward network with ReLU activation applied to the hidden layer, a linear output layer, and dropout applied to both layers [9]. To facilitate the training process, we also apply layer normalization [1] before this network and use residual connections [3] (see Figure 2, right). Layer normalization is also applied before each convolution block and after the final one. We do an ablation study to show the effectiveness of each module. The linear layers have complexity of $O(nd^2)$, and so overall complexity of our model is $O(ndl + nd^2)$. As $l < d$, our dynamic convolution is more efficient than linear layers. Also, notice that our model scales linearly in n (length of sequence).

Prediction layer: Proceeding from the output of the final convolution layer (o) at the nth time step, i.e. o_n, we calculate the prediction scores as follows: we use the cosine similarity measure to calculate similarity scores between the predicted embedding, i.e., o_n and the ground truth, e_n, i.e., $r_{o_n} = cos(o_n, e_n)$. We also compute this score for negative samples selected randomly from the total vocabulary of items. Then the loss function is:

$$\mathcal{L}_{ranking} = -\sum_{\mathbf{E} \in U} \sum \left[\log(\sigma(r_{o_n})) + \sum_{j \notin \mathbf{E}} \log(\sigma(1 - r_{o_j})) \right] \quad (3)$$

	YooChoose	Last FM	NowPlaying	Diginetica
#Sessions	80,183	196,010	113,918	63,466
#Items	12,936	107,391	239,221	38,970
#Total actions	406,979	3,419,953	1,184,815	557,048
#Avg. length	5.07	17.45	10.40	8.70

Table 1: Data Statistics (after preprocessing).

where U is the set of all session sequences, r_{o_n} is the ground truth score, r_{o_j} is the negative item score, and σ is the logistic function. This is also known as the point-wise loss function [4, 11].

3 EXPERIMENTS

3.1 Datasets

We use the same two datasets as used by our convolutional model baseline [13]: YooChoose and LastFM. Additionally, we consider two more datasets: Diginetica and NowPlaying.

YooChoose: This dataset is chosen from the RecSys Challenge 2015 [1]. We use the buying dataset for evaluation.

LastFM: This dataset is collected from Last.fm, a music streaming website.

Diginetica: This dataset includes user sessions extracted from the e-commerce search engine logs of Diginetica. We use the purchase data for our experiments.

NowPlaying: This dataset is created from music-related tweets, where users post which tracks they are currently listening to.

We preprocess each dataset as follows: any session with less than 5 items were removed. Table 1 gives the statistics of each datasets after the preprocessing. Following the settings of other session-based sequential models [13], 70% of the sessions are used for training, 10% for validation, and 20% for testing.

3.2 Implementation Details and Metrics

We implement our model in PyTorch[2]. The maximum length (n) for YooChoose and Diginetica is set to 10 and for LastFM and NowPlaying, we set it to 30, given their longer sequence lengths (Table 1). After grid search on the validation set, the other parameters are set as follows: dropout of 0.2, the embedding dimension and feed-forward network width is set to 200, the learning rate is set to 0.001, and the batch size is 128. We use 2 layers of dynamic convolution, each having a filter size l of 5. We use the Adam optimizer [8]. We evaluate the models using the two most popular Top-N metrics in recommender systems: Hit Rate and Normalized Discounted Cumulative Gain (NDCG) [4]. Hit Rate at k is the fraction of times the ground-truth next item is among the top k predicted items, while NDCG at k is a position-aware metric which assigns larger weights to higher positions. We report values for both $k = 10$ and $k = 20$.

3.3 Comparison Models

We compare our models to the state-of-the-art deep learning models with three different backbone architectures: recurrent neural network, convolution and self-attention. In each case we used the code published by the authors.

- **MostPop:** MostPop is a very simple baseline which stands for most popular items. We simply check if the correct item is in the top k most popular items.
- **GRU4Rec$^+$:** A state-of-the-art model which uses an RNN to capture sequential dependencies and make predictions [5].
- **NextItNet:** This is a recently-proposed convolutional model for next item recommendation in session-based settings[13].
- **SASRec:** This is a sequential model based on self-attention for next item recommendation [7].
- **DynamicRec:** Our method.

3.4 Performance

The performance results are presented in Table 2. We observe that MostPop performs the worst for all dataset under all metrics, as expected. DynamicRec performs better than previous SOTA models across all metrics by large margins on three of the datasets, while GRU4Rec$^+$ performs better on NowPlaying. The performance improvements achieved by DynamicRec over NextItNet, which uses fixed convolutional filters, highlights the advantages of using dynamic kernels that can adapt during test time.

We hypothesize that the superior performance of GRU4Rec$^+$ on NowPlaying can be attributed to its more sophisticated sampling strategy. Under this strategy, for each item in a session, every other item in other sessions in the same mini-batch are used as negative samples. If the item-space becomes relatively large, this sampling strategy might lead to better performance, as mentioned in their paper. From Table 1, we can see that the NowPlaying dataset has the largest number of items compared to the other datasets, so GRU4Rec$^+$'s sampling strategy could be the reason for its superior performance on this dataset.

Our model only does random sampling without any special conditions - i.e., we treat any item other than the ground truth item as a negative item. This might be the reason for the performance drop for NowPlaying. We will explore other sampling strategies in DynamicRec as a future direction. However, we emphasize that, when compared to a fixed kernel convolution method, namely NextItNet, our model always performs better, which is the main experimental result we wanted to show.

3.5 Ablation Study

In this experiment, we have done an ablation study of various components of DynamicRec. We report the results on the LastFM dataset in Table 3 as this is the largest among four. Results from other datasets are similar.

The most dramatic drop in performance is when we remove the convolutional layers, and only use feedforward networks, showing the importance of the convolutional filters. Similarly, making the convolution use a fixed kernel also results in a large drop in performance, demonstrating the power of adaptive convolutions. We also see that we achieve better performance with two layers of convolution instead of one. Preliminary experiments showed no significant improvement when we added a third layer.

Aside from the convolutional components, the residual connections were clearly important, as they reduce the vanishing gradients problem, as has been shown in many other models. Finally, LayerNorm appears to be the least important component.

[1] https://2015.recsyschallenge.com/challenge.html
[2] Our source code is available at: https://github.com/Mehrab-Tanjim/DynamicRec

Data	Metric	k	MostPop	GRU4Rec$^+$	SASRec	NextItNet	DynamicRec	Improvement
YooChoose	HitRate	10	0.0326	0.4269	0.3854	0.3757	**0.4685**	9.74%
		20	0.0566	0.5143	0.4974	0.4677	**0.5549**	7.89%
	NDCG	10	0.0150	0.1954	0.2079	0.2209	**0.2310**	4.57%
		20	0.0199	0.2170	0.2363	0.2443	**0.2554**	4.54%
LastFM	HitRate	10	0.0324	0.2023	0.1639	0.2064	**0.2430**	17.73%
		20	0.0598	0.2792	0.2408	0.2835	**0.3446**	21.55%
	NDCG	10	0.0167	0.0896	0.0936	0.1240	**0.1413**	13.95%
		20	0.0251	0.1086	0.1129	0.1433	**0.1662**	15.98%
Diginetica	HitRate	10	0.0039	0.2577	0.2553	0.1055	**0.3628**	40.78%
		20	0.0073	0.3654	0.3603	0.1605	**0.4705**	28.76%
	NDCG	10	0.0016	0.1084	0.1560	0.0538	**0.2180**	39.74%
		20	0.0025	0.1350	0.1823	0.0676	**0.2452**	34.50%
NowPlaying	HitRate	10	0.0088	**0.2029**	0.0964	0.1072	0.1713	-15.57%
		20	0.0136	**0.2379**	0.1309	0.1308	0.2296	-03.49%
	NDCG	10	0.0038	**0.1036**	0.0545	0.0825	0.0865	-16.50%
		20	0.0050	**0.1123**	0.0631	0.0884	0.1012	-09.88%

Table 2: Performance of all models

	NDCG@20	HitRate@20
1. Default	**0.1662**	**0.3446**
2. Fixed Convolution	0.1493	0.2824
3. No Convolution	0.0713	0.1822
4. Remove Residual Connection	0.1219	0.2617
5. Remove Layer Norm	0.1552	0.3162
6. Single Layer	0.1536	0.3127

Table 3: Ablation Study for Last FM dataset

#Neg. Samples	NDCG@20	HitRate@20
1	0.1479	0.2848
10	0.1524	0.3048
50	0.1603	0.3271
100	0.1614	0.3340
200	0.1659	0.3434
400	**0.1662**	**0.3446**

Table 4: Impact of negative sampling

3.6 Effect of negative sampling

Negative Sampling is an important factor in our model. It denotes how many negative items we randomly sample against each ground truth to optimize our loss function (Equation 3). Table 4 shows the impact of increasing the number of negative samples for the case of LastFM (the results for other datasets are similar). We see a monotonic improvement in performance as we increase the number of negative samples, with diminishing returns after 200 samples. To keep the model complexity low, we set the negative sample number to 400. We should note that this number is still quite low compared to other convolutional models that optimize for classification loss and consider all items.

4 CONCLUSION

In this work, we proposed DynamicRec, a convolutional model that can change its kernel dynamically based on the current input. In session-based settings, where items change dynamically, our model makes significant improvements over fixed-kernel convolution model approaches. Experimental results with four real datasets further validate the effectiveness of our approach. Our model performs better than the current state-of-the-art convolutional model on all datasets, and achieves state-of-the-art performance on three out of the four datasets.

REFERENCES

[1] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. CoRR abs/1607.06450. *Bulletin of Mathematical Linguistics (Proc. EAMT)(108)* (2016), 49–60.

[2] Bert De Brabandere, Xu Jia1, Tinne Tuytelaars, and Luc Van Gool. 2016. Dynamic Filter Networks. In *Advances in Neural Information Processing Systems*. 667–675.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR-16)*. 770–778.

[4] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 173–182.

[5] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 843–852.

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

[7] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.

[8] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[10] 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.

[11] Shoujin Wang, Liang Hu, Longbing Cao, Xiaoshui Huang, Defu Lian, and Wei Liu. 2018. Attention-based transactional context embedding for next-item recommendation. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[12] Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. 2019. Pay Less Attention with Lightweight and Dynamic Convolutions. *arXiv preprint arXiv:1901.10430* (2019).

[13] 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.