FLASH4Rec: A Lightweight and Sparsely Activated Transformer for User-Aware Sequential Recommendation

Yachen Yan
yachen.yan@creditkarma.com
Credit Karma
San Francisco, California, USA

Liubo Li
liubo.li@creditkarma.com
Credit Karma
San Francisco, California, USA

ABSTRACT

Transformers have been extensively applied to model users' dynamic and evolving historical behaviors for the sequential recommendation. Despite their success, the current Transformer architecture applied to sequential recommendation faces two major challenges: 1) the high computational complexity and over-parameterization result in substantial computational cost and memory requirement. 2) The self-attention mechanism tends to overly rely on high-attributed attention positions, leading to a higher risk of overfitting. The long-tail distribution of item popularity further intensifies this issue. To address these issues, we present FLASH4Rec, an efficient transformer architecture that utilizes gated attention layer and Sparsely-Gated Mixture of Experts (MoE) layer to replace the multi-head self-attention layer and over-parameterized dense feed-forward network, respectively. Additionally, we propose Top-K Dropout to regularize the attention weights and encourage reliance on low-attention positions, reducing the risk of overfitting. Extensive experimental studies are conducted on real-world datasets, and further prove the effectiveness and efficiency of FLASH4Rec.

CCS CONCEPTS

• **Computing methodologies**; • **Machine learning**; • **Machine learning approaches**; • **Neural networks**;

KEYWORDS

Transformers, BERT, Mixture of Experts, Sequential Recommendation

ACM Reference Format:
Yachen Yan and Liubo Li. 2023. FLASH4Rec: A Lightweight and Sparsely Activated Transformer for User-Aware Sequential Recommendation. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

A sequential recommender system is a type of recommender system that utilizes historical user-item interactions to make subsequent user-item recommendations by modeling the sequential dependencies between them. As one of the essential methods for large-scale neural retrieval systems, various methods have been designed for sequential recommenders, including SASRec [5] and BERT4rec [11], both are based on self-attention mechanism and the Transformer architecture.

We introduce a new architecture called FLASH4Rec, which efficiently models item dependencies in users' historical behavior sequences using a Gated Attention Layer and a Sparsely-Gated Mixture-of-Experts Layer. The Gated Attention Layer computes user-aware item sequence representations, while the SparseMoE Layer increases the model's capacity without increasing computational costs. To prevent overfitting, we include a Top-K Dropout mechanism that encourages the model to learn from long-tail attention positions.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 PROPOSED METHOD

In sequential recommendation setup, given the user u's historical interaction sequence $\mathcal{S}_u$=$[i_1^u, \ldots, i_t^u]$, we aim to predict the item that user u will interact with at time step $t+1$: i_{t+1}^u. In this work, we propose FLASH4Rec, which leverages the gated attention layer and Sparsely-Gated Mixture of Experts (MoE) layer to model the items' dependencies. The overall architecture of FLASH4Rec is shown in Figure 1.

2.1 Embedding Layer

Our recommender maintains an item embedding table $\mathbf{T}_I \in \mathbb{R}^{|\mathcal{I}| \times d}$, where d is the size of the embedding. For each user's historical interaction sequence $[i_1^u, i_2^u, \ldots, i_t^u]$, where t is the maximum time length. The embedding for item sequence $[i_1^u, i_2^u, \ldots, i_t^u]$ is denoted as $\mathbf{E}_i \in \mathbb{R}^{t \times d}$, where d is the size of the embedding.

Similarly, we also maintain a user embedding table $\mathbf{T}_U \in \mathbb{R}^{|\mathcal{U}| \times d}$. Given a user id, we can retrieve the user embedding vector $\mathbf{e}_u \in R^d$.

2.2 Gated Attention Layer

Inspired by [4], we utilize the gated attention layer to generate user-aware representations. The attentive gating mechanism can achieve higher parameter and computing efficiency without reducing performance. The overall architecture of FLASH4Rec is shown in Figure 2.

Firstly, we project the item sequence embedding $X_I \in \mathbb{R}^{T \times d}$ to a shared representation Z with time length T and embedding dimension k. $\mathcal{Q}$ and $\mathcal{K}$ are transformations that apply per-dimension scales and offsets to Z. Then, we apply the scaled dot-product attention to compute the attention weights $A \in \mathbb{R}^{T \times T}$:

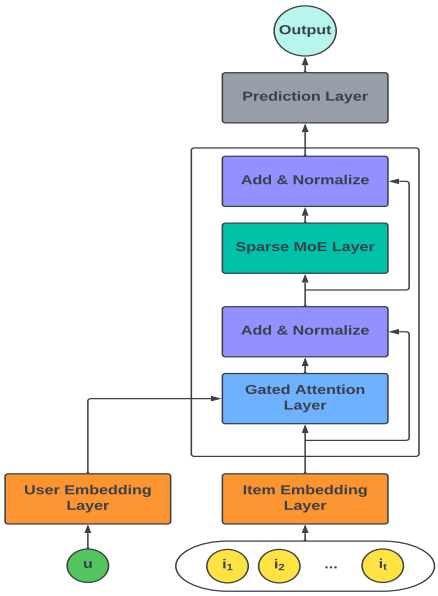

Figure 1: The Architecture of FLASH4Rec

In this example, we only include a single transformer layer consisting of GAU, MoE, and Normalization layer. While in practice, it is usually beneficial to stack multiple transformer layers for learning more complicated item relevance patterns.

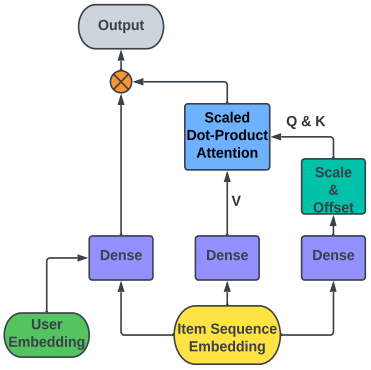

Figure 2: The Architecture of Gated Attention Layer

$$Z = \sigma(X_I W_z) \qquad \in \mathbb{R}^{T\times k} \tag{1}$$

$$A = \text{softmax}(\frac{Q(Z)\mathcal{K}(Z)^\top}{\sqrt{d_k}}) \qquad \in \mathbb{R}^{T\times T} \tag{2}$$

where W_z denotes the kernel weights of the projection layer for both query and key, and $\frac{1}{\sqrt{d_k}}$ is the scaling factor.

In order to model the sequential relations of the user's interacted items and adapt to varying lengths of sequences, we apply the rotary position embedding [10] to capture the sequential information. Suppose $R^d_{\Theta,m}$ is the rotary matrix with parameters $\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \cdots, d/2]\}$, Applying the rotary position embedding to the self-attention in Equation (2), we obtain the attention weights with the rotary position embeddings:

$$\begin{aligned} A^{\text{rotary}}_{m,n} &= \text{softmax}\Big(\frac{(Q(Z)_m R^d_{\Theta,m})(\mathcal{K}(Z)_n R^d_{\Theta,n})^\top}{\sqrt{d_k}}\Big) \\ &= \text{softmax}\Big(\frac{(Q(Z)_m R^d_{\Theta,m-n}\mathcal{K}(Z)^\top_n)}{\sqrt{d_k}}\Big) \qquad \in \mathbb{R}^{T\times T} \end{aligned} \tag{3}$$

where $A^{\text{rotary}}_{m,n}$ is the m,n-th element of A^{rotary}, $Q(Z)_m$ is the m-th row of $Q(Z)$ and $\mathcal{K}(Z)_n$ is the n-th row of $\mathcal{K}(Z)$.

Secondly, we also project the item sequence embedding X_I to V. To enhance the representation capacity, we concatenate the item sequence embedding $X_I \in \mathbb{R}^{T\times d}$ and user embedding $X_U \in \mathbb{R}^d$ along the time sequence dimension and further compute the gating weights U. In this way, gating weights U vary over different user embedding X_U and thus become user-aware. The mathematical representation of the gating weights U is given by:

$$V = \sigma(X_I W_v) \qquad \in \mathbb{R}^{T\times k} \tag{4}$$

$$U = \sigma(\text{Concat}(X_I, X_U) W_u) \qquad \in \mathbb{R}^{T\times k} \tag{5}$$

where W_v denotes the kernel weights of the projection layer for value. The concatenation operation with broadcasting concatenates item sequence embedding and user embedding to representation with dimension $T \times 2d$. The W_u denotes the kernel weights of the projection layer for gating weight.

Finally, we multiply gating weights U with the self-attention output element-wise to compute the user-aware item sequence embedding:

$$O = U \otimes AV \tag{6}$$

The Gated Attention Layer has single-head structure but exhibits competitive performance comparing with multi-head self-attention, while being more efficient via shared representation and efficient per-dimension scaling and offsetting. It fuses user embedding information into gating weights to create user-aware item sequence representations, improving the accuracy and relevance of recommendations.

2.3 Sparse Mixture-of-Experts Layer

Feed-Forward Network is another essential component to Transformer [2], other than multi-head self-attention. To increase the model's capacity, we drew inspiration from Switch Transformers [3] and incorporated a sparsely-activated mixture-of-experts layer. This layer includes several key elements: a noisy gating network, a sparse dispatcher, and load balancing regularization. Figure 3 displays the overall architecture of the SparseMoE layer.

2.3.1 Noisy Gating Network. The gating network essentially computes the gating value for selecting experts for each item representation.

For the input item embedding of gating network X_I, it is firstly processed by the gating network: a two-layer feed-forward network

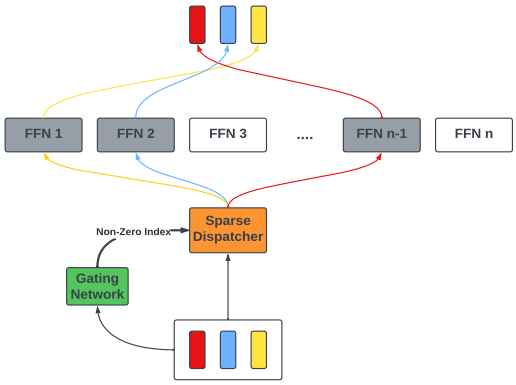

Figure 3: The Architecture of Sparse Mixture-of-Experts Layer Layer

In this example, the gating network selects the expert for each item embedding. The sparse dispatcher further dispatch each item embedding to corresponding FFN to independently compute the outputs.

with non-linear activation function computing the gating logits $h \in R^N$ which are normalized via a softmax function over the available N experts. Additionally, we applied multiplicative jitter noise to introduce exploration and promoting load balancing between different experts.

$$h = \text{FFN}(X_I \circ \text{RandomUniform}(1.0 - \text{eps}, 1.0 + \text{eps})) \tag{7}$$

For the computed routing score h, we only keep the top k values and set the rest to $-\infty$, resulting in the corresponding softmax gating values equal 0. The i-th element of the output vector of the gating network is

$$g_i = \frac{\exp\left(\text{TopK}(h,k)_i\right)}{\sum_{i=1}^{N} \exp\left(\text{TopK}(h,k)_i\right)}, \tag{8}$$

where

$$\text{TopK}(h,k)_i = \begin{cases} h_i & \text{if } h_i \text{ is in the top } k \text{ elements of } h \\ -\infty & \text{otherwise.} \end{cases} \tag{9}$$

The sparse dispatcher will use these gating values for routing item embedding to different experts. This is the essential step for achieving the sparsity of our Sparse Mixture-of-Experts layer. Note that the g is differentiable regardless of the value of k[3]. For simplicity and computational efficiency, we keep k as 1 throughout this paper.

2.3.2 Sparse Dispatcher. The sparse dispatcher [3, 9, 14] takes the input gating values and experts as input. It dispatches item representations to the experts corresponding to the non-zero gating value and lets experts generate the refined item embeddings. The output y of the Sparse Mixture-of-Experts layer is the linearly weighted combination of expert output embeddings by the non-zero gating values.

$$y = \sum_{i \in \phi} g_i FFN_i(X_I) \tag{10}$$

Where ϕ denotes the selected non-zero indices. Wherever $g_i = 0$, we don't pass the input to the corresponding expert and thus achieving computing efficiency via sparse activation.

2.3.3 Load Balancing Regularization. As stated in the previous research [3, 9, 14], the gating network tends to select only a few experts if no regularization is applied. This phenomenon is self-reinforcing since the selected experts are trained more and will be selected more frequently by the gating network. Therefore, the load balancing loss is applied to enforce the uniform expert routing.

$$L_{\text{balance}} = \lambda \cdot N \cdot \sum_{j=1}^{N} f_j \cdot P_j \tag{11}$$

where N is the number of experts, f_j is the fraction of item embeddings dispatched to expert j, P_j is the average of the router probability allocated for expert j, and λ is the coefficient for the regularization term. In practice, the λ should be sufficiently large to prevent expert selection self-reinforcing phenomenon at the initial training stage while not overwhelming the primary objective.

2.4 Top-K Dropout

Over-parameterization of Transformers for recommendation tasks can lead to overfitting with limited training data. Additionally, the long-tail distribution of item popularity can cause an imbalance in self-attention weights, resulting in over-weighting for short-tail item embeddings and under-weighting for long-tail item embeddings.

We use Top-K Dropout to mitigate this overfitting issue. This regularization approach randomly drops some high-attention positions from self-attention weights, encouraging the model to learn from low-attention positions. It identifies Top-K positions in each row of self-attention weight matrix and masks a portion of them with probability p.

Formally given a self-attention weight matrix $A \in \mathbb{R}^{T \times T}$, we firstly compute its Top-K position indicator S_A, in which each element $S_{i,j}$ is defined as:

$$S_{i,j} = \begin{cases} 1 & \text{if } A_{i,j} \text{ is in the top } k \text{ elements of } A_{i,\cdot} \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

Next, we want to randomly dropout self-attention weights within the Top-K positions to produce the Top-K mask matrix M_A with dropout rate p:

$$M_{i,j} = \begin{cases} 0 & \text{if } s_{i,j} * Bernoulli(p) = 1 \\ 1 & \text{otherwise,} \end{cases} \tag{13}$$

Similar to the inverted dropout, the Top-K dropout re-scales the self-attention weights to ensure consistency between training and serving. After the dropout is applied, we re-scale the self-attention weights by scaling factor f:

$$f = \frac{1}{1.0 - (\sum_{i=1}^{T}\sum_{j=1}^{T} A_{i,j} * M_{i,j} / \sum_{i=1}^{T}\sum_{j=1}^{T} A_{i,j})} \tag{14}$$

Note that we block the gradient when applying the scaling factor to avoid the co-adaptation issue. In the training phase, we forward propagate $A \cdot M \cdot \text{BlockGradient}(f)$ to mask the high-attention position for regularization while re-scaling to ensure the training and serving consistency. In the serving phase, we forward propagate original self-attention weights A.

2.5 Prediction Layer and Loss Function

After stacked L enhanced Transformer blocks and given the item interaction sequence with length t, we apply a two-layer feed-forward network predict on the last layer's output of the t-th item and obtain the F_t^L to predict the next item i_{t+1}. In this work, we use inner product to measure the relevance of item i as:

$$r(i_{t+1}^u | [i_1^u, i_2^u, \ldots, i_t^u]) = \langle \mathbf{F}_t^L, \mathbf{e}_i \rangle \tag{15}$$

where $\mathbf{e}_i \in \mathbb{R}^d$ is the embedding of item i. Finally, we utilize the softmax cross-entropy loss to train our model:

$$Loss = \frac{exp(\langle \mathbf{F}_t^L, \mathbf{e}_g \rangle)}{\sum_{i=1}^{I} \langle \mathbf{F}_t^L, \mathbf{e}_i \rangle} \tag{16}$$

where item g is the ground truth item, $|I|$ is the number of items. In practice, when the item corpus is large, one can use sampled softmax to further improve the training efficiency.

3 EXPERIMENT

3.1 Experiment Settings

3.1.1 Datasets and Evaluation Metrics. We conducted an experiment using three real-world datasets: ML-1M, ML-20M, and Yelp. We used a Leave-One-Out evaluation strategy to split each dataset into train, validation, and test partitions. To create the validation set, we selected the second last item interaction of randomly selected 1024 users from each dataset. For the test set, we hold the final interaction for each user; the remaining user-item interactions are used for training. We evaluated our model mainly based on two ranking-based metrics: Recall@10 and NDCG@10.

3.1.2 Baseline Models. We consider following baseline models to compare with FLASH4Rec: MF-BPR, GRU4Rec, SASRec and BERT4Rec.

3.2 Model Performance Comparison

3.2.1 Evaluation on Effectiveness. The models' ranking performance on each dataset is listed in Table 1. We can observe that our model FLASH4Rec outperforms existing methods across different datasets. The superior performance mainly benefits from the key components of our architecture, including Gated Attention Layer, Sparsely Activated Mixture-of-Experts Layer, and Top-K Dropout. We will conduct a more detailed analysis in later section.

Table 1: Performance Comparison of Different Algorithms on ML-1M, ML-20M and Yelp Dataset.

	ML-1M		ML-20M		Yelp	
Model	Recall@10	NDCG@10	Recall@10	NDCG@10	Recall@10	NDCG@10
MF-BPR	0.0740	0.0377	0.0807	0.0407	0.0191	0.0092
GRU4Rec	0.2132	0.1093	0.1544	0.0839	0.0113	0.0048
SASRec	0.1993	0.1078	0.1439	0.0724	0.0146	0.0076
BERT4Rec	0.2584	0.1392	0.2393	0.1310	0.0149	0.0079
FLASH4Rec	**0.2841**	**0.1568**	**0.2554**	**0.1487**	**0.151**	**0.0081**

3.2.2 Evaluation on Efficiency. The efficiency is compared between BERT4Rec and FLASHRec, mainly since their architectures are very similar and BERT4Rec is the most performant model among all the baseline models. The memory cost is measured with the number of parameters. The computation cost is measured with floating point operations per second (FLOPs). In this section, we use ML-1M as our benchmark dataset since the only major difference between the same architecture trained on different datasets is the embedding table. Our main goal is to compare different transformer modules. The models' efficiency performance is listed in Table 2.

We can observe that even with almost same amount of parameters, the FLASH4Rec achieves better accuracy with less computational cost. The significant acceleration mainly comes from the efficient design of the Gated Attention Layer. The SparseMoE layer contains more parameters but does not increase the computational cost because of its conditional computation capacity.

Table 2: Efficiency Comparison of BERT4Rec and FLASH4Rec on ML-1M Dataset.

	Params	FLOPs
BERT4Rec	3.08M	74.12M
FLASH4Rec	3.16M	63.10M

3.3 Ablation Study

We conduct an ablation study to analyze the key designs in FLASH4Rec, as shown in Table 3. In the ablation study, we replace multi-head self-attention layer, FFN and dropout layer by Gated Attention Layer, SparseMoE Layer, and Top-K Dropout, respectively. We can observe that both Gated Attention Layer and SparseMoE Layer improve the modeling capacity for our enhanced transformer block. The Top-K Dropout better regularizes the self-attentive network than the vanilla dropout layer.

Table 3: Abalation Study about key componenets of FLASH4Rec on ML-1M Dataset.

	Recall@10	NDCG@10
FLASH4Rec	0.2841	0.1568
w/o Gated Attention	0.2690	0.1488
w/o SparseMoE Layer	0.2787	0.1535
w/o Top-K Dropout	0.2765	0.1512

4 CONCLUSION

In this paper, we proposed a Transformer variant FLASH4Rec for the sequential recommendation. Compared with original architectures, FLASH4Rec utilizes Gated Attention Layer and Sparsely-Gated Mixture-of-Experts Layer to learn user-aware item sequence representation more effectively and efficiently. We also design the Top-K Dropout to encourage the model learning from low-attention positions to reduce overfitting. In future work, we would like to develop a linear attention version of FLASH4Rec further and consider the time interval between behaviors to efficiently test the model's performance on real-world recommendation problems with very long sequences.

REFERENCES

[1] Gabriel de Souza Pereira Moreira, Sara Rabhi, Jeong Min Lee, Ronay Ak, and Even Oldridge. 2021. Transformers4rec: Bridging the gap between nlp and sequential/session-based recommendation. In *Proceedings of the 15th ACM Conference on Recommender Systems*. 143–153.

[2] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. 2021. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *International Conference on Machine Learning*. PMLR, 2793–2803.

[3] William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res* 23 (2021), 1–40.

[4] Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. 2022. Transformer quality in linear time. In *International Conference on Machine Learning*. PMLR, 9099–9117.

[5] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*. IEEE, 197–206.

[6] Walid Krichene and Steffen Rendle. 2022. On sampled metrics for item recommendation. *Commun. ACM* 65, 7 (2022), 75–83.

[7] Aleksandr Petrov and Craig Macdonald. 2022. A Systematic Review and Replicability Study of BERT4Rec for Sequential Recommendation. In *Proceedings of the 16th ACM Conference on Recommender Systems*. 436–447.

[8] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).

[9] 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).

[10] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2021. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864* (2021).

[11] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 1441–1450.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[13] 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 *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 1059–1068.

[14] Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. 2022. Designing effective sparse expert models. *arXiv preprint arXiv:2202.08906* (2022).