Challenges and Result-Analysis of Sequential Recommendation Models

Alokam Gnaneswara Sai
CSA, Indian Institute of Science

***Abstract*—This report analyzes the performance of various sequential recommendation models on three datasets: MovieLens, Amazon Beauty, and Delicious. The models evaluated include the P5 model, a Pure RNN model, and a Transformer Decoder-based model. Detailed results are presented in terms of HR@5, HR@10, NDCG@5, and NDCG@10, and the impact of varying learning rates and hidden sizes on model performance is explored.**

I. Problem Statement

The sequential recommendation problem involves predicting the next item in a sequence of user interactions, given the previous items in that sequence.

A. Notation and Definitions

Let $\mathcal{U}$ be the set of all users, and $\mathcal{I}$ be the set of all items. For a user $u \in \mathcal{U}$, their interaction history is represented as a sequence of items $S_u = [i_1, i_2, \dots, i_t]$, where $i_j \in \mathcal{I}$ for $j = 1, 2, \dots, t$. The goal is to predict the next item i_{t+1} that the user u will interact with, given their interaction history S_u.

B. Objective

The objective is to learn a function $f : (\mathcal{U}, \mathcal{I}^*) \to \mathcal{I}$, where $\mathcal{I}^*$ represents the set of all possible sequences of items. Specifically, for a given user u and their interaction history S_u, the function f should predict the next item i_{t+1}:

$$\hat{i}_{t+1} = f(u, S_u)$$

C. Learning the Function f

The function f is typically parameterized by a model, such as a Recurrent Neural Network (RNN), Transformer, or any other sequential model. The model parameters θ are learned by minimizing a loss function over a training set of user interaction sequences. The most common loss function used is the cross-entropy loss.

D. Training and Optimization

The model parameters θ are optimized using gradient-based methods, such as stochastic gradient descent (SGD) or its variants like Adam, to minimize the cross-entropy loss $\mathcal{L}(\theta)$.

II. Approach: RNN for Sequential Recommendation

To address the sequential recommendation problem, a Recurrent Neural Network (RNN) is utilized. The RNN processes the interaction history of a user sequentially, capturing temporal dependencies.

A. Model Architecture

An RNN consists of an input layer, hidden layers, and an output layer. The hidden state at time step t is computed as:

$$\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h)$$

where $\mathbf{x}_t$ is the input embedding of the item i_t, $\mathbf{h}_{t-1}$ is the hidden state from the previous time step, $\mathbf{W}_h$ and $\mathbf{U}_h$ are weight matrices, $\mathbf{b}_h$ is a bias term, and σ is a non-linear activation function.

B. Output Layer

The output at each time step t is given by:

$$\mathbf{o}_t = \mathbf{W}_o \mathbf{h}_t + \mathbf{b}_o$$

where $\mathbf{W}_o$ is the output weight matrix and $\mathbf{b}_o$ is a bias term. The probability distribution over the next item is obtained using the softmax function:

$$P(i_{t+1} \mid u, S_u^{(t)}) = \text{softmax}(\mathbf{o}_t)$$

C. Training

The model is trained by minimizing the cross-entropy loss:

$$\mathcal{L}(\theta) = -\sum_{k=1}^{N} \sum_{t=1}^{|S_{u_k}|-1} \log P(i_{t+1} \mid u_k, S_{u_k}^{(t)}; \theta)$$

using gradient-based optimization methods such as Adam.

D. Prediction

During inference, given the interaction history S_u of user u, the RNN generates the hidden states sequentially and predicts the next item $\hat{i}_{t+1}$ by selecting the item with the highest probability:

$$\hat{i}_{t+1} = \arg\max_{i \in \mathcal{I}} P(i \mid u, S_u)$$

III. Approach: Transformer Decoder for Sequential Recommendation

To address the sequential recommendation problem, a Transformer decoder is utilized. The Transformer decoder processes the interaction history of a user in parallel, capturing temporal dependencies using self-attention mechanisms.

A. Model Architecture

The Transformer decoder consists of multiple layers of masked self-attention and feed-forward networks. The masked self-attention ensures that the model attends only to previous time steps and not future ones, maintaining the sequential nature of the problem.

1) Self-Attention: The attention scores are computed as:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)$$

where the query $\mathbf{Q}$, key $\mathbf{K}$, and value $\mathbf{V}$ matrices are derived from the input embeddings. For causal self-attention, we use a mask to ensure that each position t only attends to positions $\leq t$.

2) Feed-Forward Network: Each decoder layer also includes a feed-forward network:

$$\mathbf{FFN}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

where $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{b}_1$, and $\mathbf{b}_2$ are learned parameters.

B. Output Layer

The output at each time step t is computed by passing the decoder's final hidden state through a linear layer followed by a softmax function:

$$P(i_{t+1} \mid u, S_u^{(t)}) = \text{softmax}(\mathbf{W}_o \mathbf{h}_t + \mathbf{b}_o)$$

where $\mathbf{W}_o$ is the output weight matrix and $\mathbf{b}_o$ is a bias term.

C. Training with Teacher Forcing

During training, we use teacher forcing, where the ground truth item i_t is fed into the model at each time step. The model is trained by minimizing the cross-entropy loss:

$$\mathcal{L}(\theta) = -\sum_{k=1}^{N} \sum_{t=1}^{|S_{u_k}|-1} \log P(i_{t+1} \mid u_k, S_{u_k}^{(t)}; \theta)$$

using gradient-based optimization methods such as Adam.

D. Prediction

During inference, given the interaction history S_u of user u, the Transformer decoder generates the hidden states sequentially and predicts the next item $\hat{i}_{t+1}$ by selecting the item with the highest probability:

$$\hat{i}_{t+1} = \arg\max_{i \in \mathcal{I}} P(i \mid u, S_u)$$

If the prediction is incorrect, the model is penalized accordingly to improve future predictions.

IV. Data Preprocessing

For all datasets, the presence of a review or rating is treated as implicit feedback (i.e., the user interacted with the item), and timestamps are used to determine the sequence order of actions. Users and items with fewer than 5 related actions are discarded. For partitioning, the historical sequence S_u for each user u is split into three parts: (1) the most recent action $S_u^{|S_u|}$ for testing, (2) the second most recent action $S_u^{|S_u|-1}$ for validation, and (3) all remaining actions for training. Note that during testing, the input sequences contain training actions and the validation action.

V. Network Training

Convert each user sequence (excluding the last action) $(S_u^1, S_u^2, \ldots, S_u^{|S_u|-1})$ to a fixed-length sequence $s = \{s_1, s_2, \ldots, s_n\}$ via truncation or padding items(to the left of the sequence). We define o_t as the expected output at time step t:

$$o_t = \begin{cases} < pad > & \text{if } s_t \text{ is a padding item} \\ s_{t+1} & 1 \leq t < n \\ S_u^{|S_u|} & t = n \end{cases}$$

where $< pad >$ indicates a padding item. The model takes a sequence s as input, and the corresponding sequence o as expected output.

TABLE I
PERFORMANCE COMPARISON OF VARIOUS MODELS ON MOVIELENS, AMAZON BEAUTY, AND DELICIOUS

Models	MovieLens				AmazonBeauty				Delicious			
	HR@5	HR@10	NDCG@5	NDCG@10	HR@5	HR@10	NDCG@5	NDCG@10	HR@5	HR@10	NDCG@5	NDCG@10
RNN	0.2532	0.3128	0.1792	0.2242	0.1564	0.2032	0.1076	0.1656	0.3124	0.3452	0.2231	0.2632
Transformer(Decoder)	0.4596	0.5286	0.4408	0.4882	0.3210	0.4056	0.2042	0.2770	0.5564	0.6010	0.6042	0.6532
P5-small	0.5076	0.6162	0.4922	0.5072	0.3830	0.4134	0.2211	0.2824	0.6475	0.7080	0.6540	0.6680
FPMC	0.5655	0.7322	0.4530	0.5176	0.3255	0.3822	0.2041	0.2891	0.7553	0.7628	0.7068	0.7243
Caser	0.6446	0.7886	0.4860	0.5538	0.3346	0.4023	0.2260	0.2762	0.7579	0.7752	0.7332	0.7365
TransRec	0.6053	0.6413	0.3527	0.3969	0.4036	0.4607	0.2621	0.3020	0.8597	0.8725	**0.8323**	**0.8365**
SASRec	**0.6874**	**0.7904**	**0.5308**	**0.5642**	**0.4259**	**0.4851**	**0.2721**	**0.3250**	**0.8857**	**0.9450**	0.7438	0.7652

TABLE II
HYPERPARAMETERS FOR DIFFERENT MODELS

Hyperparameter	RNN Model	Transformer Decoder-Based Model	P5 Model
$hidden\_size$	128	256	256
num_layers	2	3	2
$num\_classes$	$input\_size$	$input\_size$	$input\_size$
$learning\_rate$	2.5×10^{-3}	1e-2	1e-3
$optimizer$	Adam	Adam	Adam
$loss$	Cross Entropy Loss	Cross Entropy Loss	Cross Entropy Loss
emb_dim	128	256	128

VI. DATASETS

For the sequential recommendation task, the following datasets were utilized

A. *MovieLens 1M*

The MovieLens 1M dataset contains 1 million ratings of movies. Each interaction consists of a user, a movie, a rating, and a timestamp, which are treated as implicit feedback for this task.

B. *Delicious*

The Delicious dataset contains bookmarks from the social bookmarking website Delicious. Each interaction represents a user bookmarking a URL at a specific timestamp.

C. *Amazon Beauty*

The Amazon Beauty dataset is a subset of the Amazon product dataset, specifically focused on beauty products. Each interaction consists of a user, an item, a rating, and a timestamp, which are used to infer the sequence of interactions.

TABLE III
DATASET STATISTICS

Dataset	#Users	#Items	#Interactions	Avgseqlen
MovieLens 1M	6,040	3,706	1,000,209	165.6
Delicious	1,867	69,223	437,593	234.47
Amazon Beauty	22,363	12,101	198,502	8.88

VII. EVALUATION STRATEGY

To assess the performance of the recommendation models, two widely adopted Top-N metrics are used: Hit Rate@K and NDCG@K.

A. *Hit Rate@K*

Hit Rate@10 measures the proportion of times the true next item appears in the top 10 recommendations.

B. *NDCG@K*

NDCG@K, or Normalized Discounted Cumulative Gain at K, is a position-sensitive metric that assigns higher weights to items appearing earlier in the recommendation list. This ensures that items ranked higher are more relevant and receive more emphasis in the evaluation.

C. *Evaluation Procedure*

To make the evaluation process more efficient, we adopt a sampling strategy. For each user u, we randomly sample 100 negative items and rank these items alongside the ground-truth item. By focusing on the rankings of these 101 items, we can compute Hit Rate@10 and NDCG@10 without evaluating every possible user-item pair.

VIII. RESULTS

The performance of each model on the datasets is summarized in Table I

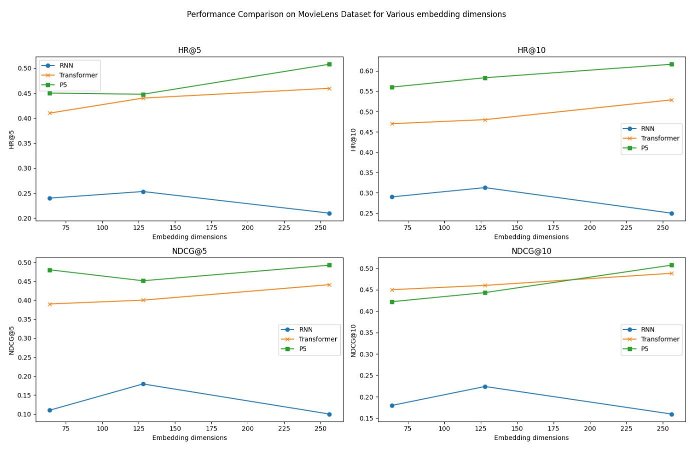

Fig. 1. Performance comparision of movielens dataset for various embedding dimensions

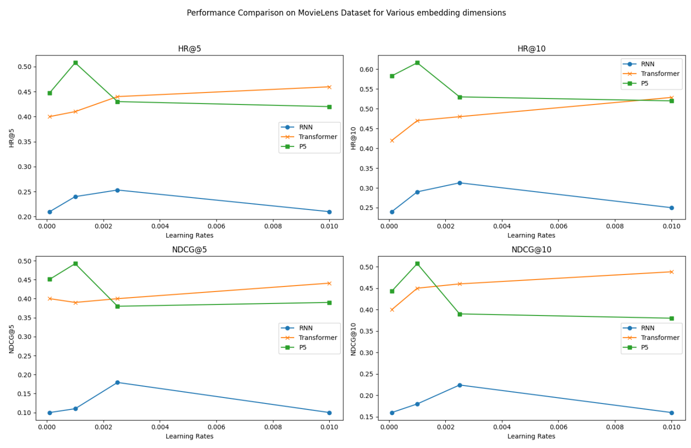

Fig. 2. Performance comparision of movielens dataset for various learning rates

IX. Hyperparameter Symbols

To ensure consistency and clarity in our report, the symbols for the hyperparameters used in the report are defined as follows II:

- $hidden\_size$: The number of hidden units in each layer of the model.
- num_layers: The number of layers in the model.
- $num\_classes$: The number of output classes, set to the input size.
- $learning\_rate$: The learning rate used for training the model.
- $optimizer$: The optimization algorithm used for training (Adam).
- $loss$: The loss function used for training (Cross Entropy Loss).
- emb_dim: The dimensionality of the embedding vectors.

X. Complexity Analysis

In this section, the time and space complexities of the Recurrent Neural Network (RNN) and Transformer models used for sequential recommendation are analyzed.

A. Recurrent Neural Network (RNN)

1) Time Complexity: For an RNN, the time complexity per time step is dominated by the matrix multiplications involved in updating the hidden state. Specifically, the time complexity is:

$$\mathcal{O}(n \cdot d^2)$$

where n is the sequence length and d is the dimensionality of the hidden state. Since the hidden state update must be computed sequentially for each time step, the overall time complexity for processing a sequence of length n is:

$$\mathcal{O}(n \cdot d^2)$$

2) Space Complexity: The space complexity of an RNN is determined by the storage of the model parameters and the hidden states. The parameters include the weight matrices and biases, with a total space complexity of:

$$\mathcal{O}(d^2)$$

Additionally, storing the hidden states for a sequence of length n requires:

$$\mathcal{O}(n \cdot d)$$

Thus, the overall space complexity is:

$$\mathcal{O}(d^2 + n \cdot d)$$

B. Transformer Decoder

1) Time Complexity: For the Transformer decoder, the time complexity is dominated by the self-attention mechanism. For a sequence of length n with hidden dimension d, the self-attention operation requires:

$$\mathcal{O}(n^2 \cdot d)$$

The feed-forward network within each layer has a time complexity of:

$$\mathcal{O}(n \cdot d^2)$$

Since both the self-attention and feed-forward operations are repeated for each of the L layers, the overall time complexity is:

$$\mathcal{O}(L \cdot (n^2 \cdot d + n \cdot d^2))$$

2) Space Complexity: The space complexity for the Transformer decoder includes the storage of parameters and intermediate states. The parameters for the self-attention mechanism and feed-forward network in each layer require:

$$\mathcal{O}(L \cdot d^2)$$

Additionally, storing the hidden states and attention weights for a sequence of length n requires:

$$\mathcal{O}(n \cdot d + n^2)$$

Thus, the overall space complexity is:

$$\mathcal{O}(L \cdot d^2 + n \cdot d + n^2)$$

In summary, while RNNs have a linear time complexity with respect to the sequence length, Transformers exhibit a quadratic time complexity due to the self-attention mechanism. However, Transformers can utilize parallel computation more effectively, making them faster in practice for sufficiently long sequences.

XI. Analysis

Experiments reveal that the P5-base model achieves results approximately as good as SASRec on our sequential recommendation tasks. This is notable given that the P5 model, originally trained on Amazon Toys and Sports datasets, demonstrates significant transferability to other domains.

The results suggest that by directly fine-tuning the T5 architecture on our specific dataset, rather than relying on the pre-trained P5 model, we can potentially achieve even better performance.

Using P5-large variants we can expect better results.

XII. Challenges in Sequential Recommendation Systems

This section discusses the unique challenges encountered in sequential recommendation systems (SRSs) from various perspectives, including sequence length, internal order, action types, user information, data structure, and the reliance on implicit feedback.

A. Challenges Related to Sequence Length

Sequences in SRSs can vary widely in length, presenting distinct challenges. Long sequences require the model to capture long-range dependencies, which can be computationally intensive and complex to learn. On the other hand, short sequences may lack sufficient interactions, making it difficult for the model to learn meaningful patterns and dependencies for accurate recommendations.

B. Challenges Related to Internal Order

The inherent order of actions within sequences is a critical factor in SRSs. Each action depends on the preceding ones, necessitating models to learn strict sequential dependencies. This ordering adds complexity to the recommendation process, as the model must accurately predict the next item based on the precise sequence of previous interactions.

C. Challenges Related to Action Types

Actions within sequences can include purchases, clicks, views, and additions to the cart. Sequences involving a single type of action are simpler to model. However, sequences containing multiple action types require the model to learn both intra-action type dependencies (within the same action type) and inter-action type dependencies (between different action types). This adds an additional layer of complexity to the recommendation process.

D. Challenges Related to Implicit Feedback

In the approaches discussed, only implicit feedback is considered, where any interaction (e.g., click, view) is treated as a positive signal. This presents a significant challenge as it does not account for explicit feedback, such as user ratings indicating their preference for an item. Ignoring explicit feedback can lead to less accurate recommendations, as the model cannot differentiate between items that users interacted with but did not like and those they genuinely preferred. Incorporating explicit feedback would provide a more nuanced understanding of user preferences, enhancing the accuracy of the recommendations but also increasing the data complexity.

In summary, these challenges highlight the need for advanced models capable of handling the complex nature of sequential data in recommendation systems. Addressing these challenges is crucial for improving the performance and accuracy of SRSs.