A survey on sequential recommender systems

Gohil Dwijesh

April 2022

Contents

1 Abstract

There are numerous web applications in today's world where the user-item interaction is sequential in nature, for example, Amazon, Spotify, Netflix, and Google Scholar. These applications have access to enormous amounts of user-item sequential interaction data. By utilising this sequential data, we can leverage both the user's long-term patterns (previous user-item interactions) and the user's current context (recent user-item interactions). The task of sequential recommendation is to recommend the next item that a user is more likely to interact with based on the sequence of previously interacted items. We review prior work on sequential recommendation and categorise it into two broad categories: non-neural methods and neural methods in this survey. Additionally, we elaborate on several seminal methods from each category.

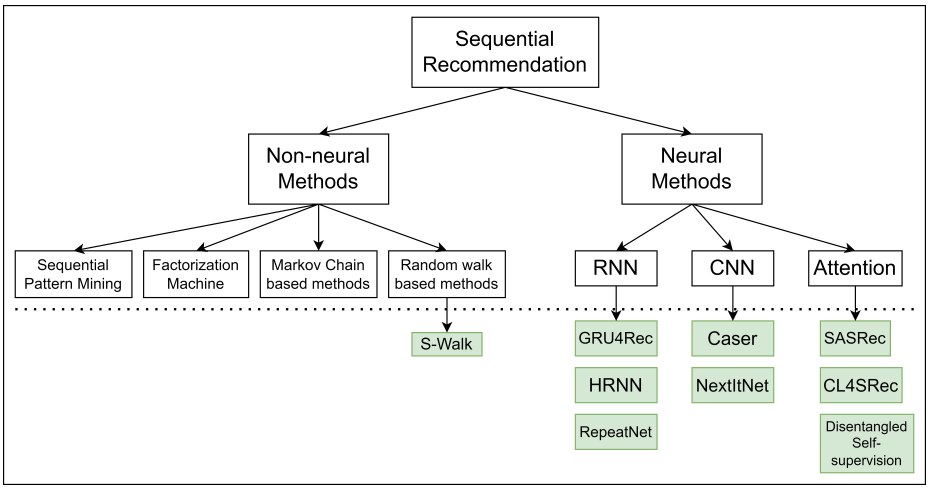

Figure 1: In this review, we classify methods into two broad classes: neural and non-neural methods. These classes are further divided into sub-classes. In this review, we will review some of the seminal methods from various classes(colored in green)

.

2 Sequential Recommendation - Classification

Sequential recommendation is a task of recommending an item that a user is most likely to interact with, given his or her historically interacted items. Figure-1 shows the classification of sequential recommendation methods. Overall we divide the methods into two broad classes: Neural and Non-neural methods. In this review, we will elaborate on some of the seminal work in sequential recommendation field. Specifically, we will review, S-walk [CKL+22], GRU4REC [HKBT15], HRNN [QKHC17], RepeatNet [RCL+19], Caser [TW18], NextItNet [YKA+19], SASRec [KM18], CL4SRec [XSL+20], and Disentangled self-supervision based method [MZY+20].

2.1 Non-neural methods

In this section, we will review sequential recommendation methods that do not involve any form of neural network learning. We further divide the methods into four categories: 1) sequential pattern mining based methods 2) factorization machine based methods 3) markov chain based methods and 4) random walk based methods.

Recently, a random walk based method called S-Walk[CKL+22] has shown state-of-the-art results outperforming many deep learning based methods. S-Walk mainly focuses on efficiency and scalability aspects. Unlike any-other random walk based methods, it tries to capture the inter-session and intra-session relationship between items to recommend the next item to a user.

2.2 Neural methods

Neural methods for sequential recommendation involve learning non-linear relationship between items of a sequence/session or across sessions(if user information is available). We further divide the methods into three categories: 1) RNN based methods 2) CNN based methods and 3) Attention based methods

RNN based methods mostly use GRU(Gated Recurrent Unit) units to capture the sequential dependency of items. RNN based methods assume strong ordering of items that all the items in a sequence are strongly ordered with respect to the time of interaction. Some representative work on RNN based methods are GRU4REC [HKBT15], HRNN [QKHC17], and RepeatNet [RCL+19]. GRU4REC considers GRU units to capture the sequential dependecies between items. HRNN [QKHC17] considers two levels of hierarchy, one level to capture intra-session item dependency and second level to capture inter-session item dependency. HRNN assumes that the user information is available at training and inference time. RepeatNet relies on an observation that users tend to repeat the item interactions. RepeatNet explicitly models the probability of a user repeating or exploring an item. It then ranks the candidate items based on these probability scores and recommends the next item.

CNN based methods consider session-item matrix ($\#rows = \#sessions$ and $\#columns = \#items$) or sequence-item matrix ($\#rows = \#sequences$ and $\#columns = \#items$) as input to a CNN network. Then a kernel of width equal to the $\#items$ is applied to the input matrix to capture the non-linear item dependency. The representative work includes Caser [TW18] and NextItNet [YKA+19]. To best of our knowledge, Caser is a first method to apply the CNN network to a sequential recommendation task. Caser method applied horizontal and vertical kernels on the sequence-item matrix to capture the relationship between items of a sequence. As a follow up work, NextItNet used dilated kernels to increase the receptive field of a CNN network. The idea of dilated kernels allowed their model to capture the skip behavior(where the next item prediction depends on a non-consecutive set of previous items).

Attention based methods usually tries to learn the importance of items within a session or a sequence. One of the seminal work is SASRec [KM18]. The self-attention part of the framework does the weighted linear combination of item representations. Where weights corresponds to the importance of the items. CL4SRec [XSL+20] and Disentangled self-supervision [MZY+20] based methods consider SASRec as a backbone model in their respective frameworks. CL4Srec on top of SASRec, considers contrastive learning loss and multi-headed attention to further improve the recommendation quality. Disentangled self-supervision based method [MZY+20] assumes that the user's intentions keep changing over time. Hence a sequence of items contains multiple of those intentions. Their framework learns the hidden intention representations and predicts an item that matches the most with one of the intentions.

3 Sequential Recommendation - Methods

3.1 S-Walk [CKL+22]

The motivation of the paper is to reduce the train/inference time complexity and memory complexity. Unlike other random-walk based sequential recommendation methods, S-Walk tries to capture the correlation between inter-session items.

Figure-2 shows the S-Walk framework. The framework consists of mainly five parts. 1) Train data preparation 2) Item transition model 3) Item teleportation model 4) Random walk with restart and 5) Inference.

Train data preparation: Consider we have n sessions: $S_1, S_2, ..., S_n$. Each session consists of items interacted by a user. Suppose $S_i = [item_1, item_2, ..., item_{|S_i|}]$ be an i^{th} session. We will consider two matrices: Y and Z. Y corresponds to input session matrix and Z corresponds to output session matrix. Consider $[item_1, item_2, ..., item_j] \mid j >= 1 \ and \ j < |S_i|$ will be one of the row(say k^{th} row) in Y correspondingly $\{[item_{j+1}, item_{j+2}, ..., item_{|s_u|}]|j >= 1 and j < |S_i|$ will be the k^{th} row in Z. Such data preparation step is repeated for all sessions and for all valid j. The number of columns

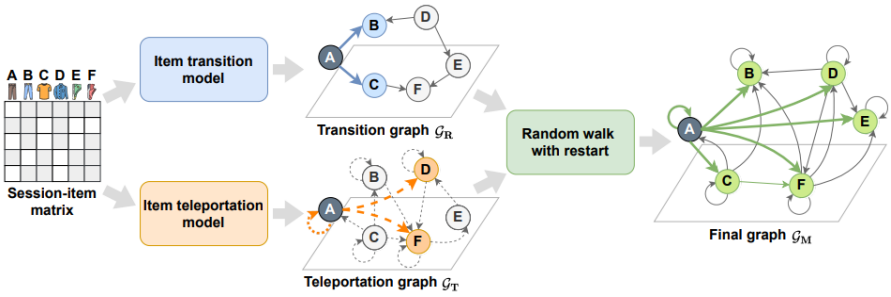

Figure 2: S-Walk framework: It consists of two linear models: item transition and item teleportation model. Each model tries to learn the probability of transitioning from one item to another. After applying the random walk with restart on a combined graph, a new graph with transition probability is constructed. The item predictions are made based on this final graph

of matrices Y and Z is equal to the number of items in the dataset. They are zero-one matrices. An entry is one if corresponding item is part of input/output sub-session otherwise the entry will be zero. Also the matrix X is the input session matrix with number of rows equal to number of sessions in the dataset and number of columns equal to number of items in the dataset.

Item transition model: Item transition model aims at learning sequential dependencies between Y and Z. Formally, the optimization problem is as follows:

$$arg \min_{B^{tran}} ||Z - YB^{tran}||_F^2 + \lambda ||B^{tran}||_F^2$$

Here, λ is a hyper-parameter and B^{tran} is an item-item relevance matrix. The optimization problem has a direct solution: $B^{tran} = (Y^T Y + \lambda I)^{-1}(Y^T Z)$. Since the entries of B^{tran} might be negative and also the row sum will not be 1, hence all negative entries are replaced with zero and row normalization is applied to B^{tran} as: $R = diagMat(B^{tran}_{\geq 0})^{-1} B^{tran}_{\geq 0}$. Here R is the item transition probability matrix.

Item teleportation model: The optimization problem for item teleportation model is as follows:

$$arg \min_{B^{tele}} ||X - XB^{tele}||_F^2 + \lambda ||B^{tele}||_F^2 \;\; s.t. \;\; diag(B^{tele}) \leq \epsilon$$

Here λ is a hyper-parameter. Most random walk-based prior work puts a constraint over B that diagonal elements of B should be zero. However, to consider the same item repetition, S-walk allows the diagonal to have some non-zero value. Similar to item transition model, to get the item teleportation probability matrix, thresholding and normalization is applied on B^{tele} as: $T = \beta\left(diagMat(B^{tele}_{\geq 0}\ 1)^{-1} B^{tele}_{\geq 0}\right) + (1-\beta)I$. Here β is a hyper-parameter which controls the proportion of self-loop to guarantee the convergence of random walk.

Random walk with restarts Figure-3 shows the S-Walk training algorithm. It aims to learn a matrix M. Semantically, M_{ij} corresponds to the probability of ending up on item j starting from item i if a random walker does a random walk with restarts for infinite times.

Inference: Suppose $S_q = [item_1, item_2, ..., item_{|S_q|}]$ is a query session. Then we will consider a binary vector x whose dimension will be total number of items in the dataset. The i^{th} entry will be 1 if i^{th} item in the dataset belongs to the query session otherwise i^{th} entry will be 0. Now we will

Algorithm 1: Training procedure for S-Walk

Input: Transition matrix **R**, teleportation matrix **T**.
Output: S-Walk model **M**.

$\mathbf{M}_{(0)} \leftarrow \mathbf{I}$
$k = 0$
repeat
  $k \leftarrow k + 1$
  $\mathbf{M}_{(k)} \leftarrow \alpha \mathbf{M}_{(k-1)} \mathbf{R} + (1 - \alpha)\mathbf{T}$
until $\|\mathbf{M}_{(k)} - \mathbf{M}_{(k-1)}\|_1 \leq \epsilon$
$\mathbf{M} \leftarrow \mathbf{M}_{(k)}$

Figure 3

weight the vector x according to the recency of items in the query session. Weight for the i^{th} item in the query session is defined as follows:

$$w_{inf}(i, S_q) = exp\left(- \frac{|S_q| - i}{\delta_{inf}} \right)$$

Where δ_{inf} is a hyper-parameter. After weighting the vector x, it is multiplied with the matrix M to get a new vector y as $y = xM$. Semantically, $y[i]$ stores the chance of ending up on an item i given the input session vector x. For recommendation, items are ordered in decreasing order of their scores in y.

3.2 GRU4REC [HKBT15]

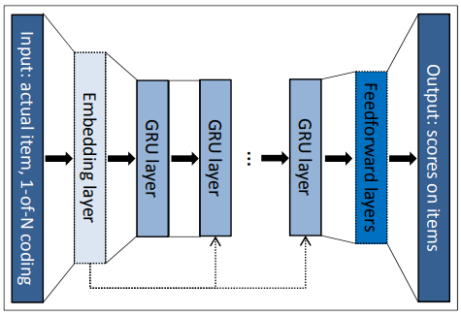

Figure 4: GRU4REC framework. It consists of embedding layer, a stack of GRU layers, and feed forward layer to score all items in the dataset

Figure-4 shows the GRU4REC's architecture. The overall architecture consists of an embedding layer, a stack of GRU layers, and feedforward layer that score all the items in the dataset for the next-item prediction task. To the best of our knowledge, GRU4REC is the first work that used RNN in the session-based recommendation(SBR) task. GRU4REC is counted as one of seminal work in the field of RNN based methods for the SBR task.

Session-parallel mini-batch: The authors introduce a batching technique called as session-parallel mini-batching. Suppose we have n sessions and the batch size is B. Then the batch-1 input will

contain first item from first B sessions and the batch-1 output will contain second item from first B sessions. If one of the session does not have left any input-output pair then it will be discarded and a new session will be added in the list of B sessions. For example, consider three sessions: $S_1 = [item_1, item_2, item_3]$, $S_2 = [item_4, item_5]$, and $S_3 = [item_6, item_7]$ and batch size be 2. Then

$$Batch - 1 \;\; input: \;\; [item_1, item_4] \;\; output: \;\; [item_2, item_5]$$

$$Batch - 2 \;\; input: \;\; [item_2, item_6] \;\; output: \;\; [item_3, item_7]$$

Bayesian Personalized Ranking: Authors use a pair-wise loss: Bayesian Personalized Ranking(BPR) loss as follows:

$$-\frac{1}{N_s}\sum_{j=1}^{N_s} log(\sigma(r_{s,i} - r_{s,j}))$$

Here, $r_{s,i}$ is the score of the ground truth next item, $r_{s,j}$ is the score of negative sample, N_s is the number of negative samples, and σ is the sigmoid function. Intuitively, the loss function tries to maximize the score of ground truth next item and minimize the score of the negative samples. To save training time, instead of considering all samples other than the ground truth item as negative samples, the authors consider the items from the current batch as negative samples.

3.3 HRNN [QKHC17]

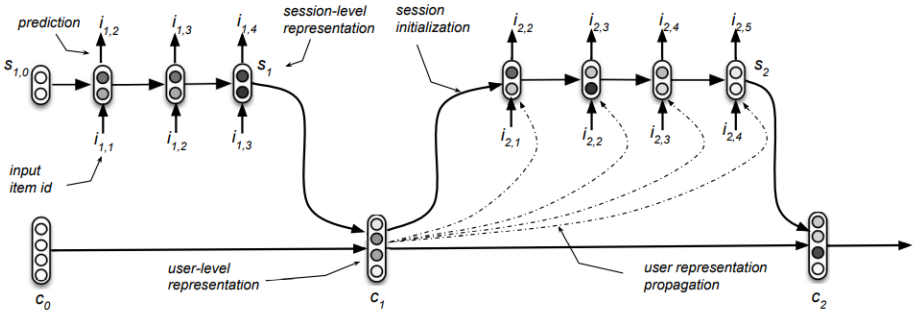

Figure 5: HRNN architecture: unlike GRU4REC, HRNN incorporates personalized session-based recommendation using a two-level hierarchy of GRU layers. The first layer learns session-level representations and the second layer learns user-level representations.

There exists many variants of GRU4REC: []. One of the variant is hierarchical RNN approach to incorporate the personalized session-based recommendations. It assumes that for a user, we have a sequence of sessions. Each session consists of sequence of items. Figure-5 shows the architecture of HRNN method. It consists of two level hierarchy of GRU units. The first level learns session-level representations and second level learns user-level representations. Suppose for example, a user u has two sessions in the dataset. Then the firstly, the session-1 is passed through the level-1 GRU units and at the end of the last GRU unit, session-1 representation is passed through the second layer GRU unit as to update the user representation. Now, items from session-2 along with user representation after session-1 is passed through the level one GRU units. And the output of the last GRU unit is passed through the level-2 GRU unit to get the user representation after session-2. Authors use the same strategy for data preparation(session-parallel mini-batch) and loss function(Bayesian Personalized Ranking loss) as GRU4REC.

3.4 RepeatNet [RCL+19]

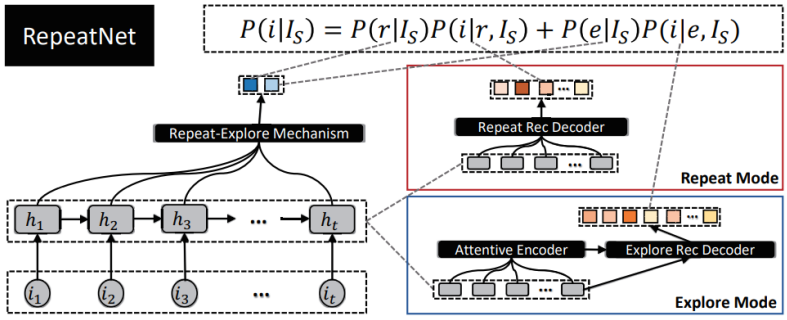

Figure 6: RepeatNet architecture: It consists of three parts. 1) Repeat-Explore Mechanism: predicts the probability of repeat vs exploration 2) Repeat recommendation decoder: predicts the item scores under repeat mode and 3) Explore recommendation decoder: predicts the item scores under explore mode

RepeatNet is yet another method based on attention and recurrent neural networks. It assumes that a user often times repeats the previously interacted items. The probability of an item i being the next item given a session I_S is formulated as follows:

$$P(i|I_S) = P(r|I_S)P(i|r, I_S) + P(e|I_S)P(i|e, I_S)$$

Here $P(r|I_S)$ and $P(e|I_S)$ are the probability of repeat and explore mode respectively. $P(i|r, I_S)$ and $P(i|e, I_S)$ are the probability of of an item being the next item under repeat and explore mode respectively. Figure-6 shows the RepeatNet architecture. It mainly consists of three parts. 1) *Repeat-Explore module:* It estimates the probability of repeat or exploration given the input session I_S. Similar to GRU4REC, this module also uses a sequence of GRU units to learn hidden state representation at each time step. 2) *Repeat module:* This module takes the learned hidden state representations and scores items under repeat mode. It only scores the items from the input session I_S to incorporate the repeatation 3) *Exploration module:* It takes hidden state representations as input and scores the items under explore mode. It scores items not from the input session I_S to incorporate the explore mode. Using the probabilistic framework mentioned above, the model scores all items and ranks in decreasing order of the scores.

3.5 Caser [TW18]

To the best of our knowledge, Caser is the first work that used convolutional neural network to approach the sequential recommendation task. The Caser method incorporates two aspects: 1) *Union-level dependency:* Union-level dependency occurs when the next item in a sequence depends on a union of items from that sequence. 2) *skip behavior:* There can be unintentional clicks when a user is interacting with items. These unintentional clicks have little or no influence on the next item prediction task. Skip behavior allows to skip such erroneous clicks in the task of next prediction task.

Figure-7 shows the Caser architecture. Given a sequence $S^u = [S^u_1, S^u_2, ..., S^u_{|S^u|}]$, Caser uses a sliding window approach to prepare the training data. It considers the window of size L as input items and immediate T following items as output. It learns item and user representations in the

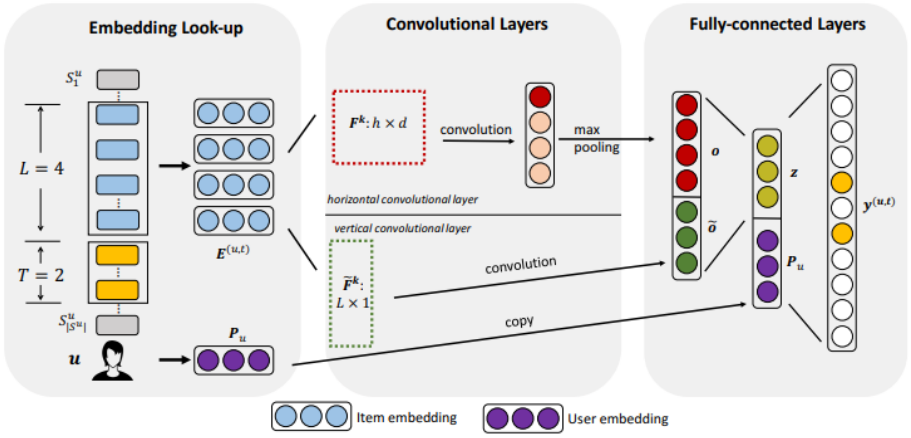

Figure 7: Caser architecture: It treats the stack of item embeddings as an image and applies horizontal and vertical kernels. It also considers user embedding at the fully connected layers. It finally scores all items in the dataset to predict the next most likely item in a sequence.

training process. Stacked item representations for the L items is considered as an image. Then k_1 and k_2 number of horizontal and vertical kernels are applied to the image respectively. The horizontal kernels' width matches with the image's width and vertical kernels' height matches with image's height. The horizontal kernels incorporates the skip behavior and union-level dependencies. Suppose the output of a horizontal kernel is of dimension p then a maxpooling operation is applied to this p dimensional vector. The output of horizontal and vertical filter is concatenated and passed through a fully connected layer. The output is further concatenated with the learnable user embedding and passed through a fully connected layer again whose output scores all the items in the dataset. The training loss tries to maximize the score of the correct next item and minimize the score of negative items.

3.6 NextItNet [YKA+19]

Despite the success behind Caser [TW18] method, there are some obvious critics. Caser method uses a maxpooling operation after applying the horizontal kernel to an input image. This operation ignores the position information and assumes that there is just one important feature per kernel output. Also in general, the CNN based approaches for sequential recommendation are criticized for not able capture the long-term dependencies due to the limited sized kernels. To overcome the above issues, NextItNet proposes a CNN based method that uses the dilated filters to increase the receptive field. Figure-8 shows the example of dilated filters. NextItNet uses a deep CNN model and to avoid the vanishing gradient problem, authors use skip connections. NextItNet does not use maxpooling operation after applying the dilated kernels to preserve the position information. Given a sequence of items: $[x_1, x_2, ..., x_{t-1}]$, NextItNet generates all $[x_2, x_3, ..., x_t]$ items. To generate an item x_i, it assumes all previous items: $[x_1, x_2, ..., x_{i-1}]$ are available. The motivation is to capture intra-sequence item-item relationship.

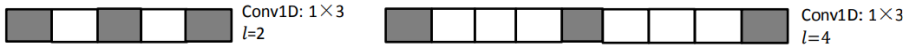

Figure 8: Example of dilated filters of different sizes but fixed 3 learnable paramters. The dark boxes represents the learnable values and white boxes represents a zero value. Both kernels have three learnable parameters but different number of zero elements.

3.7 SASRec [KM18]

SASRec is one of the seminal work in applying the self-attention to the sequential recommendation task. SASRec architecture can be divided into three types of layers: 1) Embedding layer 2) Self-attention layer and 3) Inference layer.

Embedding layer: Suppose we have a sequence $S = [S_1, S_2, ..., S_n]$. Then the embedding layer prepares the input representation of the sequence for further processing as follows:

$$E = \begin{bmatrix} M_{S_1} + P_1 \\ M_{S_2} + P_2 \\ ... \\ M_{S_n} + P_n \end{bmatrix}$$

Where M_{S_i} is learnable representation of item S_i and P_i is the position embedding for i^{th} position.

Self-attention layer: In this paper, the self-attention layer is made up of stack of two self-attention blocks. A self-attention block is made up of attention module and a 2-layer fully connected module. A self-attention block is mathematically described below

$$S = SA(E) = Attention(EW^Q, EW^K, EW^V)$$

$$F_i = FFN(S_i) = ReLU(S_i W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}$$

Here W^Q, W^K, W^V, $W^{(1)}$, $W^{(2)}$, $b^{(1)}$, and $b^{(2)}$ are learnable parameters. S is of dimension $n \ x \ d$ where n is the number of items in a sequence and d is the dimension of item representation. Overall, S stores the item representations after self-attention. F_i is again the representation of i^{th} item after applying the fully connected layers. The output of the first self-attention block(F) is fed as input to the second self-attention block(replacing E with F in above equation).

Inference: Suppose $S_q = [S_1, S_2, ..., S_n]$ be a query sequence. We will pass it through the embedding layer and self-attention layer. Suppose the output of the second self-attention block is $F \in R^{n \ x \ d}$. We also have learned the item representations $M \in R^{n \ x \ d}$. The score for an item i as a next item recommendation is formulated as follows:

$$r_i = F_n M_i^T$$

Where F_n be the n^{th} row in matrix F and M_i is the i^{th} item representation. We considered the last row of F since it turns out that j^{th} row of matrix F is actually a non-linear combination of rows above it. Hence last row covers information from all rows above it. Finally, the items are recommended in decreasing order their scores.

3.8 Disentangled Self-Supervision in Sequential Recommenders [MZY+20]

The general approach adapted by most sequential recommender systems is as follows. The system learns the representation of all items in the dataset. The system also learns a sequence encoder which

takes a sequence as input and outputs a representation for the input sequence. Finally, the items are recommended based on the cosine similarity between the query sequence representation and item representation. Motivation for this paper is that most prior work focuses on seq2item learning strategy. Instead the proposed method tries to exploit extra signals from longer-term future - that is to use seq2seq learning strategy. Unlike other methods, the proposed method learns K different sequence encoders, each encoder trying to capture a latent intention. One of the challenge is that reconstructing a sequence is exponentially harder problem compared to the next item prediction task. The proposed method can be divided into four parts: 1) Training data preparation 2) Seq2seq loss 3) Seq2item loss and 4) Disentangled sequence encoding

Training data preparation: Suppose a user u interacted with T_u items: $x^{(u)} = [x_1^{(u)}, x_2^{(u)}, ..., x_{T_u}^{(u)}]$. Then we define the earlier sequence and future sequence as follows:

$$Earlier\ \ sequence: \ \ x_{1:t}^{(u)} = [x_1^{(u)}, x_2^{(u)}, ..., x_t^{(u)}]$$

$$Future\ \ sequence: \ \ x_{(t+1):T_u}^{(u)} = [x_{(t+1)}^{(u)}, x_{(t+2)}^{(u)}, ..., x_{T_u}^{(u)}]$$

Also consider $\phi_\theta^{(k)}$ being the k^{th} sequence encoder. Where θ represents the learnable parameters.

Seq2seq loss: Figure-9 shows the equation of seq2seq loss term. It aims at making the future sequence representation($= \phi_\theta^{(k)}(x_{T_u:t+1}^{(u)})$) and the earlier sequence representation($= \phi_\theta^{(k)}(x_{1:t}^{(u)})$) to be as similar as possible. Since there is a chance that earlier sequence and future sequence may

$$\mathcal{L}_{s2s}(\boldsymbol{\theta}, u, t, k) = -\ln p_{\boldsymbol{\theta}}(\phi_{\boldsymbol{\theta}}^{(k)}(\mathbf{x}_{T_u:t+1}^{(u)}) \mid \phi_{\boldsymbol{\theta}}^{(k)}(\mathbf{x}_{1:t}^{(u)})) =$$
$$-\ln \frac{\exp\left(\frac{1}{\sqrt{D}}\, \phi_{\boldsymbol{\theta}}^{(k)}(\mathbf{x}_{T_u:t+1}^{(u)}) \cdot \phi_{\boldsymbol{\theta}}^{(k)}(\mathbf{x}_{1:t}^{(u)})\right)}{\sum_{(u',t')\in\mathcal{B}} \sum_{k'=1}^{K} \exp\left(\frac{1}{\sqrt{D}}\, \phi_{\boldsymbol{\theta}}^{(k')}(\mathbf{x}_{T_{u'}:t'+1}^{(u')}) \cdot \phi_{\boldsymbol{\theta}}^{(k)}(\mathbf{x}_{1:t}^{(u)})\right)},$$

Figure 9

not share any latent intention, the authors only consider the loss values for confident earlier-future sequence pair as shown in figure-10. Where τ is the $\lceil \lambda |B| K \rceil^{th}$ smallest value in $\{L_{s2s}(\theta, u, t, k) : (u, t) \in B, 1 \leq k \leq K\}$. Here λ is a hyper-parameter, B represents a batch, and K represents the number of latent intentions. Equation in figure-10 enforces the model to learn from only the confident training pairs.

$$\mathcal{L}_{s2s}(\boldsymbol{\theta}, \mathcal{B}) = \sum_{(u,t)\in\mathcal{B}} \sum_{k=1}^{K} \mathcal{L}_{s2s}(\boldsymbol{\theta}, u, t, k) \cdot \mathbf{1}[\mathcal{L}_{s2s}(\boldsymbol{\theta}, u, t, k) \leq \tau]$$

Figure 10

Seq2item loss: The seq2item loss is similar to what most prior work has used. Figure-11 shows the equation for seq2item loss. It tries to maximize the similarity between next item representation($h_{t+1}^{(u)}$) and the earlier sequence representation($= \phi_\theta^{(k)}(x_{1:t}^{(u)})$). One catch is that it will consider the best matched intention between the item representation and the sequence representation as shown in the numerator of the last equation in figure-11. The overall loss is then considered as summation of seq2seq and seq2item loss.

Disentangled sequence encoding: So far we assumed that there are K sequence encoders that each of them learns distinct latent intention from the input sequence. In this section we will define the

$$\mathcal{L}_{s2i}(\boldsymbol{\theta}, \mathcal{B}) = \sum_{(u,t)\in\mathcal{B}} \mathcal{L}_{s2i}(\boldsymbol{\theta}, u, t),$$

$$\mathcal{L}_{s2i}(\boldsymbol{\theta}, u, t) = -\ln p_{\boldsymbol{\theta}}(\mathbf{h}_{t+1}^{(u)} \mid \phi_{\boldsymbol{\theta}}(\mathbf{x}_{1:t}^{(u)})) =$$

$$-\ln \frac{\max_{k\in\{1,2,\ldots,K\}} \exp\left(\frac{1}{\sqrt{D}}\, \mathbf{h}_{t+1}^{(u)} \cdot \phi_{\boldsymbol{\theta}}^{(k)}(\mathbf{x}_{1:t}^{(u)})\right)}{\sum_{(u',t')\in\mathcal{B}} \sum_{k'=1}^{K} \exp\left(\frac{1}{\sqrt{D}}\, \mathbf{h}_{t'+1}^{(u')} \cdot \phi_{\boldsymbol{\theta}}^{(k')}(\mathbf{x}_{1:t}^{(u)})\right)},$$

Figure 11

k^{th} sequence encoder $\phi_\theta^{(k)}$. The authors use SASRec [KM18] as a backbone encoder model. Suppose we have an input sequence $x^{(u)} = [x_1^{(u)}, x_2^{(u)}, ..., x_t^{(u)}]$. Now we utilize the SASRec architecture and get the representation of each item in a sequence:

$$[z_1^{(u)}, z_2^{(u)}, ..., z_t^{(u)}] = SASRec\ ([x_1^{(u)}, x_2^{(u)}, ..., x_t^{(u)}])$$

We define the k^{th} sequence encoder as follows:

$$\phi_\theta^{(k)}(x_{1:t}^{(u)}) = LayerNorm\left(\beta_k + \sum_{i=1}^{t} P_{k|i}\ P_i\ z_i^{(u)}\right)$$

Where β_k is a hyper-parameter. P_i measures the importance of position i. $P_{k|i}$ is the probability that an item at i^{th} position belongs to k^{th} latent intention. Figure-12 and -13 shows the equations for P_i and $P_{k|i}$ respectively.

$$p_i = \frac{\exp\left(\frac{1}{\sqrt{D}}\mathrm{key}_i \cdot \mathrm{query}\right)}{\sum_{i'=1}^{t} \exp\left(\frac{1}{\sqrt{D}}\mathrm{key}_{i'} \cdot \mathrm{query}\right)},$$

$$\mathrm{key}_i = \widetilde{\mathrm{key}}_i + \mathrm{ReLU}(\mathbf{W}^\top \widetilde{\mathrm{key}}_i + \mathbf{b}),$$

$$\widetilde{\mathrm{key}}_i = \mathrm{LayerNorm}_3(\boldsymbol{\alpha}_i + \mathbf{z}_i^{(u)}),$$

$$\mathrm{query} = \mathrm{LayerNorm}_4(\boldsymbol{\alpha}_t + \mathbf{z}_t^{(u)} + \mathbf{b}'),$$

Figure 12

$$p_{k|i} = \frac{\exp\left(\frac{1}{\sqrt{D}}\mathrm{LayerNorm}_1(\mathbf{z}_i^{(u)}) \cdot \mathrm{LayerNorm}_2(\mathbf{c}_k)\right)}{\sum_{k'=1}^{K} \exp\left(\frac{1}{\sqrt{D}}\mathrm{LayerNorm}_1(\mathbf{z}_i^{(u)}) \cdot \mathrm{LayerNorm}_2(\mathbf{c}_{k'})\right)},$$

Figure 13

P_i uses finds attention weights for item at position i. It uses output of SASRec for i^{th} position for the key_i calculation. Here, α_i acts as position embedding. c_k in the equation for $P_{k|i}$ is part of learnable parameters.

Inference: An items $h_{t+1}^{(u)}$ will be ordered in decreasing order of the probability of next item recommendation $P_\theta(h_{t+1}^{(u)}|\phi_\theta(x_{1:t}^{(u)}))$.

3.9 CL4SRec [XSL+20]

CL4SRec is contrastive learning based method for sequential recommendation task. Figure-14 shows the CL4SRec architecture. It consists of two types of losses. Contrastive loss and next-item prediction loss. The authors use SASRec as the backbone architecture with one modification: multi-headed attention instead of single-headed attention.

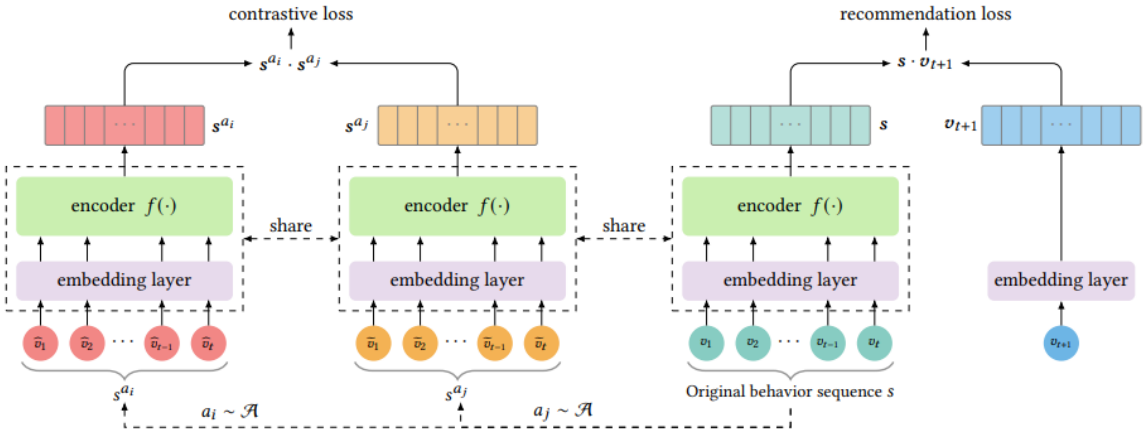

Figure 14: CL4SRec architecture: It consists of two types of losses. Contrastive loss and next item prediction loss. It uses SASRec as its backbone model.

Contrastive loss: Consider a batch of n sequences. In this paper, the authors consider three augmentation methods(item crop, item mask, and item reorder). We will randomly pick two augmentation methods with replacement and apply to a sequence. Suppose the transformed sequences are represented as s^{a_i} and s^{a_j}. Same process is repeated for all sequences in a batch. Now after the augmentation, there are $2n$ sequences in a batch, out of which (s^{a_i}, s^{a_j}) are related and (s^{a_i}, s^-) are not related pairs. Here s^- is all the transformed sequences except s^{a_j}. Hence the contrastive loss tries to give same representation to related pairs and different representation to un-related pairs:

$$L_{cl}(s_u^{a_i}, s_u^{a_j}) = -log\frac{exp(sim(s_u^{a_i}, s_u^{a_j}))}{exp(sim(s_u^{a_i}, s_u^{a_j})) + \sum_{s^- \in S^-} exp(sim(s_u^{a_i}, s^-))}$$

Next-item recommendation loss: It tries to bring the ground-truth next item representation and the sequence representation(output of SASRec) to get as similar representation as possible:

$$L_{main}(s_{u,t}) = -log\left(\frac{exp(s_{u,t}^T v_{t+1}^+)}{exp(s_{u,t}^T v_{t+1}^+) + \sum_{v^- \in V^-} exp(s_{u,t}^T v_{t+1}^-)}\right)$$

Where $s_{u,t}$ is the output of SASRec on input sequence and v_{t+1}^+ is the ground-truth next item representation. v_{t+1}^- is a negative item representation. Hence the overall loss is summation of next-item prediction loss and the contrastive loss. For the inference, the query sequence is passed through the

backbone architecture and items are recommended in decreasing order of cosine similarity between the sequence representation and the item representation.

Augmentation methods: Item crop randomly selects a L_1 sized continuous sub-sequence. Item mask randomly chooses L_2 sized indices and masks the corresponding items in a sequence. Item reorder randomly selects a continuous sub-sequence of length L_3 and shuffles it randomly. Here L_1, L_2, and L_3 are αn, βn, and γn where n is the sequence length and α, β, and γ are hyper-parameters s.t. their values lie between 0 and 1.

4 Critics

Despite being able to outperform state-of-the-art methods, every type of methods have their own pros and cons. In this section we'll look at the critics for various types of methods.

Type of SRS methods	**Critic**
Markov Chain	- Number of parameters grow exponentially with order - Captures only the several recent user-item interactions - Can not capture union-level seq. patterns - Can not capture skip behavior
Factorization Machines	- Models a seq. by the summation of item vectors and hence loses the order info.
Sequential Pattern Mining	- Generates large number of redundant patterns - High memory and time cost - Recommendation limited to popular item
RNN	- Assumes strongly ordered user-item sequence - Parallel computation not possible because of sequential dependency over previous states
CNN	- Can not capture long-term dependencies

Table 1

References

[CKL+22] Minjin Choi, Jinhong Kim, Joonseok Lee, Hyunjung Shim, and Jongwuk Lee. S-walk: Accurate and scalable session-based recommendation with random walks. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, WSDM '22, page 150–160, New York, NY, USA, 2022. Association for Computing Machinery.

[HKBT15] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks, 2015.

[KM18] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation, 2018.

[MZY+20] Jianxin Ma, Chang Zhou, Hongxia Yang, Peng Cui, Xin Wang, and Wenwu Zhu. Disentangled self-supervision in sequential recommenders. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '20, page 483–491, New York, NY, USA, 2020. Association for Computing Machinery.

[QKHC17] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 130–137, New York, NY, USA, 2017. Association for Computing Machinery.

[RCL+19] Pengjie Ren, Zhumin Chen, Jing Li, Zhaochun Ren, Jun Ma, and Maarten de Rijke. Repeatnet: A repeat aware neural recommendation machine for session-based recommendation. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019.

[TW18] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, page 565–573, New York, NY, USA, 2018. Association for Computing Machinery.

[XSL+20] Xu Xie, Fei Sun, Zhaoyang Liu, Shiwen Wu, Jinyang Gao, Bolin Ding, and Bin Cui. Contrastive learning for sequential recommendation, 2020.

[YKA+19] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M. Jose, and Xiangnan He. A simple convolutional generative network for next item recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, WSDM '19, page 582–590, New York, NY, USA, 2019. Association for Computing Machinery.