Intro to Sequential Based Modeling for Recommendation

Kaiwen Bian

November 24, 2024

Refer to CSE158 Textbook for Recommendation by Julian McAuley.

Use Cases

Consider the following scenarios which could cause changes in movie ratings or interactions over time:

1. A mundane change to a user interface, such as modifying the tool-tip text associated with a certain rating, may alter the rating distribution.
2. Users may binge-watch a series, dominating their interaction patterns for a short period.
3. Action blockbusters may be more favored during summer (or Christmas movies during Christmas).
4. Users may gradually develop an appreciation for certain characteristics of a movie as they consume more content from that genre.

Main-stream Sequential Recommendations

Baked-in Biases to Model Temporal Dynamics

The first class makes use of the *actual timestamps* of events. Each interaction (u, i) is augmented with a timestamp (u, i, t), and the goal is to understand how ratings $r_{u,i,t}$ vary over time. Our goal is to extend models such as the one below so that **parameters vary as a function of time (many many matrix or we can use a neural network directly)**:

$$r_{u,i,t} = \alpha(t) + \beta_u(t) + \beta_i(t) + \gamma_u(t) \cdot \gamma_i(t)$$

Modeling temporal dynamics in this way is effective at **capturing long-term shifts** of community preferences over time. Such a model can also **capture short-term** or 'bursty' dynamics, such as purchase patterns being affected by external events, or periodic events, such as purchases being higher at a particular time of day, day of week, or season.

Markov Probability

The second class of methods discards the specific *timestamps*, but preserves only the *sequence* (or order) of events. Thus, the goal is generally to predict the next action as a function of the previous one:

$$p(\text{user } u \text{ interacts with item } i \mid \text{ they previously interacted with item } j)$$

This type of model makes the assumption that the **important temporal information is captured in the context provided by the most recent event (Markovian)**. This is useful in **highly-contextual settings**, such as predicting the next song a user will listen to, or other items they will place in their basket, etc. In such settings, knowing the most recent action (or most recent few actions) is more informative than knowing the specific timestamp.

Recurrent Neural Networks

A fundamental limitation of the Markov-Chain-based models is that they have a very limited notion of 'memory,' due to the assumption that the next event is conditionally independent of all historical events, given the most recent observation. This assumption may be sufficient in certain scenarios, such as recommendation settings that are highly dependent on the context of the previously clicked item. However, as we begin to model text data, or sequence data such as heart-rate logs, or more complex recommendation scenarios, we will need to handle **longer-term** semantics (such as grammatical structures in a sentence, or even an individual's level of 'fatigue' in a heart-rate trace). We need more advanced methods involving recurrent neural network.

Baked-in Biases to Model Temporal Dynamics

Building from perspective of **traditional ML methods (many hard-coded heuristic approaches here rather than self-learned representation)**. Models must account for both **long-term trends** and **short-term variations** in user behavior and item popularity. This section focuses on modeling these temporal effects as described in Koren (2009).

Critically:

1. **Design Philosophy:** must balance flexibility with parsimony, capturing meaningful trends without overfitting.
2. **Bias Terms Dominate:** Most temporal variation is captured by evolving bias terms (β_u, β_i), while temporally evolving latent factors are often less critical.
3. **Adjusting for Historical Data:** aim to account for discrepancies across time periods to enable meaningful comparisons, rather than forecasting future trends.

Modeling Temporal Bias Terms

The Netflix Prize solutions emphasized **temporal bias terms** to capture variations over time. For user u and item i at time t, the baseline temporal bias model is defined as:

$$b_{u,i}(t) = \alpha + \beta_u(t) + \beta_i(t)$$

Where each variables are:

- α: Global average rating across all users, items, and time.
- $\beta_u(t)$: User-specific temporal bias.
- $\beta_i(t)$: Item-specific temporal bias.

While temporal models are highly effective, they must be **designed with parsimony** in mind to avoid excessive parameter growth. For instance:

1. Item biases can afford many parameters due to high interaction counts per item.
2. User biases must be parameterized efficiently (e.g., using $\text{dev}_u(t)$ or splines) due to the relatively sparse data for users.

Discrete Assignment (Item-TD)

Item biases (**only looking at how bias tends to change overtime**) evolve over time to capture **long-term** and **periodic** trends. A common approach is to divide the timeline into **discrete bins** and **assign bias parameters** to each bin. The item bias can then be expressed as:

$$\beta_i(t) = \beta_i + \beta_{i,\text{bin}(t)} + \beta_{i,\text{period}(t)}$$

where:

- $\beta_{i,\text{bin}(t)}$: Captures gradual, long-term variation (e.g., popularity changes over months or years).
- $\beta_{i,\text{period}(t)}$: Encodes periodic effects (e.g., day-of-week or seasonal trends).
- In Netflix dataset, ~ 30 bins of 10 weeks each were found effective to model long-term variation in item biases.

Expressive Deviation (User-TD)

For users, temporal dynamics are more challenging to model due to the sparsity of data. To address this, Koren introduced the concept of an **expressive deviation term**, which essentially sets cutoff point for temporal judgment:

$$\text{dev}_u(t) = \text{sign}(t - t_u) \cdot |t - t_u|^x$$

where:

- t_u: The mean timestamp of user u's ratings.
- $\text{sign}(x)$: Sign function, indicating whether t is before or after t_u.
- $|t - t_u|^x$: Models the magnitude of temporal deviation (e.g., $x = 0.4$ was effective for Netflix data).

The deviation term adjusts the user bias as follows:

$$\beta_u(t) = \beta_u + \alpha_u \cdot \text{dev}_u(t)$$

where α_u is a scaling parameter that controls the strength of the deviation term for each user:

- $\alpha_u < 0$: User ratings trend downward over time.
- $\alpha_u = 0$: No temporal variation in user bias.
- $\alpha_u > 0$: User ratings trend upward over time.

Latent Factors with Temporal Dynamics

In addition to biases, temporal dynamics can also be incorporated into latent factors. For a user u and factor k, the temporal latent factor is:

$$\gamma_{u,k}(t) = \gamma_{u,k} + \alpha_{u,k} \cdot \text{dev}_u(t) + \gamma_{u,k,t}$$

where:

- $\gamma_{u,k}$: Static latent factor for user u (does not change over time). Example: If k represents affinity for action movies, then $\gamma_{u,k}$ might encode a general preference for action movies for user u.
- $\alpha_{u,k} \cdot \text{dev}_u(t)$: Temporal deviation term for the latent factor.

- $\gamma_{u,k,t}$: Highly localized, time-specific adjustment (e.g., day-specific variations).

This approach captures temporal variations in user preferences, but $\gamma_{u,k,t}$ introduces many parameters and may only be feasible for users with abundant data.

Spline Interpolation (Complex User-TD)

For users with many interactions, a more complex model may be needed to capture gradual shifts in preferences. Koren proposed using a **spline function** to interpolate between control points in time. The user bias is defined as:

$$\beta_u(t) = \frac{\sum_{l=1}^{k_u} e^{-\gamma|t-t_{u,l}|} b_{u,t_l}}{\sum_{l=1}^{k_u} e^{-\gamma|t-t_{u,l}|}}$$

where:

- k_u: Number of control points for user u.
- $t_{u,l}$: Time associated with the lth control point.
- b_{u,t_l}: Bias term at the lth control point.
- γ: A decay parameter controlling how quickly the influence of a control point diminishes with time.

This formulation provides a smooth interpolation of user biases over time, allowing for gradual shifts while avoiding over fitting.

Overall Latent Temporal Dynamics Models

The predicted rating $r_{u,i,t}$ for user u, item i, and time t is given by:

$$r_{u,i,t} = \mu + b_u(t) + b_i(t) + \langle \gamma_u(t), \gamma_i(t) \rangle$$

where:

- μ: The global average rating across all users, items, and time.
- $b_u(t)$: The **user-specific temporal bias**, modeling how user u's preferences change over time:
  $$b_u(t) = \beta_u + \alpha_u \cdot \text{dev}_u(t),$$
  where:
  - β_u: Static bias for user u.
  - α_u: Scaling parameter for the temporal deviation.
  - $\text{dev}_u(t)$: Expressive deviation term, defined as:
    $$\text{dev}_u(t) = \text{sgn}(t - t_u) \cdot |t - t_u|^x,$$
    with t_u being the mean timestamp of user u's ratings and x controlling the curvature of the deviation.

- $b_i(t)$: The **item-specific temporal bias**, capturing long-term and periodic trends in item popularity:
  $$b_i(t) = \beta_i + \beta_{i,\mathrm{bin}(t)} + \beta_{i,\mathrm{period}(t)},$$
  where:
  - β_i: Static bias for item i.
  - $\beta_{i,\mathrm{bin}(t)}$: Bias term for long-term trends (e.g., weekly bins).
  - $\beta_{i,\mathrm{period}(t)}$: Bias term for periodic effects (e.g., seasonal trends).
- $\langle \gamma_u(t), \gamma_i(t) \rangle$: The **temporal interaction term**, representing the dot product of user and item latent factors:
  $$\langle \gamma_u(t), \gamma_i(t) \rangle = \sum_{k=1}^{K} \gamma_{u,k}(t) \cdot \gamma_{i,k},$$
  where:
  - $\gamma_{u,k}(t)$: The temporal latent factor for user u in dimension k, defined as:
    $$\gamma_{u,k}(t) = \gamma_{u,k} + \alpha_{u,k} \cdot \mathrm{dev}_u(t) + \gamma_{u,k,t},$$
    where:
    * $\gamma_{u,k}$: Static latent factor for user u in dimension k.
    * $\alpha_{u,k}$: Scaling parameter for the temporal deviation in dimension k.
    * $\gamma_{u,k,t}$: Localized, time-specific adjustment for user u in dimension k.
  - $\gamma_{i,k}$: Static latent factor for item i in dimension k.

Factorized Personalized Markov Chains

Markov Chains for Sequential Recommendation

Markov Chains are a type of temporal model used in recommendation systems where the next action in a sequence is assumed to depend only on the most recent action. Mathematically, let $i(t)$ represent the item interacted with at time t, and $p(i_{t+1} \mid i_t)$ represent the probability of interacting with item i_{t+1} given the previous item i_t. A first-order Markov Chain assumes:

$$p(i_{t+1} \mid i_t, i_{t-1}, \ldots, i_1) = p(i_{t+1} \mid i_t)$$

This conditional independence assumption significantly reduces the complexity of modeling sequential interactions, as the entire interaction history is summarized by the most recent item.

Personalized Markov Chains (PMC)

In a personalized recommendation context, the probability of the next item also depends on the user u. Personalized Markov Chains extend by incorporating user-specific preferences:

$$p(i_{t+1} \mid i_t, u) = p(i_{t+1} \mid i_t, u)$$

Here, user identity u acts as a conditioning variable that personalizes the sequential dynamics. For instance, different users interacting with the same item i_t may have different probabilities of transitioning to i_{t+1} based on their preferences. In practice, the model seeks to learn a **scoring function**:

$$f(u, i_{t+1} \mid i_t),$$

where f quantifies the compatibility of the user u, the previous item i_t, and the candidate next item i_{t+1}.

Factorized Personalized Markov Chains (FPMC)

Factorized Personalized Markov Chains (FPMC) extend the personalized Markov Chain model by representing user-item interactions and item-item sequential transitions as a **latent factor model**. The core scoring function in FPMC is $f(i \mid u, j)$, or the probability of seeing item i giving the joint probability of item j and the user u. We can use decomposition (essentially a generalization of the matrix factorization schemes) to make it easier:

$$f(i \mid u, j) = \underbrace{\gamma_{ui} \cdot \gamma_{iu}}_{f(i|u)} + \underbrace{\gamma_{ij} \cdot \gamma_{ji}}_{f(i|j)} + \underbrace{\gamma_{uj} \cdot \gamma_{ju}}_{f(u,j)}.$$

Here:

- $f(i \mid u)$ captures the compatibility between the user u and the next item i.
- $f(i \mid j)$ captures the sequential relationship between the previous item j and the next item i.
- $f(u, j)$ captures the relationship between the user u and the previous item j.

In practice, the latter expression cancels out when optimizing the model using a BPR-like framework because the third term is independent of the candidate next item i and does not contribute to the relative ranking of items. **In BPR we only care about relative ranking**.

$$f(i \mid u, j) = \underbrace{\gamma_{ui} \cdot \gamma_{iu}}_{\text{user's compatibility with the next item}} + \underbrace{\gamma_{ij} \cdot \gamma_{ji}}_{\text{next item's compatibility with the previous item}}$$

Intuitively, this factorization simply states that **the next item should be compatible with both the user and the previous item consumed**. The scoring function comprises two components:

1. **User-next item compatibility:** $\gamma_{ui} \cdot \gamma_{iu}$ models how compatible the next item i is with the user's preferences (user item compatiability and item user compatiability).
2. **Sequential dynamics:** $\gamma_{ij} \cdot \gamma_{ji}$ models the transition likelihood from the previous item j to the next item i.

Optimization with Bayesian Personalized Ranking (BPR)

FPMC is typically trained using the Bayesian Personalized Ranking (BPR) loss, which optimizes the model to rank observed interactions higher than unobserved (negative) interactions. The contrastive loss is defined as:

$$L = \sum_{(u,j,i,i')} - \ln \sigma \left(f(i \mid u, j) - f(i' \mid u, j) \right)$$

where:

- i is a positive (observed) next item for user u after item j.
- i' is a negative (unobserved) next item for the same context.
- $\sigma(x)$ is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$.

This loss encourages $f(i \mid u, j) > f(i' \mid u, j)$, ensuring that the model ranks positive interactions higher than negative ones.

Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are designed to handle sequential data by maintaining a notion of *memory*, encoded in a vector of **hidden states** $\{h_t\}$. These hidden states allow RNNs to capture long-term dependencies across sequences, addressing the limitation of Markov Chains, which assume conditional independence given the most recent observation.

An RNN processes an input sequence $\{x_1, x_2, \ldots, x_N\}$ to generate a sequence of outputs $\{y_1, y_2, \ldots, y_N\}$ and a sequence of hidden states $\{h_1, h_2, \ldots, h_N\}$. Formally, the hidden state and output at each time step t are updated as:

$$h_t = f(h_{t-1}, x_t; \theta_h),$$
$$y_t = g(h_t; \theta_y),$$

where:

- f is the function that updates the hidden state,
- g is the function that generates the output from the hidden state,
- θ_h and θ_y are the learnable parameters of the model.

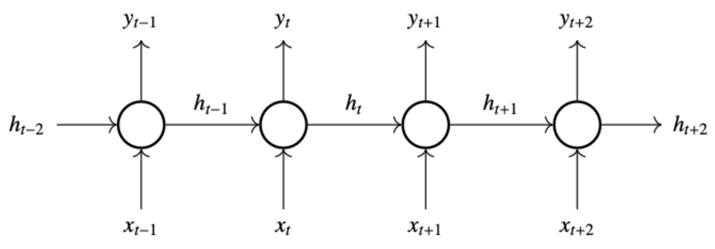

Figure 1: Simple RNN

Long Short-Term Memory (LSTM)

A fundamental challenge of RNNs is their difficulty in preserving information over long sequences due to issues like the **vanishing gradient problem**. To address this, more advanced architectures like the **Long Short-Term Memory (LSTM)** network were developed.

The **Long Short-Term Memory** network (LSTM) extends RNNs by introducing a **cell state** c_t that explicitly carries information across time steps with minimal modification. LSTMs maintain both a cell state c_t and a hidden state h_t. The key components of the LSTM are:

$$f_t = \sigma(W_f[h_{t-1}; x_t] + b_f)$$
$$i_t = \sigma(W_i[h_{t-1}; x_t] + b_i)$$
$$o_t = \sigma(W_o[h_{t-1}; x_t] + b_o)$$

$$g_t = \tanh(W_g[h_{t-1}; x_t] + b_g)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = \tanh(c_t) \odot o_t$$

Here:

- f_t (**forget gate**): Determines which part of the previous cell state c_{t-1} to retain.
- i_t (**input gate**): Controls how much of the new candidate information g_t should be added to the cell state.
- o_t (**output gate**): Controls which parts of the cell state c_t should influence the hidden state h_t and the output.
- g_t (**candidate memory**): A potential update to the cell state, calculated as a function of the input x_t and the previous hidden state h_{t-1}.
- σ: Sigmoid activation function, $\sigma(x) = \frac{1}{1+e^{-x}}$.
- $\odot$: Element-wise multiplication.
- W_f, W_i, W_o, W_g: Weight matrices for forget, input, output, and candidate memory, respectively.
- b_f, b_i, b_o, b_g: Bias terms for forget, input, output, and candidate memory, respectively.

More specifically, everything and all the weights in RNN are learned parameters:

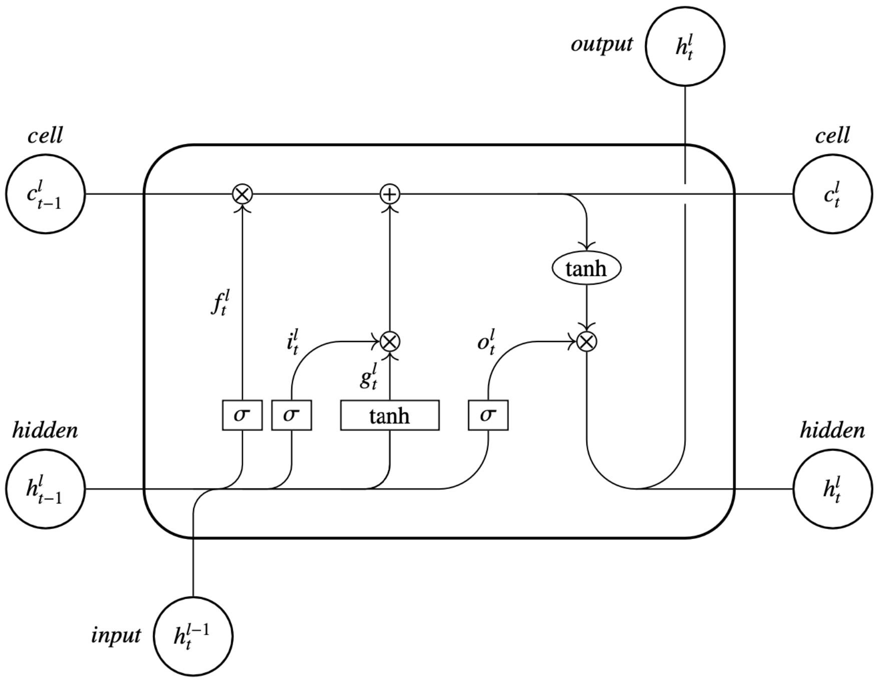

Figure 2: Visualization of an LSTM cell

1. **Forget Gate (f_t)**: Controls what portion of the previous cell state c_{t-1} to retain.

$$f_t = \sigma(W_f[h_{t-1}; x_t] + b_f).$$

2. **Input Gate (i_t)**: Determines how much new information to add to the cell state.
$$i_t = \sigma(W_i[h_{t-1}; x_t] + b_i).$$

3. **Candidate Memory (g_t)**: Proposes new information to add to the cell state.
$$g_t = \tanh(W_g[h_{t-1}; x_t] + b_g).$$

4. **Cell State Update (c_t)**: Combines the retained information from c_{t-1} with new information from g_t.
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t.$$

5. **Output Gate (o_t)**: Controls what information from the cell state is passed to the hidden state h_t.
$$o_t = \sigma(W_o[h_{t-1}; x_t] + b_o).$$

6. **Hidden State (h_t)**: Produces the output based on the cell state and the output gate.
$$h_t = \tanh(c_t) \odot o_t.$$

Recurrent Recommendations

An early paper to explore the use of recurrent networks for recommendation is Hidasi et al. (2016), which explored the problem of **session-based recommendation**, where user interactions are divided into distinct 'sessions' (typically using some heuristic based on interaction timestamps). This method seeks to pass sequences of items into a recurrent network such that the hidden state of the network is capable of predicting the next interaction. **This model does not learn a user representation: rather the user 'context' is captured via the hidden state of the recurrent network.** The paper argue that a benefit of this type of approach is that it will work well in situations where long user histories are unavailable (e.g. on niche platforms or long-tailed datasets). In such cases, techniques like those we developed for the Netflix Prize are unreliable as effective user representations (γ_u) cannot be learned from only a few interactions.