Session-based Recommendations with Recurrent Neural Networks

**Balázs Hidasi**
**Alexandros Karatzoglou**
**Linas Baltrunas**
**Domonkos Tikk**

balazs.hidasi@gravityrd.com
alexk@tid.es
lbaltrunas@netflix.com
domonkos.tikk@gravityrd.com

Session-based recommendation

Permanent cold start: where personalized recommendations fail

- **User identification**: Many sites (e.g., classifieds, video services) don’t require users to log in. Although some form of identification is possible, it is not reliable.
- **Intent/theme**: Sessions usually have a goal or a specific theme. Different sessions of the same user center around different concepts. The entire user history may not help much in identifying the user’s current needs.
- **Never/rarely returning users**: High percentage of the users of webshops come from search engines in search for some products and rarely return.

Workaround in practice

- **Item-to-item recommendations**: Recommend similar or frequently co-occurring items.

We explore item-to-session recommendations. By modeling the whole session, more accurate recommendations can be provided. We propose an RN-based approach to model the session and provide session-based recommendations.

Gated Recurrent Unit

Hidden state is the mix of the previous hidden state and the current hidden state candidate (controlled by the update gate):

$$ h_t = (1 - z_t) h_{t-1} + z_t h_t $$

The reset gate controls the contribution of the previous hidden state to the hidden state candidate:

$$ h_t = \tanh(W_x x_t + U_r h_{t-1}) $$

Reset gate: $r_t = \sigma(W_r x_t + U_r h_{t-1})$

Update gate: $z_t = \sigma(W_z x_t + U_z h_{t-1})$

Architecture

- Input: item of the actual event
- Output: likelihood for every item for being the next one in the event stream

Different layers for all items

Feedforward layers (optional)

GRU layer

GRU layer

Embedding layers (optional)

Input: actual item, T-of-N coding

Experiments

<table>
<thead>
<tr>
<th>Data</th>
<th>Description</th>
<th>Items</th>
<th>Train Sessions</th>
<th>Events</th>
<th>Test Sessions</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSC15</td>
<td>RecSys Challenge 2015. Clickstream data of a webshop.</td>
<td>37,483</td>
<td>7,966,257</td>
<td>31,637,239</td>
<td>15,324</td>
<td>71,222</td>
</tr>
<tr>
<td>VIDEO</td>
<td>Watch events collected from a video service platform.</td>
<td>327,929</td>
<td>2,954,816</td>
<td>13,180,128</td>
<td>48,746</td>
<td>178,637</td>
</tr>
</tbody>
</table>

Findings (Architecture, training & parameters)

- Single layer GRU performs best
- Pre/postprocessing FF layers are not needed
- Adam works better than RMSProp
- TOP loss is better overall than other losses
- Pointwise losses (e.g., cross-entropy) are unstable
- Feeding the network earlier events of the session (i.e. reminding it) does not improve performance
- LSTM & RNN are inferior to GRU
- The number of hidden units has the highest impact on performance

Adapting GRU to the RecSys task

Session-parallel mini-batches

Motivation:
- High variance in the length of the sessions (from 2 to 100s of events)
- The goal is to capture how sessions evolve

Approach:
- Have an ordering of all sessions (e.g., random order or order by time)
- Take the first events of the first X sessions (X = mini-batch size) to form the first input mini-batch.
- The desired output is formed from the second events of the first X sessions.
- The second mini-batch (input) is formed from the second events, etc.
- If a session ends, put the next available session in its place and reset the corresponding hidden state.

Sampling the output

Motivation:
- The number of items is generally high: 100,000s or even a few millions.
- Training scales with the number of events, hidden units and outputs ($O(N, W_{r_i})$). The latter equals to the number of items.
- Models need to be trained frequently to keep up with the changes in the item catalog and user behavior.

Approach:
- For an input, the desired output is a one-hot vector over all items.
- Always compute the score for the coordinate corresponding to the desired item. Sample the others.
- Popularity based sampling: it is more likely that the lack of an event on a more popular item means negative feedback.
- Use the items of the other examples of the mini-batch as the negative examples for each event in the mini-batch. This is a form of popularity based sampling with several practical benefits.

Ranking loss

Motivation:
- The ultimate goal of recommenders is to rank the items.
- Pointwise and pairwise rankings have been applied with great success (listwise ranking is not scalable enough in practice).
- Pairwise ranking (A is preferred over B) often performs better.

Approach:
- **BPR**: Adapt Bayesian Personalized Ranking for multiple negative samples.
  $$ L = \frac{1}{N} \sum_{i=1}^{N} \log \left( \frac{e(r_{ui} - r_{ui})}{\sum_{j \neq i} e(r_{ui} - r_{uj})} \right) $$
- **TOP**: This ranking loss was devised by us for this task. It is the approximation of the relative rank of the desired item. Regularization is added for the sake of stability.
  $$ L = \frac{1}{N} \sum_{i=1}^{N} \sigma(r_{ui} - r_{ui}) + \alpha \sigma^2(r_{ui}) $$

Try the algorithm: [https://github.com/davidvis/GRU4Rec](https://github.com/davidvis/GRU4Rec)