

Session-based Recommendations with Recurrent Neural Networks

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 RNN-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):

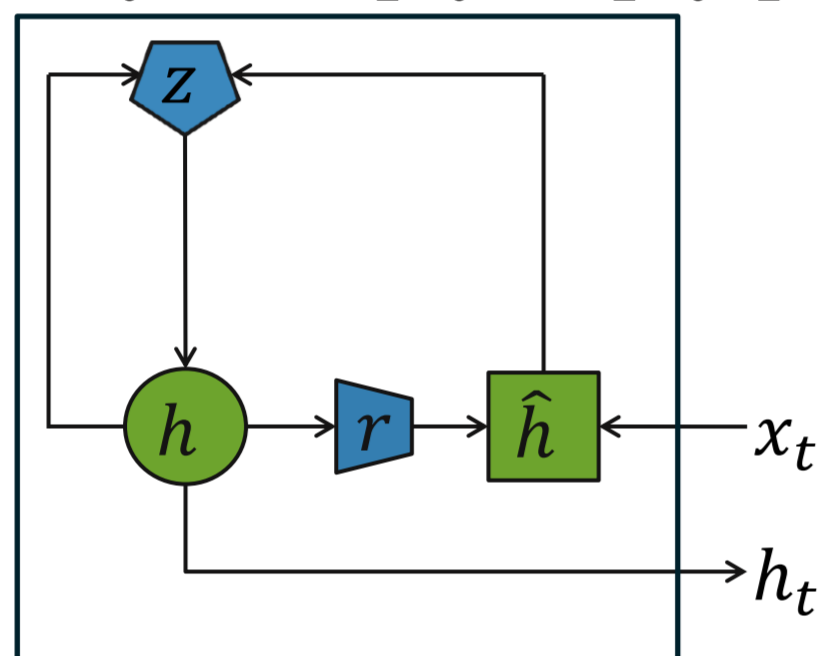
$$h_t = (1 - z_t)h_{t-1} + z_t\hat{h}_t$$

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

$$\hat{h}_t = \tanh(Wx_t + U(r_t \circ h_{t-1}))$$

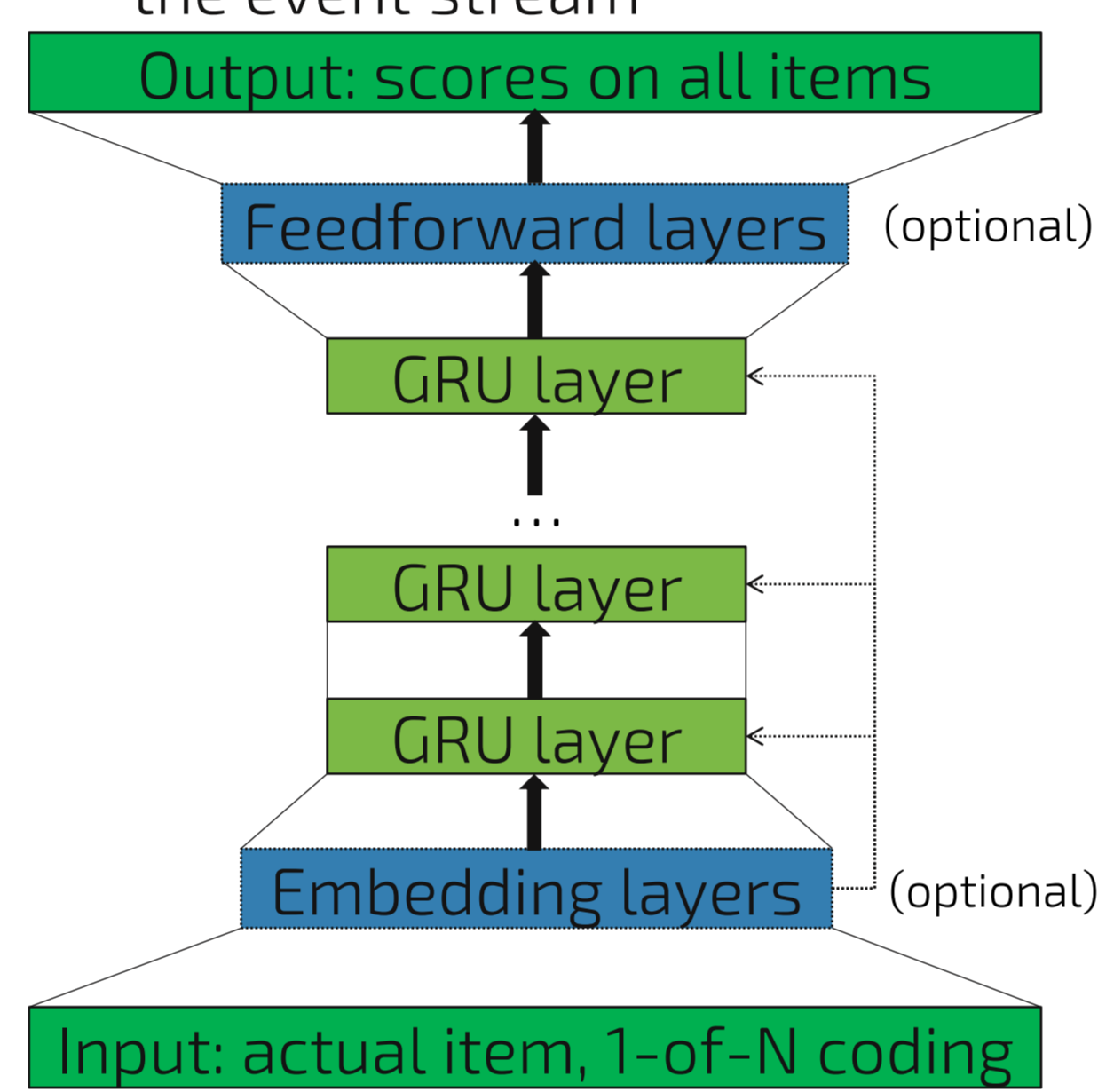
Reset gate: $r_t = \sigma(W_r x_t + U_r h_{t-1})$

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



Adapting GRU to the RecSys task

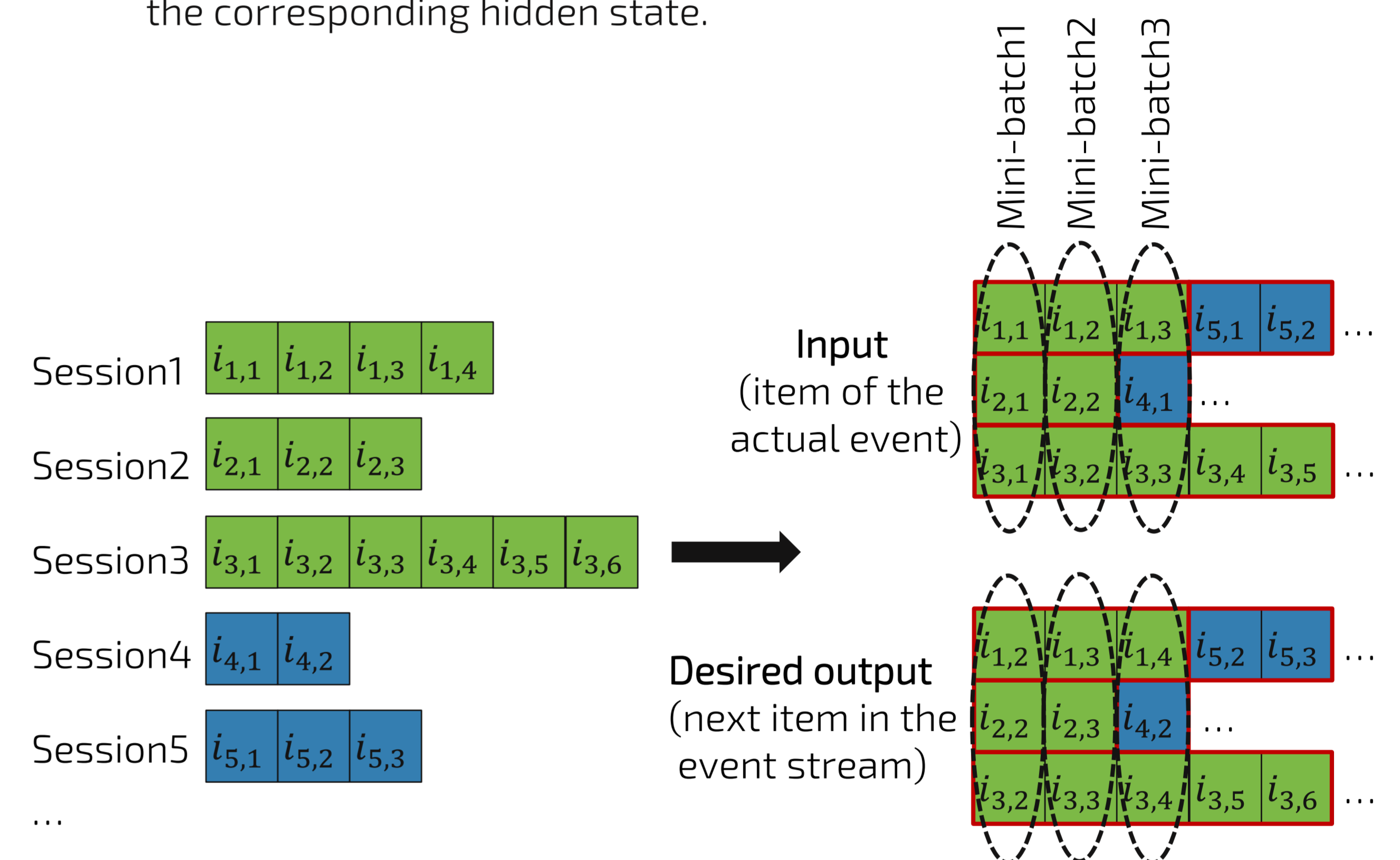
Session-parallel mini-batches

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.

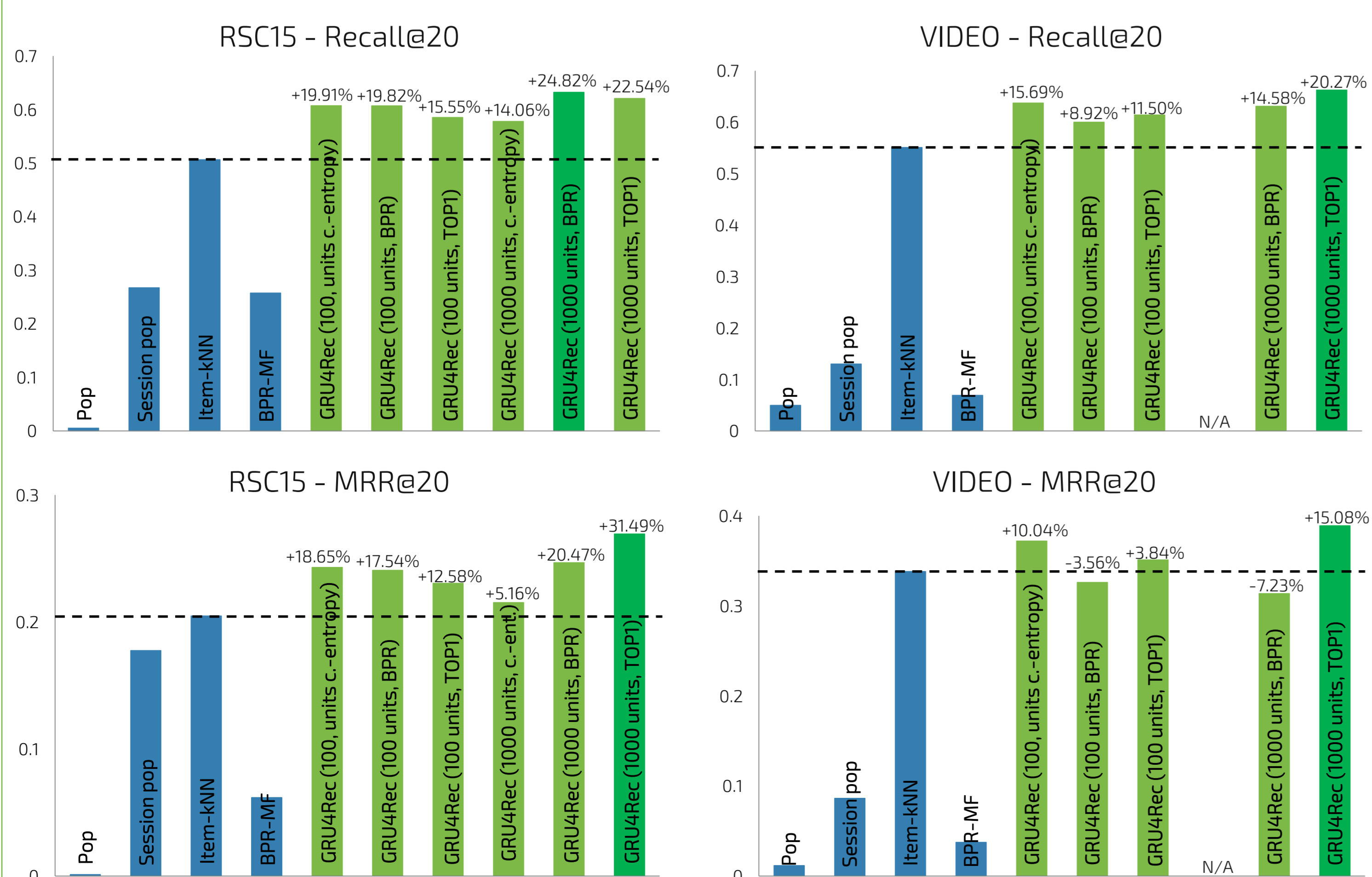


Experiments

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

Findings (Architecture, training & parameters)

- Single layer GRU performs best
- Pre/postprocessing FF layers are not needed
- Adagrad works better than RMSProp
- TOP1 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



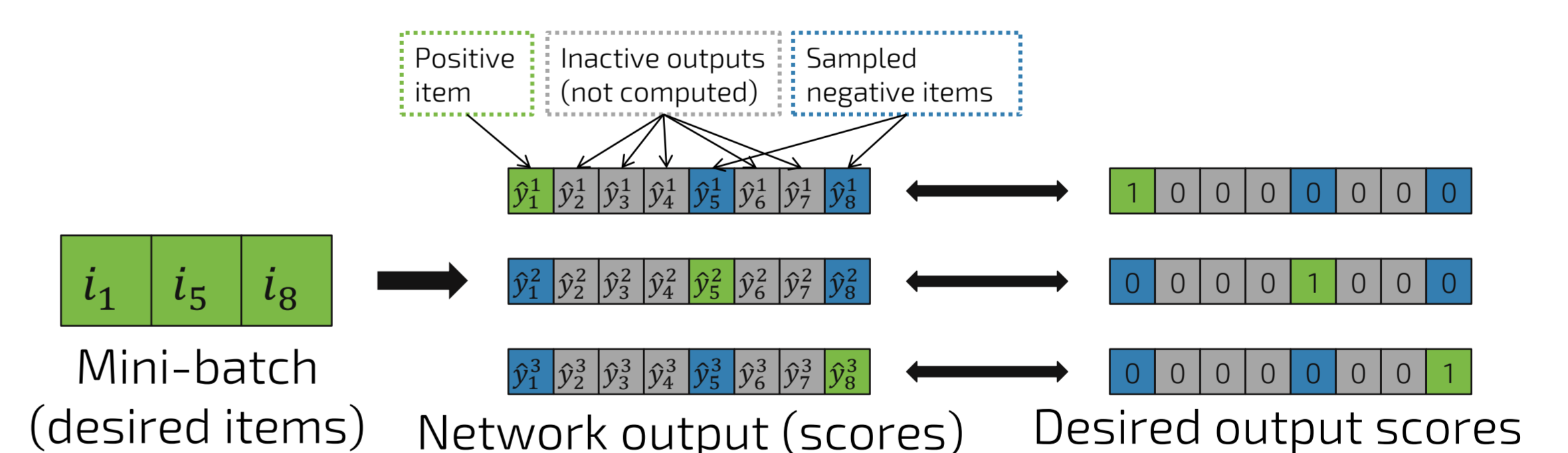
Sampling the output

Motivation:

- The number of items is generally high: 100,000s or even a few millions.
- Training scales with the product of the number of events, hidden units and outputs ($O(N_e H N_i)$). 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.

$$L = -\frac{1}{N_S} \sum_{j=1}^{N_S} \log(\sigma(\hat{r}_{s,i} - \hat{r}_{s,j}))$$

- **TOP1:** 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.

$$L = \frac{1}{N_S} \sum_{j=1}^{N_S} \sigma(\hat{r}_{s,i} - \hat{r}_{s,j}) + \sigma(\hat{r}_{s,j}^2)$$

