Sequence-Aware Recommenders

Recommender Systems

Dimitris Sacharidis

introduction

recommenders in practice

- differ from standard recommenders in three main ways:
  - long-term vs. short-term interests
  - users vs. sessions
  - richer input

long-term vs. short-term interests

- typically recommenders learn **correlations** in a ratings matrix
- by observing user behavior in the past
- that capture the **long-term** user preferences
  - e.g., tastes of users in movies, music

- **assumption**: what people look for is determined by long-term interests

- in practice, this may not necessarily hold
- **short-term** interests may be as important, or more
  - the *intent* of the user
  - e.g., when playing music, what I just listened to matters most

users vs. sessions

- in some cases, long-term profiling is not possible
- the system may not know of users
  - e.g., users not logging in, just browsing

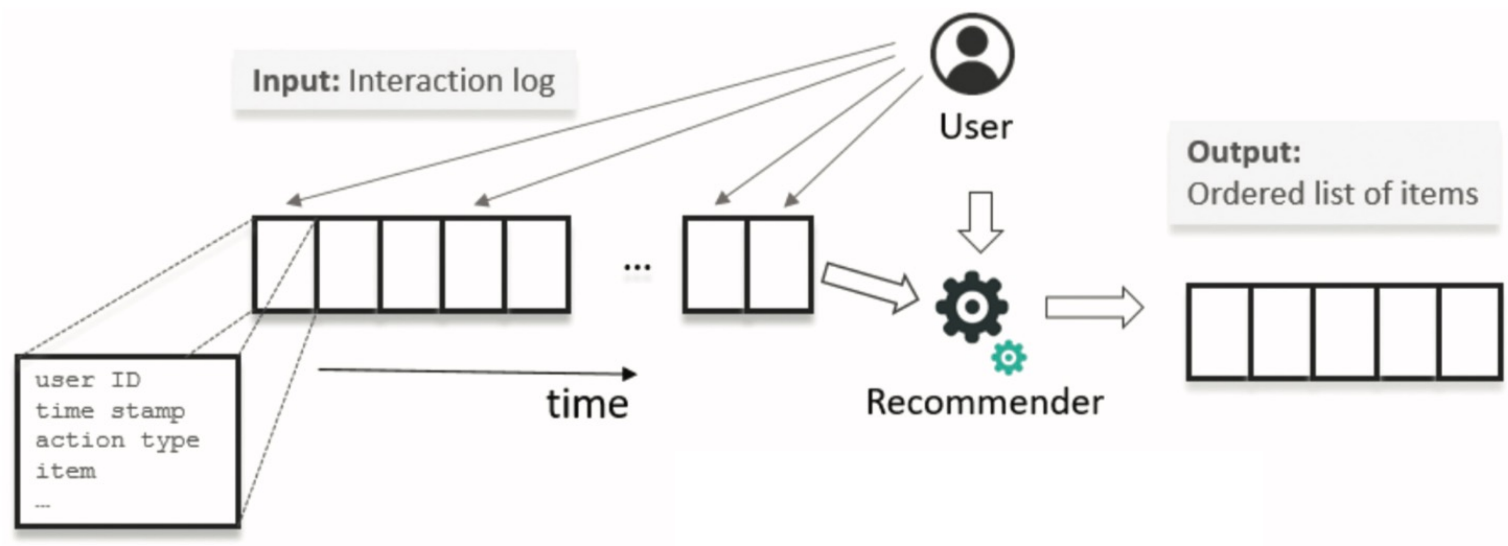
- system only sees **sessions** of activity
- captures the short-term preferences of the user
- but still needs to make recommendations

richer input

- users give feedback from which the system learns
- originally, **explicit feedback**, e.g., ratings
- then, **implicit feedback**, e.g., purchases
- now, **richer** implicit feedback, e.g., an **interaction log**
  - **multiple actions** possible for an item
    - e.g., item-view, item-purchase, add-to-cart

sequence-aware recommenders

- important distinction:
- input is a **sequence** of actions, the **interaction log**
  - order matters



[2018 ACM Comp. Surveys M. Quadrana et al.] *Sequence-Aware Recommender Systems*

input

- how much past information is used to make recommendations

- **last-N interactions**
  - sometimes only last interaction
    - e.g., next Point-Of-Interest (POI) recommendation
    - e.g., “customers who bought X also bought”

- **session-based recommender**
  - not aware of users; e.g., not logged-in, anonymous
  - *short-term* interest

- **session-aware recommender**
  - past sessions of users are known; e.g., logged-in, cookies
  - *short-term* and *long-term* interest

output

- ordered list of items, with different interpretation

- **alternatives**; e.g., other hotels
- **complements**; e.g., accessories to an item
- **continuations**
  - with restrictions on order: e.g., course prerequisites
  - without restrictions on order: e.g., next tracks in an automated playlist

conventional algorithms

U-U CF, I-I CF, Matrix Factorization

conventional methods

- do they apply? sure

- let's simplify a bit:
  - one type of action; e.g., rating, click, purchase
  - order of actions does not matter; set of previous item interactions

- can we handle sessions instead of users?

- yes! treat a **session** like a **user**
- let's revisit conventional methods

user-user CF

- user=session; a session is a set of previously interacted items
- identical to UU CF for implicit feedback
- called **session-based kNN** method in [2017 RecSys]
- shown to outperform more elaborate methods

$$\hat{r}(s, i) = \sum_{s' \in N(s)} w_{s,s'} \cdot \mathbb{1}(i \in s')$$

predicted score of target item to current session

neighborhood of current session

session-session cosine similarity

only consider neighbors that contain target item

[2017 RecSys D. Jannach, M. Ludewig] *When Recurrent Neural Networks meet the Neighborhood for Session-Based Recommendation*

item-item CF

- again, user=session
- similarity of items based on the sessions they appear in
    - or learn the weights as in SLIM
- prediction for a target item is the sum of similarities of all current session items
    - or consider only the last session item

$$\hat{r}(s,i) = \sum_{j \in s} w_{i,j}$$

predicted score of target item to current session

item-item similarity

matrix factorization et al.

- again, user=session
- one issue, must train for each new session
  - why? must learn the features of current session

- **alternative**: do not explicitly learn features for current session
- instead learn two sets of features for items
  - the *q*'s and the *y*'s (just like SVD++)
  - a session is represented by the *y*'s of the items it contains

$$\hat{r}(s,i) = q_i^\intercal \sum_{j \in s} y_j = \sum_{j \in s} q_i^\intercal y_j$$

predicted score of target item to current session

item features

item-in-session features

sequence-aware algorithms

Markov Processes, Recurrent Neural Networks

Markov Processes

- (a.k.a. Markov chains) describe transitions between **states** of the world
- S_t is the state at time t
- *"the future is independent of the past given the present"*

$$\underbrace{Pr[S_{t+1}}_{\textit{future}}|\underbrace{S_1, \ldots, S_t}_{\textit{past}}] = Pr[S_{t+1}|\underbrace{S_t}_{\textit{present}}]$$

- the present state tells you everything you need to know
- throw away history (or carefully encode it into the state!)

Markov Processes

- the world can be fully described by the **state transition probabilities**

$$P_{s,s'} = Pr[S_{t+1} = s' | S_t = s]$$

the probability of moving from state s to s'

- these state transition probabilities can be nicely organized in the **state transition matrix**

Markov Processes for Recommendations

- modeling the recommendation problem as an MP

- **state** is the **sequence** of previous user interactions
- typically sequences of length up to k

$$s = (i_1, \ldots, i_k)$$

- how many states? too many! m^k (m is the number of items)
  - so k has a small value like 3 or even 1

Markov Processes for Recommendations

- suppose state transition probabilities are known
  - (we come back to this)

- then to recommend:
  - given the **present**, find the most probable next state, the **future**
  - return the **last item** in the future state

- let $s = (i_1, \ldots, i_k)$ be the **present state**
- and assume $s' = (i_2, \ldots, i_{k+1})$ is the most probable **future state**
- then recommend item i_{k+1}

Markov Processes for Recommendations

- how to learn the state transition probabilities

- via **maximum likelihood estimation**
- which involves **counting** how many times sequences appear in the interaction log

- consider a **from** state $s = (i_1, \ldots, i_k)$ and a **to** state $s' = (i_2, \ldots, i_{k+1})$
- the transition probability is computes as

$$P_{s,s'} = \frac{Pr[(i_1, \ldots, i_{k+1})]}{Pr[(i_1, \ldots, i_k)]}$$

how many times we see the transition, i.e., ***join*** *of the* ***from*** *and* ***to*** *sequences*

how many times we see the ***from*** *sequence*

Markov Processes for Recommendations

- sparseness issue: the **state space may be too large** and the **observed transitions too few**

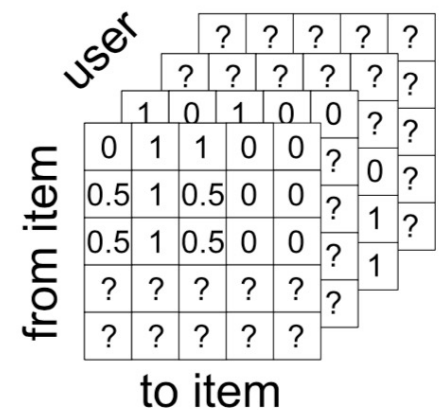
- some ideas:
  - make k=1; next item transitions
  - skipping, clustering, mixture; see [2005 JMLR G. Shani et al.]

[2005 JMLR G. Shani et al.] *An MDP-Based Recommender System*

Markov Processes for Recommendations

- MPs address **session-based** problem
  - not user personalized

- what if we have users *and* sessions, the **session-aware** problem
- transition matrix per user, based on her sessions
- i.e., a transition **cube**: from-item, to-item, user

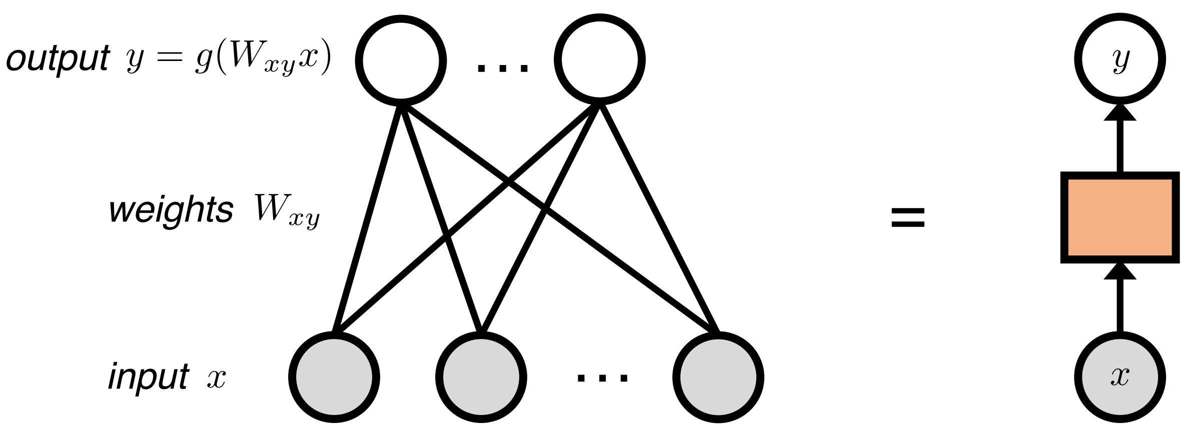


- the cube is even more sparse!
- but we can **factorize** the cube to exploit correlations across its dimensions

[2010 WWW S. Rendle et al.] *Factorizing Personalized Markov Chains for Next-Basket Recommendation*

from Neural Networks …

- an NN layer transforms an **input** vector x to an **output** vector y
- two ingredients:
  - nonlinear function (e.g., tanh, ReLU): $g()$
  - weight matrix: W_{xy}

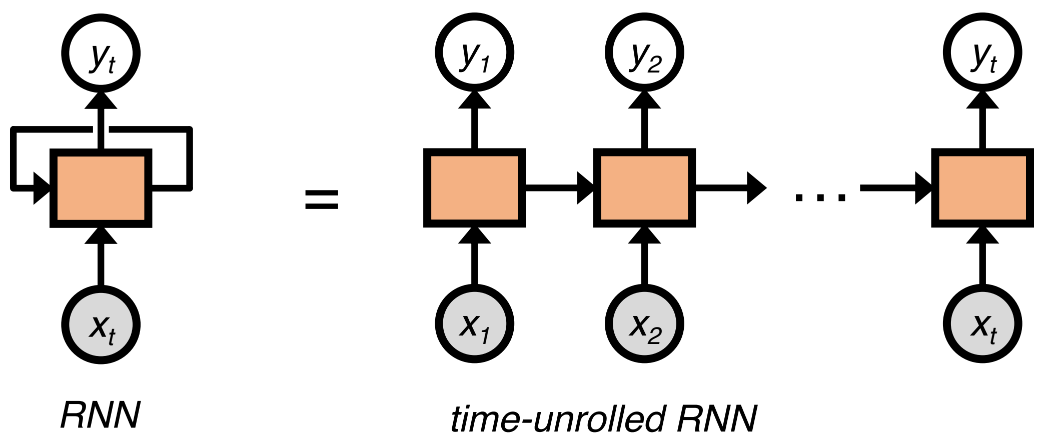


single neural network layer

abstraction

… to Recurrent Neural Networks

- can transform a **sequence** of vectors to a **sequence** of vectors
- RNNs have a hidden state that controls its output
  - a feedback loop
- different flavors: basic RNN, LSTM, GRU



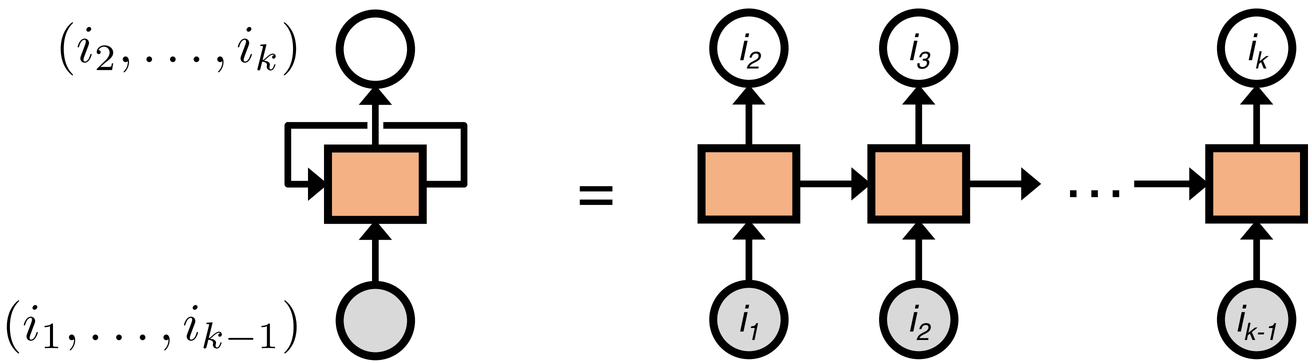
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNNs for Recommendations

for training:

- feed a session to the RNN $s = (i_1, \ldots, i_k)$
- at each step, we want the output to be the next item

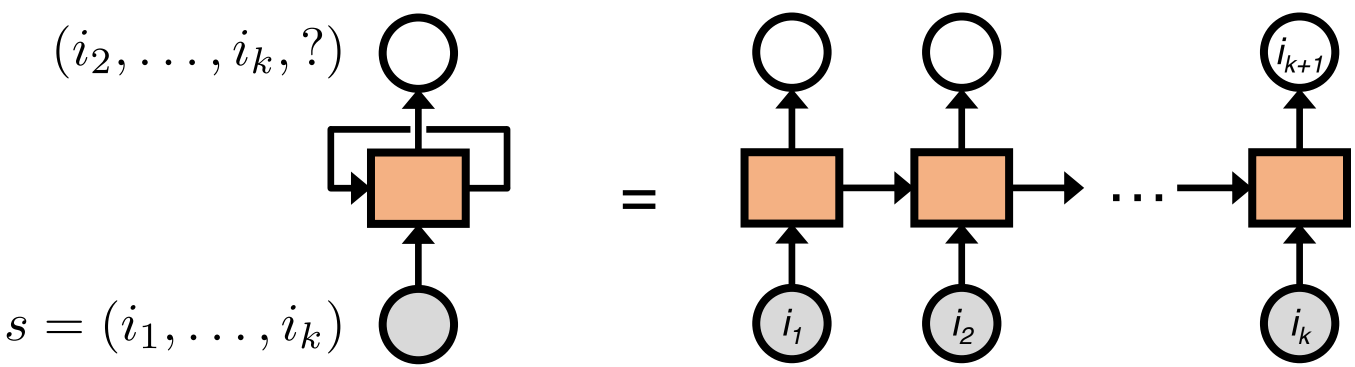


[2016 ICLR B. Hidasi et al.] *Session-based Recommendations with Recurrent Neural Networks*

RNNs for Recommendations

to recommend:

- feed the **current session**
- look at the **last output**, to select the next item



[2016 ICLR B. Hidasi et al.] *Session-based Recommendations with Recurrent Neural Networks*

RNNs for Recommendations

- not always better than conventional algorithms [2017 RecSys]
- combining them brings benefits

[2017 RecSys D. Jannach, M. Ludewig] *When Recurrent Neural Networks meet the Neighborhood for Session-Based Recommendation*