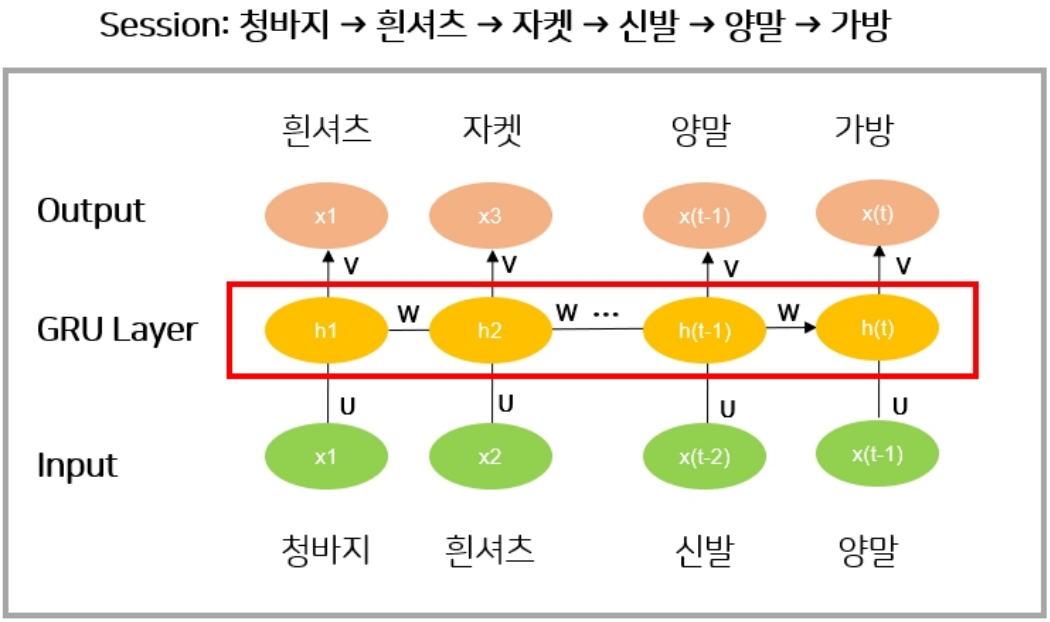
(GRU4REC) SESSION-BASED RECOMMENDATIONS WITH RECURRENT NEURAL NETWORKS

Journal/Conference	ICLR
Tags	Recsys Session-based
한줄요약	RNN의 하나의 종류인 GRU를 이용하여 next click item을 예측하는 session-based recommendation을 구현함
논문링크	https://arxiv.org/pdf/1511.06939.pdf
저자	Balazs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, Domonkos Tikk
원논문 PDF	(GRU4REC)SESSION-BASED RECOMMENDATIONS WITH RECURRENT NEURAL NETWORKS.pdf
출간연도	2016
발표자/발표일자	김소정/2023/08/20
리뷰완료	☑

▼ 0. GRU4REC SUMMARY

- session에서는 시간 순서를 고려하니까 RNN을 써야하는게 아닌가?라는 질문에서 출발하게 된 논문
- 해당 논문은 RNN 중 GRU를 사용한 모델임



- A를 입력했을 때 B를 맞추고 A,B를 입력했을 때 C를 맞추고 A,B,C를 입력했을 때 D를 맞추고 A,B,C,D를 입력했을 때 E를 맞추는 형식으로 학습함
- 예) (청바지)을 입력으로 넣었을때 (흰셔츠)를 맞추고, (청바지,흰셔츠)을 입력으로 넣었을때 (자켓)를 맞추고 (청바지, 흰셔츠, 자켓)을 입력으로 넣었을때 (신발)를 맞추는 식
- 입력 sequence를 순차적으로 넣고 next 아이템을 맞추려고 함
- GRU4rec은 마지막 히든 스테이트가 의미하는 바가 가장 처음 x(1))부터 현재시점(x(t-1))까지의 입력을 잘 aggegation해서 표현한 벡터가 h(t)임
- 이 h(t)에 마지막 아이템인 청바지의 정보가 더 많이 들어감 -> 이 부분이 주로 고려됨
- 이로 인해 마지막 아이템인 청바지를 맞추는데 있어 h(t) 뿐만 아니라 h(t-1)이나 h1등 중간 히든스테이트도 같이 고려해서 예측하는게 성능 개선에 도움이 되지 않을까 생각하여 후속 session model이 생성됨
- 기존 GRU4rec은 h(t)만 썼는데 NARM은 중간 히든스테이트도 사용함
- NARM의 글로벌 인코더는 마지막 h(t)가 x1~xt를 고려해서 만든 건데 이 h(t)를 글로벌 인코더로 사용하겠다는 것 → 입력으로 주어진 모든 정보를 h(t)가 압축하는 부분이고 이 부분은 GRU4rec과 동일함

▼ 1. Abstract

기존 추천의 한계

- only on short session-based data (e.g. a small sportsware website) instead of long user histories (as in the case of Netflix)
- matrix factorization approaches are not accurate
- overcome in practice by resorting to item-to-item recommendation

해당 논문에서의 제안점

- by modeling the **whole session,** more accurate recommendations can be provided.
- therefore propose an **RNN based approach** for session-based recommendations.

▼ 2. Introduction

기존 session recommendation 한계

- most session-based recommendation systems deployed for e-commerce are based on relatively simple methods that **do not make use of a user profile** e.g. **item to-item similarity, co-occurrence, or transition probabilities.**

▼ 3. Recommendation with RNNs

▼ 1. RNN 과 GRU

1) RNN

- Standard RNNs update their hidden state h using the following update function

$$\mathbf{h_t} = g(W\mathbf{x_t} + U\mathbf{h_{t-1}}) \quad (1)$$

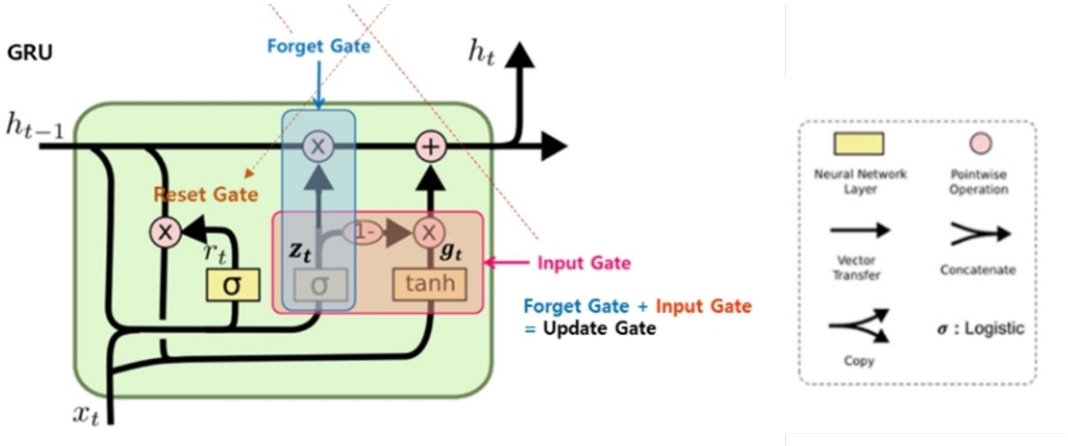
- g: a smooth and bounded function such as a logistic sigmoid function
- xt: the input of the unit at time t
- An RNN outputs a probability distribution over the **next element of the sequence**,
  given its current state ht

→ RNN은 기존 feedforward deep model과는 다르게 hidden state가 존재함

→ RNN은 current state인 ht를 가지고 next element를 예측함

2) GRU

- RNN은 sequence가 길어질수록 vanishing/exploding gradient problem 문제가 있음
- 이로 인해 LSTM과 GRU와 같은 모델들이 등장했는데 해당 모델들은 unit의 hidden state를 얼마나 업데이트 할지를 조절할 수 있음(Gate)
- GRU는 Update & Reset Gate로 구성이 되어있음



1> previous & candidate activation 사이의 조정을 통한 hidden state 산출

- Update gate인 z(t)를 통해 previous & candidate activation의 반영 정도를 조정하여 output인 hidden state를 산출함

$$\mathbf{h_t} = (1 - \mathbf{z_t})\mathbf{h_{t-1}} + \mathbf{z_t}\hat{\mathbf{h}}_\mathbf{t} \tag{2}$$

- z(t)가 이전 정보의 비율을 결정, (1-z(t))가 현재 정보의 비율을 결정함 → 전자가 Forget Gate, 후자가 Input Gate의 역할을 함
- z(t)가 1이면 Forget Gate가 열리며, 0이면 Input Gate가 열림

2> Update Gate

- LSTM의 Forget Gate와 Input Gate를 병합하여, 과거 & 현재 정보를 얼마나 반영할지 구하는 단계
- Update Gate는 x(t)와 h(t-1)으로 결정됨

$$\mathbf{z_t} = \sigma(W_z\mathbf{x_t} + U_z\mathbf{h_{t-1}}) \tag{3}$$

3> Reset Gate

- 새로운 입력 정보를 이전 메모리와 어떻게 합칠지 결정함
- 이전 상태/정보 h(t-1)에서 얼만큼을 선택해서 내보낼지 제어함

$$\mathbf{r_t} = \sigma(W_r\mathbf{x_t} + U_r\mathbf{h_{t-1}}) \tag{5}$$

- 직전 시점의 hidden state인 h(t-1)과 현재의 입력정보인 x(t)에 대해 sigmoid를 적용하여 산출함

4> Candidate activation function

- h(t) 틸다는 reset gate를 사용하여 만들어짐
- Candidate activation function은 x(t)와 h(t-1), Reset Gate로 결정됨

$$\hat{\mathbf{h}}_\mathbf{t} = \tanh\left(W\mathbf{x_t} + U(\mathbf{r_t} \odot \mathbf{h_{t-1}})\right) \tag{4}$$

▼ 2. GRU Customization for Recsys

▼ 1) Input & Output

(1) input:

- actual state of the session
- The state of the session can either be the **item** of the actual event or **the events** in the session so far

(2) output:

- item of the next event in the session.

(3) 인코딩 2가지 방식

1> 1-of-N encoding
i.e. the **input vector's length equals to the number of items**
and only the coordinate corresponding to the active item is one, the others are zeros.

2> weighted sum of these representations

in which events are discounted if they have occurred earlier.
→ the **1-of-N encoding** always performed **better**

▼ 2) Architecture

- 코어는 GRU Layer이고 Output 전에 feedforward layer를 넣을 수 있음
- output은 the likelihood of being the next in the session for each item을 의미함
- GRU layer가 여러개 사용되는 경우, 이전 layer의 hidden state는 다음 번의 input으로 사용됨
- Input 값은 GRU layer에 선택적으로 connected 될 수 있고 이는 성능을 향상시킴

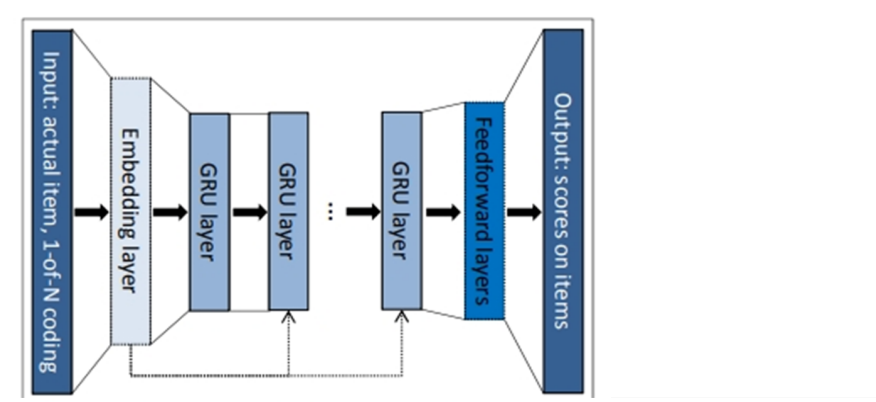


Figure 1: General architecture of the network. Processing of one event of the event stream at once.

▼ 3) (Mini-batch) Session-PARALLEL MINI BATCHES

- NLP task에서는 in-sequence mini-batch를 사용하는데 예를 들어 문장의 단어에 슬라이딩 윈도우를 사용하여 이 윈도우로 나눈 조각들을 서로 붙여 미니 배치를 형성하는 것이 일반적임
- 하지만 이 방법은 다음의 2가지 이유에서 해당 task에서는 적합하지 않음

  (1) the length of sessions can be very different
  (2) our goal is to capture how a session evolves over time,
  so breaking down into fragments would make no sense.

  → session-parallel mini-batches 사용

session-parallel mini-batches

- 세션을 병렬적으로 구성하여 미니배치 구성

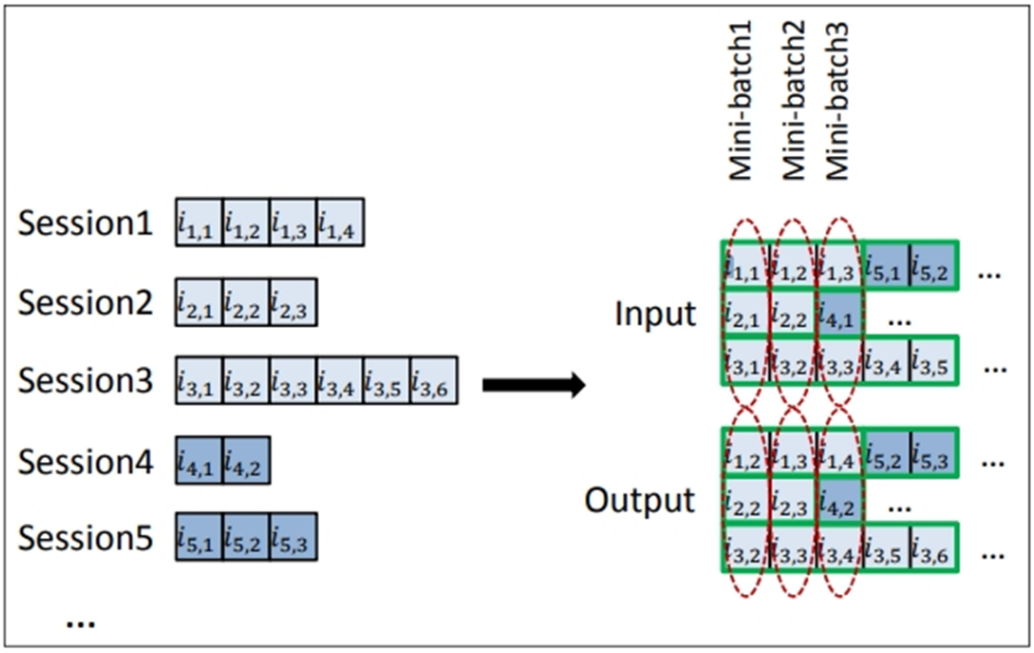


Figure 2: Session-parallel mini-batch creation

1) create an **order for the sessions**.
2) use the **first event of the first X sessions** to form the **input of the first mini-batch**
(the desired output is the second events of our active sessions).

3) The second mini-batch is formed from the second events and so on.

If any of the sessions end, the next available session is put in its place.
Sessions are assumed to be independent,
thus we reset the appropriate hidden state when this switch occurs.

→ sessionID별로 그 안에 클릭한 item들이 있을거고 해당 item의 순서대로 mini batch가 병렬적으로 생성됨

▼ 4) (Sampling) Sampling on the output

- 추천시스템의 경우, item이 많을 때 유용한데, 실제로 모든 items과 events에 대해서 score를 계산하기에는 무리가 있기에 output을 sampling하여 일부 item에 대해서만 score를 생성해야함

- 인기 있는 item일수록 user가 그 item에 대해 알 확률이 높음, missing events는 user가 dislike한다고 봄
- 그러므로 item의 인기도에 비례하여 item을 sampling 해야함
- training exmaple에 대해 separate samples을 생성하기 보다는, mini-batch의 the other training example을 negative example로 이용함

Negative sampling

- 1000개 상품 중 1번 상품만 postive이고, 나머지 아이템을 negative라고 보면
- 999개 중 랜덤 샘플링하는데 negative인 999개를 모두 학습하는 것은 비효율적이라고 보기 때문임
- 전체 negative sample을 모두 사용하여 학습하는 것보다 효율적이라고 함

Benefit

1) sampling을 skip하여 computational time을 줄일 수 있음

2) code less complex

3) popularity-based-sampling

- mini-batch의 the other training examples 안에서 item이 될 확률은 인기도에 비례하기 때문임.

▼ 5) (Loss) Ranking Loss

- 추천시스템은 item에 대한 랭킹을 매기는 것인데,
  해당 task를 classification task로 치환하여 진행을 할때 learning to rank 방식이 다른 방식보다 성능이 좋다고 알려져 있음
- LTR 관련해서는 다음 3가지로 나뉨

1. Pointwise ranking

- estimates the score or the rank of items **independently of each other** and the loss is defined in a way so that the rank of relevant items should be low.

2. Pairwise ranking

- compares the score or the **rank of pairs** of a positive and a negative item and the loss enforces that the rank of the positive item should be lower than that of the negative one.

3. Listwise ranking

- uses the scores and **ranks of all items**
  and compares them to the perfect ordering.
- As it includes sorting, it is usually computationally more expensive and thus not used often.

→ 해당 논문에서는 pointwise & pairwise ranking losses를 사용
→ pointwise ranking는 해당 네트워크에서는 unstable함
→ Pairwise ranking losses가 상대적으로 perform well
→ 논문에서는 BPR & TOP1을 사용함

▼ 6) (Loss) BPR & TOP1

1. BPR(Bayesian Personalized Ranking)

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

N_S : sample size
$\hat{r}_{s,k}$: the score on item k at the given point of the session
i : the desired item (next item in the session)
j : the negative samples

- **matrix factorization** method that **uses pairwise ranking loss**.
- **compares the score of** a **positive and a sampled negative item**
- 해당 논문에서는 positive item과 sampling된 items의 score를 비교하여 이에 대한 loss 평균을 사용함

2. TOP 1

- 해당 loss는 task에 맞게 변경됨
- It is the regularized approximation of the **relative rank of the relevant item.**
- The relative rank of the relevant item is given by

$$\frac{1}{N_S} \cdot \sum_{j=1}^{N_S} I\{\hat{r}_{s,j} > \hat{r}_{s,i}\}$$

$I\{\cdot\}$: approximate this with a sigmoid.

- Optimizing for this would modify parameters so that the score for i would be high.
- However this is unstable as certain positive items also act as negative examples
  and thus scores tend to become increasingly higher.
- To avoid this, we want to **force the scores of the negative examples to be around zero** → add a **regularization term** to the loss.
- It is important that this term is in the same range as the relative rank and acts similarly to it.
- The final loss function is as follows:

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

▼ 4. Experiments

▼ 1) Datasets

(1) RecSys Challenge 2015 = RSC15 = Yoochoose

- **click-streams of an ecommerce** site that sometimes end in purchase events.

<training>

- training set of the challenge and keep only the click events.
- filter out sessions of length 1.
- The network is trained on 6 months of data,
  containing 7,966,257 sessions of 31,637,239 clicks on 37,483 items.

<test>

- subsequent day for testing.
- filter out clicks from the test set where the item clicked is not in the train set. (?)
- After the preprocessing we are left with 15,324 sessions of 71,222 events for the test set.

(2) Youtube-like OTT video service platform = VIDEO

- Events of watching a video for at least a certain amount of time were collected.
- The training data consists of all but the last day of the aforementioned period and has ~ 3 million sessions of ~ 13 million watch events on 330 thousand videos.
- The test set contains the sessions of the last day of the collection period and has ~ 37 thousand sessions with
  ~ 180 thousand watch events.

▼ 2) 평가지표: Recall@20 & MRR@20

Recall@20

- the proportion of cases having the **desired item** among the **top-20** items in all test cases.
- Recall does **not consider the actual rank** of the item as long as it is amongs the top-N.
- Recall also usually correlates well with important online KPIs, such as click-through rate (CTR)

- Recall @N : N개 내에서 좋아하는 items/전체 좋아하는 items
- Recall@3 : (사용자 선호 아이템 2개) & (3개의 아이템 중 실제 선호하는 아이템 1개 포함) → 1/2

MRR@20

- average of reciprocal ranks of the desired items.
- MRR **takes into account the rank** of the item,
  which is important in cases where the order of recommendations matter
  (e.g. the lower ranked items are only visible after scrolling).
- 어느 위치에서 사용자가 선호하는 아이템이 나오는가하는 순서도 중요한 정보로 고려하는 메트릭
- RR: 실제 추천 리스트가 있을때 추천 아이템이 처음에 등장하는 것이 몇 번째에 있나하는 부분임 → ex) RR@3인데 0|1|0의 관련도라고 하면 → RR = 0/1 + 1/2 = 1/2임 → 이미 2번째에서 관련있는 아이템이 나왔으므로 3번째는 고려하지 않음
- 분모: K번째 아이템 & 분자: 실제로 관련있는 문서면 1 아니면 0 → 첫번째 위치가 1/2가 RR임
- RR이 유용하게 쓰이는 것은 관련있는 것을 하나만 찾고 싶은 경우로 첫번째 결과만 보고 이후 결과는 보지 않는 맛집 리스트 같은 케이스에 사용됨
- MRR: 여러 사람의 RR을 평균

▼ 3) Baselines

(1) POP

- the most popular items of the training set.

(2) S-POP

- most popular items of the **current session.**
- The recommendation **list changes** during the session as items gain more events.

(3) Item-KNN

- **Items similar to the actual item**
  similarity is defined as the **cosine similarity** between the vector of their sessions,
- Regularization is also included to avoid coincidental high similarities of **rarely visited items.**
- This baseline is one of the **most common item-to-item solutions** in practical systems,
  that provides recommendations in the “others who viewed this item also viewed these ones” setting.

(4) BPR-MF

- **matrix factorization** methods.
- It optimizes for a **pairwise ranking objective function** via SGD.
- Matrix factorization cannot be applied directly to session-based recommendations,
  Because the new sessions do not have feature vectors pre computed.
- can overcome this by **using the average of item feature vectors**
  of the items that had occurred in the session so far as the user feature vector.
- **average the similarities of the feature vectors**
  between a recommendable item and the items of the session so far.

→ Baselines 중에서는 item-KNN이 가장 우수한 성능

Table 1: Recall@20 and MRR@20 using the baseline methods

Baseline	RSC15		VIDEO	
	Recall@20	MRR@20	Recall@20	MRR@20
POP	0.0050	0.0012	0.0499	0.0117
S-POP	0.2672	0.1775	0.1301	0.0863
Item-KNN	0.5065	0.2048	0.5508	0.3381
BPR-MF	0.2574	0.0618	0.0692	0.0374

▼ 4) Results

- point wise인 cross-entropy 보다는 pair wise인 TOP1과 BPR이 성능 better
- TOP1이 2개 dataset에서 slightly better → baseline보다 20-30% gain

Table 3: Recall@20 and MRR@20 for different types of a single layer of GRU, compared to the best baseline (item-KNN). Best results per dataset are highlighted.

Loss / #Units	RSC15		VIDEO	
	Recall@20	MRR@20	Recall@20	MRR@20
TOP1 100	0.5853 (+15.55%)	0.2305 (+12.58%)	0.6141 (+11.50%)	0.3511 (+3.84%)
BPR 100	0.6069 (+19.82%)	0.2407 (+17.54%)	0.5999 (+8.92%)	0.3260 (-3.56%)
Cross-entropy 100	0.6074 (+19.91%)	0.2430 (+18.65%)	0.6372 (+15.69%)	0.3720 (+10.04%)
TOP1 1000	0.6206 (+22.53%)	**0.2693 (+31.49%)**	**0.6624 (+20.27%)**	**0.3891 (+15.08%)**
BPR 1000	**0.6322 (+24.82%)**	0.2467 (+20.47%)	0.6311 (+14.58%)	0.3136 (-7.23%)
Cross-entropy 1000	0.5777 (+14.06%)	0.2153 (+5.16%)	–	–

▼ References

https://arxiv.org/pdf/1511.06939.pdf

논문 리뷰 관련 준비 사항

Yoochoose(Recsys 15)

GRU4REC-CODE 해석

[이론] RNN/LSTM/GRU

Session-based Recommendation

Next To Do