

SASRec: Self-Attentive Sequential Recommendation

Date

2022. 12. 13


Presenter

이지현

Self-Attentive Sequential Recommendation

Wang-Cheng Kang, Julian McAuley
UC San Diego
{wckang,jmcauley}@ucsd.edu

<https://arxiv.org/pdf/1808.09781.pdf>
<https://github.com/kang205/SASRec> (tensorflow)
<https://github.com/pmixer/SASRec.pytorch> (pytorch)

Comments: Accepted by ICDM'18 as a long paper
Subjects: **Information Retrieval (cs.IR)**; Machine Learning (cs.LG)
Cite as: [arXiv:1808.09781](https://arxiv.org/abs/1808.09781) [**cs.IR**]
(or [arXiv:1808.09781v1](https://arxiv.org/abs/1808.09781v1) [**cs.IR**] for this version)
<https://doi.org/10.48550/arXiv.1808.09781> 

<https://arxiv.org> › cs ▾

[\[1808.09781\]](https://arxiv.org/abs/1808.09781) Self-Attentive Sequential Recommendation - arXiv

WC Kang 저술 · 2018 · 1011회 인용 — At each time step, **SASRec** seeks to identify which items are 'relevant' from a user's action history, and use them to predict the next item.

이 페이지를 4번 방문했습니다. 최근 방문 날짜: 22. 12. 3

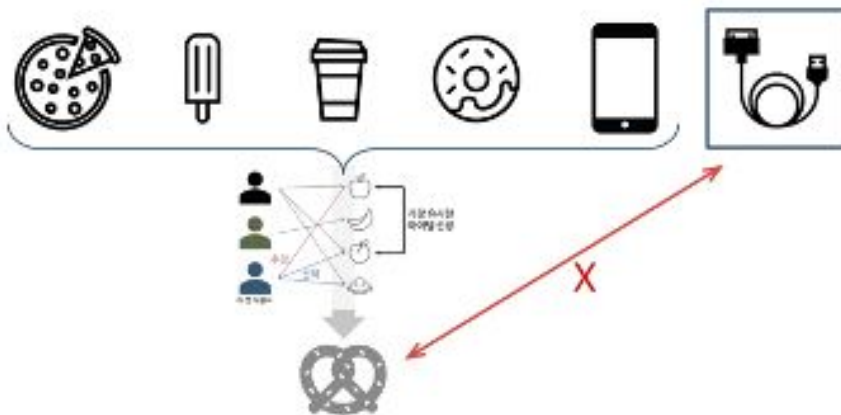
Contents

1. Introduction
2. Related Work
 - a. Transformer
3. Methodology
 - a. 구성 요소 1: Embedding layer
 - b. 구성 요소 2: Self-Attention Block
 - c. 구성 요소 3: Prediction Layer
4. Experiments
5. Conclusion

Introduction

Sequential Recommender System

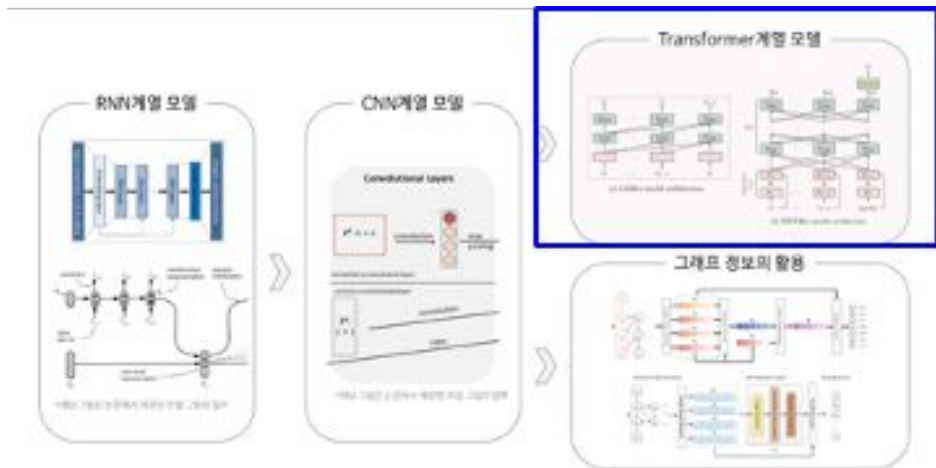
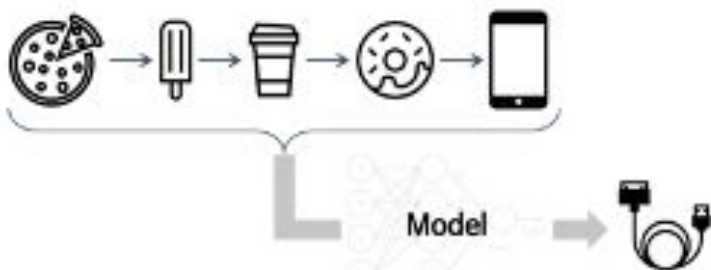
- 사용자의 과거 아이템 선택의 정보가 동일하게 중요하다는 기본 가정에서 출발한 추천 시스템 알고리즘
- 실제로 사용자가 선택을 할 때 과거 구매 정보가 동일하게 중요할까?



- 주로 선택하던 아이템과 다른 아이템이 등장할 경우 이에 대한 충분한 설명이 부족
- 협력 필터링 등 보편적으로 사용되는 추천 시스템 모델에서는 사용자가 주로 선택하던 아이템을 추천하게 됨

Sequential Recommender System

- 사용자의 선호도, 관심은 끊임없이 변화하고 발전한다는 아이디어에서 시작
- 조금 더 실제 사용자의 관심사를 잘 반영하는, 변화의 패턴까지도 잡아낼 수 있는 모델을 만들어보자
- 과거의 행동에서 유의미한 순차적인 패턴을 찾고, 최근 아이템에 더 집중하여 이를 기반으로 가장 선호할 다음 아이템 추천
- 다양한 딥러닝 모델들을 활용한 추천시스템 방법론들이 제안됨
 - RNN 계열 모델
 - CNN 계열 모델
 - Transformer 계열 모델
 - 그래프 정보의 활용



Sequential Recommender System

- 이전의 Sequential 을 고려하기 위한 방법으로 Markov Chains (MC)와 Recurrent Neural Networks (RNN) 이 있음
 - MC
 - 유저의 다음 행동이 바로 이전의 과거 혹은 몇 개 이전의 행동에 영향을 받을 것이라는 가정
 - Sparsity 가 높은 환경에서 예측 잘함
 - 그러나, 가정이 너무 over-simplify 하고, 복잡하거나 dense 한 환경에서 성능이 떨어짐

상태란 시스템에서 가능한 특정 상황을 말한다. 예를 들어 비 오는 날을 공부하고 있다면 다음과 같은 두 가지 상태가 있다.

- 오늘 비가 오고 있다.
- 오늘 비가 오고 있지 않다.



오늘 비가 내리는 경우 (R), 내일 비가 내릴 확률은 40%, 비가 내리지 않을 확률은 60%이다. 오늘 비가 오지 않으면 (N), 내일 비가 내릴 확률은 20%, 비가 내리지 않을 확률은 80%이다.

Transition Matrix

Markov Chain이 k개의 상태로 구성되어있는 경우, 전이 행렬은 각 상태에서 다른 상태로 이동할 확률을 기록하는 k*k 행렬이다. 행렬의 행은 현재 상태에 해당하고 열은 다음 상태에 해당한다. 예를 들어, 행 1과 2의 항목은 상태 1에서 2로 이동할 확률을 기록한다.

위의 예시를 이용해 전환 행렬을 만들어 보자. R이 항상 N보다 먼저 온다고 가정했을 때 첫 번째 행과 첫 번째 열은 R을 의미하고 두 번째 행과 두 번째 열은 N을 의미한다. 행은 from을 의미하고 열은 to를 의미한다.

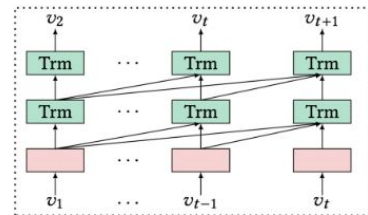
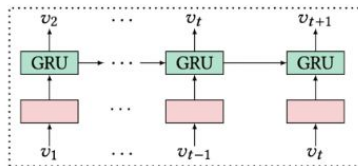
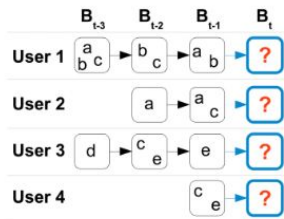
R에서 R로의 전환은 0.4이므로 행렬의 왼쪽 위에 0.4를 넣는다. R에서 N으로의 전환은 0.6이다. N에서 R은 0.2, N에서 N은 0.8이다.

	R	N
R	0.4	0.6
N	0.2	0.8

$$P = \begin{pmatrix} 0.4 & 0.6 \\ 0.2 & 0.8 \end{pmatrix}$$

Sequential Recommender System

- 이전의 Sequential 을 고려하기 위한 방법으로 Markov Chains (MC)와 Recurrent Neural Networks (RNN) 이 있음
 - MC
 - 유저의 다음 행동이 바로 이전의 과거 혹은 몇 개 이전의 행동에 영향을 받을 것이라는 가정
 - Sparsity 가 높은 환경에서 예측 잘함
 - 그러나, 가정이 너무 over-simplify 하고, 복잡하거나 **dense 한 환경에서 성능이 떨어짐**
 - RNN
 - 과거의 모든 행동을 모델의 입력값으로 넣어, 이를 요약한 정보를 가지고 다음 행동 예측
 - Hidden state 를 활용하여 이전의 모든 행동을 고려해 다음 행동 예측
 - 좋은 성능을 나타내기 위해서 **dense** 가 높은 많은 양의 데이터 요구, 따라서 **sparse** 데이터에서 **좋은 성능이 어려움**
- 본 연구에서는 이 둘의 단점을 보완하고자 새로운 모델을 추천 시스템에 적용
 - Transformer 의 self-attention 메커니즘
 - 즉, self attention 을 사용하는 Transformer 로 MCs, RNN 모델의 한계 극복 (dense, sparse 데이터셋에서 좋은



Related Work

Transformer

- 2017년, Vaswari, et al. 의 "Attention is all you need"에서 발표된 Transformer
- Transformer 특징
 - 인코더-디코더 구조
 - 인코더 : Unmasked attention
 - 디코더 : Masked attention
 - **Self-attention : (Query, Key, Value)** 를 입력으로 받아 **Scaled dot-product** 와 **Multi-head attention** 을 통해 입력들 간 유사도 계산 (즉, 각 단어가 어디에 더 많은 **weight** 를 두는지)
 - Multi-head attention

Transformer 아키텍처

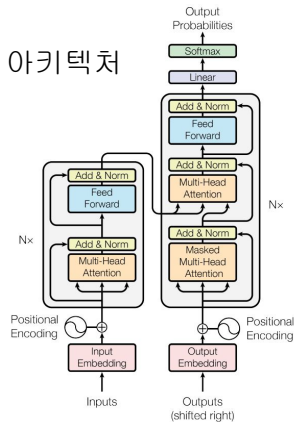
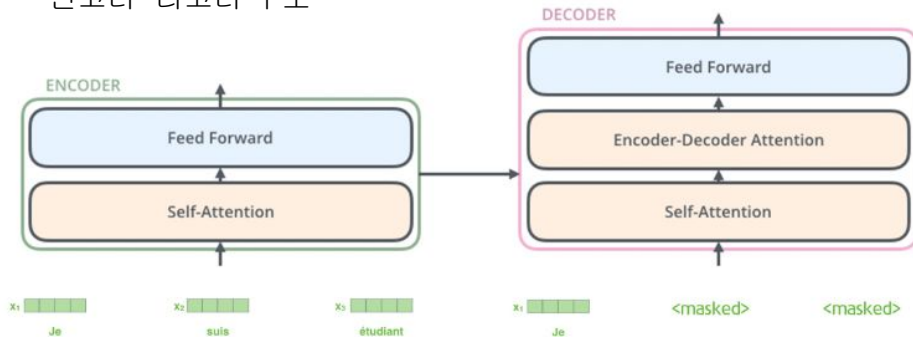


Figure 1: The Transformer - model architecture.

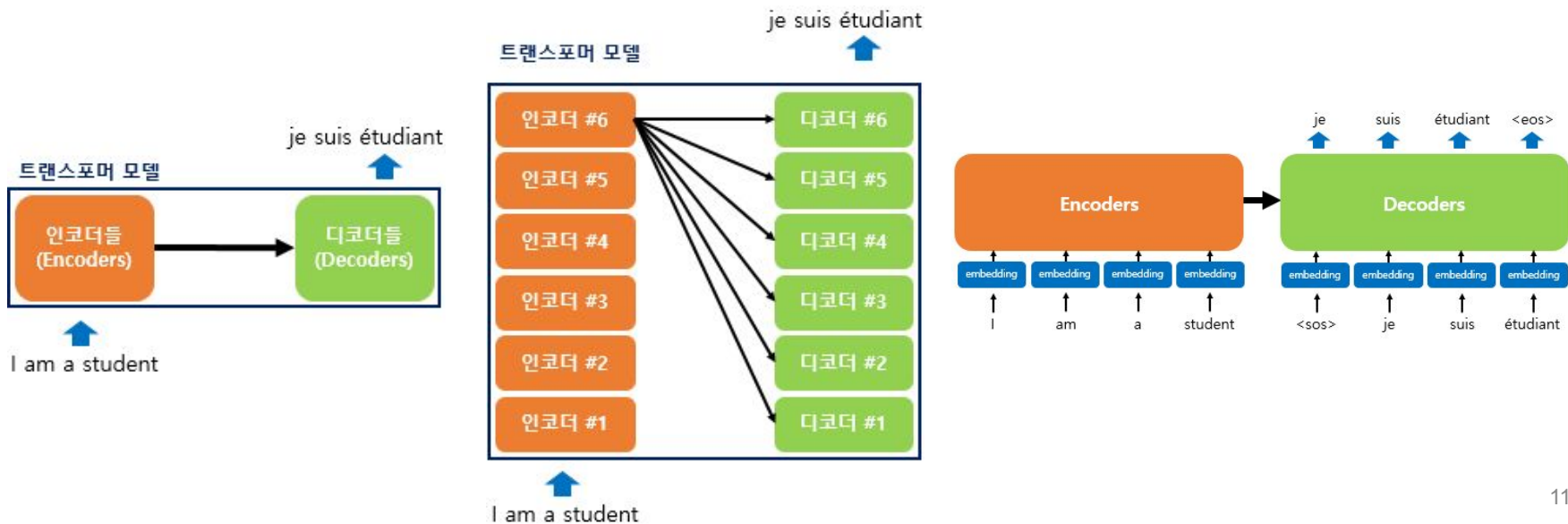
인코더-디코더 구조



Transformer

인코더 - 디코더 (6 layers)

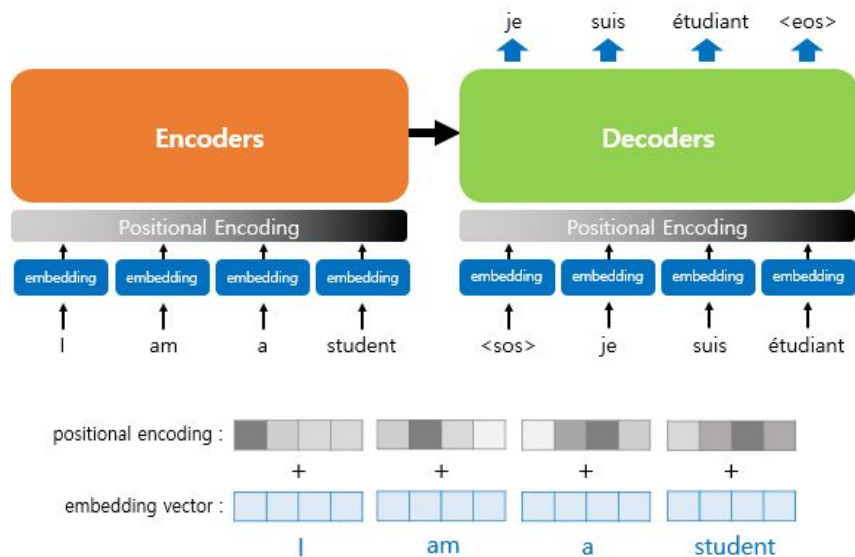
- 인코더에서 입력 시퀀스를 입력받고, 디코더에서 출력 시퀀스 출력
- 즉, 인코더로부터 정보를 전달받아 디코더가 출력 결과를 만들어 냄
 - 디코더는 seq2seq 구조처럼 시작 심볼 <sos> 와 <eos>가 나올 때까지 연산



Transformer

인코더 입력

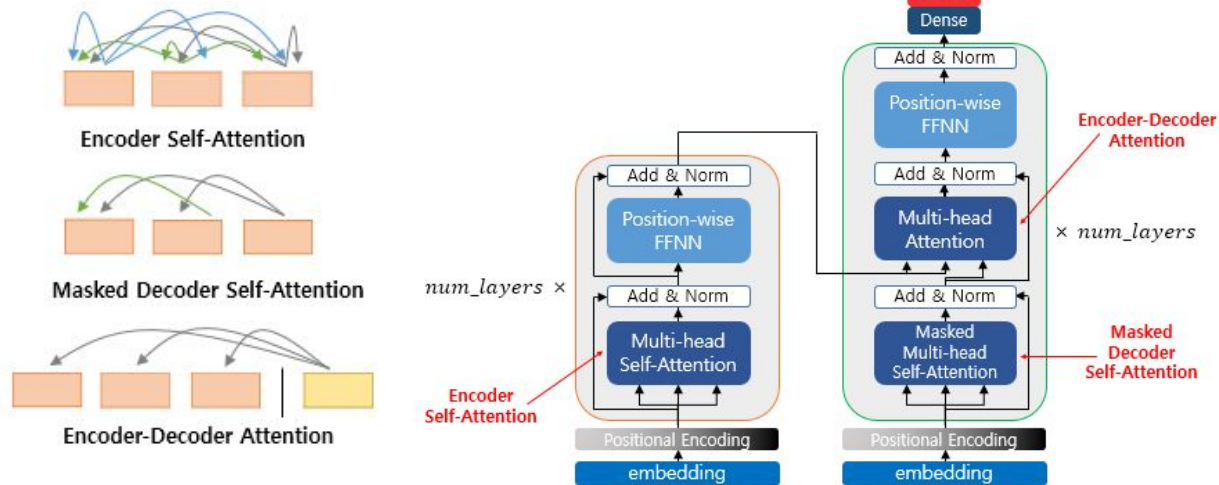
- Positional Encoding
 - 트랜스포머는 단어 입력을 순차적으로 받는 방식이 아니므로, 단어의 위치 정보를 positional encoding 을 통해서 받음



Transformer

Attention

- 트랜스포머에서 사용되는 세 가지 attention
 - Encoder **self-attention** : **encoder**
 - Masked decoder **self-attention** : **decoder**
 - Encoder-Decoder **attention** : **decoder**



Self-attention vs attention

Self-attention

- Q, K, V의 출처가 같을 때

Attention

- Q, K, V의 출처가 다를 때

인코더의 셀프 어텐션 : Query = Key = Value
 디코더의 마스크 셀프 어텐션 : Query = Key = Value
 디코더의 인코더-디코더 어텐션 : Query : 디코더 벡터 / Key = Value : 인코더 벡터

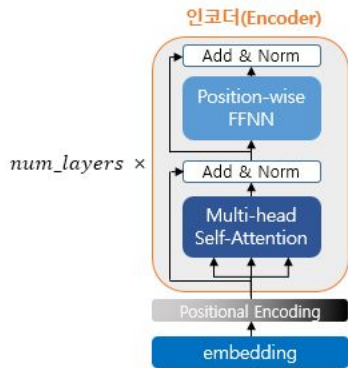
Transformer

Attention

- 트랜스포머에서 사용되는 세 가지 attention
 - Encoder **self-attention**: **encoder**

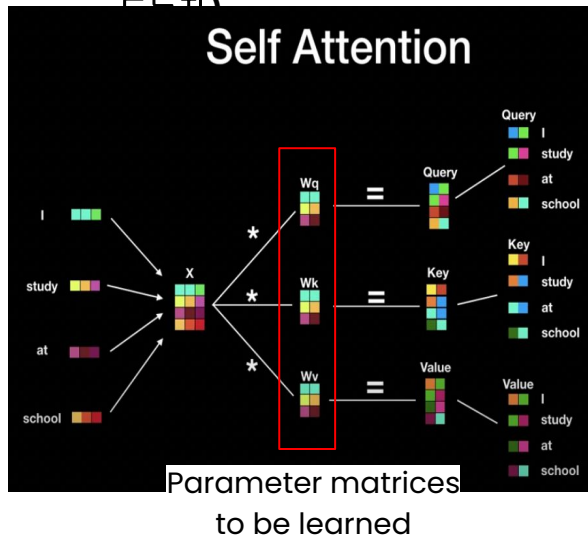
Encoder

- 하나의 encoder 는 2개의 sublayer 로 구성
 - 셀프 어텐션
 - Position-wise FFNN



Self-attention

- (Query, Key, Value) 를 입력으로 받아 Scaled dot-product 와 Multi-head attention 을 통해 입력들 간 유사도 계산 (즉, 각 단어가 어디에 더 많은 **weight** 를 두는가)



	X	Query	Key	Value
I				
study				
at				
school				

Transformer

Self-attention

- (Query, Key, Value) 를 입력으로 받아 Scaled dot-product 와 Multi-head attention 을 통해 입력들 간 유사도 계산 (즉, 각 단어가 어디에 더 많은 **weight** 를 두는지)
- **Query**: 현재 단어
- **Key**: 어떤 단어와의 상관관계를 볼 것이냐
 - **Softmax 결과값**: **key** 값에 해당하는 단어가 **query** 에 어느 정도 영향을 미치는지 나타냄
- **Value**: 각 **softmax** 결과값을 **value** 에 곱해줌, 연관성이 높으면 **value** 가 잘 살아있을 것이고, 낮으면 **value** 영향력이 미미해질 것임
- 최종 벡터: 단어 **I** 가 문장 속에서 지닌 전체적인 의미를 지닌 벡터

attention score

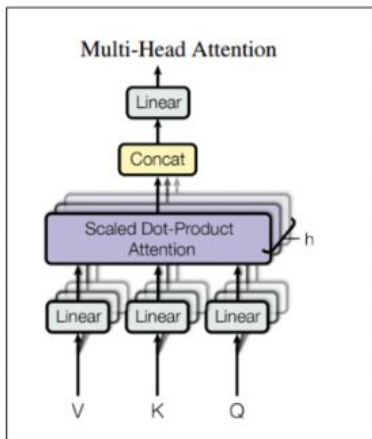
Query * Key ^T	Score	Softmax	Value	Softmax * Value	Σ Softmax * Value (Attention layer output)
I * I	130	0.92	I		}
I * study	50	0.05	study		
I * at	20	0.02	at		
I * school	10	0.01	school		

study	study * I	30	0.02		}
	study * study	110	0.70		
	study * at	20	0.03		
	study * school	70	0.25		

Transformer

Multi head attention

- Scaled dot product attention 을 여러개 수행 한 것
- Scaled dot product attention 을 concat 해서 하나로 합침



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

Methodology

SASRec

Transformer 가 **Sequential** 추천 시스템에 어떻게 적용되는가

SASRec 추천 모델은 다음과 같이 구성됨

- 구성 요소 1: Embedding layer
- 구성 요소 2: 여러 개의 self-attention blocks
- 구성 요소 3: Prediction layer

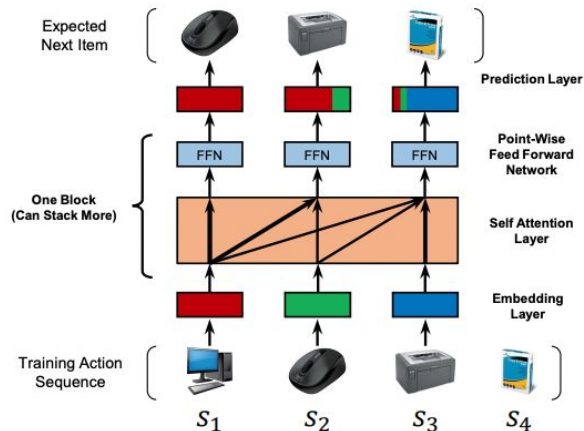


Figure 1: A simplified diagram showing the training process of SASRec. At each time step, the model considers all previous items, and uses attention to ‘focus on’ items relevant to the next action.

구성 요소 1: Embedding layer

- 학습 순서를 고정된 길이의 시퀀스로 변환
- N (하이퍼 파라미터): 모델이 다룰 수 있는 최대 길이
- 시퀀스의 길이가 n 보다 크면 잘라내어 최근 n 개만큼만 모델이 고려

시퀀스의 길이가 n 보다 작으면 길이가 n 이 될 때까지 왼쪽으로
 $(S_1^u, S_2^u, \dots, S_{|S^u|-1}^u) \longrightarrow s = (s_1, s_2, \dots, s_n)$

최종 input embedding

$$\hat{\mathbf{E}} = \begin{bmatrix} \mathbf{M}_{s_1} + \mathbf{P}_1 \\ \mathbf{M}_{s_2} + \mathbf{P}_2 \\ \dots \\ \mathbf{M}_{s_n} + \mathbf{P}_n \end{bmatrix}$$

SASRec

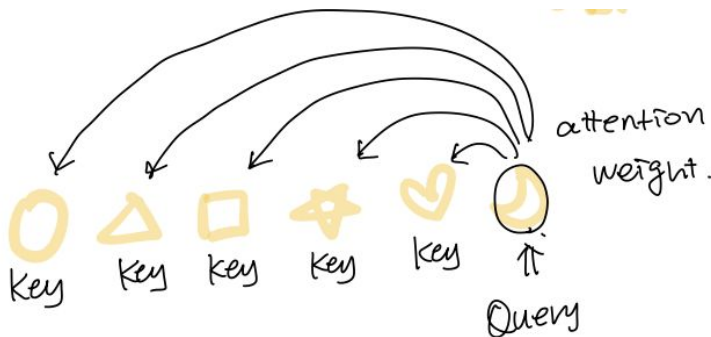
구성 요소 2 : Self-Attention Block

- 앞서 설명한 transformer 구조와 거의 유사
- Embedding layer 에서 생성한 input embedding 을 입력으로 받아 (query, key, value) 를 통해 attention layer 를 거침
- 이 때, Causality (인과관계) 측면에서 sequence 의 특성으로 $t+1$ 번째 항목을 예측할 때, t 이전 항목만 고려해야 함. 만약 t 이후 항목들이 포함되어 있다면, 해당 데이터들의 연결을 Masking 처리

The scaled dot-product attention [3] is defined as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right) \mathbf{V}, \quad (2)$$

$$\mathbf{S} = \text{SA}(\hat{\mathbf{E}}) = \text{Attention}(\hat{\mathbf{E}}\mathbf{W}^Q, \hat{\mathbf{E}}\mathbf{W}^K, \hat{\mathbf{E}}\mathbf{W}^V), \quad (3)$$



SASRec

구성 요소 2 : Self-Attention Block

- Point-Wise Feed-Forward Network
 - 모델의 비선형을 부여하고, 다른 latent 차원을 고려하기 위해 point-wise FFN 사용
 - 이후, self-attention blocks 를 여러 개 쌓아 복잡한 item transitions 을 학습하도록 함

$$\mathbf{F}_i = \text{FFN}(\mathbf{S}_i) = \text{ReLU}(\mathbf{S}_i \mathbf{W}^{(1)} + \mathbf{b}^{(1)}) \mathbf{W}^{(2)} + \mathbf{b}^{(2)}, \quad (4)$$

where $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}$ are $d \times d$ matrices and $\mathbf{b}^{(1)}, \mathbf{b}^{(2)}$ are d -dimensional vectors. Note that there is no interaction between \mathbf{S}_i and \mathbf{S}_j ($i \neq j$), meaning that we still prevent information leaks (from back to front).

$$\begin{aligned} \mathbf{S}^{(b)} &= \text{SA}(\mathbf{F}^{(b-1)}), \\ \mathbf{F}_i^{(b)} &= \text{FFN}(\mathbf{S}_i^{(b)}), \quad \forall i \in \{1, 2, \dots, n\}, \end{aligned}$$

- 네트워크가 깊어질수록 1) 모델의 capacity 가 증가하여 overfitting 발생, 2) vanishing gradients 로 학습 불안정, 3) 많은 파라미터로 많은 학습 시간 요구
- 이에 따라 Layer normalization, Dropout, Residual Connections 적용 (사실상 transformer 와 거의 동일)

$$g(x) = x + \text{Dropout}(g(\text{LayerNorm}(x))),$$

SASRec

구성 요소 3 : Prediction Layer

- 앞선 **self-attention blocks** 를 통과한 후 다음에 오는 아이템 예측
- 다음에 올 유저 행동을 예측하기 위해 적절한 **item** 의 **score** 를 최종적으로 예측

$$r_{i,t} = \mathbf{F}_t^{(b)} \mathbf{N}_i^T,$$

- $r_{\{i,t\}}$ 는 t 아이템 이후에 next item 의 타당성, 이 값이 높으면 high relevance 함
- 아이템의 관련 점수를 계산하기 위하여, **아이템 임베딩 매트릭스 N**을 사용하는데, 모델의 사이즈와 오버피팅을 예방하고자 Shared Item Embedding 매트릭스 M을 사용함
- 이 때, Shared Item embedding 을 사용할 경우 sequential 을 반영할 때 비대칭적인 item transitions 을 반영해야 하기 때문에 서로 다른 item embedding 을 사용해야 한다고 생각할 수 있으나, 제안한 모델은 FFN 을 통해 비선형 변환을 학습하기 때문에 문제가 되지 않음. 실제로 Shared Item Embedding 을 사용했을 때 모델 성능 향상 됨.
 - 예를 들어, 노트북을 사면 -> 마우스를 사지만, 마우스를 산다고 노트북을 사는 것은 아님. (비대칭적인 item transitions)

However, we can also insert an explicit user embedding at the last layer, for example via addition: $r_{u,i,t} = (\mathbf{U}_u + \mathbf{F}_t^{(b)})\mathbf{M}_i^T$ where \mathbf{U} is a user embedding matrix. However, we empirically find that adding an explicit user embedding doesn't improve performance (presumably because the model already considers all of the user's actions).

SASRec

Network Training

- 사용자가 관심있는 항목 (positive sampling) 과 관심 없는 항목 (negative sampling) 을 통해 binary cross entropy 계산
- Adam 옵티마이저

$$- \sum_{S^u \in \mathcal{S}} \sum_{t \in [1, 2, \dots, n]} \left[\log(\sigma(r_{o_t, t})) + \sum_{j \notin S^u} \log(1 - \sigma(r_{j, t})) \right]$$

positive samples negative samples

Experiments

Experiment

Dataset

- Amazon, Steam, MovieLens

Table II: Dataset statistics (after preprocessing)

Dataset	#users	#items	avg. actions /user	avg. actions /item	#actions
<i>Amazon Beauty</i>	52,024	57,289	7.6	6.9	0.4M
<i>Amazon Games</i>	31,013	23,715	9.3	12.1	0.3M
<i>Steam</i>	334,730	13,047	11.0	282.5	3.7M
<i>MovieLens-1M</i>	6,040	3,416	163.5	289.1	1.0M

- Split

- **Test**: 가장 최근에 구매한 아이템
- **Val**: 그 이전에 구매한 아이템
- **Train**: 나머지 모든 데이터

Evaluation Metrics

- Hit@10

- 전체 유저 수 대비 적중한 유저 수
- 이 때, 예측이 적중했는가를 판단하기 위해 @K 사용
- 모델이 유저 별로 K개의 아이템을 예측하고, 그 중에 정답이 있으면 적중했다고 봄

- NDCG@10

- 유저의 선호도에 따라 아이템 순서별로 가중치 값 적용
- 가장 이상적인 추천 조합 대비, 현재 모델 추천이 얼마나 좋은지

- **Amazon**: A series of datasets introduced in [46], comprising large corpora of product reviews crawled from *Amazon.com*. Top-level product categories on *Amazon* are treated as separate datasets. We consider two categories, 'Beauty,' and 'Games.' This dataset is notable for its high sparsity and variability.
- **Steam**: We introduce a new dataset crawled from *Steam*, a large online video game distribution platform. The dataset contains 2,567,538 users, 15,474 games and 7,793,069 English reviews spanning October 2010 to January 2018. The dataset also includes rich information that might be useful in future work, like users' play hours, pricing information, media score, category, developer (etc.).
- **MovieLens**: A widely used benchmark dataset for evaluating collaborative filtering algorithms. We use the version (MovieLens-1M) that includes 1 million user ratings.

Experiment

Hit@10

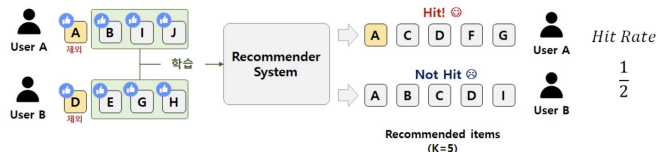
- 전체 유저 수 대비 적중한 유저 수
- 이 때, 예측이 적중했는가를 판단하기 위해 @K 사용
- 모델이 유저 별로 K개의 아이템을 예측하고, 그 중에 정답이 있으면 적중했다고 봄

Hit Rate는 전체 사용자 수 대비 적중한 사용자 수를 의미한다(적중률). Hit Rate는 아래 4가지 단계로 구할 수 있다. [그림11]의 예시를 보면 쉽게 이해할 수 있을 것이다.

- 1) 사용자가 선호한 아이템 중 1개를 제외한다.
- 2) 나머지 아이템들로 추천 시스템을 학습한다.
- 3) 사용자별로 K개의 아이템을 추천하고, 앞서 제외한 아이템이 포함되면 Hit이다.
- 4) 전체 사용자 수 대비 Hit한 사용자 수 비율을 구하면 Hit Rate가 된다.

$$\text{Hit Rate} = \frac{\# \text{ of Hit User}}{\# \text{ of User}}$$

[그림11] Hit Rate

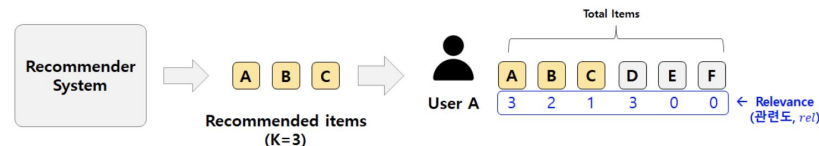


[그림11] Hit Rate 예시

NDCG@10

- 유저의 선호도에 따라 아이템 순서별로 가중치 값 적용
- 가장 이상적인 추천 조합 대비, 현재 모델 추천이 얼마나

NDCG는 원래 검색 분야에서 등장한 지표이나 추천 시스템에도 많이 사용되고 있다. 위의 두 평가 지표와 마찬가지로 Top K개 아이템을 추천하는 경우, 추천 순서에 가중치를 두어 평가한다. **NDCG@K 값은 1에 가까울수록 좋다.** MAP는 사용자가 선호한 아이템이 추천 리스트 중 어떤 순서에 포함되었는지 여부에 대해서 1 or 0으로만 구분하지만, **NDCG@K는 순서별로 가중치 값(관련도, relevance)을 다르게 적용하여 계산한다.** [그림4]의 예시를 통해 하나씩 살펴보자.



$$CG_3 = \sum_{i=1}^K rel_i = rel_1 + rel_2 + rel_3 = 3 + 2 + 1 = 6$$

$$DCG_3 = \sum_{i=1}^K \frac{rel_i}{\log_2(i+1)} = \frac{3}{\log_2(1+1)} + \frac{2}{\log_2(2+1)} + \frac{1}{\log_2(3+1)} = \frac{3}{1} + \frac{2}{1.58} + \frac{1}{2} = 4.78$$

$$IDCG_3 = \sum_{i=1}^K \frac{rel_i^{opt}}{\log_2(i+1)} = \frac{3}{\log_2(1+1)} + \frac{3}{\log_2(2+1)} + \frac{2}{\log_2(3+1)} = \frac{3}{1} + \frac{3}{1.58} + \frac{2}{2} = 5.89$$

$$NDCG_3 = \frac{DCG}{IDCG} = \frac{4.78}{5.89} = 0.81$$

[그림5] NDCG@K 예시

Experiment

4가지 Questions

- Does SASRec outperform SoTA models included CNN/RNN based methods?

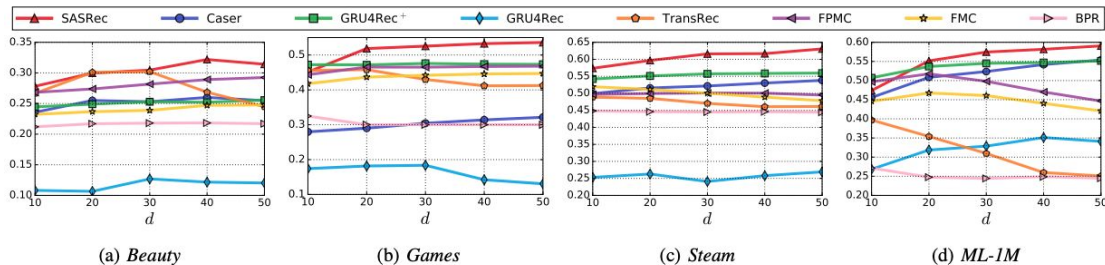


Figure 2: Effect of the latent dimensionality d on ranking performance (NDCG@10).

Table III: Recommendation performance. The best performing method in each row is boldfaced, and the second best method in each row is underlined. Improvements over non-neural and neural approaches are shown in the last two columns respectively.

Dataset	Metric	(a) PopRec	(b) BPR	(c) FMC	(d) FPMC	(e) TransRec	(f) GRU4Rec	(g) GRU4Rec ⁺	(h) Caser	(i) SASRec	Improvement vs. (a)-(e) (f)-(h)	
<i>Beauty</i>	Hit@10	0.4003	0.3775	0.3771	0.4310	<u>0.4607</u>	0.2125	0.3949	0.4264	0.4854	5.4%	13.8%
	NDCG@10	0.2277	0.2183	0.2477	0.2891	<u>0.3020</u>	0.1203	0.2556	0.2547	0.3219	6.6%	25.9%
<i>Games</i>	Hit@10	0.4724	0.4853	0.6358	0.6802	<u>0.6838</u>	0.2938	0.6599	0.5282	0.7410	8.5%	12.3%
	NDCG@10	0.2779	0.2875	0.4456	0.4680	<u>0.4557</u>	0.1837	<u>0.4759</u>	0.3214	0.5360	14.5%	12.6%
<i>Steam</i>	Hit@10	0.7172	0.7061	0.7731	0.7710	0.7624	0.4190	<u>0.8018</u>	0.7874	0.8729	13.2%	8.9%
	NDCG@10	0.4535	0.4436	0.5193	0.5011	0.4852	0.2691	<u>0.5595</u>	0.5381	0.6306	21.4%	12.7%
<i>ML-1M</i>	Hit@10	0.4329	0.5781	0.6986	0.7599	0.6413	0.5581	0.7501	<u>0.7886</u>	0.8245	8.5%	4.6%
	NDCG@10	0.2377	0.3287	0.4676	0.5176	0.3969	0.3381	0.5513	<u>0.5538</u>	0.5905	14.1%	6.6%

Experiment

4가지 Questions

- What is the influence of various components in the SAS architecture?

Table IV: Ablation analysis (NDCG@10) on four datasets. Performance better than the default version is boldfaced. ‘↓’ indicates a severe performance drop (more than 10%).

Architecture	<i>Beauty</i>	<i>Games</i>	<i>Steam</i>	<i>ML-1M</i>
(0) Default	0.3142	0.5360	0.6306	0.5905
(1) Remove PE	0.3183	0.5301	0.6036	0.5772
(2) Unshared IE	0.2437↓	0.4266↓	0.4472↓	0.4557↓
(3) Remove RC	0.2591↓	0.4303↓	0.5693	0.5535
(4) Remove Dropout	0.2436↓	0.4375↓	0.5959	0.5801
(5) 0 Block ($b=0$)	0.2620↓	0.4745↓	0.5588↓	0.4830↓
(6) 1 Block ($b=1$)	0.3066	0.5408	0.6202	0.5653
(7) 3 Blocks ($b=3$)	0.3078	0.5312	0.6275	0.5931
(8) Multi-Head	0.3080	0.5311	0.6272	0.5885

- PE (Positional Embedding)
 - The model makes recommendations based on users' past actions, but their order doesn't matter
- IE (Item Embedding)
 - Using two item embeddings consistently impairs the performance, presumably due to overfitting
- RC (Residual Connections)
 - Performance is significantly worse
- Dropout
 - Dropout can effectively regularize the model to achieve better test performance, especially on sparse dataset.

Experiment

4가지 Questions

- What is the influence of various components in the SAS architecture?

Table IV: Ablation analysis (NDCG@10) on four datasets. Performance better than the default version is boldfaced. ‘↓’ indicates a severe performance drop (more than 10%).

Architecture	<i>Beauty</i>	<i>Games</i>	<i>Steam</i>	<i>ML-1M</i>
(0) Default	0.3142	0.5360	0.6306	0.5905
(1) Remove PE	0.3183	0.5301	0.6036	0.5772
(2) Unshared IE	0.2437↓	0.4266↓	0.4472↓	0.4557↓
(3) Remove RC	0.2591↓	0.4303↓	0.5693	0.5535
(4) Remove Dropout	0.2436↓	0.4375↓	0.5959	0.5801
(5) 0 Block ($b=0$)	0.2620↓	0.4745↓	0.5588↓	0.4830↓
(6) 1 Block ($b=1$)	0.3066	0.5408	0.6202	0.5653
(7) 3 Blocks ($b=3$)	0.3078	0.5312	0.6275	0.5931
(8) Multi-Head	0.3080	0.5311	0.6272	0.5885

- Number of blocks
 - Inferior with zero blocks, since the model would only depend on the last item. The variant with one block performs reasonably well, though using two blocks (the default model) still boosts performance especially on dense datasets.
- Multi-head
 - Performance with two heads is consistently and slightly worse than single-head attention in our case

Experiment

4가지 Questions

- What is the training efficiency and scalability (regarding n) of SASRec?

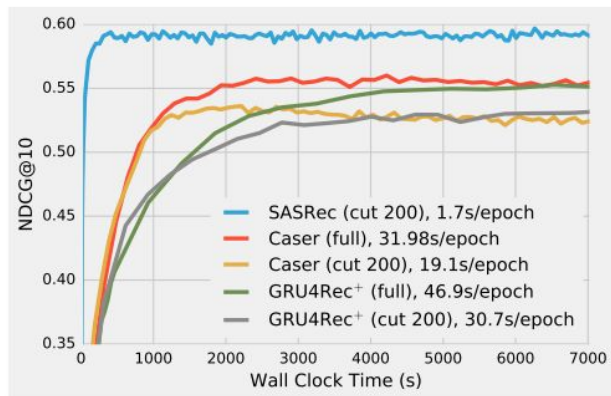


Figure 3: Training efficiency on *ML-IM*. SASRec is an order of magnitude faster than CNN/RNN-based recommendation methods in terms of training time per epoch and in total.

Table V: Scalability: performance and training time with different maximum length n on *ML-IM*.

n	10	50	100	200	300	400	500	600
Time(s)	75	101	157	341	613	965	1406	1895
NDCG@10	0.480	0.557	0.571	0.587	0.593	0.594	0.596	0.595

Experiment

4가지 Questions

- Are the attention weights able to learn meaningful patterns related to positions or item's attributes?

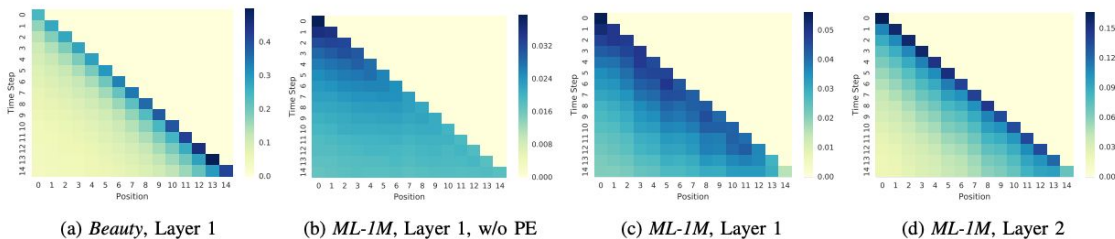


Figure 4: Visualizations of average attention weights on positions at different time steps. For comparison, the heatmap of a first-order Markov chain based model would be a diagonal matrix.

- Attention weights 시각화
 - 데이터셋, PE, Layer 에 따라 **adaptive** 하게 달라지는 **attention map**
 - (a), (c) : **sparse** 한 *Beauty* 데이터셋에서는 최근 2~3개의 아이템 정보를 기반으로 예측, 비교적 **dense** 한 *ML-IM* 데이터셋에서는 더 많은 정보 관측
 - (b), (c) : PE 효과. (b) 의 경우 **attention map** 이 **uniform** 함, 이는 순서에 대한 가중치가 없으면 학습이 안됨
 - (c), (d) : Layer 수에 따라 학습의 정도 차이. (c) 의 경우 **input sequence** 에 대해 전체 아이템들의 관계 학습하기 때문에, 넓은 **attention** 분포. (d) 의 경우, **layer1** 에서 이미 전체 정보를 받아왔기 때문에 **layer 2**에서는 최신 정보만을 주목

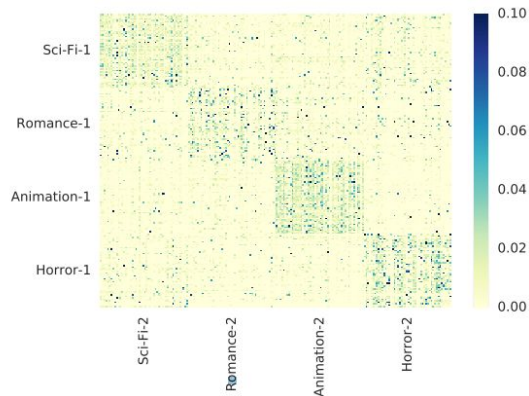


Figure 5: Visualization of average attention between movies from four categories. This shows our model can uncover items' attributes, and assigns larger weights between similar items.

- 아이템 요소들끼리 **attention**
- 비슷한 것들끼리 **attention weight** 가 강하게 들어감

Conclusion

Conclusion

- Self-attention based sequential model SASRec for next item recommendation
- SASRec models the entire user sequence, and adaptively considers consumed items for prediction
- Model outperforms SoTA baseline, and is an order of magnitude faster than CNN/RNN based approaches
- In the future, to extend the model by incorporating rich context information (e.g, dwell time, action types, locations, devices, etc) and to investigate approaches to handle very long sequences (e.g. clicks)

Thank You.