Soongsil University

Self-Attentive Sequential Recommendation

CheolHee Jung

DM Lab

05.30.2024

Kang, Wang-Cheng & McAuley, Julian

ICDM'2018

Outline

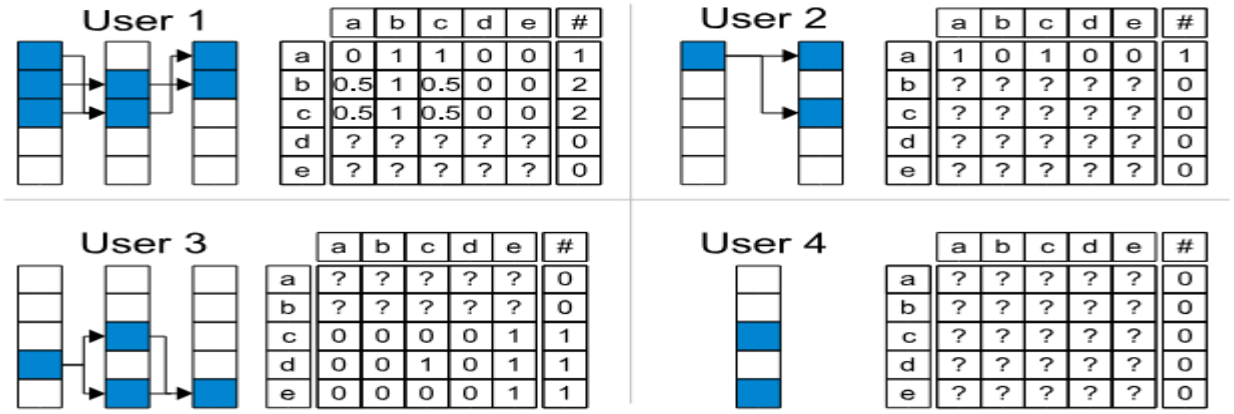
- Introduction
- Proposed Method
- Experiments
- Conclusion

Markov Chains(MCs)

❒ FPMC : 1차 Markov Chains사용

마코프 체인은 다음 행동은 이전 행동만을 조건으로 한다고 가정

희소한 데이터 셋에서 마지막으로 방문한 항목이 사용자의 다음 행동에 영향을 미치는 주요 요인인 경우가 많기 때문에 1차 MC기반 방법은 희소한 데이터셋 에서 강력한 성능을 보임



	a	b	c	d	e	#
a	0	1	1	0	0	1
b	0.5	1	0.5	0	0	2
c	0.5	1	0.5	0	0	2
d	?	?	?	?	?	0
e	?	?	?	?	?	0

	a	b	c	d	e	#
a	1	0	1	0	0	1
b	?	?	?	?	?	0
c	?	?	?	?	?	0
d	?	?	?	?	?	0
e	?	?	?	?	?	0

	a	b	c	d	e	#
a	?	?	?	?	?	0
b	?	?	?	?	?	0
c	0	0	0	0	1	1
d	0	0	1	0	1	1
e	0	0	0	0	1	1

	a	b	c	d	e	#
a	?	?	?	?	?	0
b	?	?	?	?	?	0
c	?	?	?	?	?	0
d	?	?	?	?	?	0
e	?	?	?	?	?	0

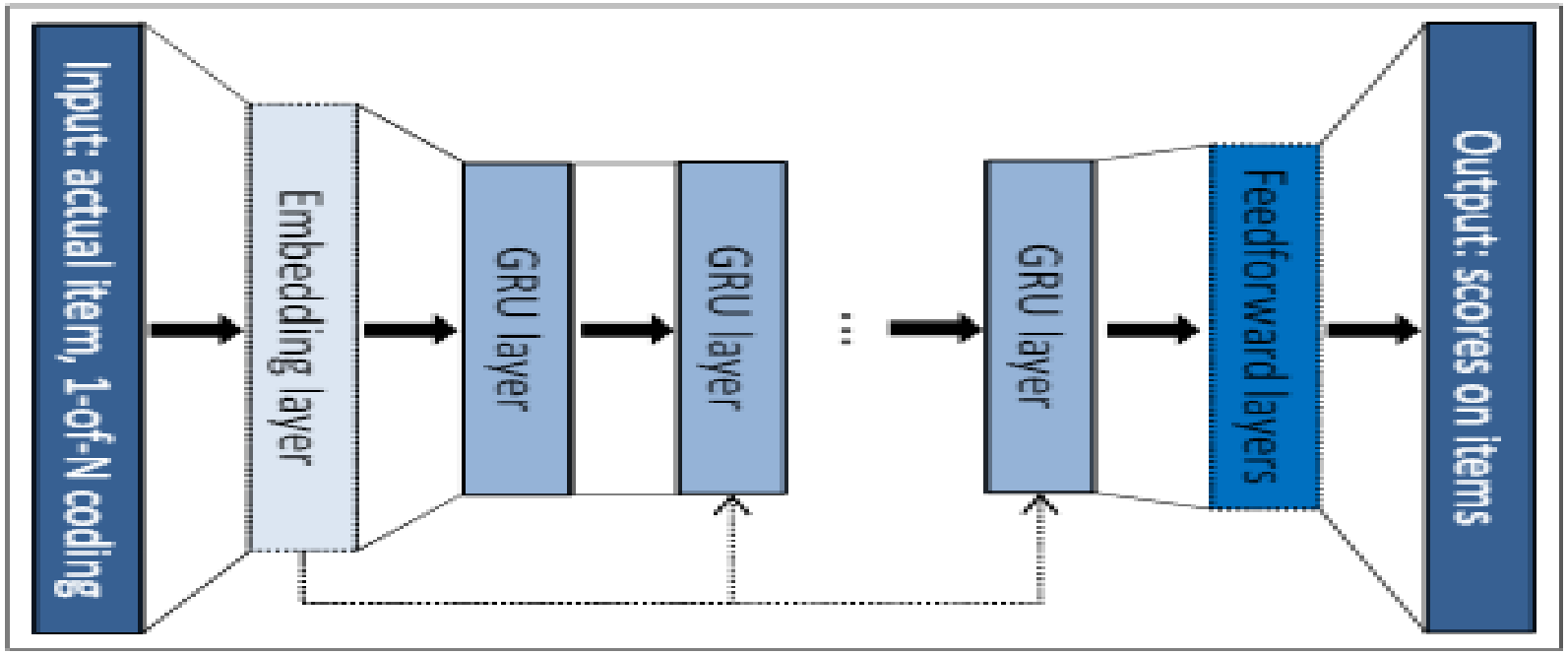
❒ MC 차수를 훈련 전에 지정해야 함
❒ 성능과 효율성이 차수에 따라 잘 확장되지 않음
❒ 복잡한 시나리오에서는 성능이 좋지 않음

"Factorizing Personalized Markov Chains for Next-Basket Recommendation"

Recurrent Neural Networks(RNNs)

❒ GRU4Rec

RNN의 경우 Hidden state를 활용하여 이전의 모든 행동을 고려해 다음 행동 예측



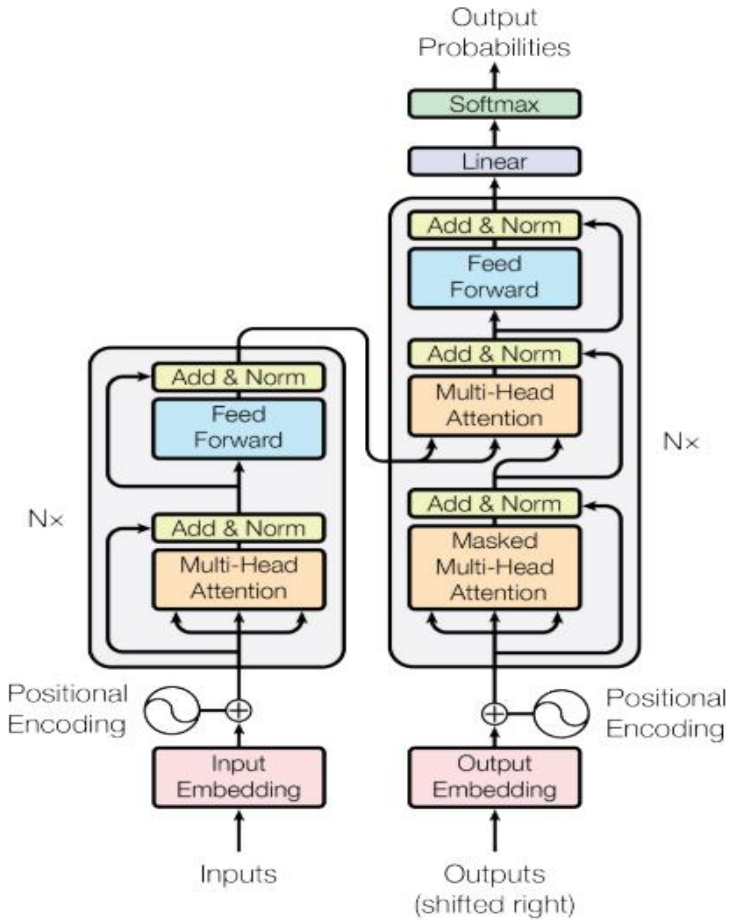
- ❒ 병렬 계산에서 비효율적
- ❒ 좋은 성능을 위해서는 밀도가 높은 많은 양의 데이터가 요구됨
- ❒ 입력 노드에서 출력 노드까지 거리가 길어질 수 있음
  - 장거리 종속성(Learning long – range dependencies)문제

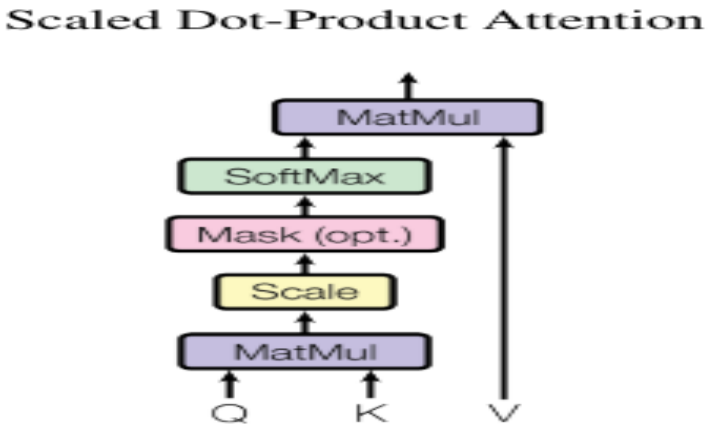
"SESSION-BASED RECOMMENDATIONS WITH RECURRENT NEURAL NETWORKS"

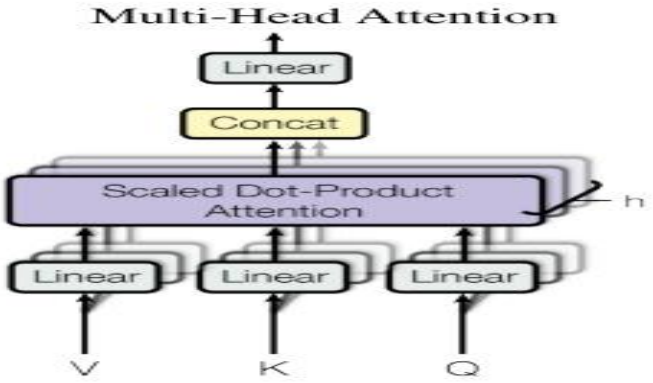
Self – attention

❒ Transformer

Query, Key, Value를 입력 받아 Scaled Dot – Product와 Multi – Head를 통해 유사도 계산







"Attention is All you Need"

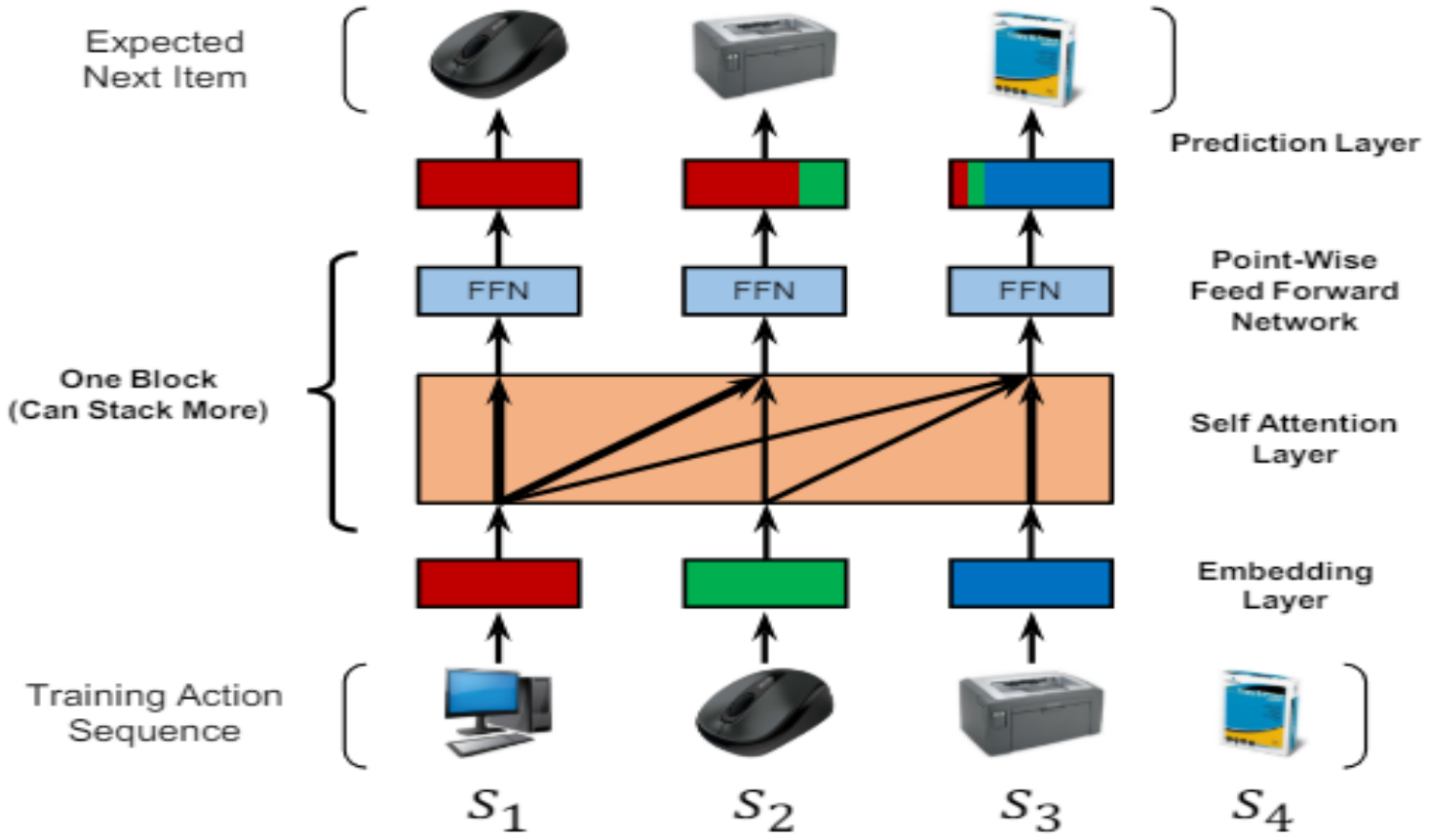
Outline

- Introduction
- Proposed Method
- Experiments
- Conclusion

SASRec

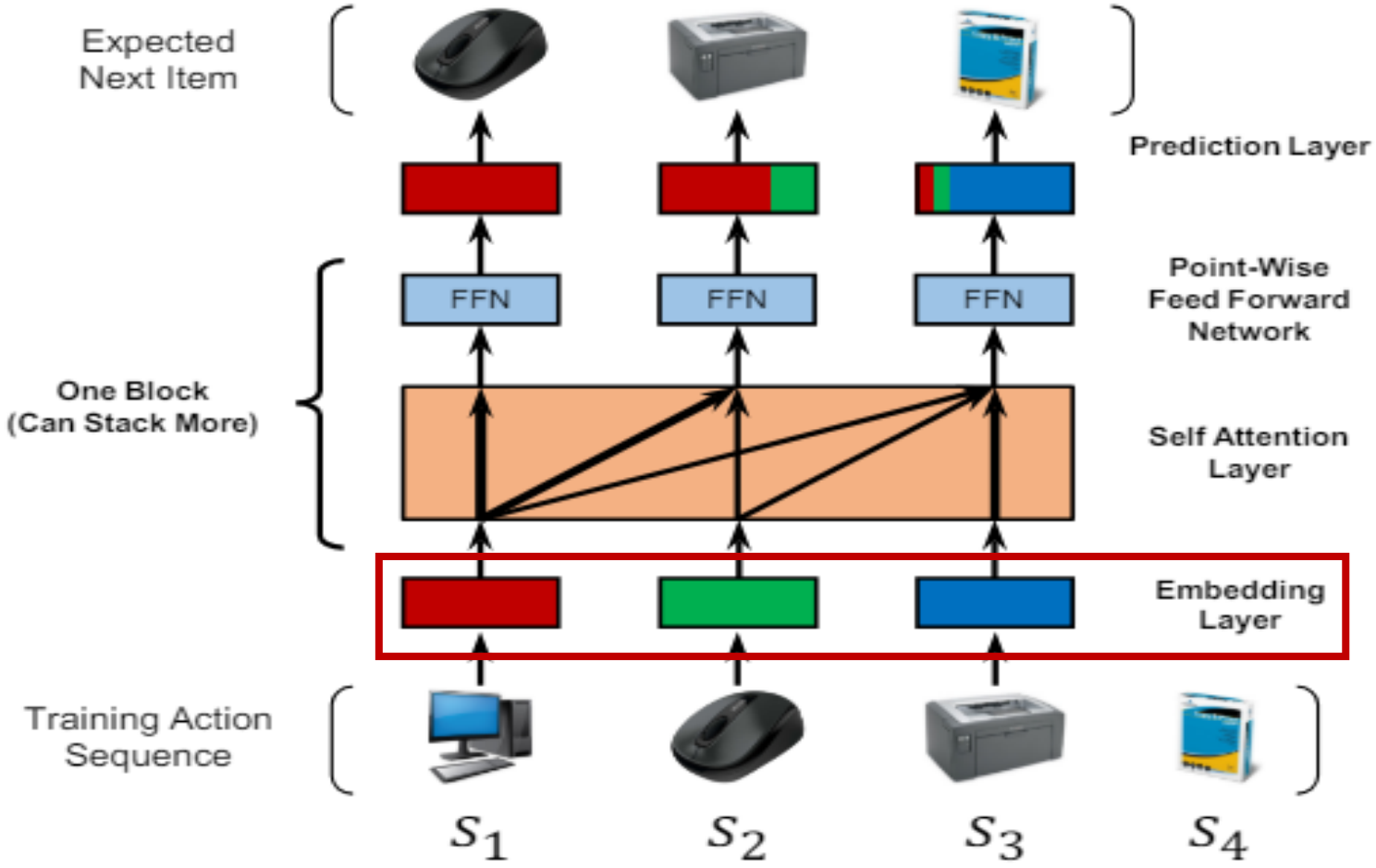
❒ METHODOLOGY

$S^u = (S_1^u, S_2^u, \ldots, S_{|S^u|}^u)$: 주어진 사용자의 행동 시퀀스
모델의 입력을$(S_1^u, S_2^u, \ldots, S_{|S^u|-1}^u)$로 받고, 예상 출력을 동일한 시퀀스의 '이동된' 버전$((S_1^u, S_2^u, \ldots, S_{|S^u|}^u))$으로 생각



SASRec

❒ Embedding Layer



SASRec

❒ Embedding Layer

학습 시퀀스$(S_1^u, S_2^u, \ldots, S_{|S^u|-1}^u)$를 n의 고정 길이 시퀀스로 변환
시퀀스 길이가 n보다 길면 가장 최근의 n개의 행동을 고려함
n보다 짧으면 n이 될 때까지 왼쪽에 padding항목을 반복적으로 추가

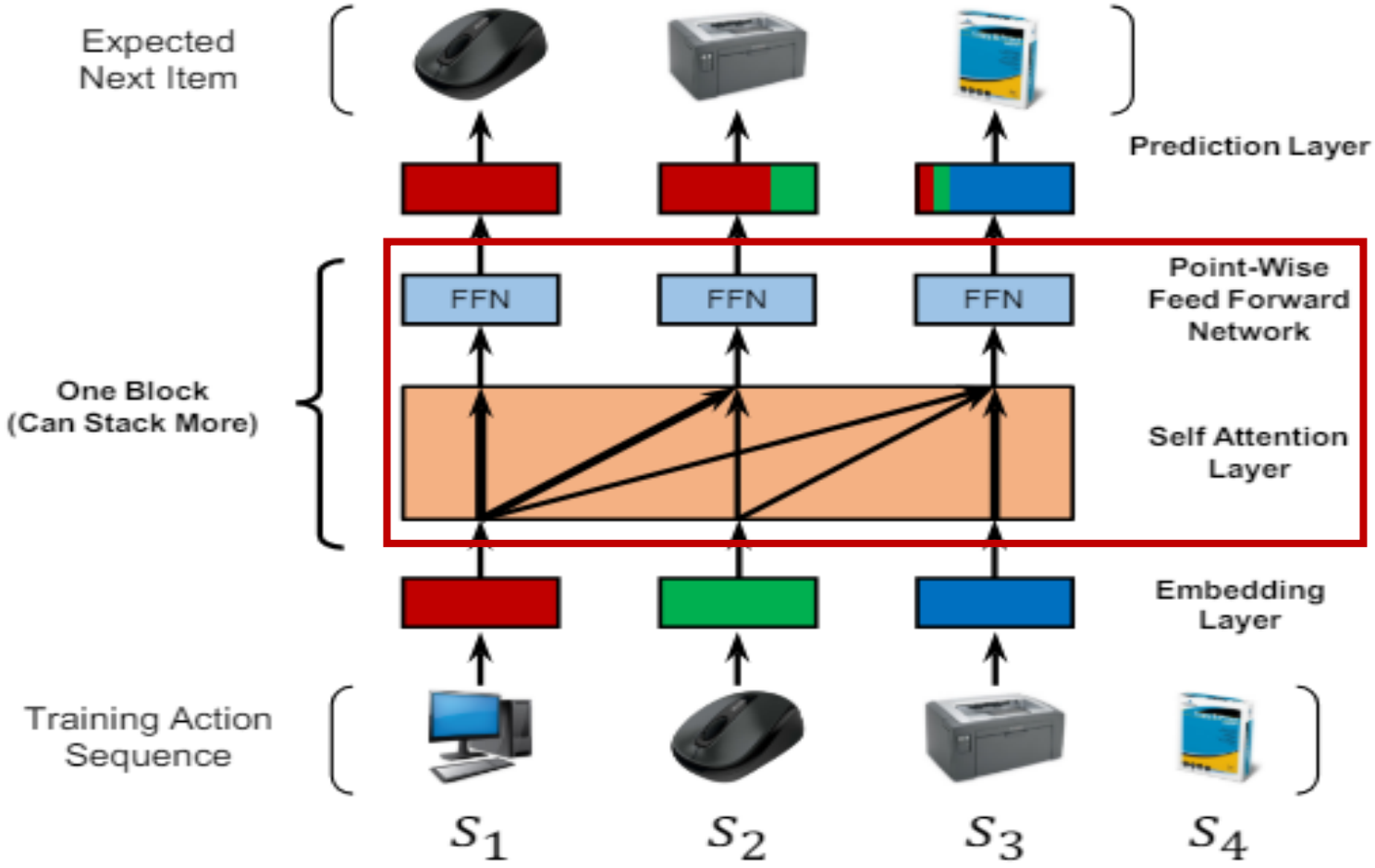
❒ Positional Embedding

Self – attention model은 이전 항목의 위치를 알 수 없음
⇒ 학습 가능한 Positional Embedding을 입력 임베딩에 주입

$$\hat{E} = \begin{bmatrix} M_{s_1} + P_1 \\ M_{s_2} + P_2 \\ M_{s_3} + P_3 \\ \cdots \\ M_{s_n} + P_n \end{bmatrix}$$

SASRec

❒ Self – Attention Block



SASRec

❒ Self – Attention layer

embedding $\hat{E}$를 Input으로 받아 linear projection으로 3개의 matrix 도출 후 Attention layer에 공급

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

❒ Causality

$t+1$번째 아이템을 예측할 때 t이전의 아이템을 고려

$\Rightarrow Q_i$와$K_j(j > i)$사이의 연결을 끊는 형태로 기존의 attention을 변형

❒ Point – Wise Feed – Forward Network

모델의 비선형성을 부여하고 잠재차원 간의 상호작용을 고려

$\Rightarrow$ Point – Wise Feed – Forward Network 적용

$$F_i = FFN(S_i) = ReLU\left(S_iW^{(1)} + b^{(1)}\right)W^{(2)} + b^{(2)}$$

$$Z_1 = S_iW^{(1)} + b^{(1)}$$

$$A_1 = ReLU(Z_1)$$

$$Z_2 = A_1W^{(2)} + b^{(2)}$$

SASRec

❒ Stacking Self – Attention Blocks

아이템 전이를 알아내기 위해 Self – attention block을 여러 개 쌓음
b번째($b > 1$) block은 다음과 같이 정의됨

$\Rightarrow S^{(b)} = SA(F^{(b-1)})\,,\ F_i^{(b)} = FFN\left(S_i^{(b)}\right), \forall i \in \{1,2,\dots,n\}$

- network가 깊어질수록 과적합, 학습시간 증가, 학습 프로세스 불안정

$\Rightarrow g(x) = x + Dropout(g(LayerNorm(x)))$

❒ Residual Connections

가장 마지막에 구매한 아이템의 Embedding을 마지막 layer에 전달하며
낮은 layer의 정보를 모델이 잘 기억할 수 있게 함

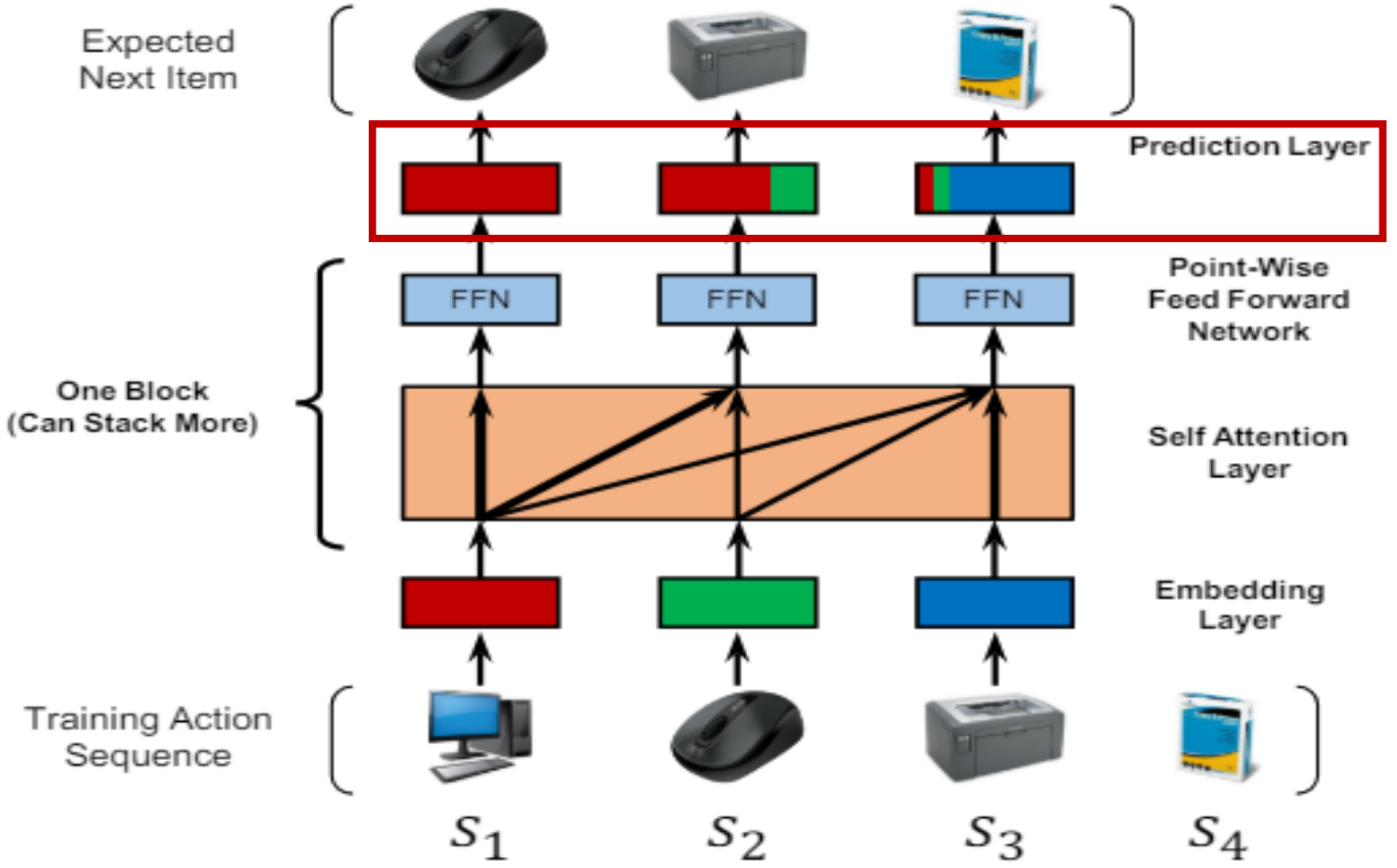
❒ Layer Normalization

Layer normalization의 역할은 input을 featur별로 정규화하여
neural network학습을 안정적이고 빠르게 만드는 것

$LayerNorm(X) = \alpha \odot \frac{X-\mu}{\sqrt{\sigma^2+\epsilon}} + \beta$

SASRec

❒ Prediction Layer



SASRec

❒ Prediction Layer

아이템 i의 interaction score $r_{i,t}$를 도출하기 위해 Matrix Factorization 사용

$$r_{i,t} = F_t^{(b)} N_i^T$$

❒ Shared Item Embedding

과적합을 방지하기 위해 하나의 item embedding M을 공유해서 사용

$$r_{i,t} = F_t^{(b)} M_i^T$$

nonlinear transformation학습

$$(FFN(M_i)M_j^T \neq FFN(M_j)M_i^T)$$

❒ Loss Function

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

Outline

- Introduction
- Proposed Method
- Experiments
- Conclusion

Dataset

❒ Dataset

Dataset	User	Items	Avg. Actions/ user	Avg. Actions/ item	actions
Amazon Beauty	52,024	57,289	7.6	6.9	0.4M
Amazon Games	31,013	23,715	9.3	12.1	0.3M
Steam	334,730	13,047	11.0	282.5	3.7M
MovieLens	6,040	3,416	163.5	289.1	1.0M

❒ Amazon data
- 매우 희소한 특성

❒ Steam data
- 아이템당 행동수가 많음

❒ MovieLens-1M data
- 밀집된 데이터

❒ 다양한 특성을 가진 데이터셋을 통해 SASRec 성능 평가

Comparison

❒ Performance Comparison

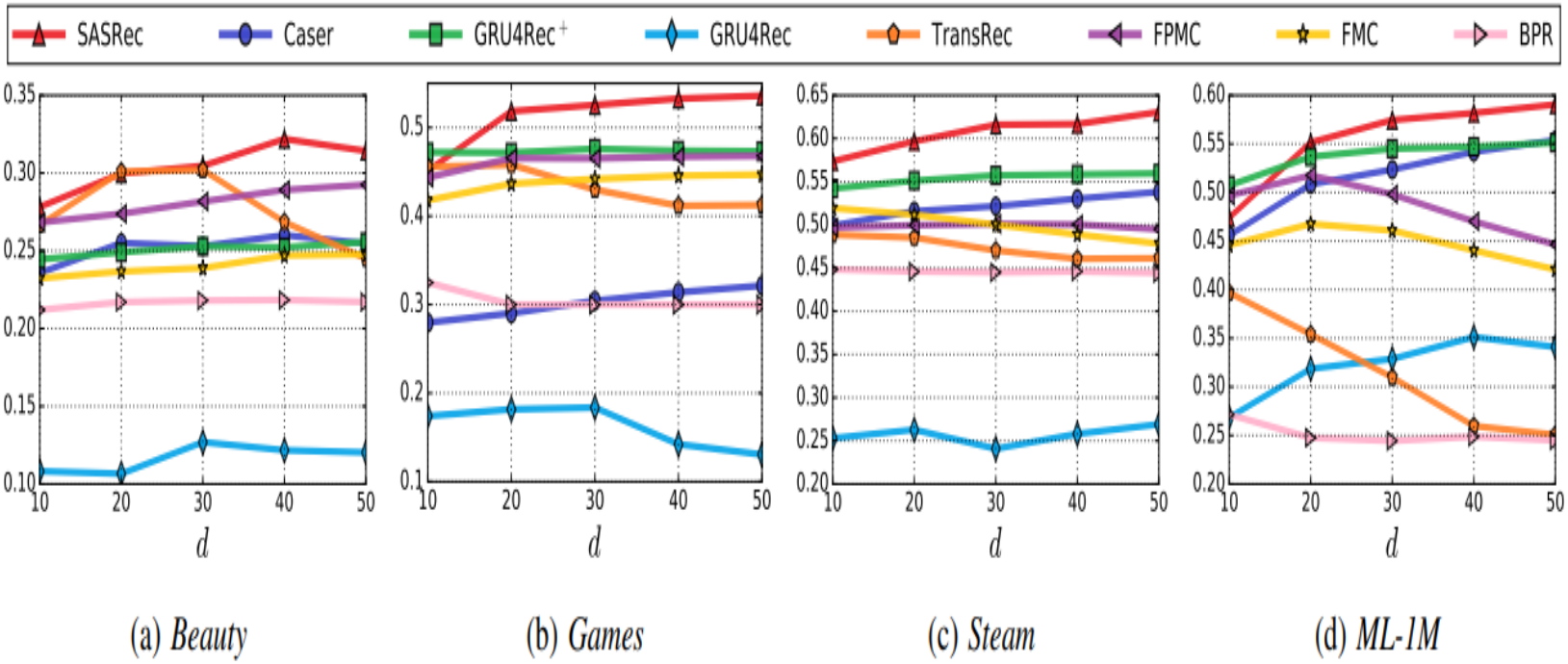
세 그룹으로 나누어 성능 비교
그룹 1 : 사용자 피드백만을 고려하고 순서를 고려하지 않은 일반적인 방법
그룹 2 : 마지막 방문한 아이템을 고려하는 1차 마코프 체인 기반방법
그룹 3 : 여러 이전에 아이템을 고려한 딥러닝 기반 순차 추천 시스템

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

Comparison

❒ Performance Comparison

잠재 차원 d의 영향 분석
d가 증가할 수록 SASRec은 성능이 향상되며, $d = 40$일 때
최고 성능 달성



(a) *Beauty* (b) *Games* (c) *Steam* (d) *ML-1M*

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

Outline

- Introduction
- Proposed Method
- Experiments
- Conclusion

Conclusion

❒ Summary

SASRec은 전체 사용자 시퀀스를 모델링하며, 예측을 위해 소비된 아이템을 상황에 맞게 고려
희소한 데이터셋, 밀집된 데이터셋 모두에서 성능이 좋게 나옴

❒ Weak Points

- 제한된 입력 정보
  - 아이템 임베딩과 위치 임베딩만을 사용하므로 풍부한 정보를 통합하지 못함
- 모델 튜닝의 어려움
  - 많은 하이퍼파라미터(잠재차원, 드롭아웃 비율 등)를 가지며, 데이터에 따른 최적의 하이퍼파라미터를 찾는 것이 어려움
- Input Sequence에 사용자가 선호도가 반영되지 않음

Method

❒ Method 구상

Input sequence에 사용자의 리뷰를 통한 아이템 간의 + - 관계를 파악(균형이론)하여 성능향상

아이템 간의 유사성 및 비유사성을 고려한 가중치를 부여하는 것

1) 각 아이템에 대한 사용자의 감정을 긍정(+), 부정(-)으로 아이템 간 유사성을 반영한 행렬 생성
   - 모든 아이템에 대한 충분한 감정 데이터를 확보하기 어려울 수 있음

2) 유사성 행렬을 사용하여 임베딩 벡터를 조정해 유사성 정보를 반영.
   - 학습 과정이 느려질 수 있음
   - 감정 데이터가 희소할 경우 정보를 신뢰할 수 없게 됨

3) Self – Attention Layer에 유사성 반영하여 어텐션 가중치 조정 이를 통해 유사한 아이템에 더 높은 가중치를 부여, 비유사한 아이템에 더 낮은 가중치를 부여
   - 유사성이 어텐션 가중치에 미치는 영향을 적절히 조정할 필요있음

Thank you!
Q & A