



연속학습을 통한 사용자의 일반적인 표현 학습 (Universal User Representation Learning based on Continual Learning)

박찬영

Assistant Professor, KAIST

Industrial and Systems Engineering
Graduate School of Data Science
Graduate School of AI

cy.park@kaist.ac.kr

BRIEF BIO



Chanyoung Park, Ph.D.

Assistant Professor

ISYSE KAIST

GSDS/GSAI KAIST

Contact Information

- cy.park@kaist.ac.kr
- <http://dsail.kaist.ac.kr/>

■ Research Interest

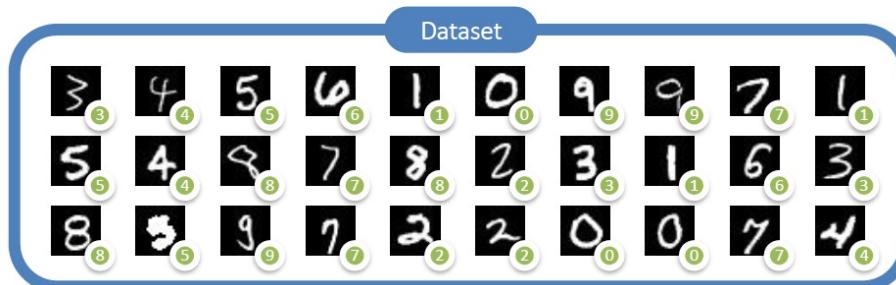
- **Multimodal Data Mining, Applied Machine Learning, Deep Learning**
 - *Mining meaningful knowledge from multimodal data to develop artificial intelligence solutions for various real-world applications across different disciplines*
 - Keywords: Multimodal user behavior analysis, Machine learning for graphs, Graph neural network, Graph representation learning
 - **Application domains:** Recommendation system, Social network analysis, Fraud detection, Sentiment analysis, Purchase/Click prediction, Anomaly detection, Knowledge-graph construction, Time-series analysis, AI4Science (Bioinformatics, Chemistry) etc.

■ Professional Experience

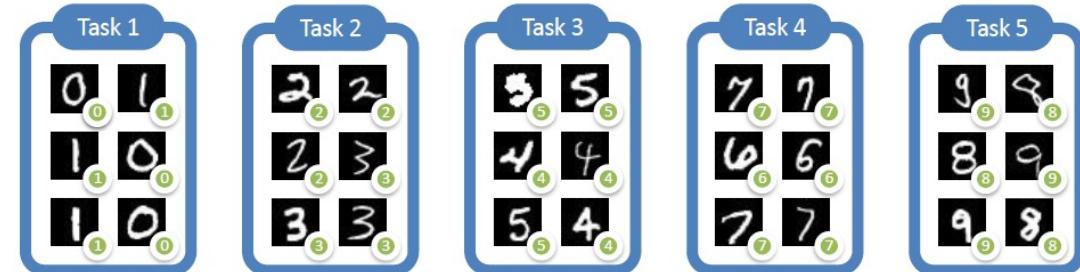
- Assistant Professor, **KAIST** (2020.11 – Present)
- Postdoctoral Research Fellow, **University of Illinois at Urbana-Champaign**, Dept. of Computer Science (2019. 1 – 2020. 10)
- Research Intern, **Microsoft Research Asia** (2017. 9 – 2017. 12)
- Research Intern, **NAVER** (2017. 3 – 2017. 6)

Continual Learning

- 여러 task를 순차적으로 학습하는 방법론
 - 사람이 초등학교, 중학교, 고등학교를 거쳐가면서 새로운 지식들을 학습하듯이, 모델에게 순차적인 지식을 학습시키는 방법
 - 과거의 기억을 잊지 않는 것이 핵심 (Prevent **Catastrophic Forgetting**)



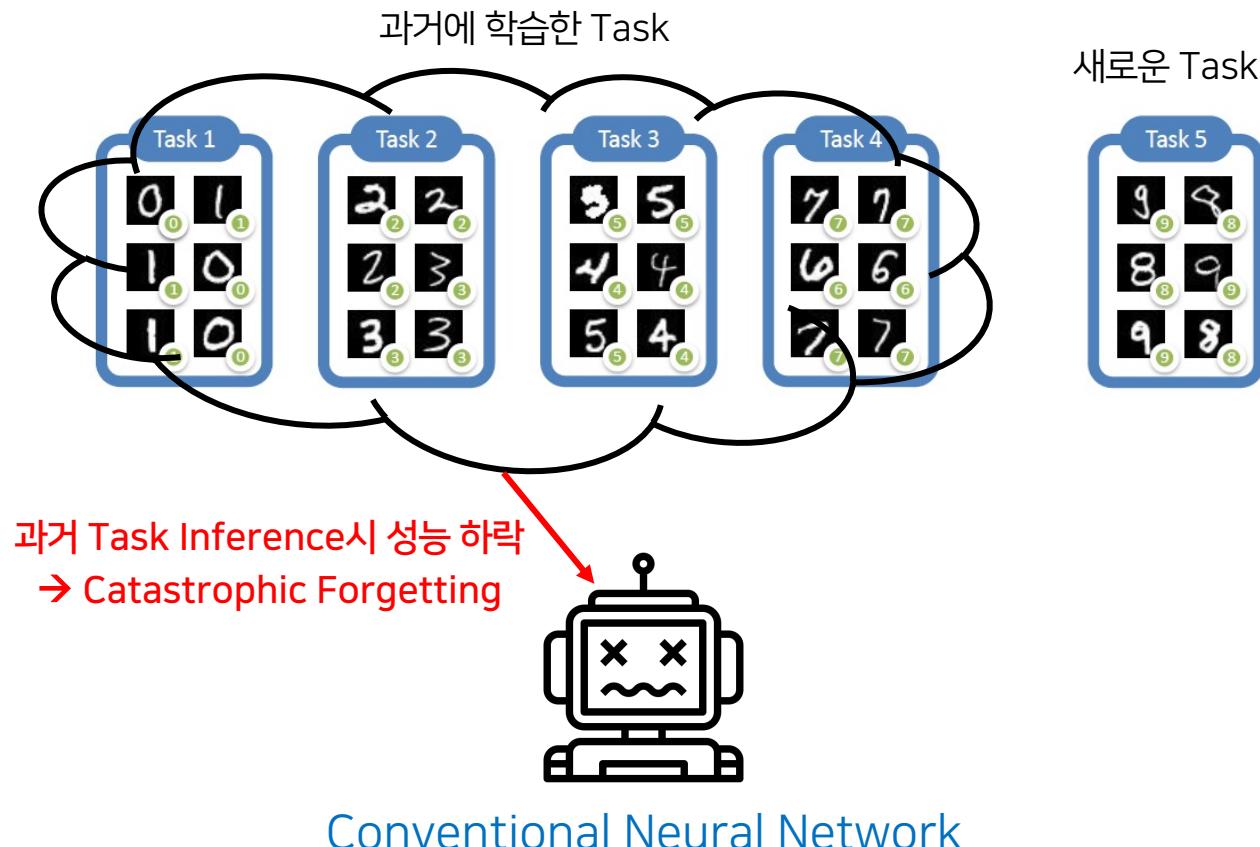
Conventional Neural Network



Continual Learning

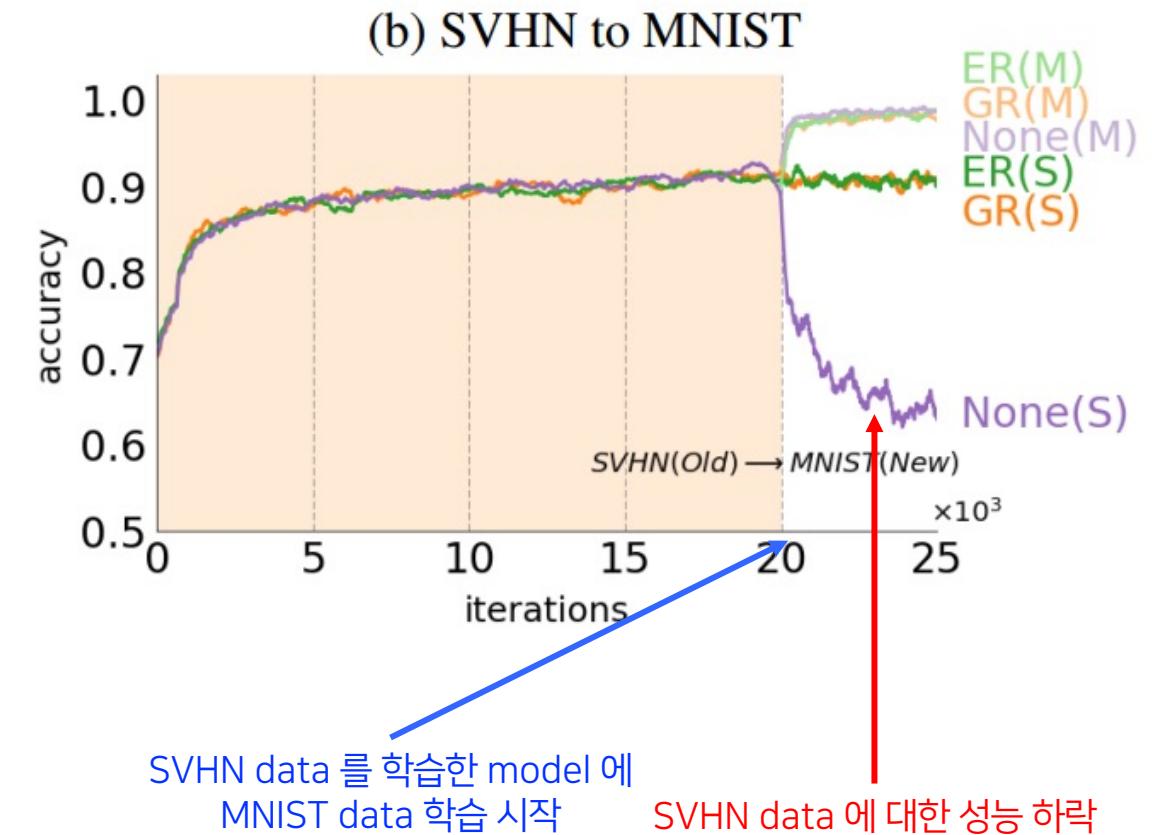
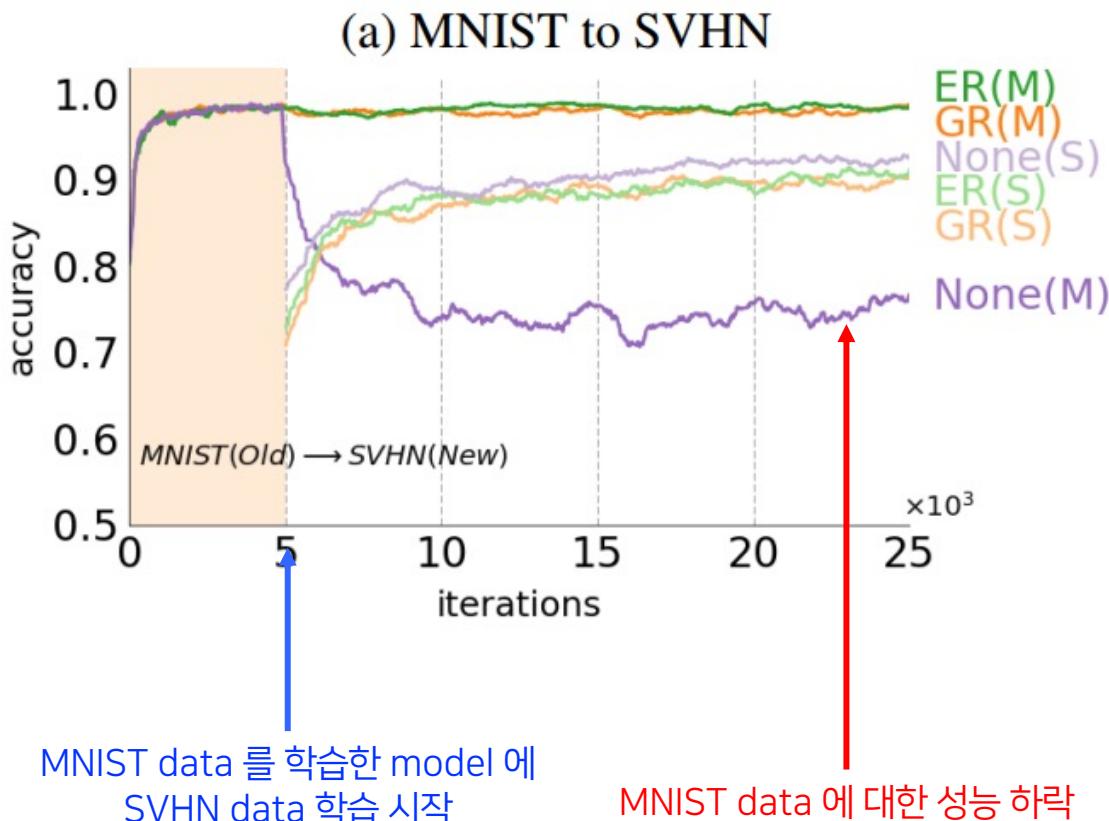
Catastrophic Forgetting

- Task의 순차적인 학습 시, 최근 task의 data distribution에 biased되어 학습됨
→ 과거에 학습한 task에 대한 성능이 현저하게 하락



Catastrophic Forgetting

- Task의 순차적인 학습 시, 최근 task의 data distribution에 biased되어 학습됨
→ 과거에 학습한 task에 대한 성능이 현저하게 하락



MNIST SVHN

Continual Learning의 목표

1. Avoid Catastrophic Forgetting

→ 이전 task의 기억을 보전해야 함

2. Positive Forward Transfer

→ 이전 task에서 학습했던 지식이 다음 task에 도움이 되어야 함

3. Positive Backward Transfer

→ 다음 task에서 학습을 한 지식이 이전 task의 성능 향상에도 도움이 되어야 함

4. Task-Order Free Learning

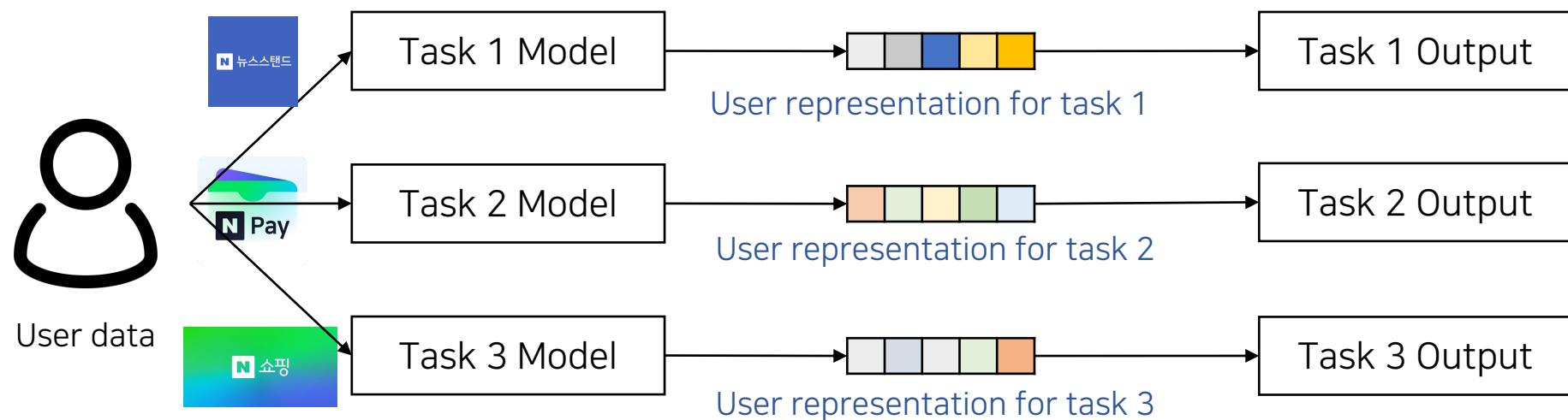
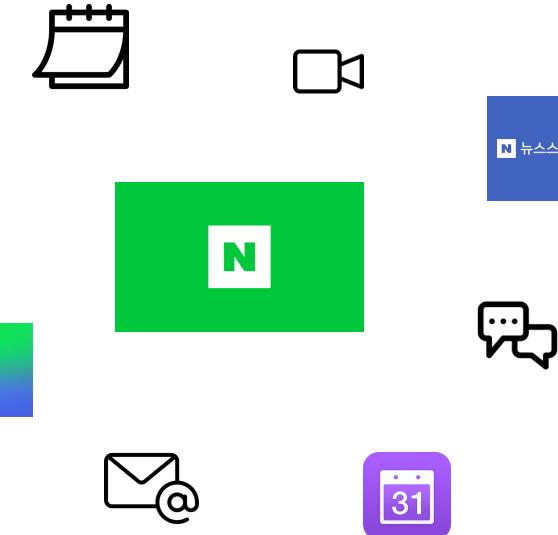
→ Task의 학습 순서와 무관하게 모든 task를 잘 수행해야함

User Representation

- NAVER는 종합 포털 서비스로 검색, 뉴스, 쇼핑, 이메일 등 다양한 서비스 제공
- Task (서비스)마다 별도의 모델이 존재하며, 별도의 유지보수 수행
→ 각 task에 맞춰 모델을 독립적으로 학습 / 평가 진행

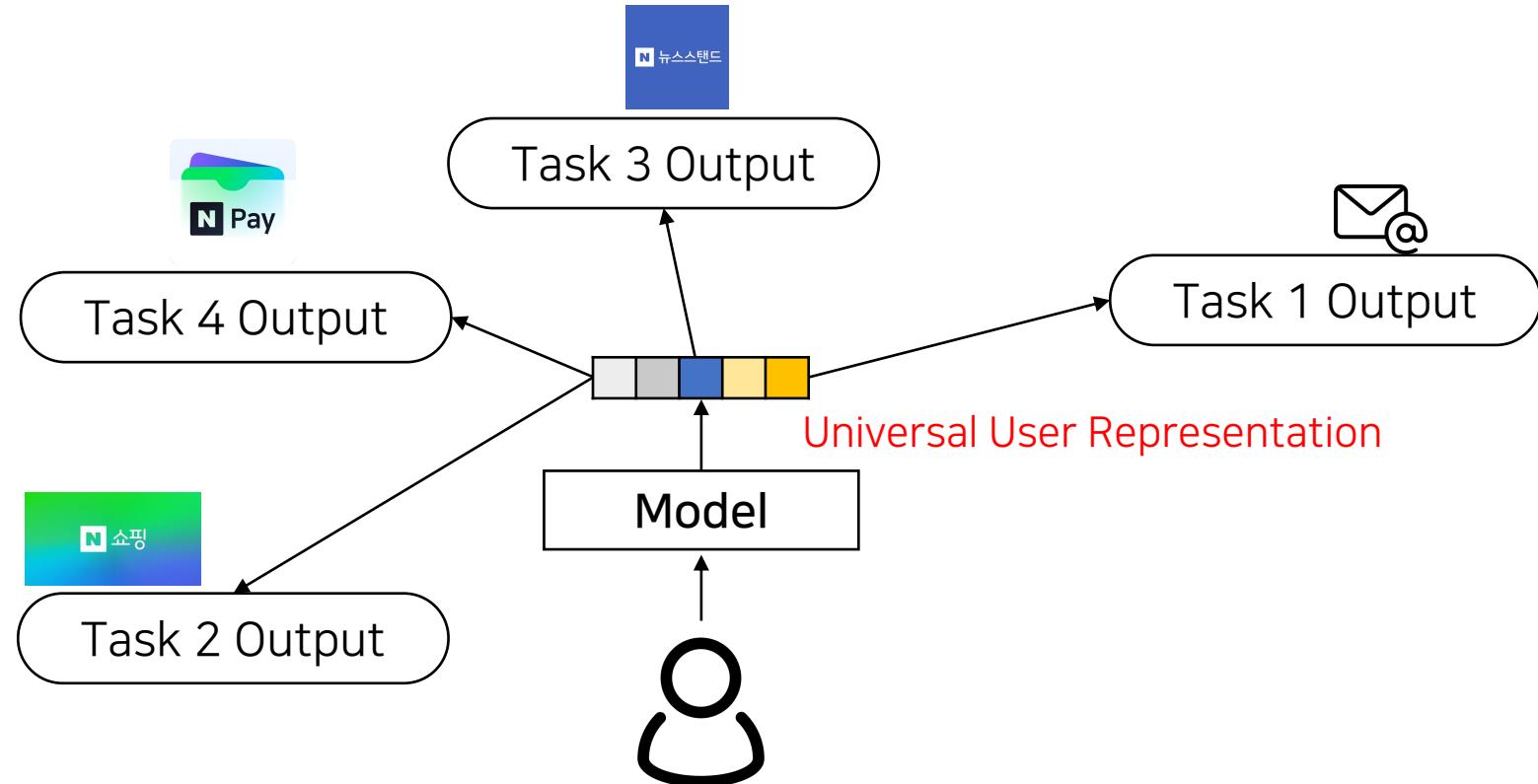
한계점

- 1) 새로운 서비스 (Task) 런칭마다 매번 새로운 모델을 만드는 것은 비효율적
- 2) 서비스간 연관성을 고려하지 못함 (e.g., 쇼핑 - 블로그)



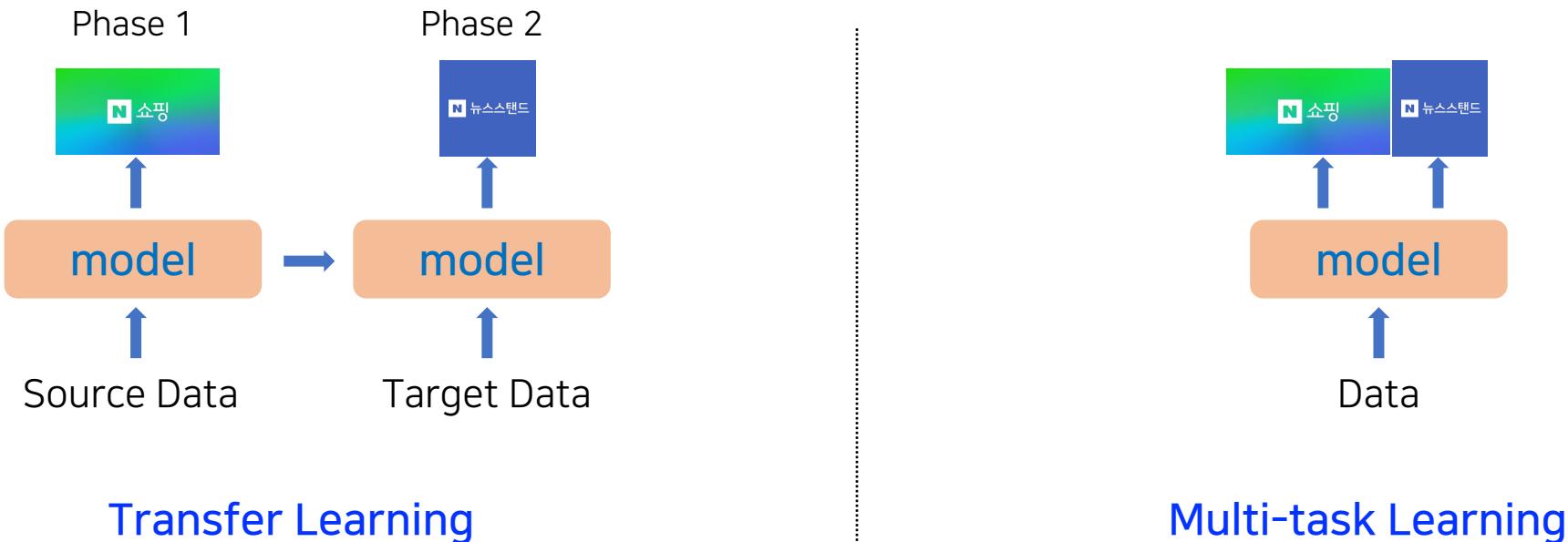
Universal User Representation

- 연구 목표: Universal한 User Representation을 통해 다양한 task를 해결하고자 함
 - 다양한 task에 사용될 수 있는 general한 user representation
 - 하나의 universal한 user representation으로 모든 task의 성능 competitive하게 유지

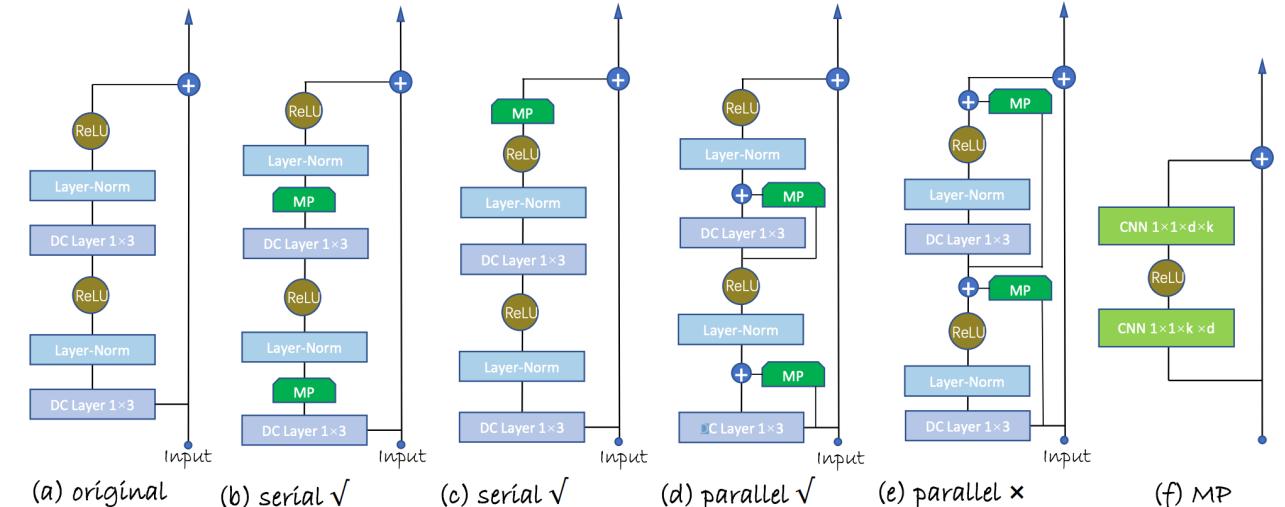
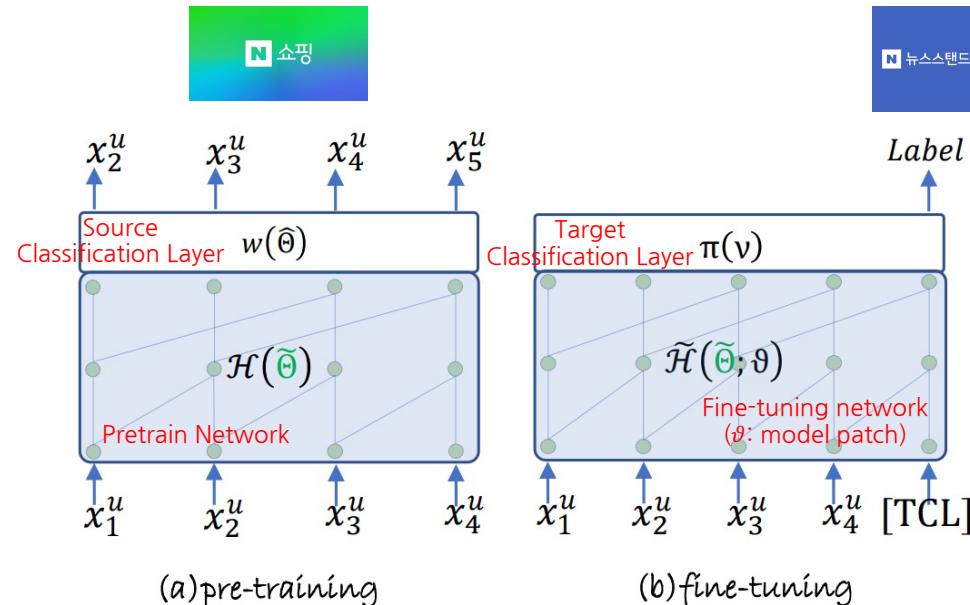
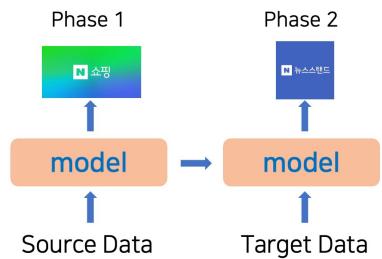


Universal User Representation 관련 기존 연구

- Transfer Learning / Multi-task Learning을 중심으로 연구되어 옴



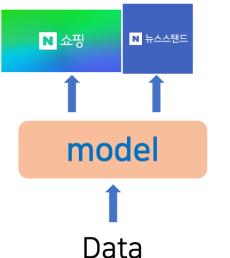
Transfer Learning 기반 방법



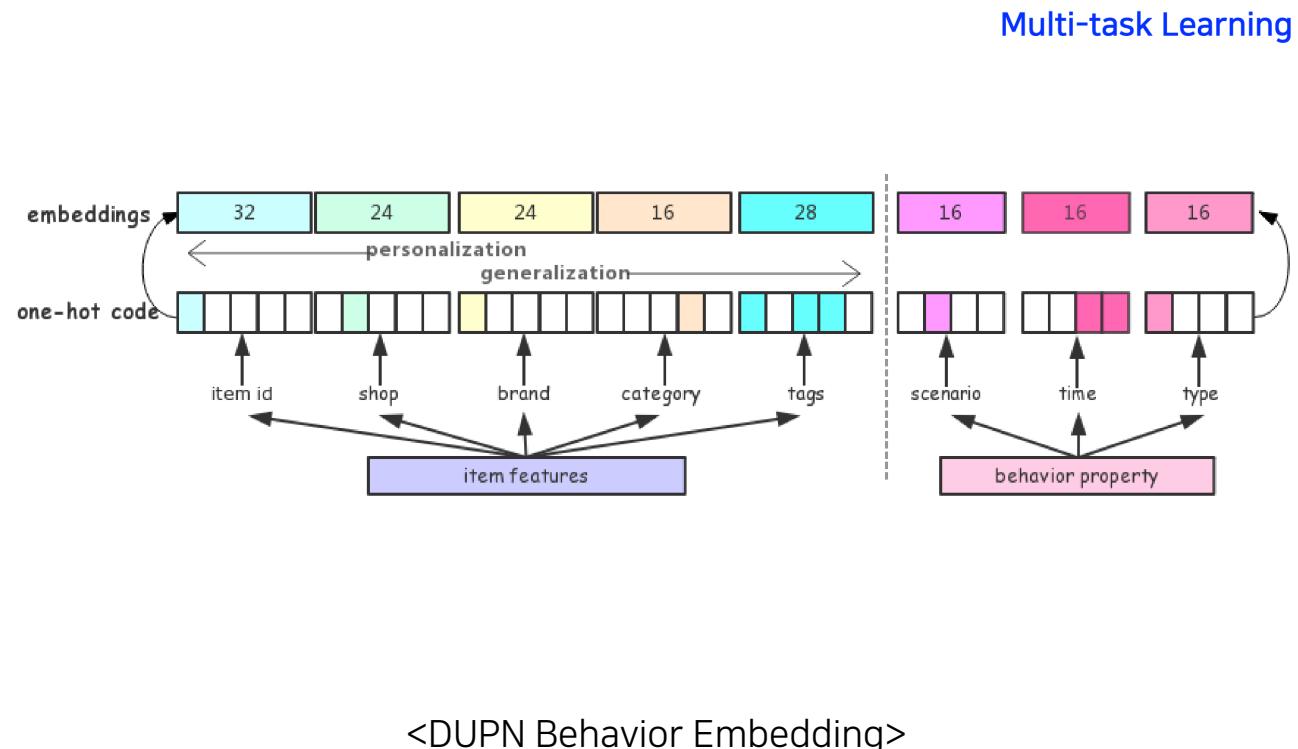
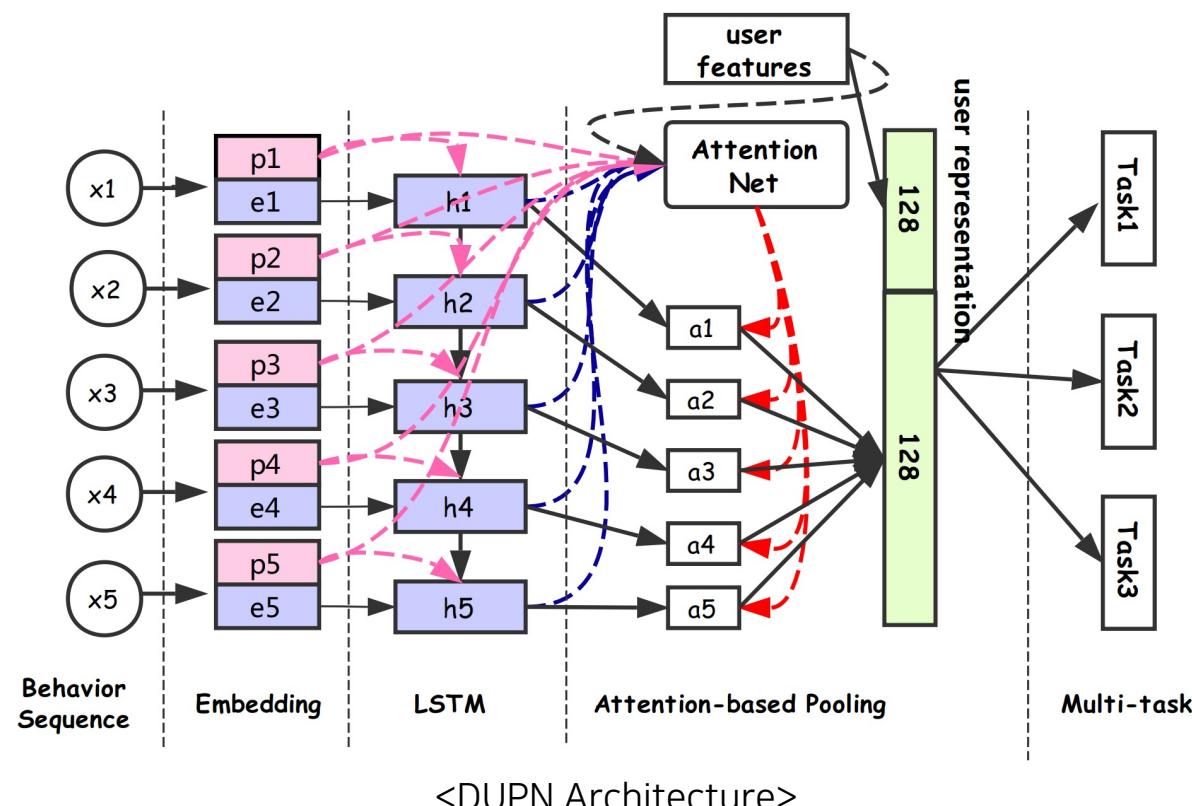
<PeterRec Transfer Learning 예시>

<PeterRec: Model Patch 예시>

- Phase 1: Source domain에 대하여 학습 (Pre-training)
- Phase 2: Pre-train된 모델에, model patch를 붙여 target domain에 대한 fine-tuning 진행



Multi-task Learning 기반 방법

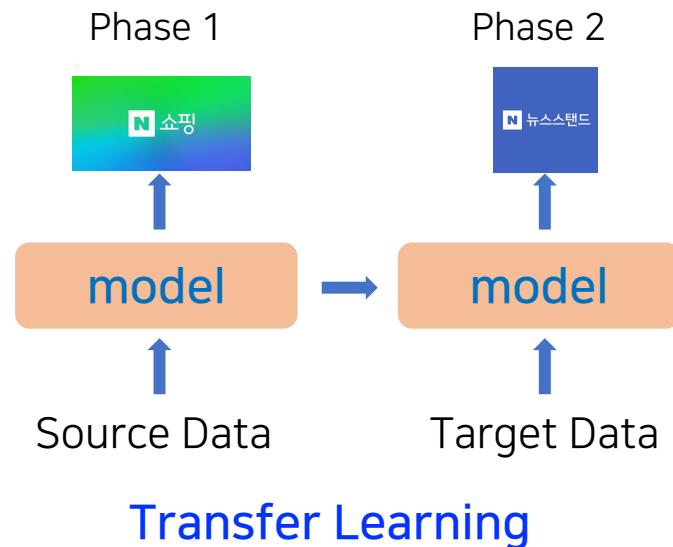


- Item feature와 Behavior property (time, type, etc)를 기반으로 Item embedding 생성
- 이렇게 생성된 item embedding의 Sequence를 기반으로 user representation 생성
- 각 task별로 task-specific feature를 붙여, multi-task learning 진행

기존 연구의 한계점

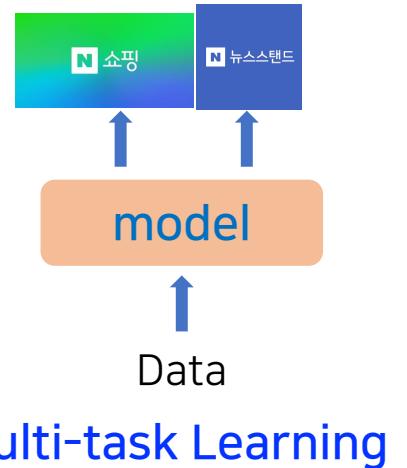
▪ Transfer Learning

- 항상 Task의 쌍이 필요 (Source → Target)
- Target task에 학습하고나면 Source task를 쉽게 잊음



▪ Multi-task Learning

- 모든 Task가 한꺼번에 주어져야 함
 - 새로운 서비스가 런칭하면 재학습 필요



지속적으로 새로운 사용자가 유입되고, 새로운 서비스가 런칭되는 상황에는 적합하지 않음
→ Continual Learning 방법론의 필요성

Continual Learning 방법론

▪ Replay-based approaches

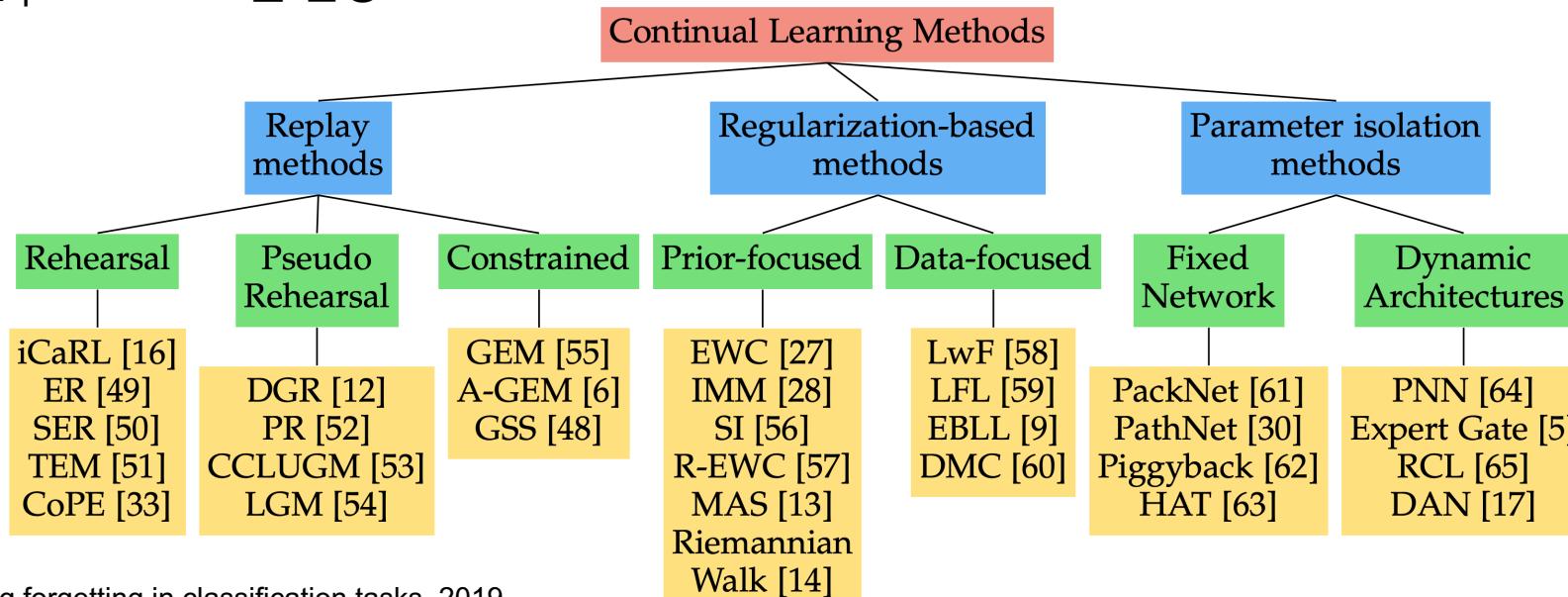
- 과거의 데이터중 일부를 저장해서 학습에 포함
- + Generative replay

▪ Regularization-based approaches

- 과거의 task에 중요했던 parameter는 task가 지나도 유지되도록 함

▪ Parameter isolation-based approaches

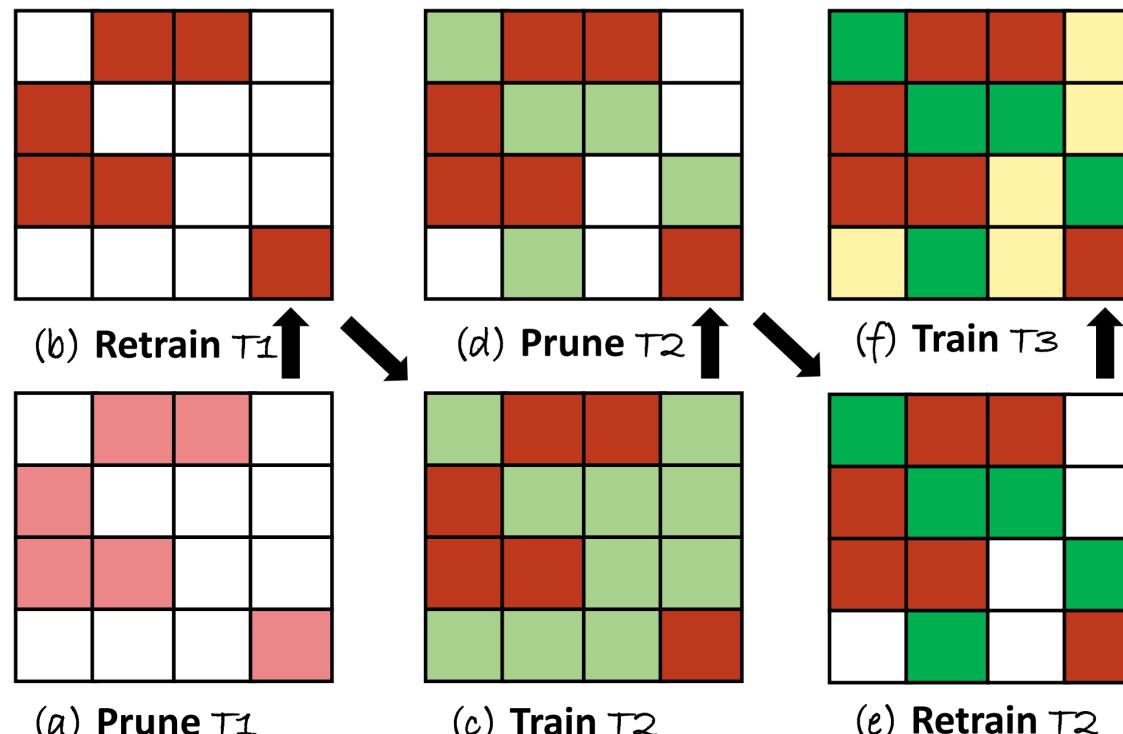
- Task마다 다른 parameter를 할당



One Person, One Model, One World: Learning Continual User Representation without Forgetting (CONURE)

- Key approach: Parameter isolation-based continual learning

- 모델 내의 parameter를 적절히 분할해 task마다 할당
- 하나의 task를 학습한 후, 중요한 parameter들을 선택
- 중요한 parameter들만 선택하여 retrain & freeze 시킨 후, 다음 task를 학습

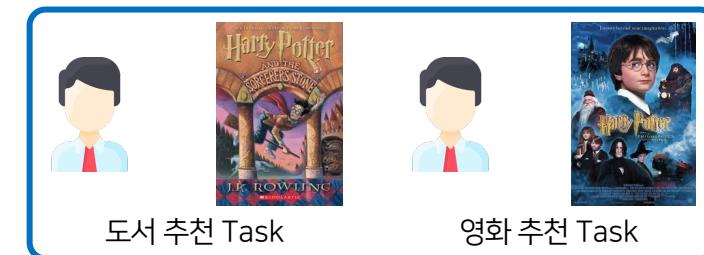


One Person, One Model, One World: Learning Continual User Representation without Forgetting (CONURE)

■ 문제점

- 1. Parameter isolation 방법론의 한계
 - a) 최근 task일수록 모델의 성능이 하락
 - Task가 순차적으로 학습이 되어 가면서, 나머지 task들의 학습에 활용가능한 parameter수 감소
 - b) 학습 가능한 task 개수가 제한됨
 - Parameter가 모두 사용된 이후에는 더이상 다른 task를 학습할 수 없음
 - c) Positive backward transfer가 일어나지 않음
 - 과거 task학습에 사용된 parameter가 고정됨
- 2. Task간 관계 무시
 - 과거 task의 학습에 사용된 parameter는 미래 task의 학습에 사용되지 않음
 - Task간 관계: Positively related tasks / Negatively related tasks
 - Task간 관계를 고려할 경우,
 - Positively related task들 사이에서 positive transfer가 가능하고
 - Negatively related task들 사이에서 negative transfer를 방지할 수 있음

Positively related task



Negatively related task





Task Relation-aware Continual User Representation Learning

Sein Kim¹, Namkyeong Lee¹, Donghyun Kim², Minchul Yang², Chanyoung Park¹

¹KAIST, ²NAVER Corporation

KDD 2023 - ACM SIGKDD Conference on Knowledge Discovery and Data Mining



Contribution

- Parameter isolation 기반 continual learning의 단점 극복
 - 최근 task의 성능 보장
 - 학습 가능한 task개수 제한 X
- Task간의 관계를 고려
 - 새로 학습한 task의 knowledge가 이전에 학습한 task에 도움을 줄 수 있도록 함 (Positive Backward Transfer)
 - Positive transfer 최대화 및 Negative transfer 최소화

Notation

- $\mathcal{T} = \{T_1, T_2, \dots, T_M\}$ Set of consecutive tasks ($T_{1:M}$)
 - $\mathcal{U} = \{u_1, u_2, \dots, u_N\}$ Set of users
 - $\mathbf{x}^{u_l} = \{x_1^{u_l}, x_2^{u_l}, \dots, x_n^{u_l}\}$ Behavior sequence of user u_l
- e.g., news/video watching history, item click history, recent search queries
 - $\mathcal{U}^{T_i} = \{u_1, u_2, \dots, u_{|\mathcal{U}^{T_i}|}\}$ Set of users associated with task T_i ($\mathcal{U}^{T_i} \subset \mathcal{U}$)
 - Note that T_1 contains all users ($\mathcal{U}^{T_1} = \mathcal{U}$)
 - $\mathbf{Y}^{T_i} = \{y_{u_1}^{T_i}, \dots, y_{u_{|\mathcal{U}^{T_i}|}}^{T_i}\}$ Set of labels associated with task T_i (e.g., Purchased item, gender, age, etc)
- Label of u_1 in task T_i

Dataset	Task 1 (T_1)		Task 2 (T_2)		Task 3 (T_3)		Task 4 (T_4)		Task 5 (T_5)		Task 6 (T_6)	
	$ \mathcal{U}^{T_1} $	$ \mathbf{Y}^{T_1} $	$ \mathcal{U}^{T_2} $	$ \mathbf{Y}^{T_2} $	$ \mathcal{U}^{T_3} $	$ \mathbf{Y}^{T_3} $	$ \mathcal{U}^{T_4} $	$ \mathbf{Y}^{T_4} $	$ \mathcal{U}^{T_5} $	$ \mathbf{Y}^{T_5} $	$ \mathcal{U}^{T_6} $	$ \mathbf{Y}^{T_6} $
TTL	Watching 1.47M	0.64M	Clicking 1.39M	17K	Thumb-up 0.25M	7K	Age 1.47M	8	Gender 1.46M	2	Life status 1M	6
ML	Clicking 0.74M	54K	4-star 0.67M	26K	5-star 0.35M	16K	-	-	-	-	-	-
NAVER Shopping	Search Query 0.9M	0.58M	Search Query 0.59M	0.51M	Item Category 0.15M	4K	Item Category 0.15M	10	Gender 0.82M	2	Age 0.82M	9

Problem Formulation

- Given: Consecutive tasks $\mathcal{T} = \{T_1, T_2, \dots, T_M\}$
 - Each T_i is associated with \mathcal{U}^{T_i} and \mathcal{Y}^{T_i}

- Goal

- Sequentially train a single model M on each task T_i
 - Predict the label of each user u_l

$$\mathbf{y}_{u_l}^{T_i} = G^{T_i}(\mathcal{M}(\mathbf{x}^{u_l}))$$

Backbone model

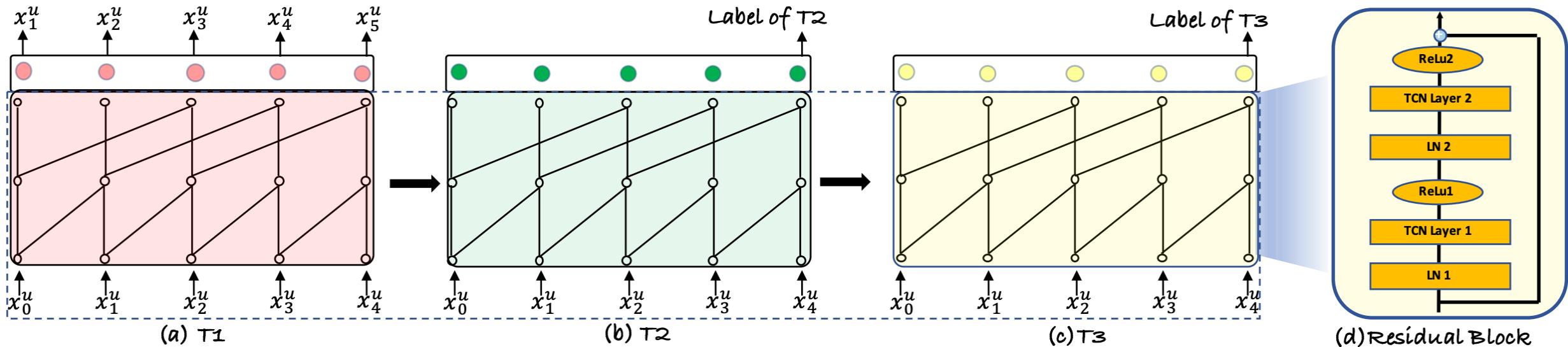
Label of user u_l
in task T_i

Task-specific
classifier of T_i

Behavior sequence of user u_l

Backbone Encoder: Temporal Convolutional Network

- Our framework is network-agnostic
- Dilated convolution

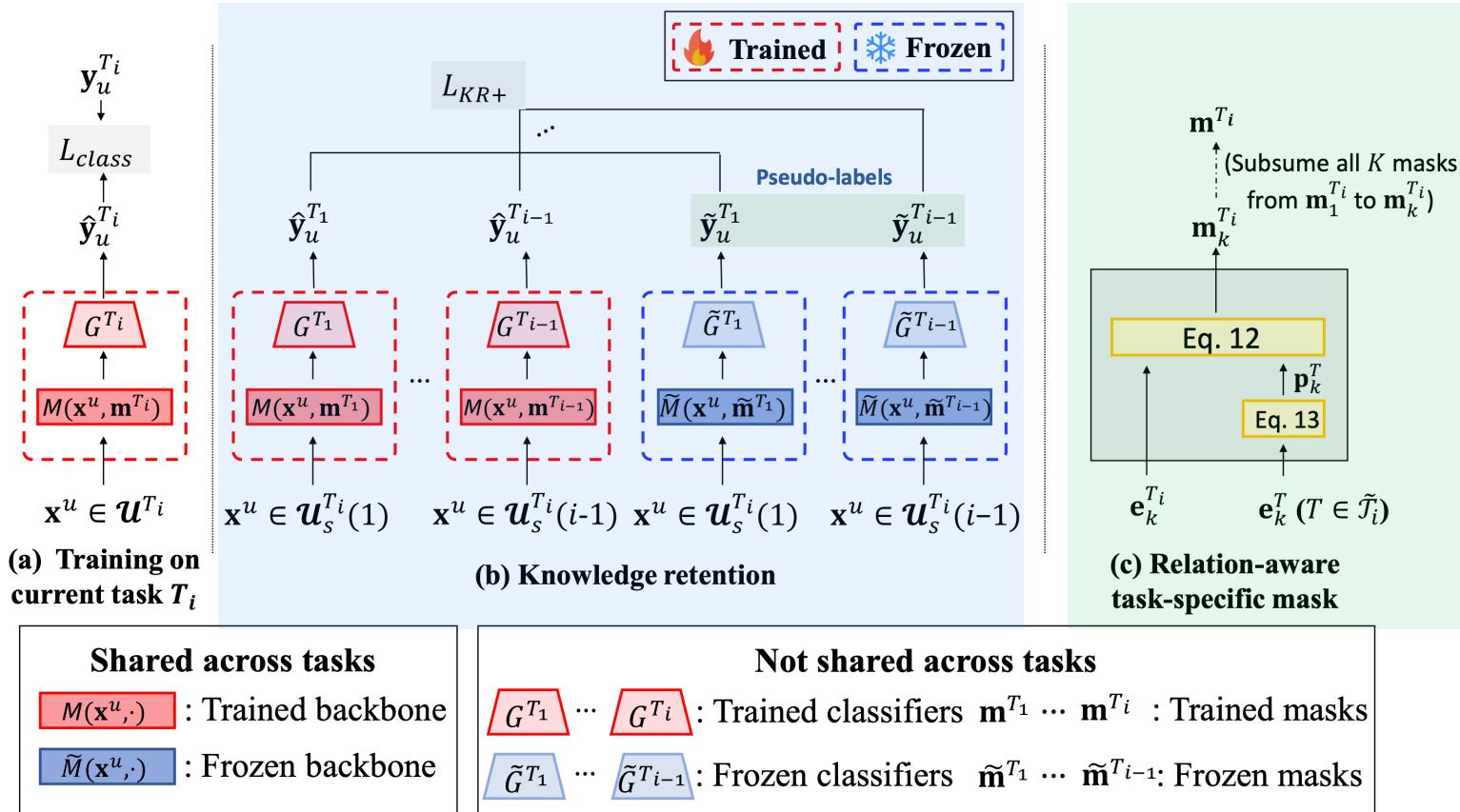


$$\mathbf{E}_k^{u_l} = F_k(\mathbf{E}_{k-1}^{u_l}) + \mathbf{E}_{k-1}^{u_l} = R_k(\mathbf{E}_{k-1}^{u_l}) \quad k\text{-th residual block}$$

$$\mathbf{E}_0^{u_l} \in \mathbb{R}^{n \times f} \quad \text{Initial embedding matrix of } x^{u_l}$$

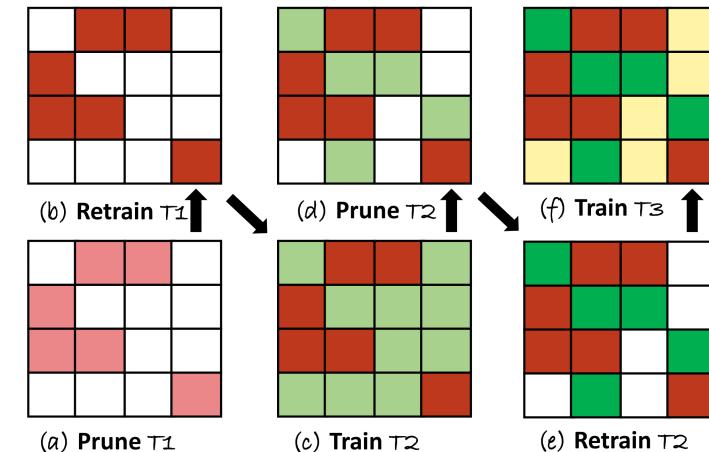
Key Components of TERACON

- 1. Task Embedding을 통해 Task-specific한 mask를 학습 (Soft masking)
 - Task간 Relation을 고려한 Task-specific mask 학습
- 2. Pseudo labeling 기법을 통한 Catastrophic forgetting 방지 (Knowledge retention)



Task-specific Mask

- Task별로 모델의 중요한 부분을 capture하기 위함
 - Soft mask
 - 모든 parameter가 각 task 학습 시 사용 가능토록 함
 - 각 task에 parameter를 exclusive하게 배정하는것이 아니기 때문에 task개수 제한 X
 - 각 layer의 output에 적용되는 mask (Mask size 관점의 효율성)
 - cf) CONURE: 모델 parameter 자체에 적용 됨



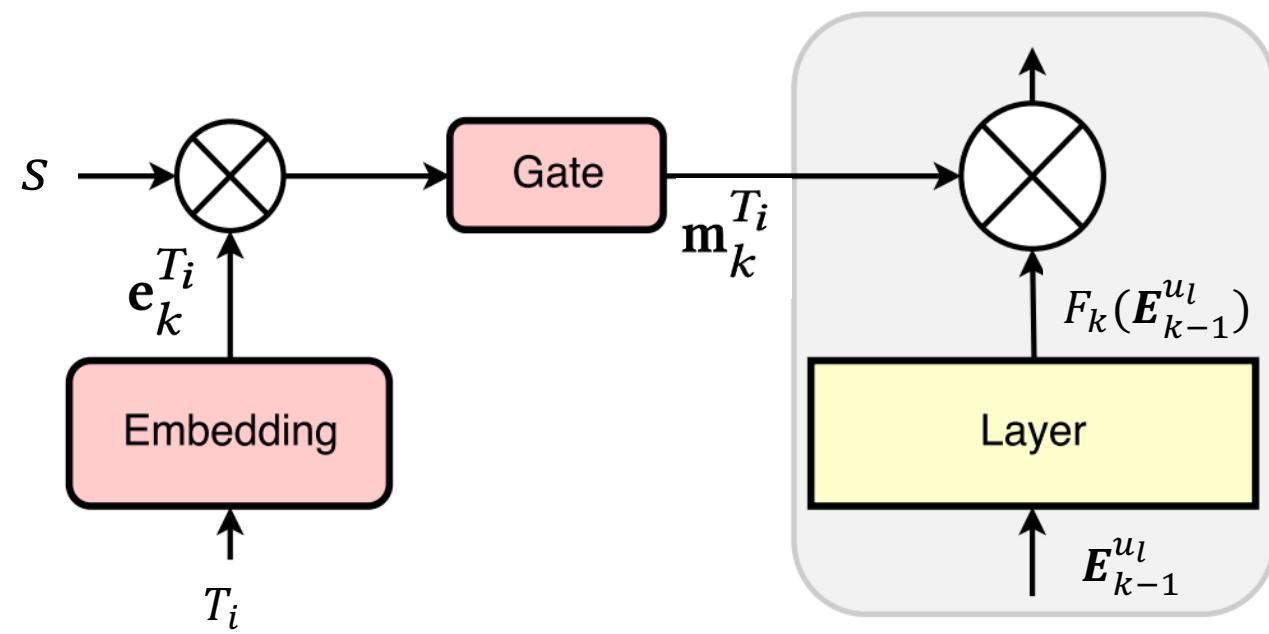
[CONURE: Hard Mask]

$$\mathbf{m}_k^{T_i} = \sigma(s \cdot \mathbf{e}_k^{T_i})$$

Task-specific mask
Scaling hyperparameter

$$F_k(\mathbf{E}_{k-1}^{u_l}) \odot \mathbf{m}_k^{T_i}$$

Element-wise product



Relation-aware Task-specific Mask

- Task간 관계를 포착하기 위하여, 과거 task와 현재 task의 정보를 aggregate

$$\mathbf{m}_k^{T_i} = \sigma(s \cdot \mathbf{e}_k^{T_i})$$

Task-specific mask Task-specific embedding
Scaling hyperparameter

Vanilla Task-specific Mask

Relation-aware Task-specific mask

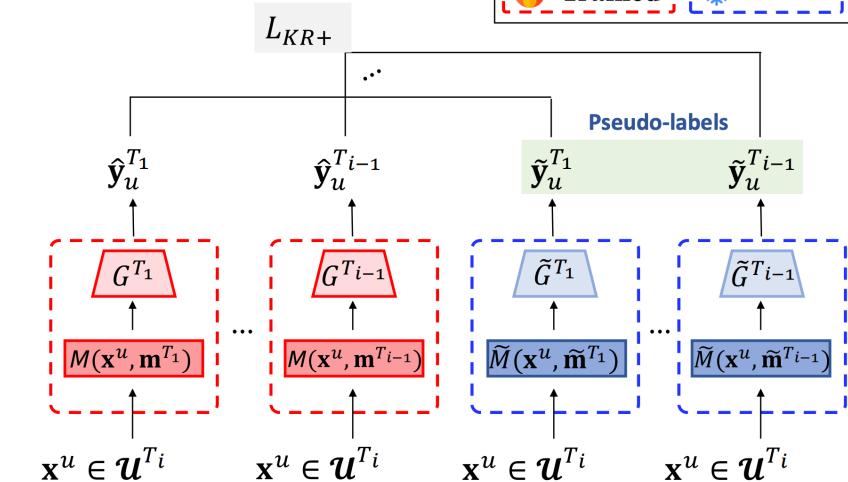
$$\mathbf{m}_k^{T_i} = \sigma \left(s \cdot f_k^{T_i} \left[\frac{\tanh(s \cdot \mathbf{e}_k^{T_i})}{\text{1-layer MLP}} \parallel \frac{(\|_{T \in \tilde{\mathcal{T}}_i} \mathbf{p}_k^T)}{\text{과거 task 들}} \right] \right) \in \mathbb{R}^f$$

$$\mathbf{p}_k^T = [\tanh(s \cdot \mathbf{e}_k^T) \parallel \tanh(-s \cdot \mathbf{e}_k^T)] \in \mathbb{R}^{2 \times f}$$

Relation-aware Task-specific Mask

Knowledge Retention

- BUT, 단순히 masking을 적용하는 것은 catastrophic forgetting 유발
 - Soft mask는 task 간 parameter 공유가 가능케 하기 때문
 - As opposed to parameter freezing
- 해결책: 과거 task들에 대한 현재 모델의 지식을 현재 모델 학습에 활용
- 접근 방법: 현재 task에 주어진 user behavior sequence를 기반으로 과거 task에 대한 pseudo-label \tilde{y} 생성
 - x 가 user behavior sequence이므로 과거 task에 user가 등장하지 않았어도 pseudo-label 얻을 수 있음



과거 task 현재 모델 과거 task
 $\tilde{y}_{u_l}^{T_j} = \tilde{G}^{T_j}(\tilde{M}(x^{u_l}; \tilde{m}^{T_j})) \quad \text{for } j = 1, \dots, i - 1$

과거 task들 과거 task 현재 모델 과거 task

$$\mathcal{L}_{KR} = \mathbb{E}_{1 \leq j < i} \left[\mathbb{E}_{u_l \in \mathcal{U}^{T_i}} \left[L_{MSE} (\tilde{G}^{T_j}(\tilde{M}(x^{u_l}; \tilde{m}^{T_i})), \tilde{y}_{u_l}^{T_j}) \right] \right]$$



- G : Task-specific classifier
- M : Backbone 모델
- m : Task-specific mask
- \tilde{y} : Pseudo-label

Relation-aware User Sampling Strategy

- 현재 task에 존재하는 모든 user들에 대해 knowledge retention (L_{KR})을 하는 것은 비효율적

$$\mathcal{L}_{KR} = \underset{1 \leq j < i}{\mathbb{E}} \left[\underset{u_l \in \mathcal{U}^{T_i}}{\mathbb{E}} \left[L_{MSE}(G^{T_j}(\mathcal{M}(\mathbf{x}^{u_l}; \mathbf{m}^{T_i})), \tilde{\mathbf{y}}_{u_l}^{T_j}) \right] \right]$$

과거 task들

- Intuition: Knowledge retention 시 더 중요한/덜 중요한 user가 있을 것!
- Key idea: 현 task와 비슷한 과거 task의 knowledge를 retention을 하는 것이 더 쉬울 것
→ 따라서, 현 task (T_i) 와 비슷한 과거 task (T_j) 일 수록 덜 sampling을 하자 (Mask간 유사도 기반)

$$\mathcal{U}_s^{T_i}(j) \leftarrow \text{sample}(\mathcal{U}^{T_i}, \rho_{i,j})$$

Sampling rate
(when sampling \mathcal{U}^{T_i} for T_j)

$$\rho_{i,j} = 1 - \frac{1}{K} \sum_{k=1}^K \sigma(c \times \cos(\mathbf{m}_k^{T_i}, \tilde{\mathbf{m}}_k^{T_j}))$$

↑
around 0.007~0.08

$$\mathcal{L}_{KR+} = \underset{1 \leq j < i}{\mathbb{E}} \left[\frac{\rho_{i,j}}{\sum_{k=1}^{i-1} \rho_{i,k}} \sum_{u_l \in \mathcal{U}_s^{T_i}(j)} \mathcal{L}_{MSE}(G^{T_j}(\mathcal{M}(\mathbf{x}^{u_l}; \mathbf{m}^{T_i})), \tilde{\mathbf{y}}_{u_l}^{T_j}) \right]$$

과거 task들

Training and Inference

■ Training

- Given current task T_i , we train the following loss

$$\mathcal{L} = \mathcal{L}_{\text{class}} + \alpha \mathcal{L}_{\text{KR+}}$$

Downstream loss (classification)

Knowledge retention loss

■ Inference

- T_M 까지 위 학습 과정을 진행 후, 이전 모든 task들인 T_1, T_2, \dots, T_M 에 대해 Inference

$$\hat{\mathbf{y}}_{u_l}^{T_i} = G^{T_i}(\mathcal{M}(\mathbf{x}^{u_l}; \mathbf{m}^{T_i}))$$

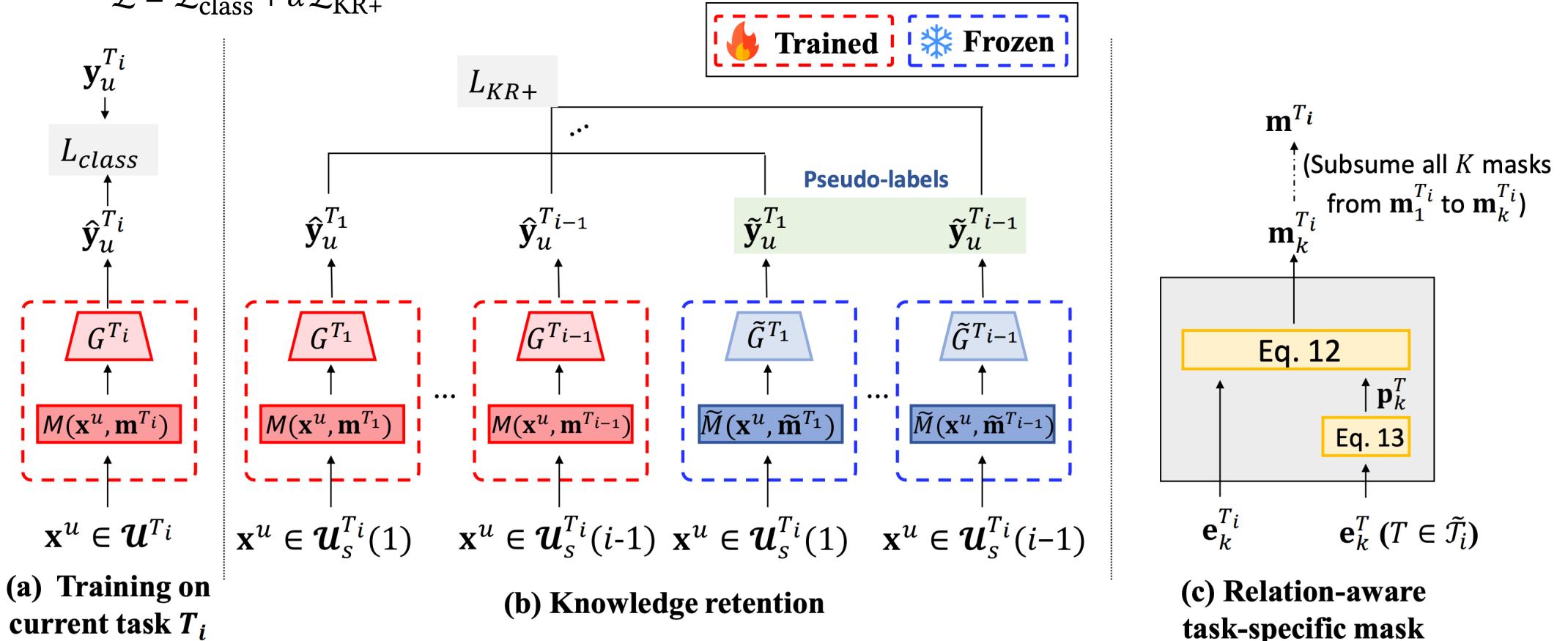
Prediction of labels of u_l in task T_i

$$\mathcal{L}_{\text{class}} = \mathbb{E}_{u_l \in \mathcal{U}^{T_i}} \left[L_{\text{CE}}(G^{T_i}(\mathcal{M}(\mathbf{x}^{u_l}; \mathbf{m}^{T_i})), \mathbf{y}_{u_l}^{T_i}) \right]$$

$$\mathcal{L}_{\text{KR+}} = \mathbb{E}_{1 \leq j < i} \left[\frac{\rho_{i,j}}{\sum_{k=1}^{i-1} \rho_{i,k}} \sum_{u_l \in \mathcal{U}_s^{T_i}(j)} \mathcal{L}_{\text{MSE}}(G^{T_j}(\mathcal{M}(\mathbf{x}^{u_l}; \mathbf{m}^{T_i})), \tilde{\mathbf{y}}_{u_l}^{T_j}) \right]$$

Overall Architecture

$$\mathcal{L} = \mathcal{L}_{\text{class}} + \alpha \mathcal{L}_{\text{KR+}}$$



Shared across tasks

- $M(\mathbf{x}^u, \cdot)$: Trained backbone
- $\tilde{M}(\mathbf{x}^u, \cdot)$: Frozen backbone

Not shared across tasks

- G^{T_1}, \dots, G^{T_i} : Trained classifiers $\mathbf{m}^{T_1}, \dots, \mathbf{m}^{T_i}$: Trained masks
- $\tilde{G}^{T_1}, \dots, \tilde{G}^{T_{i-1}}$: Frozen classifiers $\tilde{\mathbf{m}}^{T_1}, \dots, \tilde{\mathbf{m}}^{T_{i-1}}$: Frozen masks

Experiments: Datasets

- Public dataset 2개 (Tencent, MovieLens) / 네이버 쇼핑 데이터셋 1개
 - $|U^{T_i}|$: Number of users in T_i
 - $|y^{T_i}|$: Number of unique labels in T_i

Dataset	Task 1 (T_1)		Task 2 (T_2)		Task 3 (T_3)		Task 4 (T_4)		Task 5 (T_5)		Task 6 (T_6)	
	$ U^{T_1} $	$ y^{T_1} $	$ U^{T_2} $	$ y^{T_2} $	$ U^{T_3} $	$ y^{T_3} $	$ U^{T_4} $	$ y^{T_4} $	$ U^{T_5} $	$ y^{T_5} $	$ U^{T_6} $	$ y^{T_6} $
TTL	Watching 1.47M	0.64M	Clicking 1.39M	17K	Thumb-up 0.25M	7K	Age 1.47M	8	Gender 1.46M	2	Life status 1M	6
ML	Clicking 0.74M	54K	4-star 0.67M	26K	5-star 0.35M	16K	-	-	-	-	-	-
NAVER Shopping	Search Query 0.9M	0.58M	Search Query 0.59M	0.51M	Item Category 0.15M	4K	Item Category 0.15M	10	Gender 0.82M	2	Age 0.82M	9

Experiments: Task description

Tencent dataset

- T_1 : 사용자의 QQ 브라우저에서의 최근 뉴스/영상 시청 기록 100개
- T_2 : 사용자의 Kandian 플랫폼에서의 상품 클릭 예측
- T_3 : 사용자의 Kandian 플랫폼에서의 상품 좋아요 예측
- T_4 : 사용자의 나이 예측
- T_5 : 사용자의 성별 예측
- T_6 : 사용자의 Life status (미혼, 기혼, 임신, 부모, etc) 예측

Movielens dataset

- T_1 : 사용자의 최근 영화 시청 기록 30개 (별점 4,5점 제외)
- T_2 : 사용자가 별점 4점 매긴 영화 예측
- T_3 : 사용자가 별점 5점 매긴 영화 예측

Dataset	Task 1 (T_1)		Task 2 (T_2)		Task 3 (T_3)		Task 4 (T_4)		Task 5 (T_5)		Task 6 (T_6)	
TTL	Watching 1.47M	Clicking 0.64M	Clicking 1.39M	Thumb-up 17K	Clicking 0.25M	Thumb-up 7K	Age 1.47M	Gender 8	Age 1.46M	Gender 2	Life status 1M	Life status 6
ML	Clicking 0.74M	4-star 54K	Clicking 0.67M	5-star 26K	Clicking 0.35M	5-star 16K	-	-	-	-	-	-
NAVER Shopping	Search Query 0.9M	Search Query 0.58M	Search Query 0.59M	Item Category 0.51M	Item Category 0.15M	Item Category 4K	Item Category 0.15M	Gender 10	Item Category 0.82M	Gender 2	Age 0.82M	Age 9

Sequential Recommendation

Item Recommendation

User Profile Prediction

Sequential Recommendation

Item Recommendation

Experiments: Task description

■ NAVER Shopping dataset

- T_1 : 사용자의 최근 포탈 검색어 60개
- T_2 : 사용자의 다음 포탈 검색어 5개 예측
- T_3 : 사용자의 쇼핑 상품의 하위 카테고리
- T_4 : 사용자의 쇼핑 상품의 상위 카테고리
- T_5 : 사용자의 성별
- T_6 : 사용자의 나이

	Task	# Users	# Unique labels	Date
T_1	Next Search Query	0.9M	0.58M	07/01/2022 ~ 10/31/2022
T_2	Next Search Query	0.59M	0.51M	11/01/2022 ~ 11/30/2022
T_3	Minor Item Category	0.15M	4K	11/01/2022 ~ 11/30/2022
T_4	Major Item Category	0.15M	10	11/01/2022 ~ 11/30/2022
T_5	Gender	0.82M	2	-
T_6	Age	0.82M	9	-

Sequential Recommendation

Cross-domain Item Recommendation

User Profile Prediction

NAVER와 같은 다양한 서비스를 제공하는 검색 플랫폼에서
검색어를 기반으로 Universal User Representation을 만드는 것은 매우 중요
(이를 기반으로 다양한 서비스에 adapt가 가능하기 때문)

→ 본 연구가 첫 시도!

쇼핑성 검색어 사용

N | 뜻난이약과

네이버쇼핑 다른 사이트를 보시려면 클릭하세요 다른 사이트 더보기

증량 가격

전체 1000g~ 400g~500g

 <p>약과 맛집 뜻난이 모약과 약게팅 달지않은 구운 조청... 19,790원 NPay + 포인트 394원 광고 해피할피</p>	 <p>약계팅 달지않은 뜻난이 모 약과 맛집 구운 조청... 34,990원 NPay + 포인트 698원 광고 해피할피</p>
 <p>[간식]순회정 뜻난이약과 개성모약과 페스츄리 ... 9,900원 NPay + 구매 4,233 리뷰 2,563 순회 정</p>	 <p>장인약과 뜻난이약과 손 찹쌀 호박 쫄득한 꾸역... 최저 14,800원 ★ 3.9(79) 꿀 178 판매처 4</p>
 <p>정선사위 백년가게 시장 옛날 전통 장인 수제간... 16,000원 NPay + 구매 2,348 리뷰 3,238 정선사위</p>	 <p>장인 한과 파지 약과 찹 쌀약과 뜻난이 한팩 29,900원 NPay + 리뷰 2 꿀 76 오-감만족</p>
 <p>장인 명인 60호 안복자 개성 페스츄리 파지 뜻... 8,700원 NPay + 구매 172 리뷰 41 꿀 498 베이직-마켓</p>	 <p>손 찹쌀약과 매작과 장인 한과 대추 뜻난이 파지... 9,900원 NPay + 구매 209 리뷰 667 꿀 930 레이캬트니스풀</p>

네이버는 상품판매의 당사자가 아닙니다. 법적고지 및 안내

쇼핑 더보기 (122) →

쇼핑성 검색어

N | 한국 우루과이

통합 VIEW 이미지 지식IN 인플루언서 동영상 쇼핑 뉴스 어학사전 지도 ...

축구 국가대표 평가전 정보오류 수정요청

대한민국 1 경기종료 2 우루과이 경기기록 하이라이트 2023.03.28.(화) 20:00 | 서울월드컵경기장 | TV조선

하이라이트 득점정보 전력비교



설을 노렸던 클린스
만호, 우루과이전 1-2
2023.03.28. 23:00

'선발 출전' 이강민,
'클린스만호' 핵심 전
2023.03.28. 23:00

① 영상 업데이트가 지연될 수 있습니다.

관련정보 축구 국가대표 평가전 | 오픈톡 참여 | A매치데이

비쇼핑성 검색어

Experiments: Evaluation Protocol

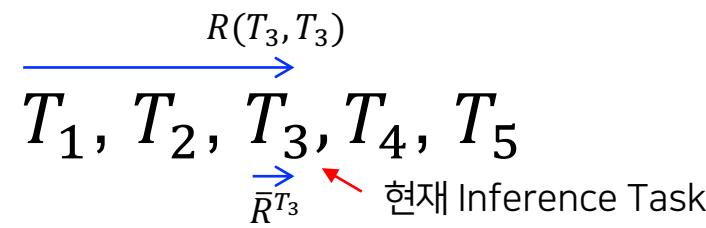
- Each task T_i : Train/Validation/Test data: 80/5/15%
- Metrics
 - Item recommendation: Mean Reciprocal Rank (MRR@5)
 - Item/Query classification: Accuracy
 - Forward transfer (FWT): T_1 부터 현재 task인 T_i 까지 학습했을때가 T_i 만 학습한것보다 T_i 의 성능이 얼마나 더 좋은지?

$$FWT^{T_i} = \frac{R^{(T_i, T_i)} - \bar{R}^{T_i}}{\bar{R}^{T_i}} \times 100$$

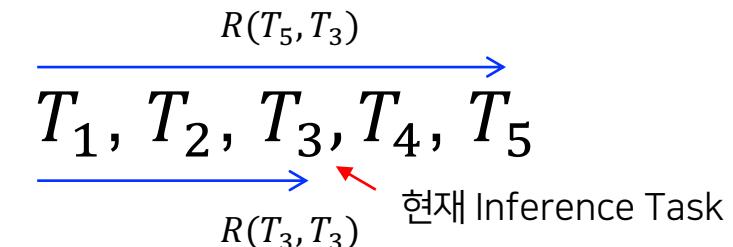
- $R(T_j, T_i)$: T_j 까지 학습했을때 T_i 의 성능 ($j > i$)
- \bar{R}^{T_i} : T_i 만 개별로 학습했을때 T_i 의 성능

- Backward transfer (BWT): 마지막 task인 T_M 까지 학습했을때가 T_i 까지만 학습한것보다 T_i 의 성능이 얼마나 더 좋은지?

$$BWT^{T_i} = \frac{R^{(T_M, T_i)} - R^{(T_i, T_i)}}{R^{(T_i, T_i)}} \times 100$$



Forward transfer (FWT)



Backward transfer (BWT)

Experiments: Overall Performance

	TTL						ML			NAVER Shopping					
	T_1	T_2	T_3	T_4	T_5	T_6	T_1	T_2	T_3	T_1	T_2	T_3	T_4	T_5	T_6
SinMo	0.0446	0.0104	0.0168	0.4475	0.8901	0.4376	0.0566	0.0186	0.0314	0.0349	0.0265	0.0292	0.1984	0.5742	0.2985
FineAll	0.0446	0.0144	0.0218	0.5232	0.8851	0.4596	0.0566	0.0224	0.0328	0.0349	0.0318	0.0332	0.2367	0.6204	0.3247
PeterRec	0.0446	0.0147	0.0224	0.5469	0.8841	0.4749	0.0566	0.0224	0.0308	0.0349	0.0317	0.0322	0.2370	0.6257	0.3258
MTL	-	0.0102	0.0142	0.4672	0.8012	0.3993	-	0.0144	0.0267	-	0.0143	0.0266	0.1372	0.4998	0.2322
Piggyback	0.0446	0.0157	0.0236	0.5931	0.8990	0.5100	0.0566	0.0214	0.0302	0.0349	0.0314	0.0322	0.2349	0.6188	0.3129
HAT	0.0424	0.0174	0.0279	0.5880	0.9002	0.5126	0.0543	0.0227	0.0372	0.0344	0.0356	0.0317	0.2411	0.6294	0.3296
CONURE	0.0457	0.0169	0.0276	0.5546	0.8967	0.5230	0.0598	0.0244	0.0384	0.0361	0.0322	0.0305	0.2403	0.6391	0.3340
TERACON	0.0474	0.0189	0.0316	0.6066	0.9048	0.5386	0.0577	0.0270	0.0459	0.0361	0.0359	0.0337	0.2444	0.6381	0.3354

- Task들 사이에 Positive Transfer가 발생함 (성능: SinMo < others)
- Continual Learning 기반의 방법들이 다른 Universal user representation 방법들에 비해 좋은 성능
- 그 중에서도 TERACON이 가장 좋은 성능 → Knowledge retention 및 Task간 관계 모델링의 중요성

Experiments: FWT/BWT & Task sequence reverse

(a) Original	T ₁			T ₂			T ₃			T ₄			T ₅			T ₆		
	MRR@5	BWT	FWT	MRR@5	BWT	FWT	MRR@5	BWT	FWT	ACC	BWT	FWT	ACC	BWT	FWT	ACC	BWT	FWT
HAT	0.0424	-11.30%	-	0.0174	-7.45%	80.77%	0.0279	-0.71%	67.25%	0.5880	-2.52%	34.79%	0.9002	-1.98%	3.17%	0.5126	-	17.14%
CONURE	0.0457	-	-	0.0169	-	62.50%	0.0276	-	64.29%	0.5546	-	23.93%	0.8967	-	0.74%	0.5230	-	19.52%
TERACON	0.0474	-0.83%	-	0.0189	0.0%	81.73%	0.0316	3.27%	82.13%	0.6066	1.23%	33.91%	0.9048	0.01%	1.64%	0.5386	-	23.08%
(b) Reversed	T ₁			T ₆			T ₅			T ₄			T ₃			T ₂		
	MRR@5	BWT	FWT	ACC	BWT	FWT	ACC	BWT	FWT	ACC	BWT	FWT	MRR@5	BWT	FWT	MRR@5	BWT	FWT
HAT	0.0422	-11.72%	-	0.5025	-4.70%	20.49%	0.8980	-0.33%	1.22%	0.5770	-1.72%	31.19%	0.0269	-0.37%	60.71%	0.0184	-	76.92%
CONURE	0.0457	-	-	0.5322	-	21.62%	0.8849	-	-0.58%	0.5546	-	23.93%	0.0164	-	-2.38%	0.0119	-	14.42%
TERACON	0.0474	-0.83%	-	0.5365	1.84%	20.38%	0.9039	0.93%	0.61%	0.6042	0.07%	34.92%	0.0313	2.62%	81.55%	0.0190	-	82.69%

- CONURE는 Catastrophic forgetting이 전혀 일어나지 않음 (Parameter isolation 방법의 장점) (BWT=0)
- TERACON은 Positive Backward Transfer가 발생 → 새로운 Task를 학습함으로써 과거 Task에 도움
- CONURE는 Task순서에 영향을 많이 받음
- **TERACON은 Task의 순서에 영향을 받지 않는 Robust한 모델**
→ 실제로 task순서가 미리 알려진 것이 아니기 때문에, TERACON이 더 실용적임

Experiments: Noisy Task 추가 (Negative BWT 여부 확인)

Noisy Task

		TTL						NAVER Shopping								
		T_1	T_2	T_3	T'	T_4	T_5	T_6	T_1		T_2	T'	T_3	T_4	T_5	T_6
HAT	0.0411	0.0165	0.0259	-	0.5424	0.8870	0.4873	HAT	0.0314	0.0302	-	0.0309	0.2357	0.6219	0.3180	
	(-3.06 %)	(-5.17 %)	(-7.16 %)	-	(-7.76 %)	(-1.47 %)	(-4.94 %)		(-8.72%)	(-15.16%)	-	(-2.52%)	(-2.24%)	(-1.19%)	(-3.51%)	
CONURE	0.0457	0.0169	0.0276	-	0.5245	0.8663	0.4469	CONURE	0.0361	0.0322	-	0.0291	0.2231	0.6202	0.3122	
	(0.0 %)	(0.0 %)	(0.0 %)	-	(-5.43 %)	(-3.39 %)	(-14.55 %)		(0.0%)	(0.0%)	-	(-4.59%)	(-7.16%)	(-2.95%)	(-6.53%)	
TERACON	0.0472	0.0189	0.0314	-	0.6022	0.9014	0.5312	TERACON	0.0346	0.0336	-	0.0329	0.2378	0.6348	0.3329	
	(-0.42 %)	(0.0 %)	(-0.63 %)	-	(-0.73 %)	(-0.38 %)	(-1.37 %)		(-4.15%)	(-6.41%)	-	(-2.37%)	(-2.7%)	(-0.52%)	(-0.75%)	

(괄호: noisy task 없을 때에 비한 성능 하락률)

- Noisy (Uninformative) Task?
 - 50% 사용자 랜덤 샘플링 후 랜덤하게 50개 클래스 중 1개 부여
 - 무의미한 task가 학습되었을 때도 그 이후 task에 대해 모델이 망가지지 않아야 함
- TERACON 은 Task들 사이에 Noisy task가 들어가 있더라도 성능이 많이 하락하지 않음
 → Negatively-related task들 사이의 Negative transfer를 효과적으로 방지함

Experiments: Sampling strategy

TTL dataset (괄호: Training time-sec/epoch)

	Sampling	T_1	T_2	T_3	T_4	T_5	T_6		Sampling	T_1	T_6	T_5	T_4	T_3	T_2
$\rho_{i,j} = \rho_{min}$	✓	0.0470 (-)	0.0184 (625.47)	0.0280 (77.82)	0.6027 (417.65)	0.9007 (510.80)	0.5385 (414.44)	$\rho_{i,j} = \rho_{min}$	✓	0.0471 (-)	0.5304 (393.53)	0.9010 (491.11)	0.6011 (449.84)	0.0307 (92.44)	0.0188 (743.94)
$\rho_{i,j} = \text{Eq.15}$	✓	0.0474 (-)	0.0189 (625.47)	0.0316 (90.79)	0.6066 (504.3)	0.9048 (583.77)	0.5386 (494.14)	$\rho_{i,j} = \text{Eq.15}$	✓	0.0474 (-)	0.5365 (393.53)	0.9039 (548.92)	0.6042 (517.22)	0.0313 (108.98)	0.0190 (873.98)
$\rho_{i,j} = 1.0$	✗	0.0475 (-)	0.0190 (1146.70)	0.0313 (151.32)	0.6143 (1179.31)	0.9047 (1355.18)	0.5403 (797.09)	$\rho_{i,j} = 1.0$	✗	0.0475 (-)	0.5366 (568.11)	0.9031 (989.42)	0.6104 (1133.54)	0.0311 (197.68)	0.0192 (1673.55)

$$\mathcal{U}_s^{T_i}(j) \leftarrow \text{sample}(\mathcal{U}^{T_i}, \rho_{i,j})$$

Sampling rate
(when sampling U^{T_j} at T_i)

$$\rho_{i,j} = 1 - \frac{1}{K} \sum_{k=1}^K \sigma(c \times \cos(\mathbf{m}_k^{T_i}, \tilde{\mathbf{m}}_k^{T_j}))$$

around 0.007~0.08

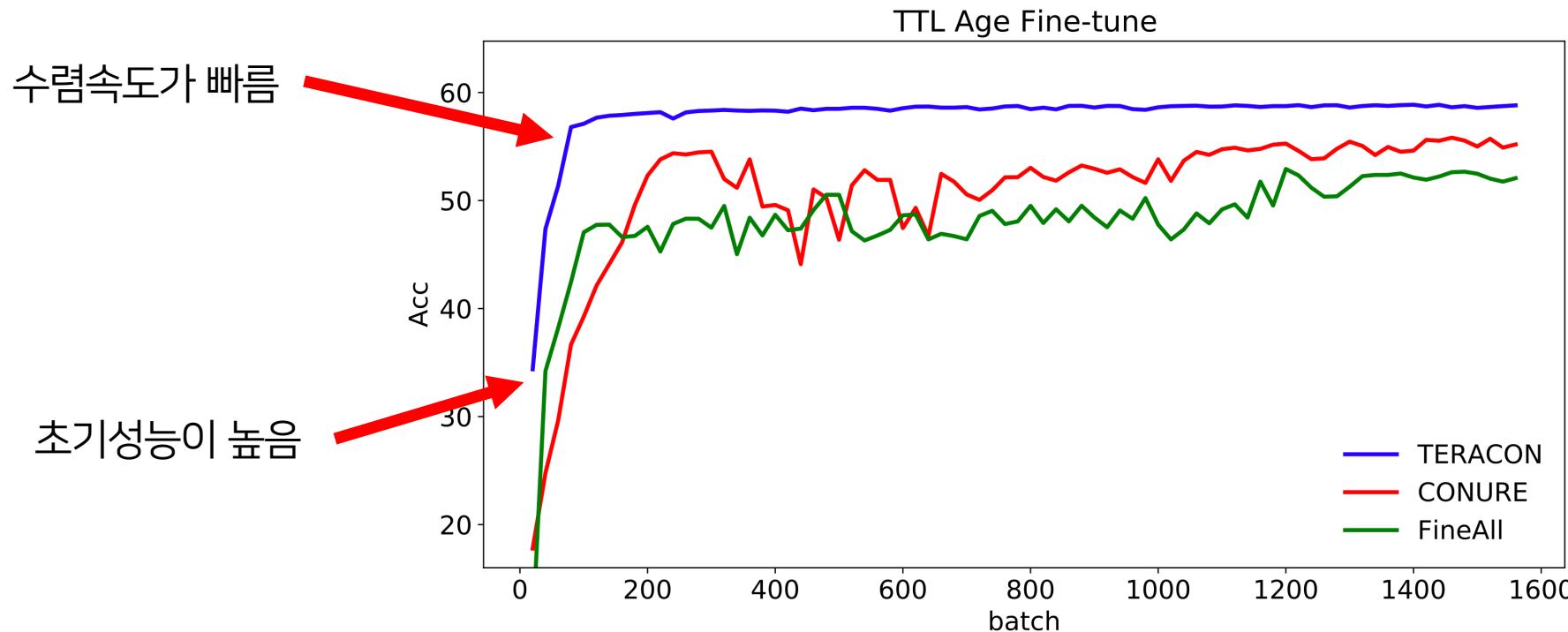
- Sampling을 했을 경우에도 모든 data 전부 사용했을 때와 비교해서 비슷한 성능을 보임 (학습시간 크게 단축)

- 0.7%~8%만의 데이터를 사용

→ Task 간 관계를 기반으로 sampling하는 것의 효과성 입증

- Task 학습순서를 바꾸어도 성능이 robust

Experiments: How well the representation adapts to new tasks



- Setup (TTL dataset): T_1, T_2, T_3 를 차례로 학습한뒤, T_4 에 학습을 시키는 상황
- TERAON의 초기성능이 높음
- TERAON의 수렴속도가 빠름

TERAON이 학습한 user representation으로 초기화 하는것이 효과적

Conclusion

- Universal User Representation 생성을 위한 continual learning 기반 방법론 제안
- Contribution
 - **기존의 parameter isolation 기반 방법의 한계점을 극복**
 - 최근 task일수록 모델 성능 하락 → 최근 task의 성능 보장
 - 학습 가능한 task개수가 제한됨 → 학습 가능한 task개수 무제한
 - **Task간 관계를 고려함 (Relation-aware task-specific mask)**
 - 새로 학습한 Task의 knowledge 가 이전에 학습한 Task 에 도움을 줄 수 있도록 함 (Positive Backward Transfer)
 - Positive transfer 최대화 및 Negative transfer 최소화 (Task 순서/noise에 robust)
 - **Pseudo-labeling 기법을 통해 Catastrophic forgetting 해결 (Knowledge retention)**
 - Relation-aware User sampling를 바탕으로 효율적인 Knowledge Retention 진행

추가 실험: 한국어 임베딩 기반 검색어 초기화

- 검색어를 한국어 임베딩을 활용해 초기화를 하면 성능 향상이 있을까?
 - TERACON 논문: 검색어를 random initialize해서 학습
- KoBERT 활용: 검색어는 일반적으로 문장/합성어로 이루어져 있음 (문장 분류에 사용된 KoBERT 모델을 사용)

KoBERT

Korean BERT (Bidirectional Encoder Representations from Transformers)



KoBERT는 기존 **BERT**의 한국어 성능 한계를 극복하기 위해 개발되었다. 위키피디아나 뉴스 등에서 수집한 수백만 개의 한국어 문장으로 이루어진 대규모말뭉치(corpus)를 학습하였으며, 한국어의 불규칙한 언어 변화의 특성을 반영하기 위해 데이터 기반 토큰화(Tokenization) 기법을 적용하여 기존 대비 27%의 토큰만으로 2.6% 이상의 성능 향상을 이끌어 냈다.

대량의 데이터를 빠른시간에 학습하기 위해 링 리듀스(ring-reduce) 기반 분산 학습 기술을 사용하여, 십억 개 이상의 문장을 다수의 머신에서 빠르게 학습한다. 더불어, 파이토치(PyTorch), 텐서플루(TensorFlow), ONNX, MXNet을 포함한 다양한 딥러닝 API를 지원함으로써, 많은 분야에서 언어 이해 서비스 확산에 기여하고 있다.

- GitHub : <https://github.com/SKTBrain/KoBERT>

검색어 전처리

- 전처리
 - 주방용품 → _주, 방, 용품
 - 스마트팜 → _스마트, 팜
 - Tokenize 하여 얻은 embedding 의 평균 값을 사용
- 기존에 pretrain 되어 있지 않는 data 에 관한 문제를 해결할 필요 역시 존재
 - 검색어의 경우 줄임말/은어/외래어/신조어로 이루어진 경우도 많기 때문
 - e.g., 이케아스투바벤치 → _이, 케, 아, 스, 투, 바, 벤, 치 / 남친룩 → _남, 친, 룩

```
[18] import numpy as np
      result = kobert_tokenizer.tokenize("주방용품")
      print(result)
      kobert_vocab = kobert_tokenizer.get_vocab()
      print([kobert_tokenizer.encode(token) for token in result])
      a = [kobert_tokenizer.encode(token) for token in result]
      ✓ 0.0s
...
...   ['_주', '방', '용품']
[[4213, 3, 2], [2267, 3, 2], [517, 7004, 3, 2]]
```

```
[17] import numpy as np
      result = kobert_tokenizer.tokenize("스마트팜")
      print(result)
      kobert_vocab = kobert_tokenizer.get_vocab()
      print([kobert_tokenizer.encode(token) for token in result])
      a = [kobert_tokenizer.encode(token) for token in result]
      ✓ 0.0s
...
...   ['_스마트', '팜']
[[2931, 3, 2], [517, 7691, 3, 2]]
```

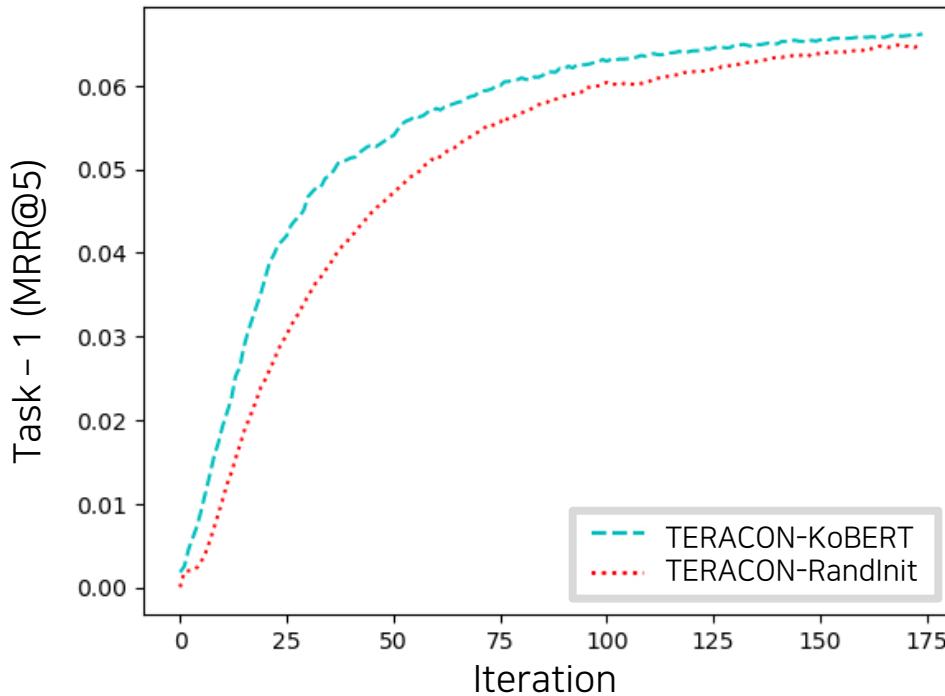
Universal User Representation 결과

	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7
Single Model	0.6597	0.0583	0.0442	0.0398	0.0326	0.4428	0.7661
CONURE	0.6622	0.0721	0.0602	0.0488	0.0424	0.4611	0.7926
TERACON - RandInit	0.6613	0.0754	0.0636	0.0531	0.0458	0.4652	0.8142
TERACON - KoBERT	0.6632	0.0802	0.0649	0.0594	0.0466	0.4651	0.8097

- Backbone: NextitNet
- Continual 방법론이 single model 보다 성능이 우수함
- TERACON이 CONURE보다 우수한 성능
- 한국어 Embedding 을 사용하면, 전반적인 성능 향상을 보임
- 하지만, 기대에 비해 미미한 성능 향상

- Task 1: User Search Query Sequence
- Task 2: User 가 “찜” 한 category 예측
- Task 3: User 가 “찜” 한 상품 예측
- Task 4: User 가 “구매” 한 category 예측
- Task 5: User 가 “구매 ” 한 상품 예측
- Task 6: User 의 “나이” 예측
- Task 7: User 의 “성별” 예측

KoBERT 임베딩의 효과



- KoBERT를 사용하여 initialize 하였을 때, 모델의 초기 성능이 증가함 → 좋은 초기 값의 중요성
 - 하지만, 후속 Task 2, 3, …, 7 의 경우, 학습 속도에서 큰 차이가 없음 (성능에서는 약간의 차이 존재)
- Task 1의 성능이 거의 비슷하기 때문에 (KoBERT: 0.6632, RandInit: 0.6613), Task 1을 마쳤을 때, 학습된 검색어 embedding 은 KoBERT와 RandInit에서 거의 비슷할 것으로 추측
- 비슷한 정보를 가진 검색어 embedding 이기 때문에, 후속 task 에 대한 학습 속도에는 영향을 주지 못할 것으로 추측

Universal User Representation 기반 Cold-start 추천

- Warm user 와 Cold user 에 대한 추천 성능 평가
 - Train 과정에서 사용된 item 수가 2개 미만일 경우 Cold user 로 간주
 - Cold user, Warm user 각각에 대하여 performance 측정
- Backbone: Neural Collaborative Filtering (NCF)

2023 1월

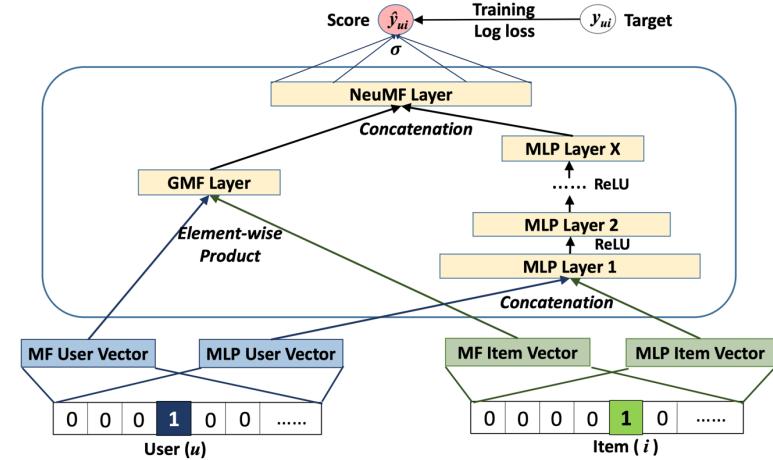
월요일	화요일	수요일	목요일	금요일	토요일	일요일
26		27	28	29	30	31 01
02	03	04	05	06	07	08
09	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	01	02	03	04	05

Task1: (Pretrain)

2023 2월

월요일	화요일	수요일	목요일	금요일	토요일	일요일
30	31	01	02	03	04	05
Task 2 ~ 5						
06	07	08	09	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	01	02	03	04	05
06	07	08	09	10	11	12

추천 모델 적용



Universal User Representation 기반 Cold-start 추천: 결과

	HR@1	HR@5	HR@10
Random Warm	0.0610	0.2687	0.5210
TERACON - RandInit (~Task 4) (Warm)	0.0661	0.3265	0.6447
TERACON - KoBERT (~Task 4) (Warm)	0.0753	0.3490	0.6492
TERACON - RandInit (~Task 7) (Warm)	0.0717	0.3402	0.6487
TERACON - KoBERT (~Task 7) (Warm)	0.0795	0.3578	0.6556
Random Cold	0.0563	0.2514	0.5092
TERACON - RandInit (~Task 4) (Cold)	0.0660	0.3252	0.6386
TERACON - KoBERT (~Task 4) (Cold)	0.0675	0.3320	0.6458
TERACON - RandInit (~Task 7) (Cold)	0.0711	0.3384	0.6420
TERACON - KoBERT (~Task 7) (Cold)	0.0690	0.3360	0.6431

- Warm 및 Cold-start 상황에서 KoBERT 임베딩을 활용하는것이 성능 향상에 도움
- 하지만, 마찬가지로 기대에 비해 미미한 향상
 - TERACON - RandInit일지라도 여러 Task의 학습을 거쳐오면서 충분히 학습이 된것으로 판단

한국어 임베딩 실험 결과 요약

▪ Universal User Representation

- KoBERT 활용: 모델의 초기 성능 증가 및 모든 Task 에 대하여 전반적인 성능 향상
- 검색어에는 줄임말/은어/외래어/신조어 등, 현재 Trend 를 나타내지만 한국어 Embedding 이 없는 단어 존재
→성능 향상을 위하여 해결해야 할 과제

▪ Recommendation

- Universal User Representation으로 상품 추천을 진행하면 더 높은 성능을 기록할 수 있음
- 특히, User representation 을 생성할 수 있기에, Warm/Cold User 에 대하여 성공적

▪ BUT, 두 task 모두 미미한 성능 향상

- RandInit도 여러 task의 학습을 거치면서, 잘 학습된 검색어 embedding 을 가질 수 있기 때문

Acknowledgement

- 이정태 박사님
- 양민철 박사님
- 김동현 박사님

NAVER

- 김세인 박사과정
- 이남경 박사과정

DSAIL Data Science & Artificial Intelligence


Thank you for listening!